Learning in Behavioural Robotics

Edward Llanddwyn Jones

Ph.D.
University of Edinburgh
1998

Abstract

The research described in this thesis examines how machine learning mechanisms can be used in an assembly robot system to improve the reliability of the system and reduce the development workload, without reducing the flexibility of the system. The justification for this is that for a robot to be performing effectively it is frequently necessary to have gained experience of its performance under a particular configuration before that configuration can be altered to produce a performance improvement. Machine learning mechanisms can automate this activity of testing, evaluating and then changing.

From studying how other researchers have developed working robot systems the activities which require most effort and experimentation are:-

- The selection of the optimal parameter settings.
- The establishment of the action-sensor couplings which are necessary for the effective handling of uncertainty.
- Choosing which way to achieve a goal.

One way to implement the first two kinds of learning is to specify a model of the coupling or the interaction of parameters and results, and from that model derive an appropriate learning mechanism that will find a parametrisation for that model that will enable good performance to be obtained. From this starting point it has been possible to show how equal, or better performance can be obtained by using learning mechanisms which are neither derived from nor require a model of the task being learned. Instead, by combining iteration and a task specific profit function it is possible to use a generic behavioural module based on a learning mechanism to achieve the task.

Iteration and a task specific profit function can also be used to learn which behavioural module from a pool of equally competent modules is the best at any one time to use to achieve a particular goal. Like the other two kinds of learning, this successfully automates an otherwise difficult test and evaluation process that would have to be performed by a developer. In doing so effectively, it, like the other learning that has been used here, shows that instead of being a peripheral issue to be introduced to a working system, learning, carried out in the right way, can be instrumental in the production of that working system.

Acknowledgements

I would like to thank my supervisor Chris Malcolm for his help, comments and guidance during my research and the subsequent production of this thesis.

The work described in this thesis, and the thesis itself, have benefited from conversations with, and comments from, John Hallam, Bridget Hallam, Graham Deacon, Myra Wilson, Prahbas Chongstitvatana, Andrew Fitzgibbon, Jeremy Wyatt, and Taehee Kim. The Forrest Hill workshop and computing staff have been of invaluable service in building the pieces of equipment I have asked for and helping to maintain the equipment in the robot lab. Special thanks to Douglas Howie for the photos.

I acknowledge using a variety of software in my experimental system which has been developed and then made available by other researchers: PVM from the Oak Ridge National Laboratory [Beguelin *et al.* 91], the CHIMERA RTOS from David Stewart and others at Carnegie Mellon University [Adv91], and the RTX controller developed here in Edinburgh by Lykourgos Petropoulakis. Many of the illustrations in this thesis were produced using Xfig, whilst Matlab, Mathematica and Gnuplot have been used to process and produce the various plots.

I would like to thank my family for all support, financial and otherwise, that they have given me which has made my time in Edinburgh far more pleasant than it would otherwise have been.

Finally, I would like to thank Fiona for putting up with me for the last few months.

During the first 3 years of the this work I have been supported by Grant no. 91306912 from SERC[1]/ACME.

[1] Now ESPRC

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Edward Llanddwyn Jones
Edinburgh
March 17, 1999

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Since the introduction of the word *Robot* into common language by Karel Čapek in his 1923 play R.U.R, set in and named after the fictional factory of Rossum's Universal Robots, neither academic nor industrial researchers have managed to approach the typically anthropomorphic, highly competent and autonomous robots created by science fiction writers, and more influentially, film makers and their script writers. Current research has branched into developing and applying two types of robots: mobiles and manipulators. Occasionally these two threads have been combined, *e.g.* [Arkin 92, Cameron *et al.* 93], but for the most part, they have stayed separate.

Although research into using manipulators (from now on referred to as *arms*) in assembly began in the 1950's, results so far have been disappointing. There are only a few systems that begin to approach the ideal of the "Production engineer's flexible friend", capable of assembling any given object from finished components [Albus & Evans, Jr. 76]. Systems have generally remained in the research lab where they were developed, though at the time of writing the InFACT system [Loughlin 92] appears to be about to become an exception to this rule. Robot assembly has turned out to be a hard problem, currently requiring costly and time consuming solutions. Outside the assembly domain, arms have only gained widespread use in a few niche applications such as welding and painting (particularly in the automotive industry), pick and place, and unpacking [Critchlow 85]. In many of these applications, successful implementation has required expensive modifications to the environment and the task.

As noted earlier, widespread use of commercial robots is restricted to a few niche

applications. Often these applications are ones where there are strong constraints on using human labour for reasons of health and safety, or quality requirements. This is because commercial arms of useful ability are expensive to buy, maintain, and program. These expenses are due to three problems that have to be resolved in every successful robotics application [Malcolm 92]:-

Variation The world is not regular — parts range in size between tolerances, buildings do not conform exactly to plans, robots change their kinematic and dynamic configuration over time and the environment changes in temperature, humidity, pressure, illumination. All of these things are variations. They can be reduced, but usually at a cost.

Uncertainty One cause of uncertainty is the economic limit of how accurately the environment can be sensed and modelled. In many applications there is a need for data to be acquired from the environment as the task is progressing to ensure correct actions are performed because of the variation in the world. The sensors that measure (assuming it can be measured) this data and/or the processing necessary to transform this data into a computer readable form have finite accuracy. For example, a joint angle sensor may only be accurate to 0.1 radians. This means that for any given reading, the joint will be within ± 0.1 radians of the reported position, *i.e.* there is ± 0.1 radians of uncertainty as to the value of the angle. For some sensors, such as infra red proximity sensors, the information returned by the sensor is a compilation of the environmental features detected by the sensor, in this case the distance to the surface which returned the reflection and the reflectivity of the surface. A weak signal could mean a poor reflecting surface, or a distant target. Such ambiguity is one of the causes of uncertainty. Another cause of uncertainty is when information about the world cannot be measured for reasons of physics, practicality or cost, but must be modelled. When having to act on such information the system is relying on the model being as accurate, as complete, and as relevant as possible. The limits on the model introduces the uncertainty. All these forms of uncertainty can be reduced, but not eliminated, through a variety of methods (Section 2.2, page 15).

Control Controlling an arm is non-trivial, typically requiring tight coupling between

sensors and actuators at least in the lower levels. When attempting to develop high level competencies, such as those required for assembly, it can be difficult to develop the connection between the high level goal and motor actuation and sensor processing. One cause of this difficulty is the need to use side effects. During a manipulation task, such as assembly, the system has to move objects through space in a particular manner, but can only achieve these motions as side effects from the motions of the (typically position) controlled arm. Side effects are difficult things to reason about, and usually require a combination of theoretical knowledge, practical experience, intuition, and luck to be successfully handled. A second cause of control problems is the complexity of the task to be handled. Research into path planning for arms has frequently been halted by the computational complexity of the problem [Latombe 91]. One way to handle this complexity is to absorb it at as low a level as possible in the controller, effecting to the user of that controller a virtual machine that can be used without knowledge of how it implements and reliably achieves the functionality it provides, an approach which VLSI designers have long followed, *e.g.* [Mead & Conway 80].

To date no general solutions to these problems have been developed. It has been possible to produce robots for specific applications, especially where the environment is highly constrained, or 'simple', as is the case with spot-welding [Critchlow 85], but generally these solutions are costly and brittle. Most of the research to date has succeeded in only showing the degree of difficulty inherent in these problems, and whilst the current collection of research systems remain costly, difficult to develop, unreliable and complex, they will remain in their home laboratory. Although mobiles suffer from these problems, it is arms that suffer most because of their physics and the tasks to which they are applied. This thesis concentrates on arms, as here the problems listed above are more noticeable, and harder to deal with because of their complexity.

Although no general solution has been developed, the approaches and ideas that have been suggested often have one thing in common: They focus on the robot's controller as being the place to develop the necessary competencies of system, *e.g.* [Malcolm 87, Lozano-Pérez *et al.* 90].

Figure 1.1 depicts, at a very abstract level, the architecture of the control software of

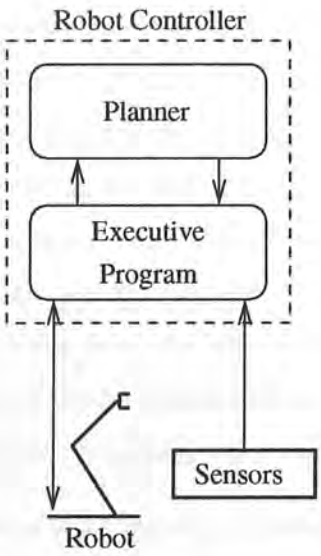

Figure 1.1: The general architecture of robot systems

most assembly robots. The planner takes a specification supplied by a user and works out a program (called a plan) that will cause the robot to perform the elements of the task in the correct order. When this plan is completed, it is sent to the executive component, which is the system controller. The executive program then implements this plan, performing the actual control of the robot, as well as the processing and interpretation of any sensor readings. Depending on the implementation and the background theories to that implementation, the executive program may send back information to the planner or it may not. In most early robotic systems (which are typically referred to as *Classical systems*), it was assumed that the planner was the most important part of the system, as it was the location of the intelligence in the system, with the sensors and robots being mere peripherals.

In the last 10 years an alternative approach has been developed by researchers dissatisfied with the classical approach [Brooks 85]. They have argued that it is hardly surprising most classical systems have not lived up to expectations as they are based on an unrealistic view of the form and location of the "intelligence" in a system, *e.g.* [Malcolm 87]. The new *Behavioural* approach draws heavily on ideas about the nature of biological intelligence. The approach, and resulting *Behavioural systems*, aims to integrate sensing and action at a very low level, with the intelligence of the system being viewed by 'purists' as being the result of the coupling of the capabilities of the

executive program with the environment and task. A typical executive program of a Behavioural assembly system would be constructed in both a bottom-up and top-down fashion from simple components. These are combined in a synergistic fashion to effect a virtual machine that as a whole is capable of performing more sophisticated behaviour than can any one of its components. Each component implements some specific task achieving behaviour of the system and will be referred to in this thesis as *activities* to emphasise the important distinction from the information processing functions of the classical systems. The question of the usefulness and scope of the planning component in such systems are still subject to discussion, *e.g.* [Brooks 91b, Malcolm 92].

Whilst these new systems share in some respects the same general form as the classical systems, their differing philosophical basis, methodological requirements, and construction techniques mean that their implementation and performance are very different. The approach places strong emphasis on the use of real robots applied to real tasks in natural environments. Whilst this approach is relatively unproven (widespread research only began in the mid to late 80's following the release of [Brooks 85, Brooks 86]), it has a philosophical base that does not suffer the weakness of the classical systems, and early results have been promising, *e.g.* SOMASS [Malcolm 87], albeit operating in a limited domain.

Activities are considered here to have two parts: the *goal*, and the subsidiary activities which the robot performs to achieve that goal. There can be several activities that can be used to achieve one goal. For example, to place an object onto a surface (the goal), the agent might drop it (activity 1), use visual servoing to guide the hand to the right position for placing the object (activity 2), or use a force torque sensor to monitor the forces produced as the hand lowers the object; when the forces are right, then the object is on the table and it can be released (activity 3). Each activity has its own pre-conditions and consequences, each might use a different sensory preconditions and/or actuator subset, each performs differently, but all achieve the same goal. A program is made up of a collection of activities whose goals are the subgoals of the task, with the activities of the modules being the things the robot does.

Within any one such activity two kinds of computation can be identified:

- deciding what to do (the *decision function*), and
- deciding how to do it (the *action-sensor coupling*).

The developer, be that a person or an AI system, has to produce and structure these activities. Generally, this has been found to be best done with a particular application in mind [Deacon 92], though one of the central ideas behind Behavioural systems is that the application for which the system is being constructed is the most general one, *i.e.* the assembly task rather than the assembly of a single product.

This definition and specification of activities has been deliberately chosen to reflect a particular design philosophy (see Chapter 2, page 11) and to reduce the risk of a conceptual conflict. Activities, as used here, are more commonly known as *Behavioural modules*, *e.g.* [Brooks 85, Malcolm 87, Wilson 92]. However this label, though commonly used, rarely refers to the same conceptual or implemented mechanism. Brook's activities are based upon an augmented finite state machine [Brooks 85] whilst those of Malcolm are closer to Pascal procedures [Malcolm 87]. By using this new label to refer to the different conceptual and implementation mechanism proposed here is it easier to identify and maintain the distinction between this work and preceding research.

This thesis addresses the question of how and why learning can be integrated into such a Behavioural robotic system to improve its reliability and reduce the development load of a human programmer or an automated planner without requiring large resources or prolonged training. The method followed to investigate this question is to study a complete agent and how that agent can be programmed to achieve a particular task. This approach will allow identification of where learning can be used.

In this thesis learning is defined as those changes that lead to an agent choosing and performing behaviours so that overall it achieves its tasks with greater speed, efficiency and reliability that results from, and is best explained by, the agent's long term history (multiple execution of one or more activities) of interaction and experimentation with the world. An example of learning would be choosing to use one search strategy over another because the history of use indicates that it is superior to the alternatives.

The motivation for this question has been that whilst the Behavioural approach has been inspired by biology, [Brooks 85], it has so far been selective in its sources and has generally ignored the use of different learning methods by some animals to achieve parts of their operational abilities [McFarland 93]. In ignoring the varied forms of learning, or treating it as an optional extra, an opportunity to obtain more useful robot systems is being missed. Animals have to deal with all of the problems noted earlier on page 2, and like artificial sensors, biological sensory mechanisms do not have infinite accuracy, are not identical between different systems, and change over time, *e.g.* the yellow shift that occurs in human colour perception as the individual ages. Biological control also has to cope with a changing anatomy and physiology, as well as be able to handle variable objects (think of all the different sizes of chips in your average fish supper). Also, like assembly systems, animals can often only change the state of the environment through side effects, *e.g.* turning a switch to illuminate a room.

There could be said to be two types of learning mechanisms that animals use to cope with variation, uncertainty and programming problems [Huntingford 84]:-

Activity Tuning: This takes place when the agent is able to perform (in some sense) the activity, but needs to adjust its performance to achieve a new goal or to handle an environment change that has caused a performance degradation.

Activity Invention: If the agent is unable to tune an existing activity to meet a goal, then a new activity must be invented. When invention takes place, the degree of innovation as opposed to reuse is dependent on the agent's activity invention mechanism, the environment, and the novelty of the goal. For example, a robot that was able to assemble a car alternator would be able to reuse all of the activities involved in that task when inventing an activity that achieves the assembly of a distributor. This is because the two goals can be achieved by exhibiting similar behaviours, though perhaps in a difference sequence [Nevins & Whitney 78]. However, if it were required to invent a brick laying activity, it would be necessary to innovate some new activities.

To date even most robotics researchers who argue for the study of complete systems have not begun to explore the use of the full extent of the biological processes

and mechanisms for coping with the world. Instead, control mechanisms centred on parameterised models (with the developer working out both model and parameterisation) are the most widely used means of implementing activities *e.g.* [Wilson 92, Pettinaro & Malcolm 95]. Some robotics researchers have introduced biologically inspired learning mechanisms with varying results, *e.g.* [Albus 75, Nehmzow 92].

This thesis will focus on the first form of learning, beginning by considering how to automatically tune models (and seeing whether models are needed at all). The point of this exercise will be to assess the usefulness of the weak learning mechanisms that can be used to do this. If these mechanisms are applicable, then one potential benefit, is that if the tuning need never stop, the agent would be able to continually modify its behaviour to respond to changing circumstances. In doing so, the learning offers a level of redundancy that would otherwise have to be hardwired into the system. If this redundancy through learning is sufficient to cover all eventualities, but without requiring the specific hardwired cases, then it will be possible to assess whether it offers a simpler solution.

1.1 Overview

This thesis describes the exploration of the idea that learning can be useful in robotics to improve the reliability without adding to the workload of the developer. Following on from this introduction are a number of chapters, each of which documents a stage of the exploration with the chapters ordered in sequence:-

Chapter 2: The robot controller has been where a lot of the research into producing reliable systems has been directed. This chapter reviews and structures that research to provide the necessary framework to support the development and experimental work described in the following chapters.

Chapter 3 This chapter contains the identification of where, and why, different types of learning should be used. From this starting point weak learning is identified as currently being the most applicable to assembly robotics before then going on to describe the profit functions that are an important part of such mechanisms.

Also during this chapter the properties that the implementations of any learning should exhibit for them to be considered useful are described.

Chapter 4: This chapter sets out the problem, namely how to make the robot arm reach out to a position specified in the coordinate frame of some sensor and details the experimental apparatus that has been used. The chapter then goes onto to describe the details of the equipment and the methodology used to evaluate the various learning activities that will be described in the following chapters. During these description explanations are given for why the problem and the robotic system are suitable.

Chapter 5: The action-sensor coupling of a reaching activity outputs the position changes required in the arm to move it to a selected sensor specified position. This initial consideration of how learning can be used in action-sensor couplings describes how by wrapping a weak model of the sensor arm relationship inside a feedback loop and then using the information gained from test moves to modify the model to be a closer match to reality (the learning), an effective reaching activity can be produced. The basic algorithm implemented in the action-sensor coupling is called QUEASY, and its variants are successfully tested using the position targets identified by the camera, or the force tray, without internal modification.

Chapter 6: The preceding chapter describes how learning is used to find good model parameters for a reaching module. In this chapter modules are proposed, designed and tested that learn good parameter settings by using optimisation methods to approximate the action-sensor coupling. Using these optimisation methods removes the need for an explicit model and all the associated work. The details of the developed mechanisms are discussed, and two different experimental results are presented.

Chapter 7: Some activities and the parameter tuning mechanism can benefit from storing and reusing past experimental results. This 'memory' can beneficially be used actively by interpolating or extrapolating from known results to suggest suitable parameter settings for obtaining a desired result. The learning here is learning of the actual relationship between the sensor and the motors of the robot,

but is done without requiring the developer to specify anything about the relationship. The results of modifying the QUEASY servoing algorithms (Chapter 5) to use this idea are presented as experimental verification.

Chapter 8: The last three chapters have focused on using learning in action-sensor couplings. This chapter describes how learning can be used when it is necessary to choose between activities that have similar goals, but different performance characteristics. The ANNIE algorithm, on which this chapter is centred, has been developed from the statistical ideas of confidence intervals to be able to learn which activity is best. The algorithm shares some features of the work on learning in action-sensor couplings in that it can learn without knowledge of the activities it is picking or the goal it is trying to achieve. Two experimental results are described which verify the algorithm and show its effectiveness.

Chapter 9: This final chapter is a review of the thesis, along with a discussion of the results and some suggestions for future research.

After the main body of the thesis there is a bibliography and then three appendices. These describe in turn the various random number generators that have used in the course of this work, and the mathematics and the filtering required to convert readings from the force sensor buried in the force table into estimates of the location of the point of contact of an object with the surface of the table.

Chapter 2

Robot Controllers

Any useful robot system should be:-

Robust Every task assigned to the robot that is within its specified capabilities should be achieved. This means that the robot will handle any uncertainty and variation (defined on page 2) in the robot, environment and task up to a defined level.

Flexible When the task changes it should not be necessary to have to redevelop the system, or any part of the system from scratch.

Economical If used for a commercial purpose, the robot system should produce a positive return on investment over its life. To achieve this, the capital investment and operating costs should be less than the cost of not using the robot.

A purchaser of such a system will not be forced to invest the time and money currently required to implement these attributes. Regardless of who implements them, such system attributes, as will be argued in this chapter, are largely the result of the skill of the software based controller. This entity is the combination of an executive agent and a planning system that converts the user specification to actions. The controller is also the place where any learning as defined in Chapter 1, and some of the adaptiveness of the system resides.

It could be said that one of the goals of AI research in assembly robotics has been to develop the control programs that can replace the skilled and experienced programmers that are currently required, *e.g.* [Lozano-Pérez *et al.* 87, Lozano-Pérez *et al.* 92]. To

date the most significant results have been in finding out how hard this can be, *e.g.* [Lozano-Pérez & Brooks 85], with many systems unable to handle the complexity of real applications.

Much of the existing robotics research relevant to this work is often classed as belonging to one of two paradigms: Classical and Behavioural. These were introduced in Chapter 1 on page 4. Both paradigms aim to provide solutions to the various problems in their own way. At the extremes, there are large differences between the paradigms. Some researchers have found it profitable to combine the two approaches, taking the best of each, *e.g.* [Hallam 91b]. Whatever the approach, the goal is the same: a task level robot programming system (defined in Section 2.1 on page 14).

The purpose of this chapter is to present an overview of the development towards this goal that provides the background for the experimental work reported in this thesis. As such, the objective is to provide the necessary conceptual and historical framework that is used in latter chapters where applications of learning are identified and developed. This overview is presented in three sections:-

- A description of the types of robot controllers that have been proposed or developed in order of increasing sophistication.
- A review of methods that have been developed for handling uncertainty, *i.e.* of activities
- A review of how activities can be structured.

2.1 Types of Robot Systems

This section reviews the types of assembly robot controllers, ordered by the sophistication of their user interface. This sophistication is judged by how well the uncertainty and variation handling mechanisms are hidden from end users and the degree of parity between a human interpretable and a controller interpretable description of the task. This list is not new, as it follows that of [Malcolm & Fothergill 86, Malcolm 94], nor is it the only one which is generally valid. However, for the purposes of the research described here it has been the most useful system for comparing and succinctly describing

different systems. This list is also useful here because it introduces and defines some of the terminology used later. Although the list is primarily structured in order of increasing sophistication, it is also in order of the sequence of development of assembly robot controllers from their beginnings up to the present.

2.1.1 Joint Level

These are the simplest controllers. Programs are specified as a long sequence of positions and speeds, and possibly accelerations, for each joint of an arm, perhaps with acceleration changes. Programming the robot at this level is very similar to programming a word-processor in hexadecimal in terms of the relationship of the program statements to the overall task. As such it is a very difficult and highly error prone activity with the resulting programs being difficult to reuse.

2.1.2 Manipulator Level

The embedding of a kinematic, and possibly a dynamic, model of the robot into this level of controller allows programs to be specified as sequences of end effector positions (usually in cartesian coordinates relative to some base frame). This simplifies the programming because it is easier to understand and describe motions. Often systems at this level are provided with computer languages that include control, assignment and processing facilities, *e.g.* the ability to index through relative spatial offsets. These facilities ease the work of the programmer. Even so, the resulting programs do not clearly resemble the overall task and can be difficult to develop, and debug. This is because the programmer is effectively working through side effects as all the robot's activities are still specified as robot motions (as with Joint level programming). This is the level of interface and control provided with most commercial and research robots, with languages such as VAL–II perhaps being widely used for arms. Controllers at this level often act as the basis for the higher level controllers that will be described in the next paragraphs.

2.1.3 Feature Level

At this level controllers are programmed by specifying the geometric relationships between the features of the components to be mated in assembly. Programmers need not be concerned with how the robot has to move to achieve these relationships. Instead the controller uses its inbuilt knowledge to deduce the required motions of the robot from the program. The RAPT programming language best typifies this level [Popplestone *et al.* 78]. Here, knowledge of the component's geometry is used in a sophisticated geometric reasoning system to parameterise the motions of the arm. If RAPT is required to reason about uncertainty and variation [Fleming 87], then the computational requirements are such that the system becomes impractical.

2.1.4 Object Level

This is the first level where the programs can be specified in terms of relationships of the whole parts, rather than in terms of the positions of the robot, or relationships between features of the parts. Programs can now be specified as 'place piece A onto piece B then insert peg'. This level of interface to the controllers is closer to how humans specify tasks to one another. This makes the programs easier to develop and debug. At this level some form of planner is required to deduce the appropriate robot motions to achieve the specified part motions or spatial relationships. The planner might also have to establish what sensory information is required to ensure successful completion of the task. Off line programming at this level becomes realistic, as programmers can work in an ideal geometric world without concern for the practicalities of the task. No clear example of an object level controller is currently known. However, one can be implemented in RAPT, and Task Level Systems (covered next), such as Handey [Lozano-Pérez *et al.* 87], could be said to contain an object level controller.

2.1.5 Task Level

Humans can follow instructions such as 'assemble this object from these parts'. A controller at this level is capable of performing tasks specified in this manner. AUTOPASS [Lieberman & Wesley 77] and LAMA [Lozano-Pérez 76] were two of the

earliest task level systems, though they were never finished. More recent task level systems such as HANDEY [Lozano-Pérez *et al.* 87] and SOMASS [Malcolm 87] are able to plan and then execute procedures for the single task of picking and then placing objects into a specified configuration. EDDIE [Deacon & Malcolm 94] is perhaps the closest of all these systems to having a true task level controller. It can be instructed to perform a variety of tasks, each being performed by a program that is invariant to the details of the task.

2.1.6 Summary

Task level robot controllers are the target for much of the current research in this area. As work progresses towards this goal the executive programs start to combine disparate elements of the low level, heavily mathematical, control software of the lower joint and manipulator levels with symbolic and geometric planners that incorporate knowledge about the robot, the environment and task. The detail of this work has turned out to be heavily dependent on the underlying mechatronics of the system, as to implement a particular functionality, appropriate action-sensor couplings have to be established. These cannot be made without reference to the hardware.

2.2 Management of Uncertainty

Developers have often handled uncertainty by linking together acting and sensing at as low a level as possible to effect a virtual machine that conceals some, or all, of the uncertainty from the next higher level. In addition, developers have used two other techniques: engineering the environment, and the application of various filtering techniques to reduce the noise in a sensor output. These two approaches will be reviewed first.

2.2.1 Engineering

Mechanical hardware approaches to reducing uncertainty generally do not impose any extra computational load onto the system. Engineering the uncertainty out of the environment is still a popular approach, if for nothing else, because of its success

rate. Nearly all developments in this are application specific and so add to the initial setup costs. Examples are widespread in assembly workcells with parts feeders being constructed to reject parts that exceed specified tolerances (reducing variation) whilst fittings and jigs help to constrain the location of objects, reducing uncertainty. Recently a number of approaches, *e.g.* [Brost & Goldberg 94, Whitney 94] have been developed for automatically designing mechanisms that use of results from constrained motion planning. However, as any design has to be realised in metal or plastic, flexibility of the system is limited. Adaptive mechanisms that can change their layout under program control [Appleton 90, Law 94] would restore that flexibility, but as yet, such systems are still research tools.

An alternative to engineering the work environment of the robot is to equip the arm with a *Remote Centre of Compliance* wrist device [Nevins *et al.* 84]. These devices have been found to be highly successful when used in part mating operations, as long as the tasks they are used for do not require non-linear compliance or large corrective motions.

However, whenever the environment is engineered, it has to be done with a specific task in mind. Even with automated means, this can still be a time consuming activity. If new fixtures have to be designed and manufactured, then the cost (financial and time) of changing between tasks can be considerably increased. A further problem with this approach is that when environmental shortcuts are available and used, the resulting system controller can have only a limited generality, *e.g.* SOMASS[Malcolm 87]. Whilst such systems can be easier to develop than a truly general system, there is usually a strict limit on the range of tasks to which the system can be applied.

Hardware approaches to reducing uncertainty should never be forgotten, especially if there are particular sources frequently encountered. After all, they do not have any computational cost and can remove a large number of problems with a relatively small amount of effort and cost, *e.g.* using a chamfer designed to cope with worst-case misalignment [Balch 92b].

2.2.2 Certain Sensing

Control theorists have developed a number of statistical approaches to handle the uncertainty present in sensor data, *e.g.* [Kalman 60, Julier & Uhlmann 94]. These approaches all aim to provide optimal estimates of the parameters of a mathematical model of the relationship between the desired information (*e.g.* the true position, velocity and acceleration of a car in the world) and the noisy, actual information available (a sequence of position estimates of the car in the image from an overhead camera). These models are then used to predict future values for the desired and actual information. Optimality is defined in this setting as being the parameter settings that minimise the difference between the predicted and the actual values next detected.

The information to sensor relationship can be simple, or it can be complex, involving several sensors sampled at differing times. There are a number of these filters which vary in their sophistication and suitability for various problems. Of these the Kalman filter is the most general, and perhaps the best known. These filters were nearly all developed for application to linear systems. They can be applied to the highly non-linear systems which frequently occur in robotic systems, although such applications usually require a degree of experimentation to determine the best underlying model and its initial parameterisation. Another problem is that the greater the approximation of true reality by the mathematical model, the harder it is to be confident that the results from the sensor processing are the correct results. These filters are statistical tools. Like any such tool, their results are not ones that can be fully trusted whenever the assumptions that lie behind these tools are violated. That the tools still work despite these violations is a testament to their power, but that power has to be used with caution. The application of the Kalman Filter in the Force table processing that is described in Appendix C (page 247) is an example of this approach. It is also an example that shows some of the limitations of filters.

2.2.3 Action-Sensor Couplings

The point of linking sensing and action is to provide the system with an operation (the acting) that can achieve a result (the sensing). The reason why this operation has been

termed an action-sensing coupling is that the aim of using the operation is to achieve a certain, detectable relationship through acting in the right way. The mechanism that implements the operation has to work out how to act to achieve that relationship. If there is no acting, then there is no way for that relationship to be established, hence the name. The determination of what action to perform will often require some initial sensing, but after the first action, the difference between the current and the target situations can be used to guide further acting. The usefulness of such coupling has long been identified. The MH-1 system included force guarded and compliant motions as atomic operations [Ernst 61]. Neurophysiological studies have frequently identified neural circuits linking sensors to muscles, one notable case being the mechanism used by some moths to avoid capture by bats [Roeder 70]. However, knowing that coupling action and sensing is good is one thing; Implementing it is another. Three types of schemes can be identified for developing implementations:-

- those that do not use sensors in the coupling, *e.g.* [Malcolm 87]
- those that are customised to fit a given application through automated reasoning, *e.g.* [Low 92]
- those that have to be hard wired by a developer, *e.g.* [Wilson 92].

In all cases, when a coupling is implemented, it is done with reference to some goal that it is designed to achieve. As a result of this, couplings are not generic because to achieve a particular goal reliably it is necessary to take into account the abilities of the equipment which will be used. This point will be revisited in Section 4.2, page 70, where a number of examples are given.

The next sections look in turn at these various options for developing activities.

2.2.4 Acting without Sensing

Sensors do not have to be used [Mason 93]. It is always worth remembering that in addition to the information provided by sensory systems at any one moment, a robot can access (if mechanisms are provided) [Mason 93]:-

- Information on its motor history.
- Advice from an external agent.
- Estimates on any future information.
- Explicit data structures as well as the information implicitly embedded in the design of the robot and executive program.

Sometimes these sources of information alone are sufficient to provide the necessary robustness without any external sensing being required. Many of the environments in which assembly robots currently operate have very low amounts of variation. The variation that does occur can usually be attributed to equipment failure, and hence is outside the control of such systems (though not their monitoring facilities). Such reduction is usually accomplished through careful engineering of the environment to produce the specialist systems that can be considered analogous to role-specific insect morphologies, making the systems inflexible and expensive. Even these engineering solutions are not static, and they do not remove all the uncertainty or variation that has to be handled.

However, such mechanical devices are not the only way to hide uncertainty and variation without sensing. One method that has been found to be very successful for acquiring and retrieving parts from initially unknown locations on a flat surface is to plan a sequence of robot motions that will sweep a part into a known orientation. The only limitation of this work is that these methods are not able to resolve symmetrical orientations. This planning requires modelling the interaction of the part with the environment as it is pushed around that environment. These methods originated in the work of the following four individuals [Deacon 97]:-

Mason [Mason 84], who researched constrained motions.

Erdmann [Erdmann & Mason 88], who looking at motion planning with uncertainty.

Brost [Brost 86], who looked at grasping strategies.

Mani [Mani *et al.* 85], who examined the use of fences that could either be held by a

robot with the part on a table, or attached to conveyor belts on which the part is placed.

Peshkin and Sanderson [Peshkin & Sanderson 87, Peshkin & Sanderson 88] have developed this line of work by showing how to calculate the longest sweep required to achieve a particular orientation. However, this calculation can fail in use [Deacon & Malcolm 94]. More recently Goldberg has extended the classes of parts for which this planning can be performed from purely polygonal ones to those that which can be described by algebraic curves [Rao & Goldberg 93]. Deacon and Wright [Deacon & Wright 95] have gone on to show how to plan pushing sequences for any shape of part. In their pure form, such strategies currently rely on the fence being of infinite length and the robot capable of infinitely long pushes and sweeps. However, it has not been difficult to develop a full implementation that does not require these assumptions, yet which works nearly all of the time [Low 92]. These methods do not require actual sensing to take place, and usually have high reliability. They do require that the initial assumptions used to develop the plans hold true, and that the parts are initially within the expected areas. Their main use is in an assembly situation, but they can be of relevance whenever an object has to be manipulated by being pushed, *e.g.* a mobile robot pushing a box [Lynch & Mason 94]. Research in this area is ongoing.

These schemes have one important advantage, if it is an advantage, over those that will be described next. There is no way of determining when they have failed during use. This is intrinsic to these schemes, as it is assumed that such an event will have been avoided. This removes the need to be concerned with how to handle errors, an advantage Malcolm made full use of in SOMASS to simplify the system [Malcolm 87].

2.2.5 Automated Synthesis

Developing activities is not easy, with times in excess of 100 hours being reported to successfully implement simple activities such as placing a brick on a flat surface, *e.g.* [Balch 92b, Balch 92a]. Consequently there has been a lot of effort made to develop procedures, which can automatically synthesise a novel activity to meet a task specification. For both of the methods described here this synthesis is performed as part of

the development of a complete execution component that will perform some specific task, and so is only part of the planning activity.

The synthesis process works through a sequence such as:-

- The outline is used to specify the goal of each activity.
- Using the goal the space of couplings is searched until one is found that will let the robot achieve that goal. This search can often be performed directly, but finding good solutions can be hard.
- Once a suitable coupling has been identified, it must be instantiated for this situation in the final step of the synthesis.
- With the synthesis of the coupling completed, the viability of the coupling within the current plan should be checked and any future impacts or earlier conflicts noted.
- In the event of a problem, then the coupling should be revised or the plan modified. Otherwise, the process can be repeated for the next coupling to be developed.

Many of the representations are geometric and are based around a particular representational device from Classical Mechanics called *Configuration Space*. This was first used in robotics by Udupa [Udupa 77]. Lozano-Pérez [Lozano-Pérez & Wesley 79] subsequently developed the generalised form and named this approach. Often used in path planning, it allows environments to be modelled in a way which lets users ignore the shape of the object for which the path is being planned. This is done by shrinking the object to a point, and expanding the obstacles proportionally. This expansion can include tolerancing information. The result of this is that path planning becomes the mathematically simpler task of moving a point through a cluttered environment as opposed to that of moving an object through obstacles. However, arriving at this general case can be very hard if the robot or objects cannot be modelled as polyhedra.

Configuration space modelling technology has been expanded to include compliant fine-motions [Lozano-Pérez & Brooks 85], as well as the effects of forces and moments

[Erdmann 84]. Articulated objects such as robot arms and multiple moving objects can both be represented. It is also possible to specify kinematic constraints on the objects and environments modelled [Latombe 91]. In configuration space it is also possible to model the variation in object position (in the case of assembly, the arm configuration), as well as variation in the environment positions, simply by adjusting the way the obstacles are expanded. Although the geometric reasoning described here does allow automated reasoning about the uncertainty and variation arising from sensing and control errors, it is just a way of modelling the environment. Unless that model can be correctly developed, the plans obtained from reasoning based around the model will be incorrect. The technique is computationally intensive, but as the model can be produced off-line from geometric plans it can be useful especially for task level planning.

The method for describing couplings that is most associated with configuration space is that of *Strategy Skeletons*. These are partially defined stereotypical descriptions of the short sequences of small motions. Taylor [Taylor 76] developed a way of completing the details of the strategy skeletons using error estimates determined by propagating the effects of errors, uncertainty, and variation through the model. Lozano-Pérez [Lozano-Pérez 76] independently developed a similar approach. Brooks [Brooks 82], before his move into the Behavioural approach, developed this method further by replacing the numerical propagation techniques with a symbolic method. He also introduced a plan checking system based around a geometric database. The system used the database to determine the effects of actions and to propagate errors backwards through the plans in order to determine which parameters of the model were critical to the overall performance of the system.

Lozano-Pérez, Mason and Taylor have developed fine motion further, with their scheme (known as *LMT* after its developers) [Lozano-Pérez & Wesley 79]. Strategy skeletons are used to select motion parameters with each motion being expressed in terms of the final relationship of the parts during the task [Lozano-Pérez *et al.* 84]. Under this scheme, guarded motions and termination conditions could be determined from the constraints imposed by the relationships. Strategy skeletons are not very useful for describing programmed compliance (force or some other sensor) motions, such as

discussed in Section 2.2.4 on page 18, as they attempt to make the robot achieve an ideal trajectory, handling the uncertainty by knowing accurately where the robot and the surface against which it is to be compliant are located. However, unless the compliant motion is very short, the cumulative uncertainty that accrues makes it very difficult for the planner to have confidence that the strategy skeleton with that compliant motion would be successful. As many assembly tasks include long compliant motions, this approach is not ideal, as there will be many situations when a skeleton cannot be fully defined.

Related to this idea is RAPT [Popplestone *et al.* 78], a feature/object level programming system. Tasks are specified as sequences of geometric relationships between identified features of parts. From this, a sequence of robot positions can be derived that will cause that sequence of part relationships. As long as all the necessary information was available, and remained true and sufficient detail was specified, then RAPT was successful. RAPT has been extended to include the kind of action-sensor couplings that have been discussed here [Yin *et al.* 84]. However, when uncertainty had to be reasoned about, like for configuration space approaches, computational complexity grew rapidly. During Fleming's demonstration of how the geometric variation of objects could be represented within RAPT using tolerance zones [Fleming 85a, Fleming 85b] (which was a development of work by Requicha [Requicha & Tilove 83]) the implementation showed the full effects of his theoretical results. Efforts to restore the tractability of RAPT have focused on changing the underlying representational framework from simple geometry to topological Euclidean groups. This, amongst other advantages, does not require symmetric rotations of objects to be handled individually. The effect of this change is to remove many of the cases in which uncertainty was having to be handled, though production of couplings to achieve a task does become more complex [Popplestone 83, Lin & Popplestone 95].

Other, more modern approaches to geometric motion planning include the use of harmonic functions *e.g.* [Connolly *et al.* 95], or potential fields, *e.g.* [Khatib 95]. Unlike the above approaches which aim to produce customised activities, these approaches use a generic activity to which a trajectory, and error measure for deviation from that trajectory are supplied. These methods still suffer when uncertainty has to be dealt

with, but are able to handle it far more effectively. However, they cannot be easily used when the robot is required to operate close to other objects within the resolution of its uncertainty [Latombe 91].

One alternative approach to those described above for implementing a coupling when compliance is required is to use *admittance* matrices. These are "embedded" in a feedback loop from the sensors to the actuators and are used to modify the robot's velocity in a manner which will achieve the specified task. This scheme for implementing activities is known as *Admittance control.* Unlike the approaches so far discussed, it does not become overly complex as the level of uncertainty grows. Peshkin [Peshkin 90] has developed an approach for synthesising an admittance matrix for coarse grained force controlled assembly tasks from geometric descriptions of the objects and estimates of the uncertainty, but has been unable to implement it due to "noise and bandwidth" problems [Asada 90].

Admittance matrices can also be learned. Gullapalli, Grupen and Barto have used a direct reinforcement learning approach to the learn the admittance matrix for a peg in hole task. They report that some 500 trials were required to achieve a good performance [Gullapalli *et al.* 92]. Asada has also worked on learning admittance matrices, using a supervised Artificial Neural Network [Asada 90]. As each matrix is specific to one task, and if learned, is specific to that robot and sensor configuration at the time of learning, then the time taken to produce all the admittance matrices required for a complete system could be large. A single admittance matrix has a limited scope, and there are few real tasks which only need a single admittance matrix. Activities that use admittance matrices would have a very fine granularity, with several different activities being required to implement even a behaviour as simple as placing an object onto a flat surface. Hirai and Iwata [Hirai & Iwata 92] have proposed a method to identify when to change between matrices according to current force sensor readings, but they are not able to identify all the changeover points [Deacon & Malcolm 94].

All the schemes described here arrange to couple sensing and action at a very low level. In doing so they encapsulate the knowledge necessary to perform a specific task and hide it from the planner. This coupling and encapsulation are two of the most highly stressed features of the Behavioural approach. Although the aim with these schemes

is to produce activities of perfect reliability, the difficulty in reality of doing so has led a number of researchers to propose error recovery schemes that can be used should an activity fail.

2.2.6 Reactive Activities

There have been a number (at least 15 [Holland 95]) of different ways proposed to implement activities that can be classed as belonging to the Behavioural approach. In all cases what differentiates these methods from those discussed in the last section is the generality of the result. Instead of producing customised activities for a specific task, the activities are imbued with the necessary functionality to accomplish most, if not all, possible instances of a task as it pertains to that robot system.

The simplest of these approaches are those which directly couple sensors to motors, perhaps the first examples of which is the turtle of Grey Walter [Grey Walter 50]. In both cases behaviour is caused by appropriate linkages being established between sensors and motors and through the world. Grey Walter's tortoise (pictured in [Arkin 98]) is an analogue device equipped with a directional photocell sensor and a bump contact sensor that could exhibit five identifiable behaviours (from [Arkin 98]):-

Seeking light: The robot would explore its environment looking for a weak light source.

Head towards weak light: Once such a source was found the robot would head towards it.

Back away from bright light: If the light source got too bright (i.e. the robot got too close) this aversive behaviour would kick in.

Turn and Push: Used to avoid obstacles, this behaviour overrode the preceding behaviours.

Recharge Battery: When the on board battery was low, the tortoise would perceive a bright light source as weak as a consequence of its construction. The recharging station had a bright light over it, so the robot would move towards it and dock,

using the second of these listed behaviours. After recharging the light source would be perceived as strong, and so the robot would be repelled.

Grey Walter's work has been developed further by Braitenberg in a series of gedanken-experiments involving the design of vehicles using, and extending the principles of analogue circuitry used by Grey Walter [Braitenberg 84]. Some of these design have subsequently been made real [Hogg *et al.* 91].

Two examples of Braintenberg's vehicles are depicted in Figure 2.1. Vehicle A has positive (excitatory) connections between its sensors and motors, and will exhibit behaviour that an observer would term aggressive because of its habit of accelerating towards sources. In Vehicle B, the connections are negative (inhibitory), and it exhibits very different behaviour that might be termed exploration. Vehicle B would slow as it approached the source, but it would also veer away and so speeds up as it moves away.

Braitenberg develops these simple designs by adding more sensors, different types (including memory and nonlinear) of connections, and suggests that the result would be increasingly complex behaviour, yet still traceable to the combination of having a number of sensors, nonlinear relationships between the sensors and motors and having the robot operating in a complex environment [Braitenberg 84]. Such robots will be described here as being *purely reactive.*

Such connections as these are rules for what to do in certain circumstances. These rules can easily be implemented in hardware, as both Grey Walter did and Braitenberg suggested. This allows for fast response times, and minimal computing requirements, except perhaps for some sensor conditioning and motor control. Including the ability to make decisions based on sensor readings can allow multiple behaviours to be exhibited. Rules can provide implicit forecasting. They can also, as research into expert systems has demonstrated, implement skills and abilities for which intelligence is usually attributed, *e.g.* fault finding in cars or managing a chemical production process [Feigenbaum *et al.* 88].

Kaelbling and Rosenschein [Kaelbling & Rosenschein 91, Rosenschein & Kaelbling 94] have developed the REX and GAPPS computer languages that for a specified goal

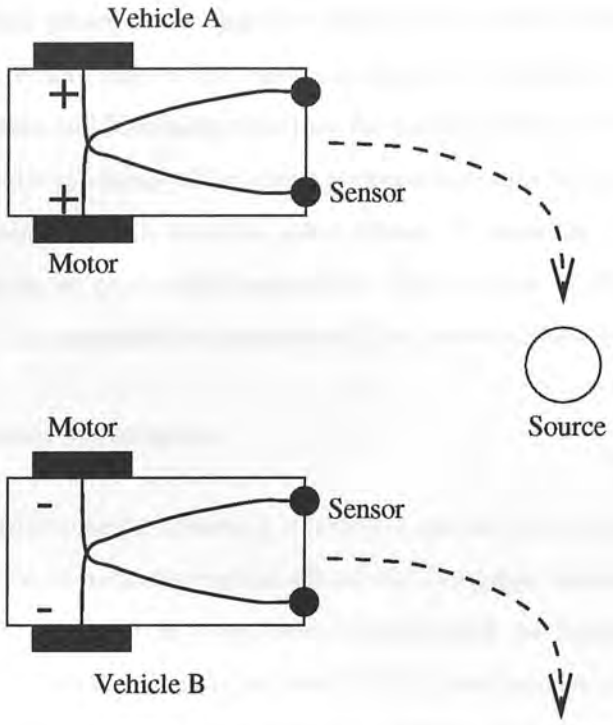

Figure 2.1: Aggression or Exploration[Braitenberg 84] p.8,11

and a model of the environment described in terms of the agent's interaction with the world, can construct a 'circuit' diagram linking sensors to actions. The resulting programs are provably correct, though only partially so as the compilers might not have selected an action for every situation so there could well be environmental states where the robot does nothing. No mechanisms are included in the final program to deactivate a behaviour if it becomes unreliable. Erdmann has proposed a method for identifying the minimum sensor information required to achieve a specific task, though his method relies on being able to develop a geometric representation of the task and in practice is more use as a post development proof of sufficiency [Erdmann 94].

A significant limitation of directly linking sensors to motors is that actions occur as direct responses to environmental situations. If the developer wishes the robot to exhibit some behaviour for which there are no environmental or proprioceptive cues available, then an alternative scheme for implementing activities must be used. For the Braitenberg vehicles in Figure 2.1 above, if the source did not exist or disappeared from the field of view, then the robot would either sit still, or head off in one direction at high speed.

Furthermore, all this presupposes that the appropriate rules for enacting behaviour can be developed. Where only a few rules are required, as above, then there is little difficulty. Rosenschein and Kaelbling note that for systems of any complexity then their approach to automatic synthesis of the robot program becomes increasingly intractable [*Ibid*,p.11]. Developing a purely reactive robot system for assembly with its very large state space is likely to be practically impossible. This because of the sheer number of rules required, and the necessity to resolve conflicts between those rules.

2.2.7 Conditional Reactions

If the wiring of a purely reactive system is structured and includes a conditional element such as a NAND gate, then that structure effectively embodies state information. This allows the system to escape from short term fixations and the necessity to always be able to sense to guide action. This is because NAND gates can be used to implement memory [Guo *et al.* 94], which means that the hitherto purely reactive system can remember information with a sophistication dependent only on the design of the agent. If memory is used, then the system can no longer be termed purely reactive, as it can now perform an action in response to some internal event or trigger. Mason has noted that a simple memory of what actions have been performed represents a significant source of information to an agent *q.v.* [Mason 93].

That memory is useful is evident in the work of many researchers who have found it useful to use state based mechanisms within activities, *e.g.* [Brooks 85, Arbib 87]. In some cases these state based mechanisms were hardwired, *e.g.* [Connell 89], though more sophisticated development tools such as the *Behaviour Language* [Brooks 90a] that can compile down high level descriptions of behaviour into the state based mechanisms are available and have been used. In addition to this, there have been a variety of formalisms produced, though in some cases these formalisms predate the widespread usage of the Behavioural approach by a number of years.

To repeat the definition given on page 26, a purely reactive system includes no internal state, though it may include many activities and mechanisms to arbitrate among those activities. As long as each activity is individually competent, there is no need for any kind of error recovery mechanism. However, should an activity fail then the

system will stall unless there is a way for that activity's successor to recognise that it has failed. One very effective way to do this is to incorporate some form of state mechanism. Wilson ably demonstrated this in her extensions of the SOMASS system [Wilson 92]. Her system had the same functionality as SOMASS, but included a robust error detection and recovery mechanism. This mechanism was based on each activity informing its successor as to whether it had succeeded or failed. If it failed, then the activity indicated which one of finite set of failures had occurred. This information was all that was required for the successor activity to behave correctly. The failure states were identified during the development phase as part of a failure mode effects analysis. In this respect Wilson's work has the same requirements and limitations as the error detection and recovery mechanisms described in Section 2.3.1, page 30.

2.2.8 Summary

Within this section seven separate approaches for managing uncertainty have been reviewed. Apart from the first approach, Engineering (Section 2.2.1, page 15), all of these have been concerned with the development of what are termed within this thesis as activities.

However complex and sophisticated each activity gets, there are not many assembly systems that can achieve realistic and useful tasks using just a single activity. In the next section the issue of how activities can be structured is examined with particular reference to the impacts this structuring has on the activities themselves. This emphasis has been chosen because of the focus on developing learning within activities, *i.e.* the objective here is to assess the impact structuring can have on learning.

2.3 Co-ordinating Activities

Regardless of how a behaviour is implemented, any real system doing a useful task exhibits different behaviours at different times. This section considers some of the ways that the controller can be programmed to implement this co-ordination, beginning with a review of the important issues that need to be addressed within a practical controller.

2.3.1 Error Recovery

Perhaps the key issue to be addressed during the development of a controller that is capable of co-ordinating different activities is specification and development of the recovery ability that need to be exhibited by the finished system. There are other issues, such as performance, maintainability, etc. which do need to be addressed during the specification and implementation. However, from the perspective of this part of this thesis, those issues are irrelevant as they do not have the same impact on the design of activities.

The ability of the system to recover from the failure of an activity is important. Even if activities are developed to handle uncertainty and effectiveness as well as possible, there always remains the possibility of failure. In some cases this might be small enough, and with sufficiently trivial consequences that any failure can be acceptably dealt with through manual intervention. In other systems, manual intervention might not be an option and the system will have to cope with all problems. In between these two extremes are a range of possible requirements, each with its associated consequences for the complexity of the system.

Three error recovery strategies can be identified within the literature:-

Failure Reason Analysis *e.g.* [Srinivas 76, Lee *et al.* 83, Hardy *et al.* 89].

Backward and Forward Error Analysis [Gini & Gini 83, Gini & Smith 86]

Plan Management System *e.g.* [Drabble & Cox 85]

Underlying these strategies is a common set of number of assumptions about what is possible, and practical [Wilson 92] :-

- Errors can be detected. To establish that the system has developed an error it is necessary to monitor every action of the system and compare the results to pre-determined expectations. It is possible to do this by updating a model of the system with information gleaned from sensors and comparing this model to an 'ideal' model. If the two differ, then there is an error. This can only be stated

with certainty if it is possible to develop the ideal model and maintain the on-line model with enough detail.

- Detected errors can be correctly diagnosed. Error diagnosis imposes the requirement on error detection of determining the context of an error, *i.e.* what the system was meant to be doing (and what state it was in) when the error occurred. It is quite difficult to work out how to do this in an efficient and application independent manner. Also, the correct diagnosis of an error requires that the system knows about the errors that can occur, and that it can relate this knowledge to the sensory results. In some cases this knowledge can only be elicited through repeated experimentation.

- Recovery is possible from diagnosed error. Once an error has been correctly detected and diagnosed, then recovery from that error should be no harder than any other stage of the process. However, unless the first two assumptions are correct, then all that a recovery activity is likely to do is to make things worse.

The assumptions are important. If, from a system perspective, some recovery ability is required, then some degree of compliance is also required, though the extent is dependent on the requirements place on the system. However, with activities that learn, some failures are inevitable. This means that if the system is to operate without, at least initially, frequent intervention, some recovery ability is essential. This means that each activity should be constructed in such a way that it is able to at least, notify the calling activity when it has been unable to achieve its goal, if not the reason why it has failed.

Summary

This review of recovery has highlighted an important requirement on activities. In the rest of this section during the consideration of a number of different schemes that have been proposed for structuring activities, note will be made, where appropriate, of how this requirement has been implemented, if at all. The value of this is that it will allow some assessment of the relative merits of the different schemes.

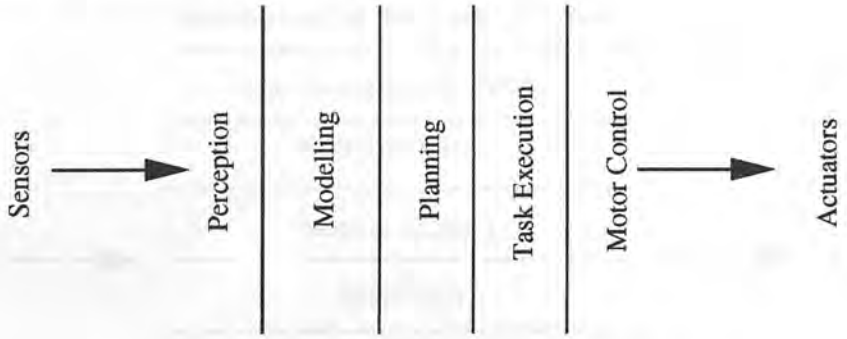

Figure 2.2: The Classical architecture [Brooks 85]

2.3.2 Classical Architecture

A representation of the stereotypical architecture for a Classical controller, taken from [Brooks 85], is depicted in Figure 2.2. Information is funnelled through the controller from the sensors to the actuators via a bottleneck that is a world model. All planning takes place using this world model. Prior to being used to update the model, the raw information is processed to extract all the facts about the world which are represented in the model. Inside the model multiple sources of the information are fused to provide a single output. The planner uses this model to work out the necessary motions to achieve the robot's goals. These motions are then passed on to the part of the controller which converts desired motions into actual motor commands. Often the planning is done off line to create a stand alone executive program that includes constructs that use information provided by the sensors to modify the performed actions. This process compresses the controller by removing much of the perception, modelling and planning stages. However, this loss increases the time and effort required for the robot to be applied to a different task as a new program has to be developed. Learning, where used, is located within the implementation of a stage, *i.e.* within an activity.

The SHAKEY system [Nilsson 69] was one of the first complete (albeit in a very limited domain) examples of this type of robot systems. It consisted of a large mobile robot guided by a high level planning system called STRIPS. The robot operated in a series of rooms, and had to accomplish tasks of moving boxes around different rooms. Sometimes it would have to search for a particular box. Sometimes it would have to push a ramp up to a box on which the target would be resting. The goal of the developers was

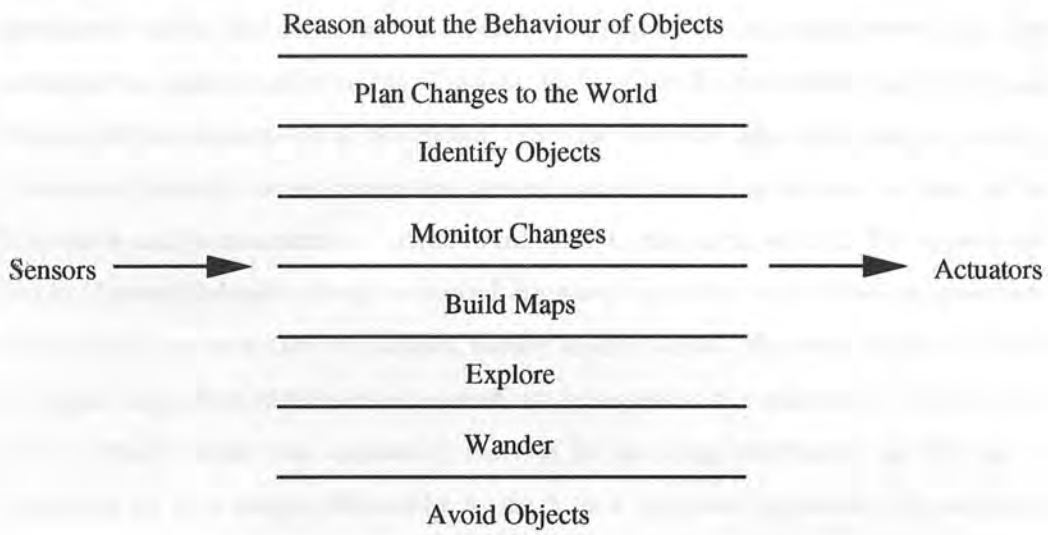

Figure 2.3: The Behavioural architecture [Brooks 85]

to create a task level system, with the planner plus a collection of lower level planner and control loops working to effect the plan designed to accomplish the task. However, the environment had been carefully chosen. The rooms were painted a uniform colour, and contained visually "simple" objects, *e.g.* cubes, wedges. The robot was the only active agent in the environment, and there were no time or economic pressures on the system to complete a task.

Even in this simplified environment things did not go as easily as expected; it was necessary to implement low level routines to handle the uncertainty, which was far more important than expected. The STRIPS planner (which incorporated learning in the form of Macro operators) was adequate for dealing with the tasks SHAKEY had to face, but it could not handle the complexity of assembly where the number of states that has to be considered is far larger. As it was, it could take SHAKEY an hour to plan a single move [Nilsson 69].

2.3.3 Behavioural Structuring

A representation, taken from [Brooks 85] of this kind of architecture is shown in Figure 2.3. Here, all components of the architecture can receive information from the sensors, and all can access the motors. This architecture is fundamentally different to that discussed in the last section. As for the Classical approach, learning is im-

plemented within the activities. As noted in Chapter 1, this architecture has been developed as an alternative to the Classical Architecture by researchers such as Brooks [Brooks 85] in response to a perception that the Classical approach was not able to produce sufficiently robust competent systems as efficiently as possible because of the brittleness and performance constraints inherent in the architecture. The representation in Figure 2.3 depicts the Behavioural approach's solution to the performance issue: the activities are executed in parallel, rather than in serial. However, Figure 2.3 does not depict how the activities resolve conflicts for access to the motors (or sensors), nor does is clearly depict the approach's method for avoiding brittleness, as this can be considered to be a design philosophy as much as a technical approach; The activities are designed to be as robust as possible, using some appropriate technique (See Section 2.2, page 15), and are then structured so that if an activity, *e.g.* Monitor Changes, fails, the system instead falls back to using a simpler activity. Whilst this can also be done within the classical architecture, this redundancy and error recovery are inherent within the approach, and within the technology used to implement the architecture, *i.e.* within the mechanism that arbitrates which activity has control over the actuators at a given time.

A number of different implementations have been developed of this architecture, *e.g.* [Brooks 85, Malcolm 87, Connell 89, Arkin 92]. All of these approaches use the ideas of the architecture, though depending on the application, the degree of actual parallelism, the level of sophistication of the controllers, and the arbitration mechanism varies considerably. Here the particular issue is how activities can be structured from the perspective of assembly.

As has been noted earlier at the start of this chapter, for assembly it is important that the system be flexible, *i.e.* can be moved between a range of tasks easily. Consequently, it is important that it is relatively simple to re-program, or redirect, the controller of the system. Unfortunately, with some of the systems developed under the Behavioural approach, this is not possible because the activities and their structuring have been hard coded for a particular task. Whilst the intent of the Behavioural approach as practised in such examples is to eventually reach a stage of sophisticated where the system would be capable of any assembly, no working examples have been produced to

validate this idea. In a sense, the situation here is similar to the situation with Reactive Activities (Section 2.2.6, page 25): simple systems can be produced relatively easily, but complex systems become very difficult to develop as the number of relationships between the activities becomes very large [Malcolm 94]. Such a complex system would also be a large system, due to the number of activities, and the complexity of the arbitration mechanism that would be required.

Two alternative, and different, approaches have been proposed and implemented for producing assembly systems: Reactive Planning and Hybrid Planning. These approaches are covered in the next two sections. Although they can both be classified as being Behavioural approaches, they both also include elements that would not be out of place in a Classical Architecture. Such hybrid systems have been developed to exploit the best of both worlds.

2.3.4 Reactive Planning

Another formal approach to describing activities which is more general than either REX or GAPPS is the *Robot Schemas* (RS) approach. This has been developed by Lyons [Lyons 85] from ideas about activities first proposed by Arbib [Arbib 81]. Activities in RS are specified as assemblages of primitive (or simpler) activities. Each activity is an independent synchronous process. The formal component of the approach stems from its basis in the port automata model of parallel interactions. This allows the overall behaviour of a particular set of activities to be formally established. Asynchronous systems can be modelled in RS, though not easily. The approach does not extend to modelling interactions with other agents.

A formal method that does handle descriptions of interacting agents has been proposed by MacKenzie and Arkin. They consider a set of activities as a society for which long term global behaviour can be predicted [MacKenzie & Arkin 94]. This work naturally extends from a single individual to multiple robots and is inherently asynchronous. Unlike RS this approach has yet to be implemented, and it may be that the generality of the proposed formal description will have to be restricted to achieve workable implementations.

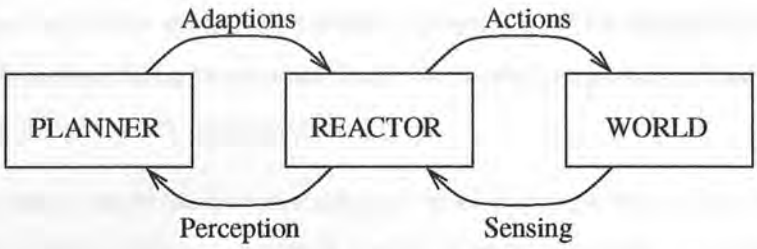

Figure 2.4: Planner-Reactor Systems [Lyons & Hendriks 92]

If the formal description of activities allows prediction of long term system behaviour, then it becomes possible for those activities to be modified to move that behaviour towards a more suitable pattern for a given application. Although REX/GAPPS and RS are both effectively programming languages, it is still necessary for the human developer to design and construct the system controller. Under the classical approach, nearly all the effort has been aimed at automating the human designer out from the process of developing the system controller. Even though the long term aim of the Behavioural approach is to have general purpose system controllers, human development is still necessary. These formal descriptions allow a way for classical reasoning technology to be used to produce, or at least modify the system controller.

Lyons has developed this approach into what he terms *reactive planning.* He has successfully applied this work to assembly robots [Lyons & Hendriks 92]. This work is closely related to the Case Based Reasoning approach, such as used in the ACBARR system [Ram *et al.* 92] (see Section 6.3.1, page 129), though this is less general. It is also related to the work of Segre [Segre & Turney 93]. He has considered how Explanation Based Learning (see Section 6.3.2 on page 130) could be used within reactive planning to allow the planner to learn from experience. This work was all done in the context of the Golddig computer arcade game.

The structure of reactive planning systems is shown in Figure 2.4. The reactor is developed in some formalism (RS in the case of Lyons, predicate calculus by Segre). When implemented it can act and sense in the world at appropriate rates. Sensory and reactor state information are copied to the planner. Here the interaction between reactor and world is compared against a model of performance developed from a specification supplied by the user to determine what changes need to be made to the reactor. Once

these changes have been made the process is repeated. The representational limits of the formalism describing the reactor limit the modelling powers of the system and constrain how its goals can be expressed.

Initially the reactor must be at some minimal level of competence. The model of the environment must be realistic, though it need only be expressed in the formalism used to describe the reactor. The planner also requires a specification of the goal of the system. The amount of initial knowledge required in Segre's system is reduced by equipping the system with the ability to extend its own knowledge base. The tradeoff here is that the system must then be given the ability to analyse its own performance in a way that a non-learning system would not. Whatever the knowledge input, it is specific to the application. However, once it is available, it is theoretically possible for the system to modify the reactor to perform any possible task of the system. The problem lies in making sure enough information has been entered. For complex domains this would not be a simple matter, or one which was economic of computer memory. Even so, systems that are built with such planners should be smaller and simpler than a system which was designed and constructed to handle all of the cases that the planner can customise its reactive partner to meet. These systems are the closest examples of how the Behavioural approach can be applied in a similar manner to the classical systems described in Section 2.2.5 on page 20 which attempt to customise activities to suite a particular application.

When the planners modify the system controllers in the work just described, they also scheduled the activities of the system controller. Such scheduling has so far been implicit in the description of how system controllers could be constructed. It has been assumed that the structuring of the activities included that information as required. In a purely reactive system there is no need, or possibility for scheduling except that provided by the environment. One of the features of more complex tasks that makes such purely reactive systems inappropriate is the need to accomplish a strict sequence of tasks. To do this some form of scheduling is required. The aim that is repeatedly defined in this thesis for the Behavioural approach is to produce system controllers that can reliably achieve any instance of the general task for which they have been constructed. One feature of such tasks is that the scheduling required changes between

instances. So far the ways that have been described here to handle such a change are to redesign the system controller to have the necessary structure (the classical approach, or the human development of Behavioural systems), or to use a planner to modify the structure.

2.3.5 Hybrid Planning

The label *hybrid system* is used here to refer to the combination of a classical planner that performs the scheduling to a Behavioural system controller that defines a virtual machine. The controller conceals the detail of how it is implemented from the planner, and provides the planner with a simplified, tractable world.

This is a return to the classical AI planner/execution program split of the kind found in SHAKEY. Here, however, the relative competencies of the components are very different. SOMASS, which has already been mentioned several times, is an example of such a hybrid system. Of the two components, the role of the planner is to:-

1. Determine how the robot could build a given object using the specified set of Soma parts [Gardner 61].
2. Convert that plan into a sequence of activities.

That sequence, along with any parametrisations are passed to the controller. This then calls the activities in the required sequence. The planner is hierarchical, and planning is done in terms of behaviours that are defined in terms of the particular part motions. These behaviours are then achieved as reliably as possible by the activities written in VAL–II on the vendor supplied robot control computer.

SOMASS was developed to explore an alternative paradigm of robot (and general intelligence) programming whose philosophical and historical routes are very different from the classical GOFAI[1] systems that had previously dominated AI research. The insight behind that lead to this architecture was that [Malcolm 87]:-

[1] **G**ood **O**ld **F**ashioned **A**rtificial **I**ntelligence [Haugeland 78]

> a hybrid system, comprising symbolic and non-symbolic components interfaced by carefully chosen virtual machines, which appropriately conceal problem complexity from the symbolic levels, constitutes an efficient solution to a problem apparently involving both reasoning and manipulative skill.

The original SOMASS system required approximately six man months to develop. It comprises some 4000 lines of Prolog and a few hundred lines of VAL–II. Most of the Prolog is concerned with determining how to construct various specified objects from Soma parts. Most of the time spent by the planner is used to determine how to build a particular shape. The initial version worked with no sensors other than robot proprioception, yet could plan and execute the assembly of a wide variety of objects using a roughly constructed Soma-4 shape set with a reliability rate of 97% [Malcolm & Smithers 88]. Sensing has since been developed and introduced into the activities *e.g.* [Chongstitvatana 92, Kim 92]. These changes did not require any modifications to be made to the planner. As described on page 29, the original version has been enhanced with the introduction of an error detection and recovery mechanism.

The idea of a hybrid system embodied in SOMASS has been used to develop CARS [Hallam 91b, Hallam 91a]. This is a system that performs real-time control of 1/24th scale Porsche 962c model cars moving around a model town. Each car is continuously controlled and is tracked by an overhead camera. The tracking information is used by the planner to determine appropriate trajectories for each car that are passed to the appropriate controllers. The controllers hide from the planner the detail and intensity of calculations necessary to make the cars perform. The planner, operating in a symbolic world uncluttered by the complexities of reality, is left to handle those tasks at which current planning technology excels: coordination, scheduling, and the conversion of high level goals into car trajectories. The modularising of specialities suggested by applying the hybrid approach here enables the critical real time performance to be handled by a suitable low level controller, leaving the planner to operate on a more relaxed and appropriate time scale.

Some of the advocates of Behavioural programming argue that the hybridisation such as used in SOMASS and CARS is unnecessary, and wrong, *e.g.* [Chapman & Agre 87,

Brooks 90b, Brooks 91b, Brooks 91a]. Instead they suggest that the functionality of an AI planner can be replicated by appropriately high level behavioural activities. However it is not clear how this could be done with the generality of the kind of classical planning system used in SOMASS or CARS. The debate over this is not helped by there being no similar systems to SOMASS or CARS that could be used to assess how their complexity compares to a purely reactive, or a purely planning system. Given that activities (however implemented) and classical planners are good at two different kinds of things (getting things done and working out what to do), it seems logical that to use two specialists to achieve a task would be simpler in development and operation than trying to use either alone.

The hybrid approach, and to a lesser extent reactive planning systems, make the assumption that the underlying system controller on which they are based is competent enough to achieve almost anything that the hardware is capable of. Both these options accept the need to perform some rescheduling or make some small modification to the activities. Both also note the need to appropriately parameterise the activities. However, this requirement to undertake some system tuning is a very different requirement from being able to automatically construct a complete system from just a requirement.

2.3.6 Summary

In this review of how to co-ordinate activities, four schemes have been considered: Classical Architecture, Behavioural Structuring, Reactive Planning, and Hybrid Planning. Other schemes have been produced (see [Holland 95] and [Arkin 98] for examples), but can be considered as derivatives or relabelled versions of these four approaches. Each of these four schemes represents a different way of developing a controller to achieve a tasks. Each has its limitations, and each has its proponents.

However, for the purposes of this thesis, there is a thread can be identified, namely that each scheme represents some way of structuring activities, and each scheme contains somewhere an "interface" between the activities and the mechanism that decides how to structure those activities. For the Classical architecture, the Reactive Planning and Hybrid Planning approaches, this mechanism can be a GOFAI planner. In the Behavioural Structuring approach, at its extreme (*e.g.* [Brooks 85]), there is al-

most no "explicit" planning, but rather activities are simply "added" until appropriate competence is obtained. In returning to the structuring, the first part of this review considered how error recovery might be implemented. Perhaps the key element, and consequent requirement on activities, is that activities must provide to the calling activity an indicator of success or failure.

2.4 Summary of Chapter

There are parallels between the historical progress of types of system controller and approaches to building robotics. Following the initial attempts that were made with good intent but a poor (with hindsight) conception of what the problems were, successive systems have appeared which attempt to tackle the failures of the preceding approach. Each of these systems has in turn had its own problems. Through analysis of these problems has come the realisation that for any kind of symbolic reasoning mechanism to be used, then the rest of the system must be constructed in such a way as to present to the symbolic reasoner as simple a world as possible. The need for such a reasoner has always been acknowledged whenever the complexity of the task and the flexibility required for that task grows beyond a certain point.

Of the two approaches the older Classical approach attempts to develop systems that can automatically produce a task level system for a specific task. The newer Behavioural approach instead concentrates on building systems that are capable of achieving any specified task. Both approaches have settled on using the same kind of basic computational entity as the building block of their systems, the activity introduced on page 5 of Chapter 1, though it is called different names and implemented in a variety of ways.

The aim given at the start of this chapter was to review the various approaches that have been taken to produce an effective robot software controller. A common element of these approaches has been that each, in its own way, implements a particular action-sensor coupling, and each takes parameterisation of some form that specifies the detail of the goal and the control parameters of that coupling. Another element has been that the structuring of the activities is important for the robustness of the system.

Under the Hybrid Planning approach, this structuring should be done in such a way as to create a virtual machine which defines a world in which a planner that is necessary to perform the scheduling of activities can operate tractably. This explicit strategy can be identified as an implicit element in many, if not all, of the other structuring approaches.

Although this has been a lengthy chapter, it has covered much ground. Importantly, in identifying action-sensor couplings and structuring as common "components" of a robot controller, it has provided much of groundwork for the remainder of this thesis. The remainder of the groundwork will be provided in the next two chapters, beginning with an identification of where learning can be used and then going on to look at the experimental system that has been developed for this work.

Chapter 3

Where and Why to Apply Learning

This thesis is researching the hypothesis that using learning in assembly robots is a good way to improve the flexibility and robustness of such systems and reduce the workload of developers. This is because learning can be used to automate the continual adjustments to the system that are necessary for the system to operate in a changing environment. So far the background to activities has been presented. The task in this chapter is to consider how and where learning can be used in assembly robotic systems.

To briefly recap, an activity is a computational entity that makes the system act to achieve a specific goal. It has task and control parameters. An activity can use other activities. If so, these other activities are structured and coordinated, sometimes with sensor mediation, sometimes using a local planner, sometimes both. Many activities are specific to the type and configuration of hardware used in a system. The internal computation occurring with an activity determines how it can achieve its goal. However, its observable performance is ultimately determined by the combination of the task and control parameter settings and the environment in which it is applied. The internal structuring and computation of an activity are determined during development of the controller. The task, and to a lesser degree, control parameters have to be determined for each application. These entities are used because they can hide the uncertainty that has to be dealt with to achieve that activity's goal from its user. If that user is an AI planning system, then the activities in the controller define a simpler world that does not include uncertainty or variation.

Controllers are developed by assembling the various activities that will be required. These activities can be either written from scratch or taken from an existing controller. As the number of activities available grows, they are structured to form successively more sophisticated activities. The process of activity development stops when a level is reached that can be interfaced to by the user that will schedule what the system does and when. That level might be the very top of a task, *e.g.* 'Pick the litter up around the university campus', or it might be to a level which can easily be represented in a symbolic planner, *e.g.* 'Put Screw into Hole 1'. Once the controller is complete, all the necessary parameterisations must take place to configure the activities.

This description of the development process provides some clues as to where to put learning. These clues are followed up in the first of the four sections in this chapter. The second section describes the profit functions that were an essential part of the learning mechanisms in the experimental work described later in this thesis. The third section continues the theme of studying things common in all the learning mechanisms by describing the properties which those mechanisms should display. The final section then summarises this chapter.

3.1 Learning in Robotics

The main motivation, described on page 6 in Chapter 1, for this investigation into the use of learning in robotics is the hypothesis (first described on page 8) that learning can be used to make the system better able to respond to changes in itself and the world.

To consider how learning might be used in robotics, the purpose of this chapter, some form of classification system will be required to group and assess related ideas. Brooks and Mataric have produced such a classification of the types of learning that have, and can be used in robotics [Brooks & Mataric 93]:-

Strong Learning: begins with a controller that contains no activities or structuring. Aims to learn everything, *e.g.* [Drescher 91], but takes a long time and a great deal of computation to learn very little.

Weak Learning begins with nearly everything predefined. Learning is used to fill in a few small details, though depending on the application, filling in these details may take some time, *e.g.* [Sutton 84, Thrun 92]

Subsumable Learning uses weak learning to complete gaps in its predefined knowledge, but can use the methods of strong learning when major changes of ability are required, *e.g.* [Ram & Santamaria 93].

According to this classification (which has been developed from engineering rather than biological or psychological principles), systems that incorporate weak learning will produce predictable results. The caveat to this statement is that this situation will only occur if the developers are fully aware of the details, requirements and assumptions of whichever learning mechanism they use. The quote on page 57 from [Chapman 91] is an illustration of what can go wrong. This predictability is a consequence of the learning mechanisms that are used. At the heart of nearly all such mechanisms is a generate and test cycle that is related to the process of searching a state space using classical AI techniques. If the search space is shallow and narrow, then the amount of searching required to find a good solution is minimal. This corresponds to weak learning. Strong learning is required when the search tree is broad and deep due to the far greater range of options that has to be considered. However, strong learning mechanisms operate differently from searching. Instead of generating a number of different options, and evaluating them to find the best, a learning mechanism attempts to generate the best solution. The knowledge gained from evaluating the results of each proposed solution is then used to improve the next guess. In other words, a learning mechanism performs a series of experiments, using the evaluation of the previous experiment to guide what the next experiment should be.

In the rest of this section the three different kinds of learning will be considered separately.

3.1.1 Weak Learning

In robotics, three of the kinds of things that can be learned using weak learning are [Brooks & Mataric 93]:-

- The control parameters of activities.
- Action-sensor couplings.
- Which activity to use when there is a choice.

A common thread that runs through the elements of this list is that in each case a function is being approximated by the learning mechanism from the control and possibly the task parameters of an activity to the output of that activity. This output can be motor commands or commands to other activities. Any realistic function belonging to any one of these classes will be ill-defined. For an assembly robot, a well-defined function requires full knowledge of the environmental situation, state of the system before and during the performance of the action, and what the system was about to do when a decision was being made. Some of this information (the *defined* parameters) will be available, but most of it, especially the environmental parameters, can not be easily obtained with the required accuracy (the *undefined* parameters), resulting in the functions being ill-defined. It is these undefined parameters that the learning mechanism is approximating, with feedback from the experiments being used to refine the approximation until the final performance is within a specified tolerance.

Furthermore, if this list is compared against the kinds of activities that developers perform when developing a system controller, then it is quickly obvious that the three things weak learning can do are things that the developer would have to do, and are things that the developers, in general, find difficult. If the reason for that difficulty is considered, then the reason why these kinds of learning will be useful becomes clear. Determining action-sensor couplings to achieve a task, establishing good parameter settings and identifying which activity to use, all require a certain amount of experimentation with the robot to gather enough information from which the appropriate informed decision can be made. When the task, robot or environment changes in a way which requires these decisions to be repeated, those experiments have to be repeated. By introducing suitable weak learning mechanisms which embody the generate and test cycle just described into assembly robotics, a number of the developer's current tasks will be removed or assuaged. Providing the mechanisms are implemented as on-line mechanisms then they will be able to continuously adjust the approximation

in response to changes during operation.

There are a variety of methods from which mechanisms that can be used to perform function approximation through learning can be developed, including:-

- Numerical Optimisation *e.g.* [Press *et al.* 90]
- Reinforcement Learning *e.g.* [Sutton 84]
- Genetic Algorithms *e.g.* [Yuret 94]]
- Artificial Neural Networks *e.g.* [Barto 90]

These four methods can be said to implement, at a very high level, a common approach to approximating a function: a test approximation is produced, evaluated against reality, and modified. This sequence continues until the difference between the result of the approximation and reality is within a set tolerance. Each of the examples of these learning methods has different implementation, operational and learning properties. The detail of the important elements of these properties are presented in Section 3.3 (page 56).

3.1.2 Strong Learning

Strong learning can be used to:-

- Develop novel activities
- Construct the structural framework of activities

When weak learning was described on page 45 the operation of the learning mechanism was described as approximating a function. Here the task of the learning mechanism is to define, and then approximate that function. As Brooks and Mataric have noted, *q.v.* [Brooks & Mataric 93], strong learning mechanisms aim to start from nothing and produce a complete system, but often fail. If the task of strong learning is seen as a generate and test cycle over the possible search space of activities and their possible

structuring, then it is not surprising that strong methods fail as those search spaces are very large, and contain many unworkable entries.

Some of the mechanisms that have been proposed to do strong learning, such as the Schema Mechanism [Drescher 91] or Genetic Programming [Koza 92], or even types of reinforcement learning, *e.g.* [Sutton 90], are all based on the generate and test model. All of these methods require a large number of trials, *e.g.* Koza repeatedly reports using in excess of 100,000 trials to learn a single function. He also reports that in most of these trials failure is expected. Simulation could be (and has been, *e.g.* [Cliff *et al.* 92b]) used to allow much of this learning to take place off line. Results can be poor unless the simulation is a fair representation of the actual system, and the learning in the simulation is periodically evaluated on the robot with the results reintegrated back into the simulation. If both of these steps are taken the reported experimental results indicate that useful transference of discovered behaviours is possible [Cliff *et al.* 92b, Cliff *et al.* 92a]. Unfortunately, such a route is not readily available for assembly robotics. As has been noted, the implementation of a system controller which achieves a specific functionality is dependent on the underlying hardware of the robot, the environment in which the robot is embedded, and the task to which the robot is to be applied. This coupling is unlikely to be identical to the one established in the simulator and consequently be of marginal value. Furthermore, as Hoyle [Hoyle 84] noted of research into insect neuroethology, the results obtained from developing strong learning for one type of system are unlikely to be easily transferable to another.

The most significant problem of these approaches which directly try and learn the system controller is the number of failures they encounter. An interesting, though very new approach, attempts to avoid this problem by following the example used in biological evolution. This work starts from the observation that animals or plants are 'constructed' from their genes. As it is these genes that are changed though evolution, then one way to construct a robot system controller is to specify an appropriate gene sequence. This gene sequence is converted into a controller by applying a development cycle that is modelled on biological development: the genes express how to construct cells, which in turn interact to produce the agent. This approach allows a valid controller to be derived from much of the large genetic search space

[Angeline & Pollack 93]. This in turn increases the chance of finding a useful result, though only if useful results are present. Such a high degree of *closure* [Koza 92] can be produced with other strong learning systems, but at a cost of limiting the power of the underlying representation. At present many of these strong learning systems produce only flat purely reactive systems, which, as has been pointed out in Section 2.2.6 (page 25), are difficult to use in situations where environmental cues to the next action are not readily available. Accordingly, the development process has to be repeated whenever the environment or task is changed. If that agent is were able to learn for itself, with the role of the development process to install the right kind of framework, then using learning becomes far easier [Dellaert & Beer 94]. However, the process of turning a genetic description into an agent that is capable of performing complex tasks is not yet known.

Research into strong learning algorithms could have a very useful result. Learning mechanisms of this type that can replace most of the development work would be an effective solution to the goal of producing a task level system controller. Strong methods have not been used here. This is partly because there is no agreement for the kinds of thing that strong learning should do, or how it should do it (the nativism versus constructivism debate being an example *e.g.* [Drescher 86]). It is also because the mechanisms currently available are not useful in the more complex kind of activities that assembly robots have to perform due to the large amount of computation and, more importantly, testing that is required. Some of these issues could be resolved by constructing one competent system, or several using different methods, to provide a target point from which strong learning methods could be developed or tested. The COG project at MIT is such a project [Brooks & Stein 93], with an expected duration of 10 years before a final complete robot is produced.

The next section looks at how the weak and the strong learning mechanisms described so far could be combined.

3.1.3 Subsumable Learning

Subsumable, or multi-strategy, learning methods have only recently received attention from robotics researchers, *e.g.* [Ram & Santamaria 93]. They have however been con-

sidered by planning researchers for some time, *e.g.* PRODIGY [Carbonell *et al.* 90]. In these methods strong learning is used whenever the task of the agent exceeds its current abilities to develop the appropriate new functionality. The new, and existing, functionality is then fine tuned using an appropriate weak learning method. This method, like strong learning, is a way of developing a system automatically, but unlike strong learning it uses weak learning to complete the alignment of the produced system to its task and situation. As such, it combines the best of both worlds. Such mechanisms are considered here as being justifying examples for the hypothesis proposed at the start of this section that the right kind of learning can make the system controller simpler, but as effective as a controller which did not incorporate learning or that incorporated the wrong form of learning.

At the moment, subsumable learning is an interesting, but impractical idea. In addition to the problems that are present with strong learning mechanisms, subsumable methods require the system to be able to identify when it should extend its abilities as opposed to having to re-tune the existing abilities. Like strong learning, subsumable learning is potentially a powerful tool, but one which will require much research before a successful application will be available.

3.1.4 Summary

From this review it would appear that the weak learning methods are the best to consider here. They can be used to approximate the ill defined functions which predominate in robot controllers. Such learning methods will have to be built into the activity and structural frameworks that comprise the executive component, or system controller of a robot system. During the last chapter when discussing methods for developing and structuring activities (Section 2.2.3, page 17) the three activities that were repeatedly mentioned as being difficult were parameterisation, definition of sensor action couplings, and specification of the structural framework.

When discussing the automated development mechanisms they were described as having to search for the correct answers. It is obvious that a human developer would have to do the same. Given that these things can be performed by the three weak learning methods discussed above, if implemented correctly, then it would appear that

the hypothesis proposed on page 6 that learning could be useful in a robotic system is worth pursuing. The rest of this thesis will describe the experimental verification of that hypothesis and the details of the learning mechanisms that are used to do so. In the rest of this chapter the profit functions that will play an important part of this learning and define what properties should be found in correct learning mechanisms are described.

3.2 Profit Functions — Economics of Activities

For many learning mechanisms, including two of the four used in this thesis, the role of the profit function[1] is to convert observations of the mechanism's performance provided by *cost functions* into a single numerical answer indicating the mechanism's performance relative to some benchmark.

The output of the profit function is used in the updating of the structures being learned, with the learning mechanism selecting the structure that provides the best profit values. Consequently the profit function must instantiate the correct relationship between desired behaviour and observed results.

Developing profit functions is a difficult task [Sutton 90]; they are application specific, and for robotics, their final form is dependent on the interaction between the robot, the learning mechanism, the structure being learned, the system's environment and the task. However, as long as the highest profit values are located at the 'right' place, and the structure of the profit function is such that these values can be found by the learning mechanism, then it does not matter if the specified profit function does not exactly match the true relationship between actions and results. As an example, consider if the true relationship between a profit p and the result r_a of some action, a was

$$p = -10r^2 + 1.6 \tag{3.1}$$

but the developer's profit function, developed from observations was

[1] More commonly known as error, cost or penalty functions. This terminology stems from the details of how profit functions work.

$$p = -2r^2 \tag{3.2}$$

The maximum profit of both functions will occur when $r = 0$, despite the functions being different.

The theory behind the profit functions is that of decision making according to expected maximum utility, *i.e.* agents choose activities which provide them with the maximum possible perceived benefit. This theory underlies most explanations for why animals (including humans) choose one activity over another [McFarland & Bösser 93], and is widely used in biology and the social sciences. When using the theory as an explanatory tool it is usual to assume that the decision maker is:-

- Rational.
- Has a limited amount of resources.
- Has constraints in the amount of resources which can be given to a particular activity.

The theory can be used to predict how agents will behave, or given a pattern of behaviour, to understand its constraints and resources.

There are four steps in producing a profit function for an application. Firstly, it is necessary to state what behaviour is desirable. In the case of reaching (the problem which will be used, described fully in Section 4.1, page 67), the best activity is the one which only takes one move taking no time to position the gripper at target within the desired tolerance. From this statement the second step of identifying and implementing the cost functions can be performed. The ith cost function of a profit function, specified formally as

$$f_i^c(p_i) \tag{3.3}$$

converts a measurement p_i (which might be a vector) of some aspect of the system's performance into a scalar real valued cost. Here a value of 0 indicates that the measurement is ideal, and 1 that the measurement is as far from the ideal as possible.

Studies that argue for the use of cost functions by animals (see [McFarland 93] for a wide ranging summary) are based on the assumption that most animals act in such a way as to minimise costs. More formally, Erdmann, considering how to design sensors and sensor processing procedures for use in particular activities [Erdmann 94] has argued the same point. He shows that doing so creates a feedback loop which will maximise the profit accrued by an individual during that activity.

The conversion of measurement into cost can be simple, or it can be complex depending on the desired relationship. The mechanisms of cost functions reflect the underlying nature of the measurement and depend on the assumption that it is possible to specify an ideal (which might be a point, a curve, a surface, a volume, etc.) and a limiting value for the measurement. In robotics, this assumption is reasonable. There are physical limits on the hardware of the system, and whenever the robot is performing a specific task then there will be physical situations where the task can be said to have been successfully completed (the ideal measurement) and situations where it will not. Behind this assumption is another, more limiting one; that it is possible to measure the aspects of performance which you are trying to get right. If not then it is necessary to rethink the initial statement on what makes a desirable performance, or to develop and install an appropriate sensing mechanism. In practice once the relevant performance aspects can be identified and monitored, the limits and ideal situations become self evident for a given application.

Now the designer has a set of cost functions, each of which returns a value in the closed interval $[0, 1]$. The next step is to decide how the different cost functions relate to one another. In the experiments with the QUEASY reaching variants, described in Section 5.1.6, page 105, it was usual for one cost, such as the position error, to decrease whilst another, such as the number of steps required, increased as the error tolerance of the variant was reduced. In choosing a particular tolerance a developer is specifying a compromise cost for the activity, a value which will be referred to as the *system cost.* The third step of developing a profit function is to express the system cost and how it is calculated from the different cost functions. A general equation to do this is:

$$f^s(\mathbf{p}) = \frac{\sum_{i=1}^{n} c_i f_i^c(p_i)^2}{n} \tag{3.4}$$

where $f_i^c(p_i)$ is the i cost function (Equation 3.3), and c_i is the coefficient applied to the squared output of a cost function.

The implementation of the system cost function as a quadratic linear sum follows McFarland. He has argued that animals use such quadratic cost functions [McFarland 71, McFarland & Bösser 93]. This is because any reduction in a measurement then produces a corresponding reduction in cost which is greater when the measurement is larger than when it is smaller. When using an optimisation procedure such a relationship speeds convergence to the optimum. The coefficients associated with each cost function, each in the closed interval $[-1, 1]$, allow the developer to express the relative importance of the costs. This final decision that has to be made by the developer has a similar effect, but a different mechanism than the assignment of utility to profits, which is the final step in most applications of profit functions.

When the form of profit functions is discussed, *e.g.* [McFarland & Bösser 93], it is from the point of view that the agent using that function is a consumer who wishes to maximise the benefit, or utility, of its actions. For an observer to understand why the agent acts in a certain manner in particular circumstances, it is essential to know what utility the agent ascribes to potential outcomes. However, profit functions in machine learning are used to bias the performance of the learning mechanism into a pattern that the developer considers optimal. In this respect they are engineering rather than analytic tools.

Establishment of the utility of a result which is necessary to bias the learning mechanism has to be done from the point of view of the learning mechanism. This is rarely a straightforward process if performed literally. Instead, here the learning mechanism is viewed as a producer (which is a consumer of a different kind [Begg *et al.* 94]) of a service, that seeks to maximise profits through minimising losses. The problem of establishment of utility now becomes the problem of identification of the costs of making a decision. This is a problem that can be solved if a system cost function of the form described in Equation 3.4 is used. The implicit assumption here is that for a producer, utility is directly linked to profit. Maximising profit then maximises utility. As the only producer, the aim of a learning mechanism will be to find which of its 'products' produces that profit. Since the use of profit here refers to internally (to the system)

assigned values, the final step in the development of a profit function is to link the system cost function with a decision function and a calculation which establishes the relative worth of different options, which is what the learning mechanism needs. This definition and use of profit functions is a solution to what is usually referred to as the *structural credit assignment problem* [Sutton 84].

The decision function (*e.g.* function $activity\_successful()$ in Algorithm 1) used should return true only if the activity to which the profit function is applied to has succeeded. Although a performance might not have been ideal, it still could have been successful. Examples of this are evident in the experimental results presented in the Section 5.1.6, page 105. If the activity was successful, then it could be said to be profitable (hence the name of these functions). If it failed then it will have made a loss. To go from measuring costs to assessing how much profit a decision has produced requires some assumptions about the relationship between the success and failure of an activity and the profit or loss:-

- A successful activity will never represent a loss for the system
- A failed activity will never represent a profit for the system
- The least costly activity will bring in the most profit and the least loss
- The most costly activity will bring in the least profit and the most loss.

Algorithm 1 has been derived from these assumptions and the preceding work. It computes the profit of an activity's performance. Here $f^s(\mathbf{p})$ is the system cost function, determined for each application, *e.g.* Equation 8.7 (page 198). $activity\_successful()$ the decision function. The details of $activity\_successful()$ are specific to each activity where a cost function is used, but every implementation of the function will return *true* or *false* depending on whether that use of that activity was successful or not. This is the fourth, and last, step in the production of a profit function. The details of expanding this algorithm into implemented code are trivial. However, implementing $activity\_successful()$ might be very difficult.

A final note on the cost function and the coefficients of the profit function. Restricting them to the interval $-1 \leq a_i \leq 1$ does not limit the adjustment of the relationship

Algorithm 1 Relative Profit

```
if activity_success() then
  rel_profit := 1 − f^s(p)
else
  rel_profit := 0 − f^s(p)
end if
```

between individual measurements. It does ensure that the profit or loss values never exceed the set limits. This allows standardisation of the interface between learning mechanisms and profit functions. Such standardisation is useful because it simplifies the application of the same learning mechanism to different tasks. This is because the profit values are constrained to a known range, so the computations they perform and their control parameterisations can be left unaltered. Changing the range of profit values would require the developer to adjust the learning mechanisms. This is a requirement which would make the implementation of those mechanisms as re-entrant[2] difficult. One drawback to this standardisation is that scaling might make certain outputs indistinguishable because the error difference between them is smaller than the resolution of the numerical system. Such differences are unlikely to be significant as their magnitude is far less than the current resolution of many sensors and robots.

3.3 Desirable Properties of Robotic Learning Mechanisms

When stating on page 46 that any learning mechanism can provide additional benefits to the system if implemented correctly correctness was not defined. This section contains that definition in terms of the properties any learning mechanism used in robotics should have. Some of these properties apply to all learning mechanisms. Others relate to the special circumstances of assembly robotics applications. These properties have been used to guide the selection, and implementation of learning mechanisms that constitute the bulk of the experimental work that has been performed. The testing that occurred was from the outset designed to allow verification of the existence of properties such as correctness and convergence. The descriptions of the mechanisms have been presented in a manner which allows identification of properties such as re-entrance that cannot be experimentally demonstrated.

[2] Meaning that the implementation of each mechanism can have multiple non-interacting invocations

The starting point for identifying these properties has been the Holland's list of of design decisions that need to be taken and justified when using learning [Holland 75]:-

- What structures are to be adapted.
- What are the initial structures.
- What is the evaluation procedure.
- What are the operations that change structures.
- What is the state (memory, history, variables) of the system at each stage.
- How is the process terminated.
- What is the method for designating a result.
- What are the parameters that control the process, and what are their settings.

For any given application arriving at the ideal choice for all of these categories is not an easy process, and typically will involve trade offs due to limits imposed by the application, available resources, time, and the researcher. This quote from [Chapman 91] provides a good example of some of the problems that can occur:

> Looking carefully at the backpropagation code [3], I noticed with dismay that it contained several global numerical constants, with comments explained*[sic]* that their values had been empirically chosen for performance in a particular domain. I had naively thought that backpropagation was a fixed algorithm, whereas in fact it is tuned for each application.
> *page 255, paragraph 1*

From this statement and other reports on the application of learning mechanisms a number of common problems that occur when using learning can be identified:-

[3] Short hand term widely used to refer to software which implements a multi layered feed forward Artificial Neural Network using the *Backpropagation* algorithm [Rumelhart *et al.* 86a, Rumelhart *et al.* 86b] as part of the training algorithm.

- Naive application of one of the rapidly growing number of learning algorithms is unlikely to be successful.
- Even with the right algorithm, finding its optimal control parameter settings for a specific application is generally non trivial.
- The solution to be learned must be achievable by the learning algorithm. In [Minsky & Papert 69] is the publication of a proof that one kind of learning algorithm could not be used to learn many of the things to which it was applied. When dealing with more complicated algorithms such analysis might not be possible.
- The right algorithm for a particular application depends on where that algorithm is located in the computational structure of the application.
- The more complicated the application, and the less constrained the activities, the harder it is:-
  - To learn quickly.
  - To learn correctly.
  - To remain responsive to change without degrading performance.

The problems with the mechanisms themselves are compounded when applied to a real assembly robot because a single trial may take minutes (as opposed to many thousands per second in a simulation), and that single trial has (at least during the early stages) a chance of failing. For an assembly robot system failure means that the robot does not succeed, and possibly with disastrous consequence. Even if no damage occurs, user intervention is often required, so adding to the time (and cost) needed, so is to be avoided if possible. For this reason alone the use of learning needs to be evaluated carefully. Only if a learning mechanism is of benefit to the robot should it be considered for use, though this benefit could lie in the robot being simpler to develop.

From this coverage of the problems inherent with learning mechanisms and the complications of applying them to assembly robotics, it is now possible to describe those

properties of learning mechanisms that can be considered assets. These different properties can be grouped into three classes, implementation, operational and learning. Each of these classes is considered separately in the following sections.

3.3.1 Implementation Properties

The two important implementation properties that are required for using learning mechanisms in robotics are that the mechanism operate on line, and that it be reentrant. Both of these properties could be placed into one of the other two classes being discussed in this section. However, they are both properties whose existence can most readily be established through examination of the implementation.

An on line implementation is one which is able to generate an experiment and evaluate the subsequent results during the normal operation of the system. If the system instead needed a separate experimental stage prior to the operational stage it would initially perform better, but as the system changed, it would not be able to respond. To avoid unnecessary experimentation the learning mechanism should include thresholds that specify when performance is adequate. Only when performance drops below the thresholds should learning be performed.

Insistence that learning be carried out during operation encourages use of efficient mechanisms that will not require significant resources. It also encourages the use of mechanisms that make experiments which cannot be easily distinguished from ideal decisions. This last feature could be a problem. Sometimes it is useful to make deliberately bad mistakes as they can provide a lot of information, *e.g.* deliberately inducing a skid in a car is a common training exercise in advanced driving courses as it provides a lot of information to the drivers on a car's performance limits. However, in robot assembly applications there are plenty of sources of errors which do not need to be increased. One of the main reasons for a commercial robot application is to gain better performance than could be obtained from a human, preferably with less effort. If the learning mechanism spends time deliberately causing failures that require intervention to correct, then any longer term benefits could quickly be discounted. Having the system perform correctly, if sub-maximally, is a more acceptable situation.

The initial reason for using the kinds of learning described in Section 3.1.1 on page 45 was to reduce the effort required from the developer when making the system work with a novel application. For learning mechanisms to be useful to developers, they should require little or no customisation. Large amounts of customisation take time and so cost money. If the learning mechanisms are developed to be re-entrant then the only customisation the developer need perform is to create (and initialise as required) a copy of the data structure used by that mechanism for each applications. There are two advantages to this:-

1. re-entrance provides for an economy of space. Only one copy of the actual learning mechanism need be in any given system, instead of having one for each application. If only a single copy is present, then it could potentially be compiled down to hardware, either permanently, or by using a Field-Programmable Gate Array device, *e.g.* [Bowen *et al.* 94, Guo *et al.* 94], with a concordant speed and memory saving.

2. re-entrant learning mechanisms have no internal structures that need to be specified before they can be applied. Instead, the data structure which is supplied as a parameter to the mechanism which is modified by the mechanism represents a template which the developer completes with the details of each application. These details could include the size of the application (though this could be implicit), or which sensors and which actuators that learning mechanism is connected to. As long as the application can use the template, there will not be any problems. Should a different template be required, then probably a different form of learning will also be required. The requirement for learning mechanisms that do not need to be supplied with application specific knowledge by the developer is consistent with a number of other of the properties given in this section.

3.3.2 Operational Properties

Three of the operational properties that any useful learning mechanism should display are that it:-

- Always be tractable.

- Scale linearly.
- Have only a few, evident, control parameters and performance should be monotonic and robust with respect to the settings of those parameters.

The rest of this section expands on these properties, providing a more detailed definition, as well as a justification.

Tractability means that at no time will the mechanism require unobtainable resources, either of computing power or time. Robots have to operate in a world which very rarely waits for them. The decision making process must be completed long enough before an activity has to be completed so the activity can be initiated. There is rarely any benefit to making late decisions, as the right activity at the wrong time can be disastrous. The time required by the mechanism to make decisions and to learn the best decision should scale linearly with the size of the problem that has to be learned, hence the second property in the above list.

If the time taken does not grow linearly with the size of each application, then a constraint is introduced on the structuring and scope of activities. If the task to which a learning mechanism was applied should double in complexity, and as a consequence the mechanism would then take four times as long to learn to a comparable standard of performance, it would seem sensible to split that task between different activities to reduce the size of the learning problem to a tolerable level. If the initial scaling of the learning reflects a quantitative rather than a qualitative change in application, a non-linear scaling learning algorithm might require a qualitative change in the structuring of the system to maintain viable performance. Such change runs contrary to the desire to have application of a learning mechanism require the minimum of user involvement.

The last of the listed operational properties relates to the user interface. In general devices which only have a few controls, with each separate control having an obvious effect whose changes are consistent with changes in the control are easier to use. If the mechanism is robust with respect to changes in the control, then it will operate within a range of environmental conditions without changes being required to the control settings. For robot systems that are to be applied to a wide range of different tasks, such an ability is useful as it avoids the necessity of having to change parameters for

every single task.

3.3.3 Learning Properties

Learning is changing. For a learning mechanism to be useful in a robotic system it should:-

Be correct as if the right thing is not learned then the system in which the learning mechanism is embedded will never perform correctly. The existence of this, and all of the properties listed here, is determined by the underlying algorithm used in the learning mechanism. The ways in which correctness, and the next three properties, can be achieved in the various learning mechanisms that are described in Chapter 5 to Chapter 8 are specific to the type of learning mechanism, and will be discussed where appropriate.

Be optimal as for the mechanism to be useful instead of just being applicable as it would be if it were only correct, the answer it produces must be the best, *i.e.* the optimal answer. If it can find that answer in the shortest possible time from the information it has access to then it will have been learning optimally.

Have low regret[4] as not making many mistakes is important in robotics applications. This because recovery from a failure can be difficult and time consuming, and because a failure might cause damage to the robot or its surroundings. If the class of error which caused the failure was not identified during development, that recovery will be much harder, if at all possible (see Section 2.3.1, page 30,). Another reason to avoid failures is that they can also damage the robot, or the environment to the extent that future use without repair is impossible.

Be expedient as if the learning mechanism does not learn any faster than if it were making random choices, then whatever processing that is required by the mechanism, and the mechanism itself are wasteful overheads.

Handle a non-stationary world because as discussed in the last chapter, and as will be made clear in the next chapter, the relationship between robot and sensors

[4] A widely used term for the numerical estimate of implications of an action [Kaelbling 93]

rarely remains fixed, and during operation the robots themselves change. If the learning mechanism becomes 'fixed' once it has found a good solution, then it will have to be restarted whenever the world has changed in a way to invalidate that solution. Simple procedures for doing this include only calculating statistics over a limited number of previous examples, or using statistics that are normalised with respect to the number of samples used. More complicated procedures such as identification of the most informative results, which are then used in any calculations require knowledge to be embedded into the learning mechanism that will allow it to make those choices. Unless that knowledge will be relevant to all of the applications of that learning mechanism, such approaches should not be used unless otherwise necessary.

Handle a real valued performance measure because if it were only able to handle a binary valued performance measure the learning mechanism would be unable to perform as well in applications where a real valued performance result were available, as is the case in any of the applications listed on page 45. As an example of this consider the number of iterations required to find the optimum of a function such as

$$z = \sin \sqrt{(x^2 - y)^2 + (1 - x)^2 + (1 - y)^2} \tag{3.5}$$

If the only information acceptable to the optimiser were that its last suggestion for a optimum was either better or worse, *i.e.* a binary valued performance measure, than its last but one suggestion, then it is highly unlikely to find a global optimum. Instead it is likely that the optimiser will be trapped into a local optimum or it will cycle. Modern optimisation methods *e.g.* [Yuret 94] are able to use the information contained in real valued results to obtain performances that are typically orders of magnitude better than those that can be obtained by simplistic hill climbing. On a practical note the range of reinforcement is restricted here to the range $[-1, 1]$. As described in Section 3.2 (page 3.2) , defining a standard interface between profit function and learning mechanism allows either to be replaced by an alternative without then having to modify either to suit the new pairing.

Have constant inertia so that as the number of experiments that have been performed grows, the learning mechanism will still be as responsive to any change in situation as it was when only the minimum number of experiments had been performed. If the learning mechanism does not have a constant inertia with respect to the number of experiments it has performed then the impact of new results will be reduced as the history of the mechanism gets longer. Figure 3.1 illustrates this point by showing how the Pearson correlation coefficient between the differences of a sequence of uniformly distributed random numbers in the range $[0, 10]$ and the difference of the running means of that sequence decreases with the increasing length of that sequence. The declining correlation coefficient indicates that the influence of changes in the values used to produce a statistic declines as the number of results on which that statistic is based increases.

In this list, the terms in the **bold font** are the formal names for these properties that are commonly used in the Machine Learning literature, *e.g.* [Narendra & Thathachar 89], with the meaning of these items following from their standard English usage.

The existence of all of these properties can be established from the description of the implementation of the mechanism. However, it is always necessary to back up such analysis by experimental testing. Only by comparing the operational behaviour of the learning mechanism against expectations of what the behaviour would be if these properties existed will it be possible to confirm their existence.

3.4 Summary of Chapter

This is the last of the principally discursive chapters that have laid the groundwork for the experimental work that takes up the rest of this thesis. Together with the last chapter the issues covered here have provided identification of where, and why learning should be used, and what properties it should exhibit when it is used. This identification of using learning to find optimal parameter settings, or good action-sensor couplings, or to choose which activity is best came from matching the kinds of thing weak learning mechanisms are good at with the kinds of things that developers who are producing activities find difficult. That activities are common to both the

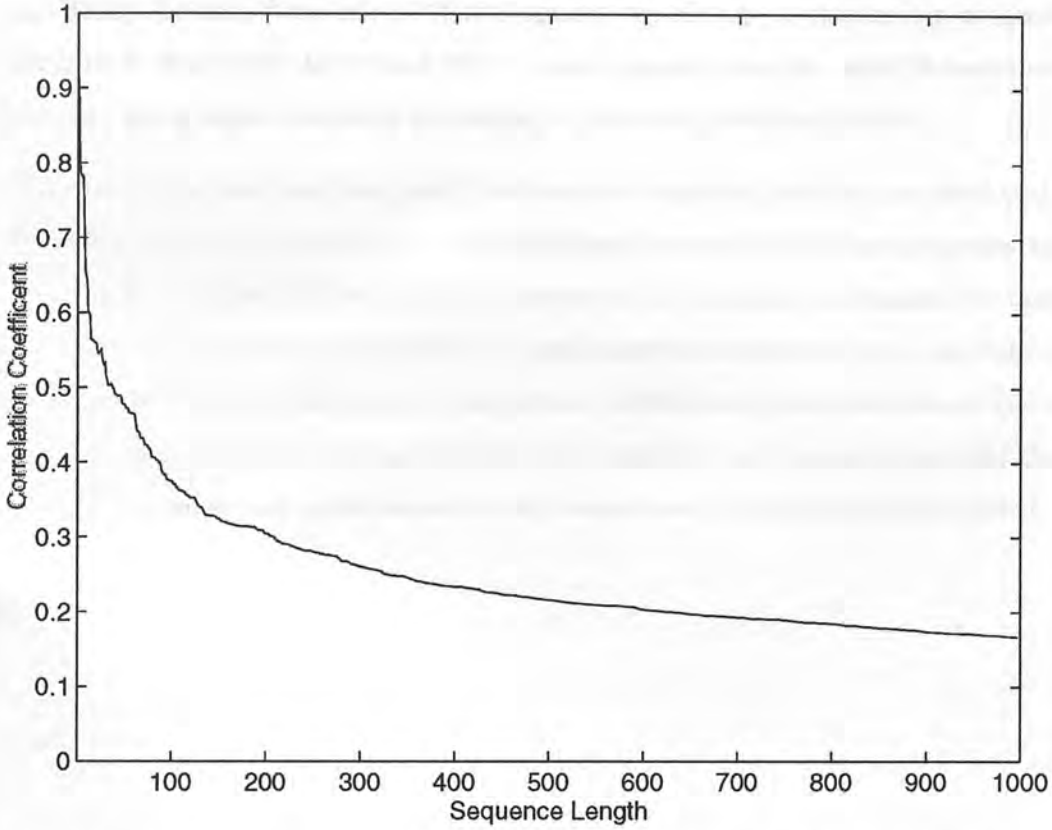

Figure 3.1: Correlation between differences in running mean and sequential tests for different length sequences

older Classical approach (in the technical sense) and the newer Behavioural approach was established in the comparisons made during Chapter 2.

At the same time as this commonality was being uncovered, the principal difference between the two approaches was coming out. This was that the newer Behavioural approach aims to produce systems capable of achieving any instance of a task and openly embraces the point of hiding uncertainty through sensor-action couplings. This is a point which the Classical approach has been forced to adopt in its attempts at producing systems which are customised for each instance of a task. However, the Behavioural approach is deliberately attempting to produce what has been termed here a 'language of action', consisting of a robust and wide syntax coupled with an effective grammar and understood semantics which enables complicated tasks to be specified at levels of abstraction easily handled by planners. When compared and assessed against the Classical approach that in essence re-works each task from physical first principles,

e.g. Handy [Lozano-Pérez *et al.* 92] and against the history of engineering progress [Fredrick P. Brooks 82, McConnell 96], it would appear that the 'new' Behavioural approach has a higher chance of succeeding in producing complex systems.

This chapter has also described profit functions and explained how they are developed. These functions will be used in two of the four learning mechanisms that are presented after the next chapter. Their role is to convey to the learning mechanism the task specific information that is required. The next chapter concentrates on a description of the test problem and the experimental system. In this description examples are given of the dependence of the implementation of the controller on the mechatronics of the system. This important point has so far only been stated, or indirectly demonstrated.

Chapter 4

Subject and Object: the testbed problem and system

The point of the research described here has been to look at learning in robotic systems. To do that effectively a task (or rather a *problem*) and a robot system are required. This chapter describes both of these items, beginning with the problem before going onto the robot system.

4.1 A Problem to Study

For the purposes of this research, a good problem is one that:-

- Can be achieved by a non-learning system, so that a comparison can be drawn between the performance and relative difficulty of learning and non-learning solutions.
- Be sufficiently complex to provide a good test of the ideas that are to be explored.

For the assembly robots, activities such as mating two parts, or acquiring a part, are not the most elementary things the system does. Instead, an arm performing a task spends most of its time moving, or *reaching* from A to B. During repeated assemblies of the same object, the paths followed during each assembly will be repeated for each object. One of the simplest ways to program paths is as a sequence of taught positions, *i.e.* at the joint level (Section 2.1.1, page 13). However, as noted in that section, this kind of programming is difficult and error prone.

In Chapter 2 a variety of techniques and methods were described that have been proposed for improving the level of robot programming. Bar two exceptions, described in Section 2.2.1 (page 15) and Section 2.2.4 (page 18), all of these methods seek to use action-sensor couplings, *i.e.* for robot motions, whether they are reaches or part mating motions, to be guided, in some sense, by the information processed from sensors. In the context robot reaching, this means that that the target locations may have to be acquired and defined by sensing, or calculated, and the arm then guided to them. This requires a transform to be determined that relates points in the coordinate frame of the sensor to points in the coordinate frame, or control space, of the robot. This problem will be used as the subject of the experiments with learning mechanisms that take up the next four chapters.

This is good problem to choose because it is one where there an potential application of weak learning. One route for implementing reaching is to use a non-learning, or "fixed" method. This is where no change is made to the transform during the operation. These methods acquire the necessary knowledge before the system goes operational during a calibration stage. That calibration can either be manual, or automatic, but once performed it is never updated. As long as the relationship between the arm and the sensor systems is static, then the calculated transform will remain correct, but it is very difficult, and rare to have a truly static system. The following quote illustrates how even a high quality robot can change its operating characteristics in use, and that those changes can lead to problems if not accounted for.

> We saw sluggish response when reversing direction, and mechanical drift when making orthogonal changes in direction. We only noticed these effects because we taking measurements to the thousandth of an inch, and these effects were on the order of 20 to 30 thousandths. For this same reason, we had to "exercise" the robot for about 45 minutes before testing. The robot, being aluminium would grow some 20 or so thousandths of an inch as it warmed up. We also noted that this warm-up changed the friction at the joints, and that affected the drift and hysteresis of the system. Again, all these effects were dependent on the arm configuration.
>
> [Parma 92] *page 138, paragraph 3*

From this example, it would appear that even if the relationship between sensor and arm is static, the arm itself is not, and inaccuracies will appear in the transform that will not be detectable by the system.

Learning can be used here to continually update the knowledge maintained about the relationship, allowing the same accuracy of reaching to be maintained irrespective of the changes in the relationship between sensor and arm. Different kinds of learning can be used depending on what information is required by the different reaching modules, each of which will have a different implementation.

Here, for further simplicity, reaching will only be performed in a horizontal plane. This is a specialisation of general reaching, a restriction which will be borne in mind as the various reaching modules are developed. For an RTX, the type of robot used here, operating in the plane does not significantly reduce the number of singularities, or locations in its workspace that can be reached by multiple arm configurations. If the various reaching mechanisms that will be developed are able to handle these singularities, then they can easily be integrated into a complete system, especially if they are developed with this end in mind. Once a number of reaching modules have been developed, the issue of how to learn in decision functions can be explored by seeing which of those modules is best.

Using reaching as the subject of this research will avoid the technical complexities of part acquisition or mating, and allow the repeated testing that will be required to assess the behaviour and performance of the learning activities. The modules that are developed here with arms should be applicable to moving a mobile robot to a target. As long as it is possible to sense, or calculate the relationship between the robot and the target, and that this is all that is required by the reaching module, then transfer of the module should be possible.

Such 'toy' problems as proposed here have been used before in assembly robotics research. SOMASS, Malcolm's demonstration [Malcolm 87] of behavioural robotic assembly used SOMA parts as the components. Although these parts are simply wooden blocks glued together, they retain the variability of real parts, and can only be fitted together in ways which depend on their shape, both of which are important properties

of the components of a real object. The final aspect of this world was the uncertainty in initial part location. Together these three features defined a micro-world which exhibits the characteristics of the real world which SOMASS was intended to handle, but without the complexity of the real world, allowing the demonstration of a conceptual point with an efficiency of development and experimentation.

In the next section the system that was constructed to be the testbed environment in which to explore this problem will be detailed.

4.2 The Research Testbed

To recapitulate from Chapter 1, the interest here is how, and if, learning can be used to improve the reliability and ease the development of a robot system. The problem that the system will be applied to is that of determining the transform between sensor and robot coordinate frames that will allow a successful reach to a sensor selected target point with an arm. The experimental system used to try out solutions to this problem must exhibit the uncertainty, variation and control problems characteristic of robotics. These problems are found throughout any full scale applications, so to retain the generality of any results, the system used here should retain these problems.

To do this an arm and two sensor systems were used that guaranteed uncertainty. A review this equipment occupies the rest of this section.

The arm and the force table, one of the sensory mechanisms used, are shown in the photograph in Figure 4.1. The camera, which is not shown, but is the other sensor mechanism, is positioned vertically above the force table pointing downwards.

Figure 4.2 is a schematic of the computational architecture of the system.

4.2.1 The IT Components

The robot, the vision system, and the force table are connected, and controlled, through a Sun 3/260 workstation running SunOS 4.1, and an Ironics IV-3220[Iro88] board mounted within the workstation. The Ironics board was managed using the Chimera RTOS [Adv91]. This workstation also served as the development and test platform for

Figure 4.1: RTX robot and Force Table

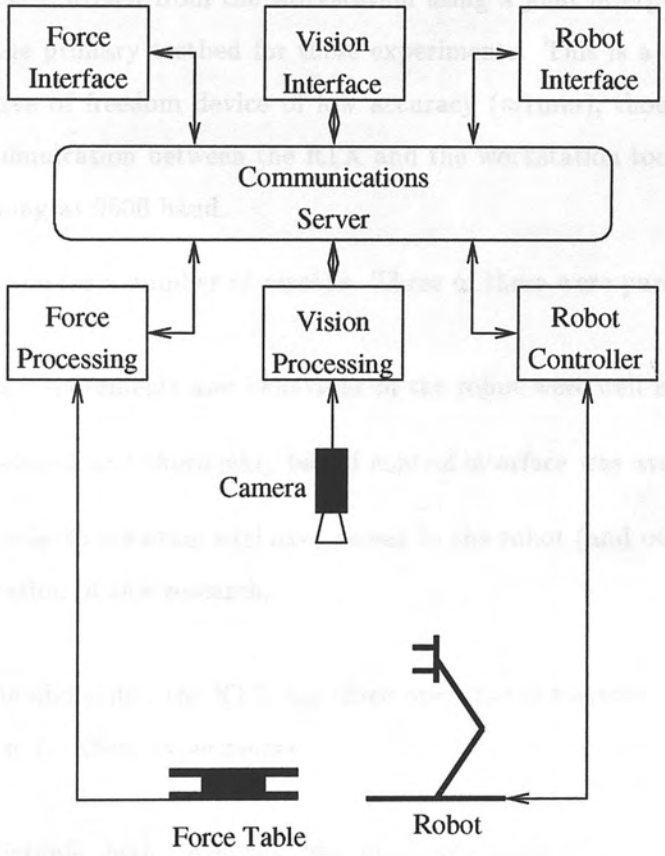

Figure 4.2: System Layout

all software developed and used during this work.

Of the software developed during the research that this thesis documents, only the core algorithms have been listed, as opposed to verbatim reproductions of source code, *e.g.* Algorithm 2, page 77.

4.2.2 The RTX

A UMI[1] RTX robot, driven from the workstation using a joint interpolated cartesian controller, was the primary testbed for these experiments. This is a relatively cheap (≈£5000) 6 degree of freedom device of low accuracy (≈1mm), though of adequate resolution. Communication between the RTX and the workstation took place over an RS-232 link running at 9600 baud.

An RTX was chosen for a number of reasons. Three of these were purely pragmatic:-

- The design, requirements and behaviour of the robot were well known.
- A well developed and thoroughly tested control interface was available.
- It was possible to maintain exclusive access to the robot (and other equipment) for the duration of this research.

In addition to the above list, the RTX has three operational features in that makes it eminently suitable for these experiments:-

- An unpredictable, high frequency, low amplitude oscillations about the current position following some moves as a consequence of the PID parameterisation of the servo controllers[2].
- A slow drift during use in the coordinate system that is dependent on activity patterns. For example, after undertaking 55 random reaches within the horizontal plane, the euclidean distance between arm positions after directing the arm to

[1] now Oxford Intelligent Machines

[2] Adjusting the PID values to remove one oscillation merely causes another at a different time. The values used represent the least bad settings achievable for general motions

return to its first target was on average 9.65mm, with a standard deviation of 2.34 (statistics calculated from 20 runs).

- A divergence, over time, of the operational characteristics of the arm when compared to identical models that have experienced different operational regimes. Examples of this have been observed every year here in the AI department during a practical for the *Intelligent Assembly Systems* course. Participating students use one of a number of RTX robots to develop a program that can knock over and then reassemble a tower of wooden blocks. Only a few students transfer their programs between different robots because each robot requires different parameter values for the various operations required.

For the reaching problem these features will make the task of the controller harder because they ensure that over time the positional accuracy of the robot controller relative to an external coordinate frame of the robot will degrade. These various features also define much of what the controller has to do in order to fulfill its specified task. Going through them here shows why the RTX is a good choice of arm for the testbed because of the problems it generates for the controller. That these problems have an impact is evident in the various test results of the learning mechanisms, *e.g.* see Section 5.1.6 on page 105 or Section 7.5 on page 171.

The last of these features is particularly interesting, as this variation is a reason why learning should be considered for inclusion in a controller, if for nothing else to handle and monitor such long term creep. The alternative to using learning would be to enforce a strict maintenance and replacement schedule. Although this would merely be an extra cost for some uses of robotics, such as factory assembly, for others such as planetary exploration where the robot will not be attended, it can never be a serious option. Even in the case of maintenance being possible and affordable, learning to handle creep should increase the intervals between maintenance. From the data gathered as part of the learning process, it should be possible to obtain a clearer picture of how the system is changing, and so what maintenance is needed when for that system.

A final reason for using an RTX is a political one. Many industrial applications of robots use high cost, high precision arms with little sensing. If by coupling sensing to

an RTX with a good controller provides the same, or a higher level of competency as those more expensive systems but for a lower cost, then robotics usage might become more attractive. Although the system proposed here is meant for a single task, it has been deliberately developed to be the basis for a full scale, competent system. Section 9.2, page 219, of the concluding chapter considers what would be involved in the production of such a system.

In the remainder of this section, a description of the RTX controller and the calibration routine for the robot will be provided before the review of the testbed moves onto the vision system.

The RTX Controller

The low level controller of the robot is implemented using joint interpolated position controller. This controller is given locations specified as cartesian positions. It then uses an inbuilt kinematic model to transform the specified locations into arm joint angles which are then sent to the servo controllers of the relevant joints. The controller can be queried to determine the cartesian location of robot, from its internal sensors. The controller does not contain any dynamic modelling, and it is difficult to predict the trajectory of the arm's links during a given motion.

The software that implements this controller was initially developed by Lykourgos Petropoulakis[3]. Subsequently, the author has extensively developed the controller to improve its performance, functionality, and interface.

Calibrating the RTX

The controller includes a calibration routine, which was used before each experiment. This calibration consisted of driving all of the arm's joints in sequence, beginning with the Z-axis, to their end limits. Once each joint reached its end limit, detected by an signal being generated by the robot's on board controller, the motion was halted and the software counter for incremental positional encoder on that joint set to zero.

[3] Now at Strathclyde University

Image Size	Transfer Time (ms)
64x64	12.5
128x128	50
256x256	200
512x512	800

Table 4.1: Image transfer times

4.2.3 The Vision System

The vision system has been built around a Digimax Datacube digitiser and framestore that are mounted inside the workstation. Images are collected from a Sony WV 100-KT camera equipped with a 1.4f auto focus zoom lens of focal length between 10.5 and 84mm and an auto-exposure facility, both of which were used. Two auxiliary monitors were used, one to show the output from the Datacube and a second showing the output of the camera. No special care was taken with the illumination of the scene apart from normal laboratory lighting.

All of the processing and control for the vision system occurred on the workstation. On command the current video frame is frozen into the framestore before being copied into the workstation memory. The extent of the frozen image that is copied to the workstation memory is controllable. Images could be acquired at up to 50Hz, but the times required to transfer the image to memory (shown in Table 4.1) limit the maximum rate achievable. Processed images could be sent back to the framestore for display on the dedicated monitor connected to its video outputs. This facility was used during debugging and demonstrations, but otherwise disabled to save time. The software used to implement this functionality was developed from software produced by Chongstitvatana and Conkie [Chongstitvatana & Conkie 90a, Chongstitvatana & Conkie 90b] during their experimental work. However, the design and implementation of the gripper localisation mechanism, described next, was done specifically for this project.

Visual Finger Recognition

The QUEASY, P_MOVE and I_INTERP reaching algorithms, described in Section 5.1 on page 95, Section 6.4 on page 136 and Section 7.3 on page 164 rely on being able to detect the location of the midpoint of the gripper fingers.

Algorithm 2 The Image Differencing Algorithm

```
for x=0 to image1_x_size do
  for y=0 to image1_y_size do
    if abs(image1[x][y] − image2[x][y]) ≥ d_thresh then
      image3[x][y]=255
    else
      image3[x][y]=0
    end if
  end for
end for
```

The initial approach to localisation was to use an activity which takes two equal sized images, one with the grippers closed (*image*1), and one with them open (*image*2). The difference between these two images is then calculated using Algorithm 2. This algorithm takes as input two images, referred to as *image*1 and *image*2 and does a pixel by pixel comparison. If the absolute difference between two pixels is greater than some threshold *d_thresh*, then the corresponding pixel in an output image *image*3 is set to a value of 255, which in for this system corresponds to white. If the difference is less than *d_thresh*, then the pixel in *image* is set to 0, or black. This algorithm is dependent on the tips of the robot's fingers have been painted white, so that after Algorithm 2 is applied *image*3 just contains four white blobs that the location of the fingertips in *image*1 and *image*2.

By calculating the centre of mass all the pixels with a value of 255 in *image*, an estimate is generated for the location of the midpoint of the fingers.

During testing, two disadvantages of this approach became apparent, the first of which was expected:-

1. It was not possible to determine gripper orientation with respect to the camera from a single localisation. At best there is a $\pm\pi$ rotational ambiguity. Two separate localisations, the second following a specified arm motion, will allow gripper orientation to be established.

2. Unless the background is a dark colour, snapping the fingers causes more to appear in the differenced image than the fingertips. This noise affects the calculation of the midpoint. Attempting to set *d_thresh* to a sufficiently high value to

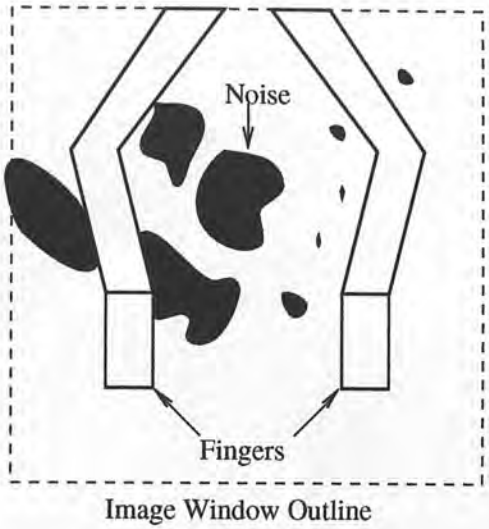

Figure 4.3: Example of image noise when snapping fingers

exclude this noise results in part or all of the fingertips also being excluded.

Figure 4.3 shows a typical result of snapping the fingers when noise has occurred. A two stage approach was used to handle this noise:-

1. The robot gripper was converted into a "webbed" hand by gluing a piece of thin and slightly elastic material between the fingers. This effectively removes all the noise that occurs inside the outline of the two fingers shown in the diagram.

2. The localisation activity was extended to first rotate the hand with its fingers open by 180°, then take a third snapshot, and finally, to restore the hand to its initial orientation. The difference is then determined between this image and the image of the gripper the right way up with its fingers open using the same operator that extracts the initial fingernail image. This new image should only show the fingernails, the noise having been cancelled out. In practice, because the hand moves slightly during rotation and of the aliasing effect in the camera, there is some residual noise. This noise was handled by only searching for the fingernails in the image region where they were initially detected.

The error between the reported and actual midpoint of the gripper fingers in the image after this final stage is on average 2 pixels, or 1–2mm in the x-y plane of the robot's

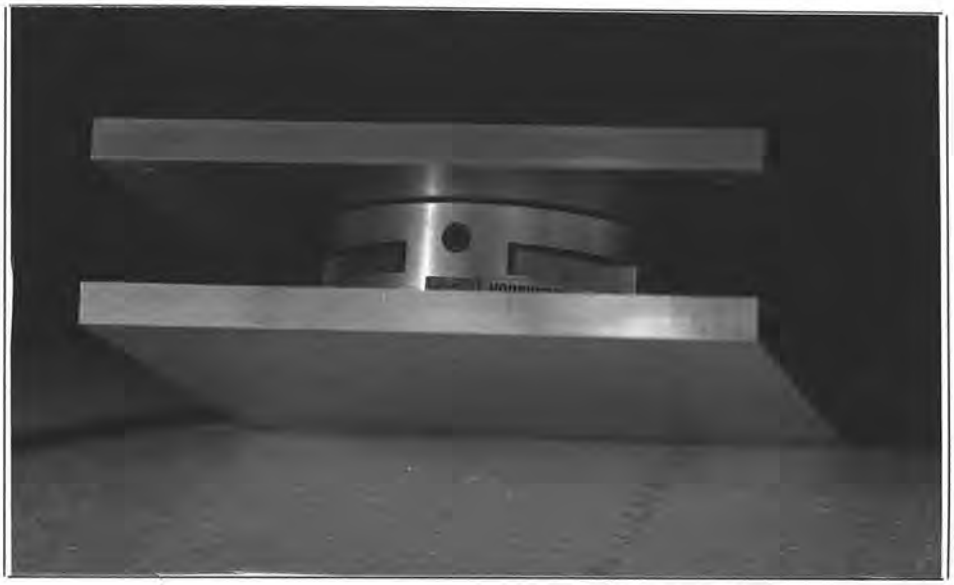

Figure 4.4: Underside view of Force Table showing Lord F/T sensor

coordinate frame. This resolution is felt to be accurate enough for the purposes of this research, and no further work was undertaken to improve it. In Chapter 9 the question of how the gripper can be located accurately in the image is considered further.

4.2.4 The Force Table

The Force Table, shown in Figure 4.4, is a wooden tray securely fixed to a Lord 15/50 6 Axis Force Torque sensor. The tray was made from two laminated sheets of Marine ply and has a 1cm lip protruding up around the perimeter. Data from the sensor is pre-processed by vendor supplied electronics before being passed into the Ironics Board along an RS-232 line where it is filtered and converted into a table xy position.

The forces and torques resulting from a contact being made with the surface of the table are output by the sensor as a 6 element vector at an effective rate of 55Hz. The values of the elements of the output vector are integer multiples of a manufacturer specified generalised force (*uf*) and torque(*in-uf*), where $1uf$ is equal to $\frac{1}{5}oz$. During this work, the real world values of a force are not required and are so ignored (See Appendix B).

The resolution and the quantisation error of the sensor are shown in Table 4.2 in units of *uf*.

	FT Component					
	fx	fy	fz	tx	ty	tz
Resolution	1.0uf	1.0uf	3.0uf	2.0uf	2.0uf	2.0uf
Quantisation	±0.5uf	±0.5uf	±1.5uf	±1.0in-uf	±1.0in-uf	±1.0in-uf

Table 4.2: Resolution and quantisation error for a Lord 15/50 FT Sensor [Lor87]

The mathematics of converting from the forces and torques registered by the sensor to a position are described in Appendix B on page 243. These calculations are performed on the Ironics board to remove an otherwise CPU intensive task from the Sun. Unfortunately, naive implementation of these equations is not sufficient to produce useful information. Noise, evident from the varying estimates for a single unmoving point of contact, is significant, and has to be dealt with. Even without the noise the output of equations B.6 and B.7 can only be treated as an estimate of the current position of applied force because:-

- The FT sensor is only capable of finite resolution.
- The Force table is not mechanically isolated from the environment, so external vibrations introduce extra applied forces of variable and unpredictable duration.

When noise is factored in, the situation becomes complex. Figure 4.5 shows how the position estimate can vary over 1 second (55 samples) for a 20g weight in the normal operating environment for the table. That there are only 15 estimates shown in this figure reflects the fact that many of the position estimates are repeated within the sample. This pattern of estimates, and the repetition, is typical for the table, with the estimates falling into an elliptical area, the axes of which are aligned to the diagonals of the table. The extent of the ellipse varies with the location and mass of the objects and with the distance of the object from the centre of the table.

This pattern might appear to be surprising because Table 4.2 suggests that there would be a circular distribution of estimates around the true position. The distortion of the pattern is principally because the force table is rectangular.

If the robot were to be servoed in for a grasp using these estimates, it is unlikely to succeed. Every time a comparison was required between the location of the grippers

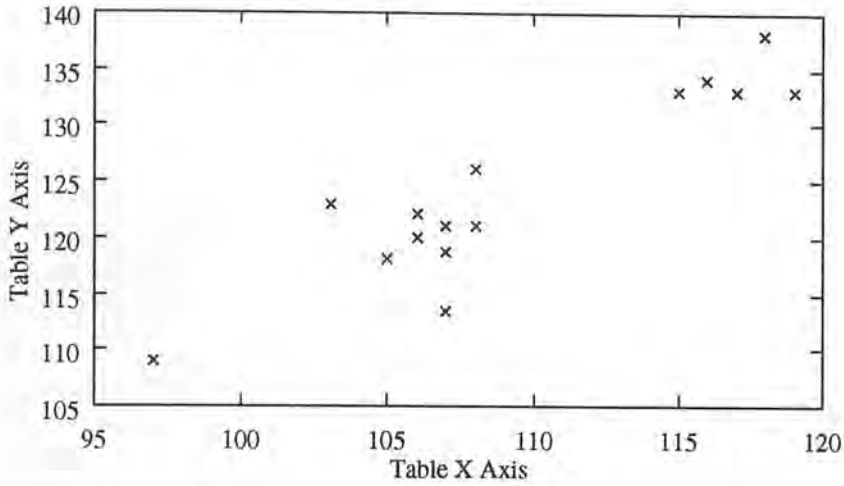

Figure 4.5: Position estimates over 1s for a stationary object

and the target, the target would have apparently moved. Repeated localisation of the grippers without moving the robot would suggest that the robot had moved. Two approaches have been used to try and improve the localisation of the contact point:-

1. The running averaging filter built into the F/T sensor's processing unit was activated.

2. Kalman Filter was implemented in software on the Ironics processor.

These two methods were combined to try to obtain the best possible localisation. The detail of these filters is given in Appendix C on page 245. The effects of using the best kind of Kalman Filter combined with the averaging filter are shown in Figure 4.6.

For the application considered here of localisation of contact points on a table, the stability of estimate is a valuable property. This is because attempting to servo an arm to a randomly moving point is neither an easy prospect, nor an activity consistent with the goal of trying to pick up an static object from the table. The gating used in the second Kalman Filter (Algorithm 10, page 251) provides this stability at the cost of requiring two extra parameters to be set. Whilst it would be possible to choose nominal values and otherwise ignore this area of the system, it is sometimes an advantage to be able to adjust the sensitivity of the parameters to take into account features of an application. Noise from the F/T decreases dramatically as more force is applied at a point, so heavier objects are localised more accurately than are light objects. For these

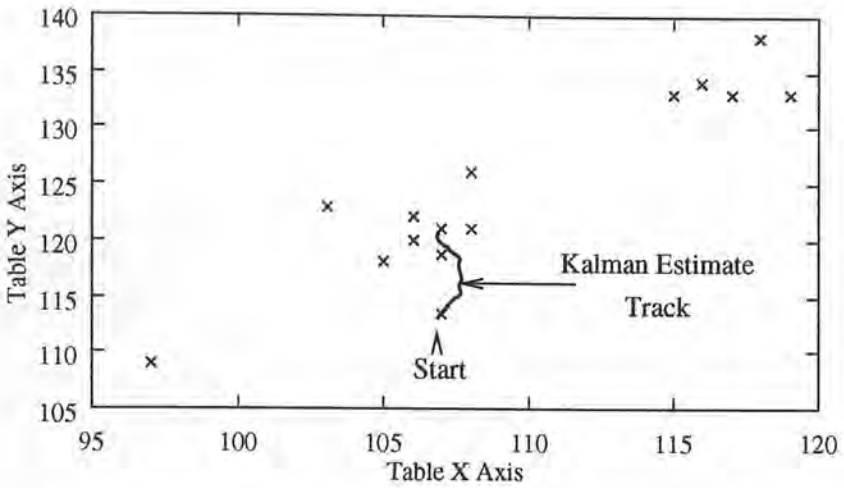

Figure 4.6: Position estimates over 1s for a stationary object using the Kalman Filter with updating

two cases then different parameter settings might be considered to obtain more or less sensitivity to change in the situation, *e.g.* a sensitive parameter setting would enable changes in the location of the heavy object to be rapidly determined, but would not provide a sufficiently accurate estimate for a light object.

Locating the Fingers

The activity for locating the fingers with respect to the table is, like the similar localisation module for the vision system, simple but robust.

The fingers are tilted downwards until contact is made while holding the rest of the robot stationary. This probe is a guarded move which does not need to be supplied with the exact height of the table in the robot's coordinate space. The move has to be performed at a slow speed to allow the contact to be determined and a halt command sent to the robot. As the stop command takes 64ms to be executed the gripper cannot be moved too far on any one move otherwise excessive and potentially damaging forces are generated. The tilting of the fingers causes the point of contact to be offset from the actual finger position. To correct for this the fingers are initially displaced in a direction to counter this offset, and when contact has occurred, then the new position of the fingers is compared against the original position. If there is a significant difference then the offset is adjusted and the procedure repeated. This solution is needed to

ensure correspondence between the localisation of the gripper performed by the vision system and the localisation by the force system. The procedure used to achieve the solution is not the most elegant, but for this application, it was effective.

When a contact is made then the averaging filter and the Kalman filter discussed in the last two sections are used, with the averaging filter feeding into the Kalman filter, in calculating an estimate for the point of contact. By taking 100 readings, requiring $\approx$ 2 seconds, and processing through the filters, a good estimate of the contact point can be established. Before beginning the localisation, the force readings are sampled, and then this sample is subtracted from all subsequent readings during the probe to avoid confusing the robot's contact with any other objects resting on the table. Each application of this tapping module takes 32 seconds on average, most of the time being spent in waiting for the robot to make contact with the table.

This activity is an example of a low-level activity that would be called from another activity. There is an action function where sensing and acting are tightly, and specifically, linked in an implementation of a guarded move, while there is decision function that checks the result of the guarded move, possibly calling for a repeat if necessary. The estimation of the location of the point of contact from the sensed forces and torques makes use of a model of how the forces detected by the strain gauges in the sensor relate the applied forces and torques (embedded within the electronics of the force controller), and a model of the relationship between the forces and torques produced by a particular point of contact. All of this was developed by hand and required extensive testing to achieve confidence in the output. Scaling up from producing one activity to a whole system would require the process of development used to produce the tapping module repeated many times and the introduction of a structural framework to organise the different modules. Tapping was quite easy, as the relationship between input and output could be analytically derived, but for activities such as reaching, such analysis would not be possible.

4.3 Experimental Details

In each of the next four chapters the results of experiments using a variety of activities are reported. These experiments have all been performed following template that is described in this section. This description has three parts:

- Experimental Goals: The reason for performing the work.
- Experimental Environment and Stages: The environment and the stages for the experiments.
- Experiment Parameters: The number of trials, etc. and the target points used in each experiment

4.3.1 Experimental Goals

The goal of each experiment was to assess the performance of the activity in question against that of the other activities developed during the research documented here. This comparison is important here because the hypothesis being explored here is that learning can be used to improve performance while reducing the development workload. Only if the activities that use more learning can be shown to have equal, or better performance at the same tolerance (See Section 4.3.3, page 86) will this hypothesis be validated.

However, this assessment depends on establishing a common measurement framework. The specification of an ideal performance is the route that has been used here to generate that framework.

Here, an ideal reaching performance would consist of the arm moving from the initial to the final location in a single step move executed at the maximum possible speed, with the arm arriving exactly on target. This definition can be translated to saying that the best variant is the one with the smallest number of steps, highest accuracy, highest efficiency and requiring the shortest time. Hence, the ideal, and unobtainable performance would be a single step, zero position error, maximum efficiency motion requiring zero time.

This specification of an ideal performance is next used in Section 4.4.1, page 90 where the meanings of accuracy and efficiency are described, along with the other information that has to be collected to allow meaningful analysis of the behaviour of the activities to take place.

4.3.2 Experimental Environment and Stages

All the experiments took place within the workcell in the Robotic Assembly Research Laboratory, Department of Artificial Intelligence. During the experiments normal laboratory lighting (no natural light sources exist within the lab) was used. Also, during the experiments, other work continued within the laboratory, though not within the workcell.

The stages of each experiment were:

1. The parameters (See Section 4.3.3, page 86) for the experiment are entered into the system.
2. All equipment is powered on, if not already on.
3. The software system is started and initialised. From this point on the testbed is under automated control.
4. All necessary calibration routines are executed.
5. The experiment is then performed.
6. All necessary close down routines are executed.
7. The software system removes temporary files and closes. The testbed is returned to manual control.
8. As appropriate, equipment is powered down or set to stand by.
9. The data files produced during the experiment are manually verified to check experiment has completed.

Whilst the testbed was under automated control status information was displayed on the Sun Workstation screen. If problems occurred during an experiment that were beyond the scope of the automated system to resolve, standard practice was to manually halt the system and end the experiment.

4.3.3 Experimental Parameters

Three types of experimental parameters were used during each experiment:

- The control parameters for the experiment.
- The control parameters for the reaching activity, or in the case of the experiment in Chapter 8, page 180, the activities, with the parameters for the ANNIE algorithm being described in Section 8.2.2, page 190.
- The targets for the reaching. This is the information supplied as the task parameters to the reaching activities.

Each of these types will now be explored in turn.

Experiment Control

The two parameters used in the control of each experiment were the number of trials to take place, and the name of the experiment. The latter was used as the root name from which the file names for the various data log files were derived. The number of trials specified the number of times the system was to attempt to reach a target, each attempt being to a different target. For all experiments this was set to 50.

Activity Control Parameters

Each of the activities that is described in the next four chapters is controlled by two parameters:-

- The tolerance of the reaching, or the maximum acceptable Euclidean distance, measured by the sensor system, that can exist between the gripper and the target

at the end of the activity for success to be declared. This parameter was varied from 1 to 5, *i.e.* for each activity the experiment will be repeated five times, using a different tolerance value each time. The meaning of the tolerance parameter is dependent on the sensory system being used: pixels for the vision system, millimetres (mm) for the force table.

- The maximum number of iterations, or attempts, allowed for the activity to try to reach the target with the arm's gripper. This parameter was set at 10 for all experiments.

Targets

As noted in Section 4.1, page 67, much of the time spent by a robot during assembly is in moving the gripper, or reaching, from one location, or target, to another. A good set of target data will then, for this work, be statistically similar to the data that would be produced by recording points reached by the gripper during an actual assembly.

However, two issues need to be addressed before the source the actual data is identified:-

- In what frame of reference will the targets be specified.
- Will the same, or different sets of targets be used. If different sets, how many and how different do they need to be.

In many robot applications, *e.g.* [Nevins & Whitney 78, Malcolm 87], the targets of the arm's reaching are specified in the robot's frame of reference. As such, it is not unusual to read the source code for these systems and find tens, if not hundreds of explicit positions. Whilst this works, it is dependent on an assumption being maintained over all of the cycles of the system, even when sensor systems are used. This assumption is that the position of the robot relative to the sensors and to the work environment remains constant. In Section 4.3, page 84, a quote was presented to show how, for what was a heavily engineered system, a slight violation of this assumption was sufficient to cause problems.

An alternative approach, is to specify the task in terms of the sensory system, *e.g.* [Harris & Conkie 92, Balch 92b, Chapman 91]. Providing the control system is able

to use the information provided by the sensor system to guide the arm, then it is potentially a far more robust method. Now, the relationship of the robot, the sensor system and the work environment need not be as rigidly fixed. The relaxation of this constraint, which has been found to be very difficult to establish, let alone maintain, *e.g.* [Parma 92], has been found to be a very powerful step. One example system that uses this approach is able to complete a task providing that the underlying assumption of the gripper and object concerned are in the same image at the start is true [Harris & Conkie 92].

This approach has been continued here, with the targets and the tolerances being specified within the experiments in the coordinate frames of the sensor systems. This means that when positions are being referred to then the units will be either pixels for the vision system or millimetres for the force table. This convention will be followed throughout this thesis. Using this convention means that it will not be possible to directly compare the results of the different sensor systems accurately. It does however, allow some assessment of the independence of the activities from the sensor mechanism.

However it is worth noting that the two systems have similar resolutions, i.e. 1 pixel in the image is approximately equivalent to 1mm on the force table, though this is exactly equal to 1mm in the robot's horizontal plane. This means that if a pattern of targets in one sensor system represents one set of target positions for the arm, then transferring that pattern to the other sensor system will result in a similar (though possibly rotated and translated) set of target positions for the arm. It should be noted that this relationship between the vision system and the force table was not deliberately created.

The second question raised at the start of this section concerned the number of target sets. If the purpose of this work were to fully assess the behaviour of each activity, then it would be appropriate to use more targets than the 50 stated in Section 4.3.3, page 87 through larger target sets or multiple experiments with different target sets. However, here the interest is in the comparative performance of the activities. Although it is important that their behaviour be tested, it is more important that that testing takes place in a consistent environment, *i.e.* , it uses the same target set.

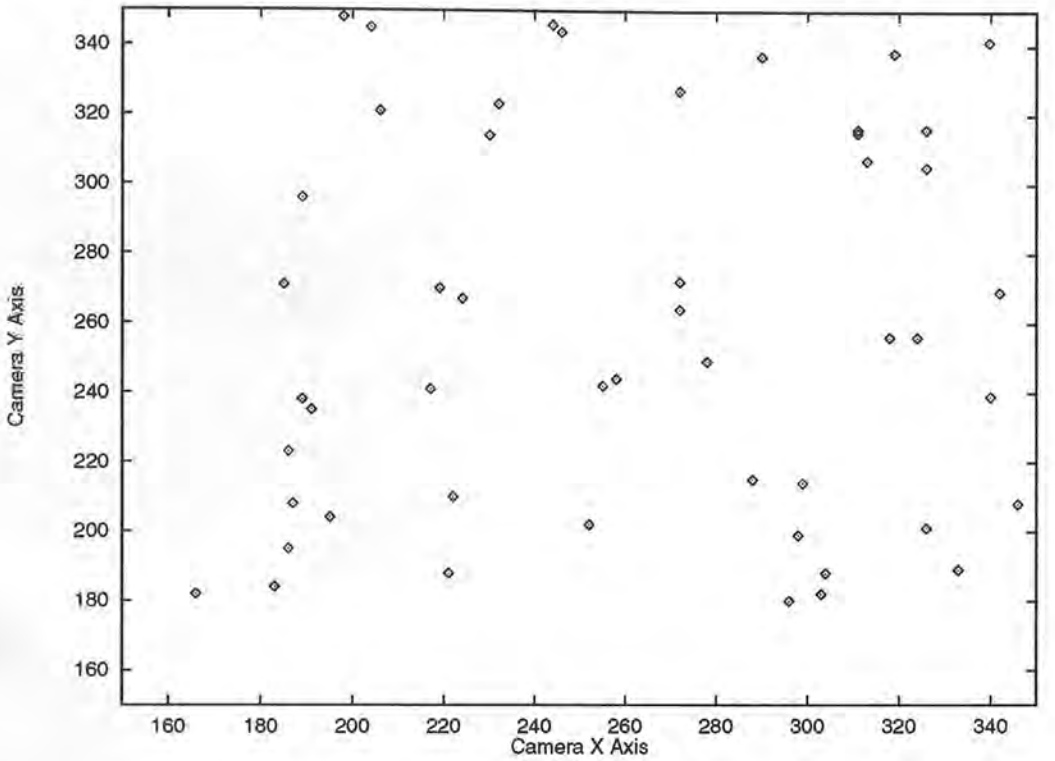

Figure 4.7: The Target Set for Reaching used during the experiments

From this discussion the two issues can be resolved; here, a single target set will be produced and reused in each experiment. That target set will be specified in terms of the coordinate frame of the sensor system being used.

The contents of the data set are easier to specify. Statistical analysis of data from undergraduate practicals and other research work indicates that it is reasonable to characterise the pattern of targets used during any one assembly as 'random' if the actual task being performed is ignored, as is the case here. From this observation, the simplest method for generating the target data set was used: a uniform random number generator (**ran1**, described in Appendix A, page 241)was called 100 times to produce 50 x coordinates and 50 y coordinates. An alternative, such as transferring an example task into this system would have required more work, and not have produced a statistically different target set.

These target points are shown in Figure 4.7.

4.4 Data Collection and Analysis

During the experiments described in the next four chapters, each following the template given in Section 4.3, page 84, data was collected and logged. This section describes this data, and then how it was analysed. This description does not cover any of the other information that was logged during each experiment by the system, such as state, system progress, etc. as this was used purely for system debugging and verification purposes.

4.4.1 Data Collection

Each experiment consisted of 50 trials (see Section 4.3, page 84). Each trial entailed moving, or reaching the arm from its then current location to a target. The same set of targets was used in each experiment.

During the experiment the results of each trial were logged. Once the experiment was over then this log was processed to produce the following information:-

Position Error The mean (μ_{er}) and standard deviation (σ_{er}) of the Euclidean distance between the final observed position of the robot and the target position. If the distribution of final positions about the target point was not isotropic, a special note was made. The measurement units used here are dependent on the sensor system: pixels for the vision system and millimetres (mm) for the force table.

Efficiency The mean (μ_{ef}) and standard deviation (σ_{ef}) of the quantity

$$\frac{\sqrt{(\vec{r}_{final} - \vec{r}_{initial})^2}}{\sum_{i=1}^{n} \sqrt{(\vec{r}_i - \vec{r}_{i-1})^2}} \tag{4.1}$$

where $\vec{r_{final}}$ is the robot's final location, $\vec{r_{initial}} = \vec{r_0}$ is the initial point of the robot, and $\vec{r_i}$ is the robot's location after iteration step i. This measure tends to 1 as the route taken by the robot approaches a straight line.

Iterations The mean (μ_i) and standard deviation (σ_i) of the number of moves taken to reach the target.

Failures The number of times the algorithm has failed at a particular tolerance.

Time The mean (μ_t) and standard deviation (σ_t) of the time taken by each variant to reach the target.

Euclidean Distance The mean μ_E and standard deviation σ_E of the Euclidean distances of the outcome of each trial from the ideal outcome. The Euclidean Distance d_e between an actual and the ideal result is calculated by

$$d_e = \sqrt{(PositionError - 0)^2 + (Efficiency - 1)^2 + (Iterations - 1)^2} \quad (4.2)$$

Mahalanobis Distance The mean μ_M and standard deviation σ_M of the Mahalanobis distances between the ideal and actual outcome. The Mahalanobis Distance d_m between the actual and ideal result is calculated by:

$$d_m = \sqrt{(\mathbf{m} - \mathbf{i})^T \left(\frac{\mathbf{P}}{n}\right)^{-1} (\mathbf{m} - \mathbf{i})} \quad (4.3)$$

where $\mathbf{P}$ is the covariance of the ideal and the actual results. $\mathbf{m}$ is the mean vector of the actual results ($(PositionError, Efficiency, Iterations)$). $\mathbf{i}$ is the ideal result ($(0, 1, 1)$), and n the number of samples in the actual results. If $\mathbf{P}$ is singular, as when the one or more of the statistics obtained during a test has a zero variance, then the inverse of $\mathbf{P}/n$ is determined using the Moore-Penrose pseudo-inverse. This produces a usable matrix, but with the relevant row(s) and column(s) set to zero. If the results are normally distributed, then values of d_m^2 will be from a χ^2 distribution of $\dim(m) = 4$ degrees of freedom [Manly 86].

These metrics have been developed using the framework specified in Section 4.3.1 on page 84. The last two metrics, the Euclidean and Mahalanobis Distances, are derived from the first three metrics and are used during the analysis stage.

Once sufficient data had been gathered, it was then subject to an analysis stage. This is described in the next section.

4.4.2 Data Analysis

The goal of analysing the data produced during the experiments described in Chapter 5 (page 94), Chapter 6 (page 121) and Chapter 7 (page 152) is to compare the behaviour of the different activities. In Chapter 8 (page 180), the role of the analysis is different, and will be described in Section 8.3, on page 196. This comparison is relativistic, *i.e.* it is to see which activity is best.

This comparison had two stages:-

1. Experiments were ranked using a 1 sided T-Test at the 95% Confidence level. The best result was ranked 1, the worst 5. Where there was no statistically significant difference, equal ranks were assigned. The resulting ordering provided an objective assessment of the relative performance of each activity.

2. The rankings produced during the first assessment stage do not allow assessment against either the ideal performance, or the magnitude of the differences between the activities. Assessment of these properties was undertaken using the statistics calculated for the Euclidean and Mahalanobis distances of each experiment.

The results of each assessment are presented as required during the review of each experiment, *e.g.* Table 5.11, page 114.

4.5 Summary of Chapter

In this chapter the reaching problem that will be the subject of the experiments with learning mechanisms that are be presented in the next four chapters has been described. This chapter has also covered the equipment with which this problem must be handled. Together they will be a good test of the value of learning because the problem contains all the problems of uncertainty and variation and difficulty of control that are amplified by the mechatronics of the system that will be used. This is because to solve the reaching problem it is necessary to determine, and then maintain, the coordinate transform that converts a location in the coordinate frame of a sensor to the coordinate

frame of the controller of the robot. Maintenance is required because of the variation over time of the relationship of the robot to the two sensors.

In the next chapter a mechanism called QUEASY is presented that is an example of using weak learning to first determine, and then maintain, the required transform to enable the arm to reach selected target points.

Chapter 5

Learning the Model Parameterisation

This is the first of four chapters where the experimental verification of the ideas that have so far only been described are put into practice. In Chapter 3, on page 45 there was a list of three robotics applications where learning could be profitably applied, the first of which is to use learning to discover the control parameters of an activity. In the last chapter the reaching problem, and the experimental system used to assess the solutions proposed in this thesis, were described.

In this chapter a particular, novel, model based approach to reaching is developed, and from this a scheme is proposed, and developed, for learning the control parameters. Two versions of this scheme have been implemented and tested, one using the vision system (Section 4.2.3, page 76) and one using the force table (Section 4.2.4, page 79).

These reaching activities will serve as a baseline against which the other reaching models proposed in the next two chapters that use learning in different ways can be compared. The use of learning here is a typical example of how weak learning has been used until now in manipulator robotics, *e.g.* [Hsia 86], or [Aboaf 88]. Before going on, it is worth noting that Aboaf has proposed an alternative form of learning applied to models [Aboaf 88], where the task parameters of the model are adjusted in the same way that a person throwing a ball adjusts their aiming point. Such learning, as Aboaf demonstrated, can be very effective. However, it only applicable in situations where the task specification can separated from the outcome of the task, and where the specification can be altered to produce different outcomes.

5.1 QUEASY: A Example of a Robot Activity

The availability of a fast and simple visual servoing routine that effectively guides a gripper (or other tool) mounted on a robot arm to a given target provides the system with a more general reaching capability than could be achieved through only pre-taught position information. The module will be centred on a novel algorithm that has been called QUEASY[1], that performs visual servoing without calibration of the visual system or the mapping between robot motion and image motion. Discussion of this algorithm takes up the first part of this section, and this is followed by test results.

5.1.1 Previous Work

The starting point for this work has been the experiments of Chongstitvatana and Conkie [Chongstitvatana & Conkie 90b, Chongstitvatana 92]. They report a number of experiments during which increasingly sophisticated calibration free visual tracking systems were developed. From the initial system that used a single fixed position camera to control the servoing Chongstitvatana and Conkie's final system used a stereo pair carried in a second robot to first track, and then servo the arm being used for assembly. In this final system there was no calibration between the separate components, and no explicit communication between the assembly arm and the camera carrying arm. At the core of all the experiments was the same idea: by establishing a local mapping between the arm and the visual system in terms of joint motions of the arm and motions of a tracked point in the image space, then it would be possible to derive the motions of the arm needed to servo the arm onto a target point in the image space. The mapping was constructed by the visual system observing the motions of the tracked point on the arm resulting from a sequence of stereotyped motions of the gripper; these motions, together with the motions of the arm in joint space were used to derive the mapping (See [Chongstitvatana 92] for details). The mapping, expressed as a matrix, was then used to convert a desired image vector into a motion vector for the arm in terms of the joint space of the arm.

As this mapping was linear and local then it was necessary to limit the actual motion

[1] Quick and easy servoing

of the arm to a certain percentage of the calculated motion. Whenever the arm crossed a singularity, or moved too far away from the initial point then accumulated errors in the mapping would cause serious position errors in the location of the arm relative to the desired location. At this point the mapping had to be re-established.

When a single camera was used the problem of parallax was dealt with by specifying in advance the heights of the gripper and the target point, in terms of the coordinate frame of the arm.

Once a stereo-pair of cameras had been introduced, this information no longer needed to be specified in advance. However, rather than fuse the output of the two cameras to obtain a depth map, a separate mapping was established between each camera and the arm, with the calculated motion vectors for the arm being fused to give a single response. The only assumption required to use the stereo pair was that both the gripper and the target point be visible in both cameras.

This idea of using information extracted from images of the robot to guide its motion has been developed in a number of different directions, which can be classified into one of two groups depending on whether the research has required that the coordinate transform from image to robot be established before or during the operation of the servoing activity. In both cases work has been done using single, or multiple cameras, *e.g.* [Coulon & Nougaret 83] or [Ohmori 86]. In the systems where the transformation was established before operation, *e.g.* [Yin *et al.* 84], then even small changes in the physical relationship between sensor and robot could render that transformation useless [*Ibid*]. Such a fact holds whether the transformation was established manually or automatically. Systems that produced the transformation during the servoing activity, of which Chongstitvatana and Conkie's is one example, vary in the sophistication of the model of the transformation used from the very simple, as they used and as is described in the next section, to the complex, *e.g.* [Rizzi & Koditschek 94]. As the model becomes less accurate then the role of feedback increases, as a poor estimate of the transform is applied over and over to gradually correct the difference between the robot's position and the target.

5.1.2 Theory

The details of this algorithm are best explained by first returning briefly to Chongstitvatana and Conkie's work. They used a local mapping derived from observing the motions of the gripper in the visual system to calculate the appropriate arm motion vector necessary to move the arm to the desired point, as seen by the visual system. This motion then had to be scaled down, and the process iterated because it was not possible to guarantee in the general case that mapping, and hence the motion, would be valid for all of the desired travel.

One of the problems with using local mapping is that the joint space of the robot does not correspond in a linear fashion to the cartesian space typically used for position specification and visualisation of arm location. Another problem is in the spherical distortion inherent in using a camera which results in the image space being of variable resolution, with the highest resolution being at the centre of the image. This problem is made worse by any imperfections in the lens or the CCD mechanism, which combined with the quantisation of the image into pixels makes it extremely difficult to obtain two identical images even when scene, environment and camera are all static.

Unless these problems can be overcome without establishing and maintaining a global mapping then it will be necessary to iterate the procedure of developing the local mapping, then calculating the appropriate move. The starting point for this work is here: if you accept the idea of an iterated visual servoing approach, then instead of trying to establish an accurate mapping from which you calculate the appropriate motion vector, why not establish a simple, approximate relationship between the arm and the vision system. This can then be used to estimate the direction and distance in which to move. Now simply make a move in this direction that is expected to place you within the desired positional tolerance from the target point. After the move a check should be made to see if the desired tolerance has been obtained, and if not, repeat this process until it has.

For the case of an arm operating in a plane, with a camera mounted along a normal to the plane, pointing at the plane, then one relationship between the $\vec{r_i}$ current robot position $[x_i, y_i]^T$ and $\vec{p_i}$, the current image position $[u_i, v_i]^T$ is:

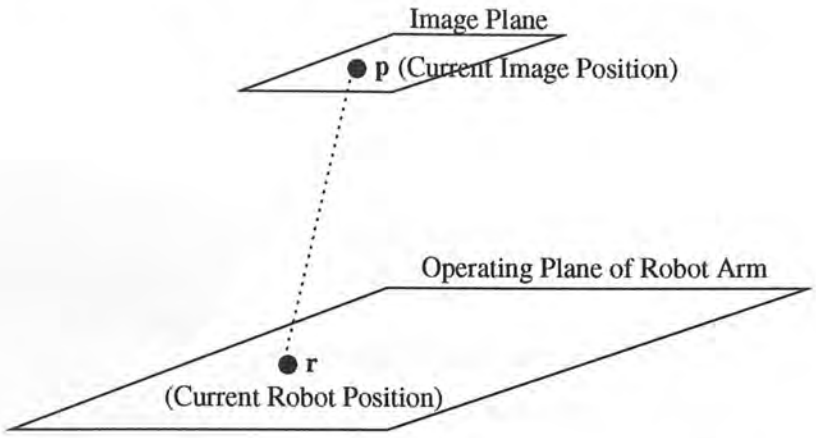

Figure 5.1: Diagram of the relationship described by Equation 5.1

$$\vec{r_i} = J_i \vec{p_i} \tag{5.1}$$

Figure 5.1 depicts this relationship graphically. In Equation 5.1 J_i is the matrix

$$\begin{bmatrix} p_i & q_i \\ r_i & s_i \end{bmatrix}_i . \tag{5.2}$$

These four variables hold the rotational and scaling factors between $\vec{r_i}$ and $\vec{p_i}$.

Equation 5.1 can be expanded into:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} p_i & q_i \\ r_i & s_i \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{5.3}$$

We want to know a value for J_i, so taking two successive image and robot displacements ($\vec{r_i}, \vec{r_{i-1}}, \vec{p_i}$, and $\vec{p_{i-1}}$), and extracting and combining the X rows gives us:

$$\begin{bmatrix} x_{i-1} \\ x_i \end{bmatrix} = \underbrace{\begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} p_i \\ q_i \end{bmatrix} \tag{5.4}$$

and similarly for the y rows:

$$\begin{bmatrix} y_{i-1} \\ y_i \end{bmatrix} = \overbrace{\begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}}^{\mathbf{A}} \begin{bmatrix} r_i \\ s_i \end{bmatrix} \tag{5.5}$$

As long as **A** is non-singular then these two equations can then be solved separately and the results combined to give an expression for J_i in terms of the components of $\vec{r}$ and $\vec{p}$:-

$$\begin{bmatrix} p_i & q_i \\ r_i & s_i \end{bmatrix} = \begin{bmatrix} \frac{v_{i-1}x_i - v_i x_{i-1}}{u_i v_{i-1} - u_{i-1} v_i} & \frac{u_{i-1}x_i - u_i x_{i-1}}{u_{i-1} v_i - u_i v_{i-1}} \\ \frac{v_{i-1}y_i - v_i y_{i-1}}{u_i v_{i-1} - u_{i-1} v_i} & \frac{u_{i-1}y_i - u_i y_{i-1}}{u_{i-1} v_i - u_i v_{i-1}} \end{bmatrix} \tag{5.6}$$

Calibration will require the robot making two separate moves, with the image displacement recorded for each move. If the initial two robot moves are orthogonal then **A** will be non-singular. Early tests showed that these moves should produce an image displacement larger than 10 pixels to avoid the displacement being obscured by errors in estimating the hand location and to provide results that were not significantly affected by rounding errors (±1mm). Robot displacements of 2cm in the robot's operating plane were found to be the minimum that meet these criteria as they resulted in displacements of around 20 pixels in the image. This initial calibration produces a local mapping J_i.

This local mapping, stored in J_i, should contain the necessary rotational transformation and a scaling factor to allow displacements in the image to be converted to displacements of the robot. Neither the scaling factor or the rotational transformation are likely to be exact, so there will be some error. However, they should be exact enough to allow the arm to reach the target through feedback.

J is easily calculated from the two previous robot and image displacements. By maintaining a short history of these displacements it would be possible to recalculate J after every move, as long as matrix **A** were non-singular, in which case the last valid J could be used instead. This variant will be referred to as ALWAYS_QUEASY.

During early use of this variant it was noticed that occasionally the robot would make a series of inappropriate moves, even though J was still valid. Monitoring of the calculations revealed that this situation occurred when the robot was near the target point and had made a succession of small moves, or moves along an almost straight line, resulting in either the scaling information and/or the directional information in J becoming incorrect as **A** became near singular. New versions of J only occur whenever **A** is non-singular. The criteria for rejecting an update can be extended to cover the

situations that produce a near singular **A**. J is a linear transform which rotates and scales the image motion to produce the appropriate robot motion vector, so J can be decomposed into:

$$\begin{bmatrix} p & q \\ r & s \end{bmatrix} = scale_x scale_y rot_z = \begin{bmatrix} k_x \cos\theta & -k_x \sin\theta \\ k_y \sin\theta & k_y \cos\theta \end{bmatrix} \tag{5.7}$$

where

$$scale_x = \begin{bmatrix} k_x & 0 \\ 0 & 1 \end{bmatrix} \tag{5.8}$$

and

$$scale_y = \begin{bmatrix} 1 & 0 \\ 0 & k_y \end{bmatrix} \tag{5.9}$$

which will scale the image vector by k_x in the x direction and k_y in the y direction, and

$$rot_z = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.10}$$

which will rotate the image vector by θ around the z-axis in an anti-clockwise direction. Appropriate manipulation of Equation 5.7 reveals that

$$-\frac{p}{q} = \frac{s}{r} = \cot\theta \tag{5.11}$$

Equation 5.11 allows the assessment of the quality of J_i. Only if $-\frac{p}{q}$ is equal to $\frac{s}{r}$ does J_i provides a high quality linear transform. If, as is normally the case, $-\frac{p}{q}$ differs from $\frac{s}{r}$ then J will not be as accurate. The closer the two values are to each other, the better J. This third variant of the algorithm, CAUTIOUS_QUEASY, calculates the absolute difference between $-\frac{p}{q}$ and $\frac{s}{r}$. If this value is smaller than the value of the difference for the current J, then J is replaced with the new version. Returning to the first variant, which we will refer to as EASY_QUEASY, it is worth noting that as long as the initial calibration is good enough, *i.e.* establishes an approximate directional

Algorithm 3 QUEASY

```
get_rlocation(i_rx, i_ry)
rx := i_rx
ry := i_ry
get_slocation(i_sx, i_sy)
vx := i_sx
vy := i_sy
do_initial_calibration(rx, ry)
itcount := 1
while (abs(vx − target_x) > TOL ∧ abs(vy − target_y) > TOL) ∧ itcount ≤ ITMAX
do
  calc_next_move(rx, ry, vx, vy, δx, δy)
  abs_move(rx + δx, ry + δ)
  get_slocation(vx, vy)
  get_rlocation(rx, ry)
  update_calibration(rx, ry, vx, vy)
  itcount := itcount + 1
end while
```

and scaling relationship, then we would expect the iterative nature of the process will absorb the errors induced by moving away in space and time from the initial calibration.

All of the QUEASY variants were developed under an assumption that the target would be stationary, but it soon became clear that this was not necessary. If the target is mobile, then the algorithm will simply calculate the motion necessary to move the robot to the target's last known position. If instead of following the target, the robot was commanded to move to the predicted next, or future position of the target, then it would be possible to either follow the target more closely, or to be waiting for the target object. Such skills would be dependent on the predictive abilities of the tracker, which is external to the algorithm discussed here.

5.1.3 Algorithm

The algorithm at the heart of these three variants is described in Algorithm 3. This is an iterative algorithm that attempts to command the robot to reach a particular location. The reaching is considered to be successful if the arm is determined to be within the specified tolerance within the number of attempts allowed. Otherwise, it is considered to have failed.

Within this algorithm, the various functions and variables and their roles are:-

get_slocation(x, y): Returns the current position of the gripper in the sensor coordinate frame.

get_rlocation(x, y): Returns the current position of the gripper in the robot coordinate frame.

abs_move(x, y): Moves the robot to the specified x and y robot coordinates.

do_initial_calibration: Implements the initial calibration routine, described in Section 5.1.2 on page 99.

calc_next_move: Calculates the next move using the current calibration J to convert the target location to the target robot location.

update_calibration: This function is present in ALWAYS_QUEASY and CAUTIOUS_QUEASY. Following the move, it is called to update the calibration, using the Equation 5.6 In CAUTIOUS_QUEASY this function is modified to use Equation 5.11 to ensure that updates only occur when they are of value.

i_rx, i_ry, rx, ry: Variables used to store the initial and current robot positions.

target_x, *target_y*: The coordinates in the sensor frame of the target location.

δx, δy: The calculated displacements estimated to move the gripper to the target from its current position.

TOL: The target tolerance for the reaching move.

$ITMAX$: The maximum number of attempts that are allowed.

Before presenting the experimental results and comparison of the QUEASY variants this section describes how parallax errors have been handled, and by describing the two central assumptions around which these variants have been constructed.

5.1.4 Parallax

Parallax is intrinsic to a single camera system. If the target point is a feature of a part to be grasped, then simply visually servoing to that point will result in the arm not

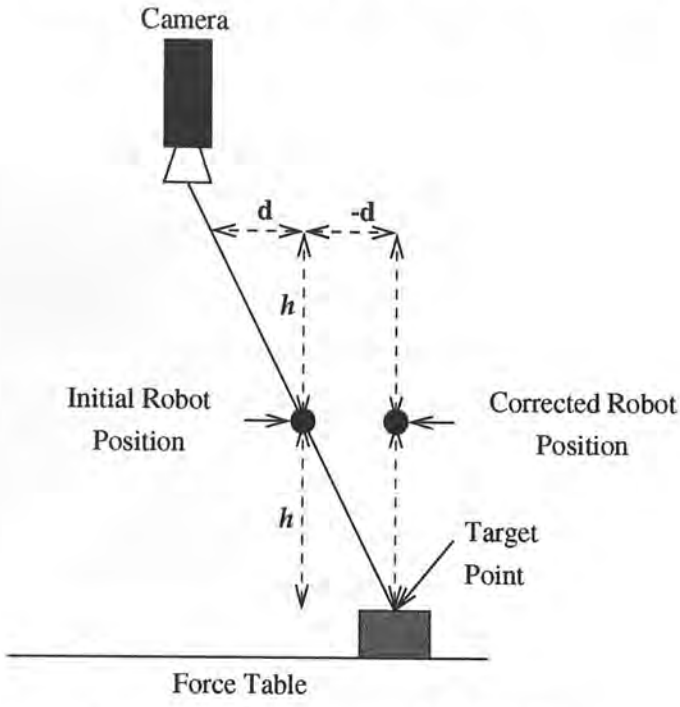

Figure 5.2: An example of 1d Parallax

being vertically above that feature. Instead it will be collinear with that point and the centre of the camera. There are two ways that this parallax error can be overcome:-

- Using knowledge about the relative heights of the robot and the target feature
- Using Multiple Cameras

Both of these approaches are commonly used, though both have problems.

The limitations of pre-supplied height information are obvious. Not only is it necessary to alter this information whenever the task changes, but it assumes that the initial calibration of the arm will be correct and that the calibration will not significantly drift during operation. Whilst the first part of this assumption is generally true, the second part is not. Simply operating the robot causes temperature changes that can lead to position errors, these errors becoming increasingly significant as the cost of the arm decreases.

Multiple cameras are the best way to overcome parallax; by mounting two (or more) cameras so their images overlap multiple lines of sight into the scene are provided.

Under a Classical, or Marr type approach, the images from the separate cameras are fused to provide explicit depth information that can be used to reconstruct a 3d model of the imaged scene. This model is then used to perform servoing. Chongstitvatana and Conkie have managed to avoid the need for the complexity of fusion by developing a joint-image mapping for each camera, as has been described earlier. Although this reduces the number of assumptions that must be made, and maintained in the physical world, it still requires multiple cameras, and the resources necessary to process the images generated by those cameras. These resources are well within the capability of modern processing systems.

The first approach to handling parallax has been used here as it is able to provide the necessary information for the problem that is being tackled, as it is possible to take advantage of operating the robot at a fixed height to overcome the approach's limitations. In the context of Figure 5.2, the height of the grippers above the target point h is known. The procedure used is this: the gripper is moved vertically up a distance h, and then located in the image. The difference between the new position and the old position is a vector $\mathbf{d}$. The gripper is returned to the original position and then displaced in the x-y plane by $-\mathbf{d}$. This will approximately locate the gripper directly above the target point, the point labelled in the diagram as the corrected robot position. The corrected position will not be perfectly above the target, partly because of inaccuracies in the robot motions, and partly because the spherical distortion due to the camera lens, as well as noise in the camera, means that $\mathbf{d}$ cannot be determined accurately. However, the resulting error is low, $\approx$ 2mm in the x-y plane of the robot's coordinate frame.

5.1.5 Assumptions

Underlying the variants of QUEASY are two assumptions about the nature of the task and the environment in which it will be applied:-

- At any one time the gripper and the target are identifiable in the same image.
- The robot is operating in some plane, with the camera mounted approximately on a normal of that plane.

The first assumption can be removed by giving the system a mobile camera, the ability to move between pointing at the gripper and at the target and the ability to set way points as it changes its viewpoint. These way points would be the intermediate targets that the arm could be moved to in sequence to arrive at a situation where both gripper and target were simultaneously visible.

The second assumption is a property of the environment in which much assembly occurs, namely a flat, horizontal work surface across which the arm has to move to acquire parts or to perform mating operations on objects that are positioned on that surface. As long as the arm is able to rise clear of any obstruction, then it can move freely in a plane. As long as the arm is visible, then even if it has to change heights during a move QUEASY will work, though it would be necessary to re-calibrate following any change in height.

The continuous recalculation of the J_i in CAUTIOUS_QUEASY introduces a great deal of robustness with respect to the angle between the normal to the plane and the camera. As the angle increases the difference between the image plane and the operating plane increases, but because the algorithm is a feedback mechanism that acts to reduce the distance between the current and target positions, and does so using a continuously updated estimate of the rotation at and scaling between the image and the robot, the gripper will eventually reach the target. Failure will occur when the angle becomes such that it is no longer possible for the gripper to be commanded to occlude the target point in the imaging system, or when the arm is unable to make the motions necessary to establish the information to correct for parallax.

5.1.6 Experiments

Each variant, with the label *Visual* prefixed to the variant name to denote the use of the vision system, has been validated through a single experiment. All the experiments have followed the method described in Section 4.3 (page 84). The intention of this validation was to show that the servoing works, *i.e.* the algorithm could command the robot to move so that its gripper would end up in a certain position in space as identified by the vision system.

Tolerance (pixels)	Position Error			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	11.45	43.34	1	0.57	0.30	1	3	3.60	0.75	1
4	7.82	41.46	1	0.51	0.25	2	1	3.78	0.84	1
3	9.81	39.98	1	0.54	0.27	2	3	3.84	0.99	1
2	6.10	32.82	1	0.52	0.29	2	2	3.98	0.74	4
1	6.80	41.97	1	0.46	0.25	2	1	4.26	0.94	5

Table 5.1: Summary results of Visual EASY_QUEASY batch testing

Tolerance (pixels)	Position Error			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.90	2.27	4	0.51	0.27	1	1	3.80	1.69	1
4	3.82	8.54	4	0.52	0.25	2	2	3.84	1.59	1
3	2.57	4.22	1	0.48	0.24	3	2	4.16	1.99	1
2	3.87	14.67	1	0.43	0.23	4	2	4.32	1.50	1
1	1.37	4.05	1	0.42	0.19	5	1	4.42	1.19	1

Table 5.2: Summary results of Visual ALWAYS_QUEASY batch testing

Tolerance (pixels)	Position Error			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.34	1.61	5	0.37	0.21	1	0	3.78	1.38	1
4	1.91	1.19	3	0.36	0.19	1	0	3.80	0.85	1
3	1.81	1.04	3	0.35	0.18	1	0	3.86	0.88	1
2	1.38	0.72	2	0.35	0.21	1	0	3.80	0.63	1
1	0.67	0.60	1	0.31	0.18	1	0	4.34	1.04	5

Table 5.3: Summary results of Visual CAUTIOUS_QUEASY batch testing

The summary results of the experiments with the three variants that are presented in Table 5.1, Table 5.2 and Table 5.3 are what might be expected given the different nature of the three variants. In all cases the final position error and the efficiency of the variant decrease and the number of steps increases with the tolerance, though these changes are not always significant. Of the three variants CAUTIOUS_QUEASY has on balance the best accuracy, though it is the least efficient and requires the most steps. This result alone is justification for having continued the development of EASY_QUEASY into CAUTIOUS_QUEASY.

However, the data from these experiments can be analysed to provided more information than these summary results In the rest of this section an analysis is presented that identifies the weaknesses of the underlying model and the learning mechanisms. This

analysis is split into five parts:-

- Negative Results – what happened on failed trails.
- Positive Results – what happened on successful trials.
- Movement Traces – what happened during a trial.
- Analysis of Recorded Times – how long trials lasted.
- Which is best? – comparing the behaviour.

Negative Results

It is worthwhile to split the results up into those which failed and those which succeeded, as this provides as better picture of the variants. Looking at the negative results, shown in Table 5.4 and Table 5.5 (CAUTIOUS_QUEASY never failed during these experiments), first, it would appear that whenever a variant of QUEASY fails, there is no correlation between the degree of failure and the tolerance at which it is working.

However, when ALWAYS_QUEASY fails, it usually does so closer to the target position than EASY_QUEASY. The explanation for this difference stems from the nature of the target points on which the variants failed. If the target point was at, or very close to a singularity in the control space of the robot, then the initial calibration used by EASY_QUEASY becomes highly incorrect, and will not be able to parameterise the model of the mapping from image to gripper in the right way for that area of the image and robot workspace, hence the large errors. ALWAYS_QUEASY is able to change the parameterisation to better approximate the mapping, but fails when that parameterisation makes the model singular. Once this situation arises ALWAYS_QUEASY is trapped as the subsequent motions determined from the model do not, unless by chance, allow the appropriate parameterisation to be devised. This behaviour is a consequence of how the model, and the learning mechanism in ALWAYS_QUEASY work.

The indices of the failed trials that occurred during the testing of EASY_QUEASY and ALWAYS_QUEASY are shown in Table 5.6 for the different tolerances. Whilst there is some similarity for experiments using the same variant at different tolerances, there

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	151.70	120.71	1	0.25	0.01	4	3	4.66	1.15	2
4	295.74	0.00	4	0.13	0.00	2	1	6.00	0.00	4
3	138.94	111.97	1	0.45	0.31	1	3	4.00	1.00	1
2	123.63	154.84	1	0.25	0.10	4	2	5.00	0.00	3
1	297.63	0.00	5	0.13	0.00	2	1	6.00	0.00	4

Table 5.4: Negative results of Visual EASY_QUEASY batch testing

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	14.56	0.00	1	0.10	0.00	5	1	12.00	0.00	4
4	38.32	31.40	4	0.31	0.07	1	2	6.00	0.00	4
3	20.83	11.02	2	0.12	0.01	4	2	12.00	0.00	1
2	63.64	55.59	5	0.23	0.18	2	2	8.00	5.65	3
1	29.20	0.00	3	0.25	0.00	2	1	6.00	0.00	1

Table 5.5: Negative of Visual ALWAYS_QUEASY batch testing

Tolerance (pixels)	Failed Trials	
	EASY_QUEASY	ALWAYS_QUEASY
5	3, 21, 24	2
4	26	2, 9
3	3, 4, 24	2, 24
2	21, 24	2, 3
1	26	24

Table 5.6: Indices of failed Visual EASY_QUEASY and ALWAYS_QUEASY trials.

is almost no correspondence between the two variants. The pattern that exists is that failures of EASY_QUEASY usually occurred for trials where the target point required the arm to be moved around a singularity in joint space, while the failures of ALWAYS_QUEASY occurred whenever the updates to the calibration matrix resulted in it becoming near singular, a situation from which it became unable to recover. Given this pattern and the fact that CAUTIOUS_QUEASY restricts updates to the calibration matrix only if they are an improvement, it is not surprising that CAUTIOUS_QUEASY did not fail on this test set, as it is designed to not be affected by either of these two problems.

Positive Results

The separation of the positive results, shown in Table 5.7, Table 5.8 and Table 5.9 from the summary results shows how much the poor results of failing dragged down the EASY_QUEASY and ALWAYS_QUEASY variants. The final position error and number of iterations for all three variants is about the same for positive results, though the efficiency of CAUTIOUS_QUEASY is far lower than the other two variants. The explanation for this difference can be found in how CAUTIOUS_QUEASY recovers from the problem that causes ALWAYS_QUEASY to fail. When the gripper is approaching a target point which is at, or close to, a singularity in the control space of the robot, any updates to the parameterisation made by the learning mechanism can lead to the matrix **A** in Equation 5.6 on page 99 to become singular if those updates are accepted without question, as done in ALWAYS_QUEASY. As the matrix becomes singular the moves which are commanded become smaller and smaller, producing the situation of commanded moves not resulting in any detectable motion. When this happens the overall efficiency of the move is high. The selective updating of CAUTIOUS_QUEASY prevents singularities, but can lead to less efficient moves being calculated. Despite this poorer efficiency, its accuracy is equal to that of ALWAYS_QUEASY, while its success rate is higher.

Analysis of Movement Traces

Here, a review is presented of the movement traces produced by the variants on two particular trials. Figure 5.3, Figure 5.4 and Figure 5.5 show what happened on the 17th trial of the three variants, whilst Figure 5.6, Figure 5.7 and Figure 5.8 show the traces of the 21st trial, all for a tolerance of 2. The things that stand out from these traces are that:-

- The first three calibration moves are different for each trial, even though the robot is commanded in exactly the same way each time.
- The initial guess (move 4) puts the robot within 50 pixels (or $\approx 50mm$ in the robot's control frame) of the target, but after that either it is able to make a single move to reach the target, or 6 further moves are required.

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.50	1.57	5	0.59	0.29	1	47	3.53	0.68	1
4	1.94	1.31	3	0.52	0.25	2	49	3.73	0.78	1
3	1.57	0.97	3	0.54	0.27	2	47	3.82	1.00	3
2	1.20	0.76	2	0.53	0.29	2	48	3.92	0.72	3
1	0.86	0.55	1	0.46	0.24	2	49	4.22	0.91	5

Table 5.7: Positive results of Visual EASY_QUEASY batch testing

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.66	1.54	4	0.52	0.27	1	49	3.63	1.21	1
4	2.38	1.51	4	0.52	0.25	2	48	3.75	1.56	1
3	1.81	1.10	3	0.49	0.23	3	48	3.83	1.19	1
2	1.38	0.70	2	0.44	0.23	3	48	4.16	1.03	4
1	0.80	0.54	1	0.43	0.19	3	49	4.38	1.18	5

Table 5.8: Positive results of Visual ALWAYS_QUEASY batch testing

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.34	1.61	5	0.37	0.21	1	50	3.78	1.38	1
4	1.91	1.19	3	0.36	0.19	1	50	3.80	0.85	1
3	1.81	1.04	3	0.35	0.18	1	50	3.86	0.88	1
2	1.38	0.72	2	0.35	0.21	1	50	3.80	0.63	1
1	0.67	0.60	1	0.31	0.18	1	50	4.34	1.04	1

Table 5.9: Positive results of Visual CAUTIOUS_QUEASY batch testing

Tolerance (pixels)	Time					
	EASY_QUEASY		ALWAYS_QUEASY		CAUTIOUS_QUEASY	
	μ_t	σ_t	μ_t	σ_t	μ_t	σ_t
5	174.38	105.71	194.72	179.18	206.85	228.55
4	186.88	140.49	199.05	174.35	175.06	88.56
3	178.79	94.15	218.75	206.21	183.74	108.11
2	191.86	120.55	219.38	167.83	186.86	90.66
1	202.29	128.84	211.74	127.89	228.95	163.36

Table 5.10: Mean (μ_t) and standard deviation (σ_t) of the times taken (in seconds) by Visual QUEASY variants

In the latter case all of the subsequent moves were centred around the target. These two trials are unusual in that they have almost similar start and target points. The very different behaviour of the variants on the separate trials is explained by one target point (trial 21) being a singular position of the robot arm. This is a situation, that has been noted above, requires large motions of the robot to achieve small gripper motions. motions which produce actual position differences that are undetectable by the robot controller. The fact that in all cases, except for ALWAYS_QUEASY the arm was able to be positioned on target using the visual reaching mechanism is a demonstration of the mechanism's robustness in the face of uncertainty.

Analysis of Recorded Times

The final statistic to be reported from the experiments is the time taken by the variants at the different tolerances. These values are tabulated in Table 5.10. Given the other performance results, this table is not surprising, as the time taken by a variant will be a function of the number of motions commanded by that variant at a particular tolerance. Of all the results, this one is most determined by the experimental hardware rather than problem, as faster image processing and communications time would reduce these values.

Which is best?

Three variants of QUEASY have been described, each of which appears to produce a different performance when applied to the same test problem. These variants have been compared using the procedure described in Section 4.4.2, page 92, with the results of

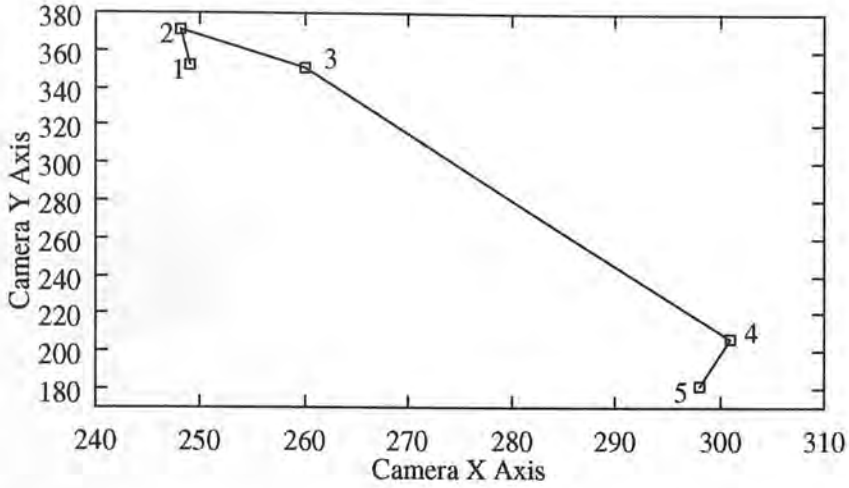

Figure 5.3: Movement Trace of Visual EASY_QUEASY on Trial 17

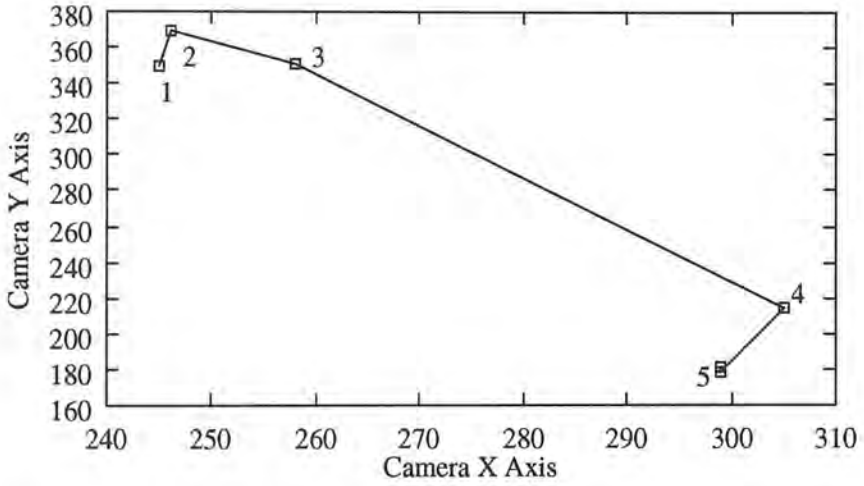

Figure 5.4: Movement Trace of Visual ALWAYS_QUEASY on Trial 17

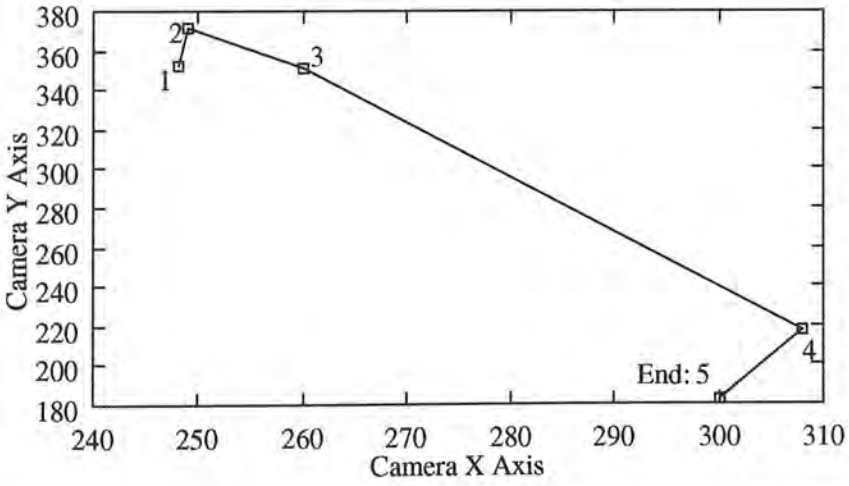

Figure 5.5: Movement Trace of Visual CAUTIOUS_QUEASY on Trial 17

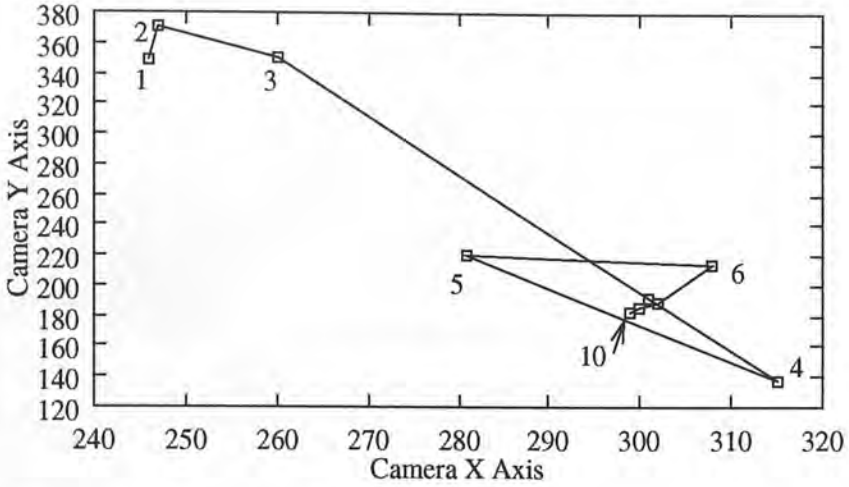

Figure 5.6: Movement Trace of Visual EASY_QUEASY on Trial 21

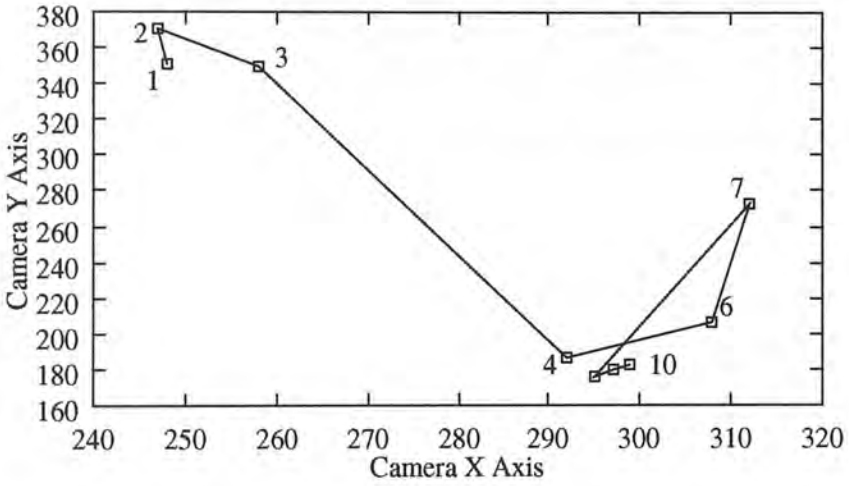

Figure 5.7: Movement Trace of Visual ALWAYS_QUEASY on Trial 21

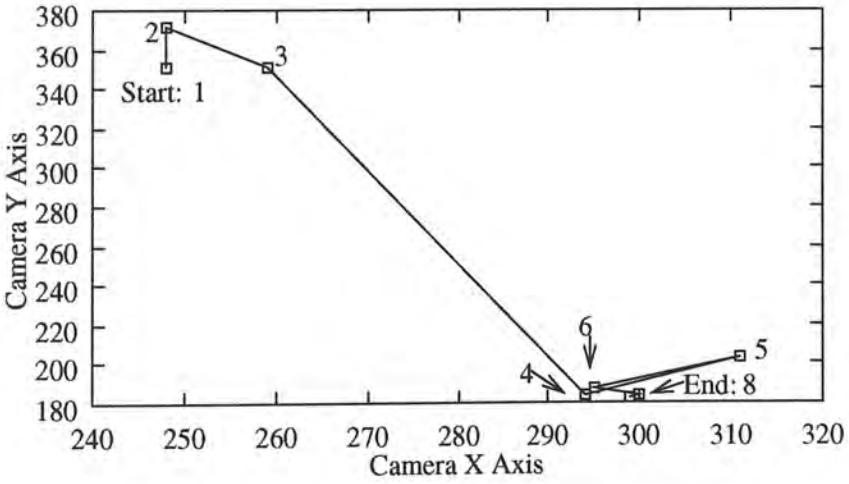

Figure 5.8: Movement Trace of Visual CAUTIOUS_QUEASY on Trial 21

Tolerance (pixels)	Euclidean Distance					
	EASY_QUEASY		ALWAYS_QUEASY		CAUTIOUS_QUEASY	
	μ_E	σ_E	μ_E	σ_E	μ_E	σ_E
5	176.59	111.38	194.78	179.18	207.51	230.68
4	188.19	145.04	199.21	174.44	175.49	89.92
3	181.18	98.47	218.80	206.25	184.36	108.85
2	192.86	123.54	219.81	168.01	187.09	90.68
1	204.21	133.76	211.91	127.92	229.13	163.83

Table 5.11: Mean Euclidean distances between actual and ideal outcomes

Tolerance (pixels)	Mahalanobis Distance					
	EASY_QUEASY		ALWAYS_QUEASY		CAUTIOUS_QUEASY	
	μ_M	σ_M	μ_M	σ_M	μ_M	σ_M
5	34.02	7.68	29.11	7.87	28.33	8.42
4	36.49	7.86	24.69	8.91	32.35	8.25
3	25.15	7.90	26.22	8.47	25.87	8.41
2	36.03	7.95	27.62	8.71	38.93	8.41
1	32.20	8.35	32.24	8.67	42.30	7.59

Table 5.12: Mean Mahalanobis distances between actual and ideal outcomes

the comparisons shown in Table 5.11 and Table 5.12.

Using these distances, it is possible to see that using any of the variants with a tolerance of 3 will, on average produce the nearest to ideal performance. The smaller distances of the EASY_QUEASY values compared to the other two variants using both metrics is a result of EASY_QUEASY having a higher, more constant efficiency and requiring fewer motions to complete the reach. The higher accuracy and success rate of the other variants does not offset their poorer efficiency and need to make more moves.

Although EASY_QUEASY has the best overall performance, CAUTIOUS_QUEASY, with the best accuracy and success rates in this experiment, will alone be tested in the experiments described in the next section where a force table is used instead of the camera to localise the robot. In a real application the relatively poor accuracy of EASY_QUEASY would result in a higher number of failures, which for present robotic systems which have almost no ability to recover from errors means that more outside intervention would be required than for CAUTIOUS_QUEASY, which would be slower, but more accurate.

5.2 Generalising QUEASY

QUEASY is not dependent on intensity based imagery. As long as both arm and target can be localised in some common measurement space, and motion trajectories in that space can be converted into real trajectories for the agent, then this algorithm is applicable. As an example of this CAUTIOUS_QUEASY has been used in the implementation of a servoing algorithm using the force table described in Section 4.2.4 on page 79.

Exactly the same algorithm used for the visual servoing has been applied here. The resulting activity is different, as the task is now one of reaching to a point on the surface of the tray, with a corresponding change in the method of localising the grippers is used. This localisation is the tapping process described in Section 4.2.4, page 82. Unlike the vision localisation, this process provides poor estimates of the gripper's position, which coupled with the equally poor estimates of the location of the target object on the tray mean that it is unlikely the performance of reaching using the tray will be equal to that of visual reaching. For this activity the assumptions about being able to detect the target and the grippers simultaneously, and that the 'imaging' mechanism is along the normal of the plane in which the robot operates, are guaranteed by virtue of the nature of the Force Tray.

5.2.1 Experiments

The summary, negative, and positive results of the activity using an experimental regime identical to that used for visual reaching are respectively displayed in Table 5.13, Table 5.14 and Table 5.15. In Table 5.16 are listed the numbers of trials on which failures occurred. Figure 5.9 and Figure 5.10 show the best and the worst performance of the servoing during a reach from one set position to another. Like the data used to generate the sequences shown in the corresponding figures for the QUEASY variants, these results show a situation in which sometimes the activity will succeed and sometimes will fail because the final position is one the robot has difficulty in moving to.

The higher number of failures seen with Force CAUTIOUS_QUEASY compared to the perfect success rate of Visual CAUTIOUS_QUEASY are entirely due to the variable es-

Tolerance (mm)	Position Error			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	3.33	1.71	1	0.39	0.16	1	0	5.1	1.19	1
4	19.33	59.63	5	0.28	0.15	4	11	6.5	2.21	2
3	9.85	28.17	1	0.3	0.15	2	10	7.18	2.12	2
2	4.21	9.72	1	0.3	0.14	2	8	6.7	1.79	2
1	4.61	7.27	1	0.24	0.14	4	25	8.84	1.67	5

Table 5.13: Summary results of Force CAUTIOUS_QUEASY batch testing

Tolerance (mm)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	–	–	–	–	–	–	0	–	–	–
4	77.45	112.49	4	0.14	0.09	4	11	9.09	2.07	1
3	39.27	55.82	3	0.18	0.08	3	10	9.0	2.16	1
2	16.53	21.28	2	0.31	0.18	1	8	10	0	3
1	8.17	9.02	1	0.24	0.14	2	25	10	0	3

Table 5.14: Negative results of Force CAUTIOUS_QUEASY batch testing

Tolerance (mm)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	3.33	1.71	5	0.39	0.16	1	50	5.1	1.19	1
4	2.93	1.37	3	0.32	0.15	2	39	5.76	1.62	2
3	2.51	0.97	3	0.32	0.15	2	40	6.72	1.88	4
2	1.87	0.63	2	0.31	0.13	2	42	6.07	1.15	2
1	1.04	0.44	1	0.24	0.15	5	25	7.68	1.71	5

Table 5.15: Positive results of Force CAUTIOUS_QUEASY batch testing

Tolerance (mm)	Failed Trials
5	No Failures
4	2, 6, 7, 15, 24, 27, 38, 46, 47, 48, 49
3	4, 8, 15, 19, 32, 33, 38, 39, 44, 50
2	1, 3, 6, 14, 16, 27, 30, 40
1	1, 3, 4, 9, 10, 13, 15, 16, 17, 18, 19, 21, 24, 27, 32, 35, 36, 37, 39, 41, 42, 43, 45, 49, 50

Table 5.16: Indices of failed Force CAUTIOUS_QUEASY trials

Tolerance (mm)	μ_t	σ_t
5	226.21	51.72
4	371.42	167.17
3	479.10	152.24
2	336.19	107.32
1	485.75	128.31

Table 5.17: Mean (μ_t) and standard deviation (σ_t) of the times required in seconds to complete Force CAUTIOUS_QUEASY reaching

timates in the robot's position produced by the force table. The effects of this can be seen in Figure 5.10 where although the arm has been moved into the right area very quickly, it is unable to reach the target point within the desired tolerance, instead oscillating around that point. Although the overall position errors are isotropic, the pattern of movement around each failure is not, being distributed roughly along an axis parallel to one of the natural measurement axes of the sensor. This distribution is a consequence of the quantisation of the table's surface and the inability of the table to produce the same estimate for a point of contact even though the cause of that contact might not have moved, a result first produced in Section 4.2.4, page 79. Even with a Gated Kalman Filter post processing the output of a moving Average Filter, there can be considerable variation in the estimate for a point of contact, even when the gripper does not move. Whilst this variation is small, it is greater than the highest tolerance used with force CAUTIOUS_QUEASY. The impact of this variation has been greatest whenever the target point is at a singularity in the robot's control space, as the errors that are induced by the large motions of the robot required to effect small motions of the gripper are exaggerated by the errors in the localisation procedure.

From the system's point of view, whenever the robot makes a move, it moves far more than expected. In practice, the coordinate frame of the sensor in which the robot's position is located is also moving. If this last fact were not known then any attempts to improve Force CAUTIOUS_QUEASY would focus on the reaching module, and ignore the sensor processing. All of the results follow on from this source of problems, with overall performance reports being a consequence of good successes being dragged down by bad failures, all of which consisted of CAUTIOUS_QUEASY commanding more than the limited number of moves available.

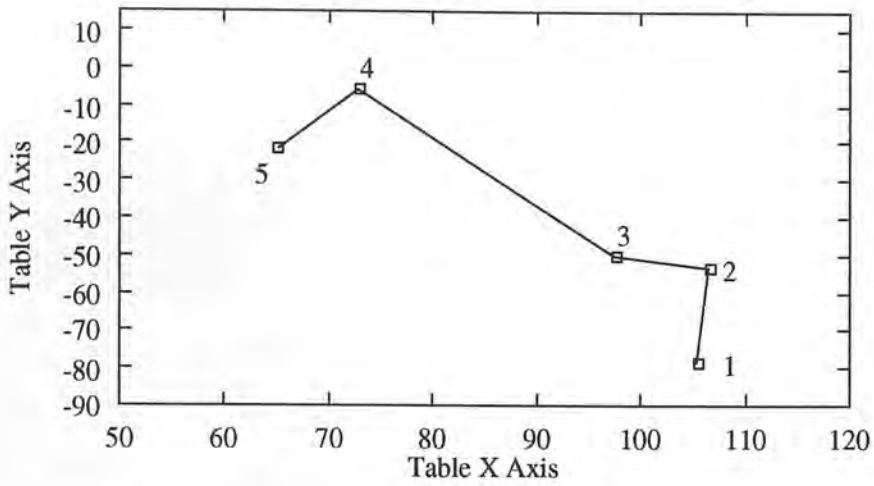

Figure 5.9: Movement Trace of Force CAUTIOUS_QUEASY on Trial 11

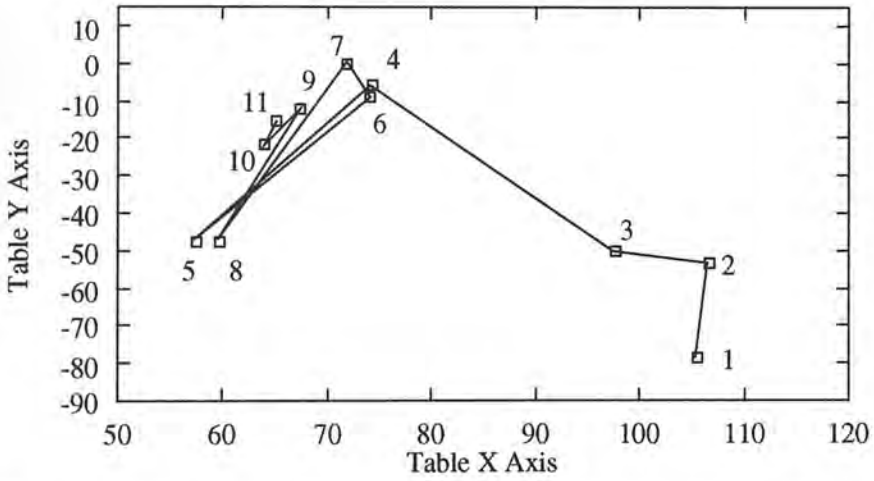

Figure 5.10: Movement Trace of Force CAUTIOUS_QUEASY on Trial 27

Tolerance (mm)	Euclidean Distance		Mahalanobis Distance	
	μ_E	σ_E	μ_M	σ_M
5	226.29	51.72	51.22	7.48
4	376.29	167.94	34.24	7.88
3	480.55	150.64	35.81	7.32
2	336.35	107.52	41.65	7.64
1	485.89	128.31	46.02	7.03

Table 5.18: Distance of Force CAUTIOUS_QUEASY performance from ideal

These results, with the greater number of errors and poorer overall performance when compared to the QUEASY results do not suggest that force QUEASY will be a useful reaching activity. Developing force QUEASY has been a useful exercise. In addition to being another demonstration of the validity of the idea behind QUEASY, it has highlighted that QUEASYis unreliable when the variation in localisation data is larger than the set tolerance, a limit that would not have been noticed if only the vision experiments had been performed.

5.3 Summary of Chapter

From the results obtained by the experimental system it can be seen that these algorithms discussed here achieve the aim of providing fast, simple, accurate servoing of the robot gripper to a point in space corresponding to a user selected point in the image space.

The generality of the QUEASY algorithms have been demonstrated by the last experiments where the force-torque sensor has been used to detect where the robot's fingers make contact with the tray. The point here is that as long as the distance between the gripper and the target point can be detected, and that measurement converted into a motion vector for the arm, then these algorithms are applicable. This generality could extend to mobile robots as well, though as it must be possible to localise both the agent and target in a common measurement space which is typically defined by an external sensor to the agent, these algorithms are not as valuable as other agent based methods. These experiments have highlighted that problems with the QUEASY variants occur when the target point is close to, or at, a singularity in the control space of the robot, problems which are exacerbated when the estimations of the position of the robot determined by the sensor is of low quality.

In this chapter a fairly typical application of learning to robotics has been worked through. Back in Section 3.1, page 44, the use of learning to find good parameterisations was listed as one of the three useful applications of weak learning in robotics. If, as has been described here, that learning has to find a good parameterisation of a model, then the details of that learning mechanism are specific to that model, and so to

the activity which is based on that model. In developing and testing three related, but different variants of a sensor guided reaching activity called QUEASY, a demonstration has been produced of how learning can be used to reduce the load on the developer and provide a more robust system through automation of the selection of the right parameters. However, to get to this stage, the developer needs to determine the model and the learning mechanism. If the activity uses an action-sensor coupling to achieve its goal then the model is a computational approximation of the relationship between the sensor and the agent that can be used to solve (or predict, depending on the model) what action will cause a specified sensor result. Learning is used to correct the model until its predictions about what an action will do are close enough to the real outcome of an action.

Whilst learning here is useful in imparting an otherwise unobtainable degree of robustness to the system, it does not make the task of the developer, or the system simpler. The kind of action-sensor coupling introduced here requires the developer to provide a model of the relationship between robot and sensor, which is rarely a simple task. From this model it is possible, as has been done here, to derive a learning method that will find and maintain the optimal parameterisation of the model with respect to some performance criteria, making this kind of learning in action-sensor couplings equivalent to learning the optimal parameter setting. The hypothesis being investigated here is that using the right kind of learning can make development and system simpler while still providing appropriate robustness. If the use of task specific models complicates situations, then they need to be discarded. The next chapter continues to look at how learning can be used to select optimal parameter settings, but this time by selecting appropriate learning methods it is possible to perform the same task (and some others) as described here, but without the model and with a better performance.

Chapter 6

Parameter Tuning

The automatic selection of good control parameter settings was identified as a useful application of learning to robotic assembly in Section 3.1 on page 45 because poor parameter settings will result in poor performance. Sometimes these control parameters can be derived from the task specification, or through analysis. However, this could be difficult, and might require assumptions to be made about the performance of that module and the state of the environment. In other cases it might only be possible to produce a model of the relationship of the control parameters to sensor values, as was done in the last chapter as part of the development of the QUEASY variants. Although such a model helps, some experimentation is still required to obtain the correct parameterisation.

If the need to undertake some experimentation is accepted, then one valid question that might be asked within the context of this thesis is whether it is possible to avoid the work required to construct the model that is at the heart of the QUEASY variants, *i.e.* to have the system develop, or learn, the model for itself. This chapter addresses that question through reviewing the issues and technologies and then describing the details and evaluation of a reaching activity called P_MOVE that is based on a direct numerical optimisation algorithm developed from Powell's method [Powell 64, Press *et al.* 90] to implement such learning.

Powell's method is an example of a large, well developed and well understood class of mechanisms that are able to find good parameters through repeated experiments. Such mechanisms are guided in their searches by the error values calculated by a profit

function from the experimental results. In doing, so the mechanisms approximate the model of the relationship between control parameters and sensor values that would otherwise have to be produced and implemented by the developer. The experiments with the P_MOVE activity in this chapter demonstrate that this approach has comparable performance to the best QUEASY performance.

The principal difference between this work and preceding work, such as that reviewed in Section 5.1.1 (page 95), or the work of Aboaf [Aboaf 88], is that here no attempt has been made to associate the state in which an activity is performed with the resulting error value for the parameter setting of the activity during the performance. Behind this non-association is the assumption that if an activity is being performed at all, then it is being done in environmental states where the chance of success, and the quality of that success are solely dependent on its parameterisation. This is an advantage because it removes the need to track and maintain records of the error value returned for performing an activity in a state with a parameterisation, which is a process that is at the heart of any model based activity. Not requiring such a process means that no task specific information need be embedded into an activity. Instead, a profit function must be constructed for each task that an activity using this kind of parameter learning is applied to. The argument put forward here is that for a new task, it is easier to construct a profit function for the task than it is to develop a model of how that task is performed. The details of how profit functions are constructed have been provided in Section 3.2 (page 51).

Before describing the activity and the experiments, the issues involved in using parameter tuning are considered before the two different methods are reviewed. After this review a short description of direct numerical optimisation is presented before a number of the alternative approaches are briefly considered.

6.1 Issues in Tuning

Why is parameter tuning difficult? It is a very common activity, but often exasperating because of the number of experiments required to find the optimum parameter settings. In robots, where repeatable experiments can be hard to arrange and each experiment

takes minutes, optimality is often sacrificed for satisfaction.

In automating the discovery of optimum parameter settings, (or the *optima* points in the parameter space [Pierre 86]), there are two types of issues that have to be handled: those caused by the relationships of the parameters to the error value, and those caused by the selected optimisation method. Whilst these issues are present in manual tuning, they are rarely handled explicitly and are usually dealt with on an ad hoc basis as they arise. The first set of issues are:-

Dependency In addition to being dependent on the environment, some of the parameters of some activities are interdependent so that behavioural modifications caused by adjusting one parameter are reversed by adjusting another parameter. For example, the number of corners detected by Malcolm's corner filter [Malcolm 83] is a function of four parameters. The relationship between these parameters is complex, with for a given object the same result being returned by different combinations of parameters. The effect of this is that locally, some parts of the parameter space are very similar to other parts of the parameter space.

Value Quantisation If the control parameters of an activity take only integer values, then the resulting error surface will be discontinuous, or stepped. This will present a series of apparent optima to many minimisation algorithms which assume a continuous error surface.

Nonlinearity of Response Ideally for a small change in parameter settings there would be a corresponding small change in the cost value. For some applications this might not be true with a small change parameters resulting in a large change in performance. In the extreme case the nonlinearity manifests itself as cliffs in the error surface. Cliffs can appear in an otherwise well behaved error surface if there are parameter settings which cause the module to fail outright. Nonlinearity is something which manual tuning can find very difficult to handle, especially when the nonlinearity increases, or the developer is unfamiliar with the system. Automatic tuning mechanisms can handle the phenomenon better, but as the nonlinearity increases, their performance degrades, especially when

discontinuities appear. One approach to handling nonlinearity is to design the profit function to remove, or reduce the problem to a manageable level. This places the burden of work back on to the developer, and in practice will require a number of experiments to find the best profit function. Error functions are hard enough to develop anyway, and there will be situations when it simply is not possible to reduce the nonlinearity to an insignificant degree. For these reasons the aim here will be to use tuning methods that do not need to be protected from the true realities of the application.

Turning now the second source of difficulty, the optimisation methods, the issues which have to be dealt with before these methods can be used are:-

Knowledge Engineering All methods require at the very least information about what parameters they are to adjust, and how to access them. This is one of the first issues that has to be addressed, and while it can be simple, it can also require a great deal of effort, *e.g.* Genetic Algorithms (GA) require parameter settings to be encoded and decoded from binary strings. If a straight forward approach is taken, assigning one section of the string to represent one component of the setting and then converting the value of that component into binary the GA will have to deal with Hamming cliffs [Goldberg 89a], where to go from a parameter setting component value 20 (binary 11001) to one of 21, (binary 10101) it will have to use a very different binary string. This problem can be overcome by using a Grey encoding from the base 10 values to the binary values. Neither GAs nor direct numerical optimisation methods require knowledge of the functional relationship between the parameters and the profit values, though direct numerical methods work better when that relationship can be approximated locally by quadratic functions. Other methods which use knowledge about the relationship need that knowledge supplied before they can begin work. Unless the relationship can be analytically derived, which it cannot for robotics (See Section 3.1, page 46 for why), then establishing that knowledge can be a time consuming and error prone activity.

Initialisation Most methods require some initialisation before beginning operation.

For some this is critical as if they begin their search in the 'wrong' place of the parameter space, or if they search in the 'wrong' direction there is a reasonable chance that they will never arrive at the correct value. Direct Numerical methods are especially prone to this, *e.g.* Powell's method requires an initial point and initial search directions [Powell 64, Press *et al.* 90]. These are usually set to be the basis vectors of the profit function.

Plateau Handling Any method used has to handle the situation of a cost function having an area where for a range of parameter settings the cost is constant. If a method relies on the cost function having some form of slope to determine the best move, then the lack of slope in such a plateau region effectively becalms that method. One option would be to engineer the cost function to remove the plateau, but this is a return to the policy of modifying the environment to the system work, with all the limitations that creates (see Section 2.2.1, page 15). Instead, any method used has to be able to escape from a plateau, or be able to recognise the situation and terminate the optimisation if it is unable to escape.

Deciding when to Update When the activity, or process, which is the subject of tuning has a natural conclusion, as for the corner filter discussed in the next section, then this question has an obvious answer: update after the process has finished. When the process does not have a clearly defined end, such as with a filter, or a homeostatic activity then resolving this topic becomes problematic. The options for when to update include:-

- After every iteration of the control loop.
- After a certain number of iterations.
- After a certain time interval.
- When particular environmental conditions occur.
- When the process moves through a particular state.

The selection of update method will affect the profit function as different measurements will be available. Sometimes the problem of which option to select can be resolved through the limits induced by the stated goal for that process. If that goal is to maintain a stable estimate of a period of time, then the natural

update interval is temporal and periodic. This problem will have to be resolved for each application where it is a factor, the exact resolution being dependent on the requirements for that application.

Termination All optimisation methods use a threshold setting to determine if they are close enough to the goal; selecting an appropriate value for this is not a problem. Most methods also include thresholds on the total number of iterations performed by the optimisation method, and additionally in some cases, the number of iterations allowed before acceptable progress has been made. Here acceptable progress is defined as the amount (either absolute, or relative to previous gains) by which the performance of the function being optimised must have improved over the set period. This last kind of termination is an important part of the plateau handling requirement described above.

In reviewing these issues there is a clear distinction between those that can be handled by the mechanism, *e.g.* plateau handling, nonlinearity and quantisation, and those that require the developer to specify control values, *e.g.* the boredom threshold, and when to update. The latter are the control parameters for the tuning process. The answers required to problems such as initialisation and sensitivity to nonlinearity will be determined by the specific mechanism used. The selected method must also comply with the requirement made here that it can operate without detailed knowledge of the task to which it is being applied. In addition to setting the control parameters, the developer must also specify the profit function which will be used to assess a module's performance under a particular parameter setting. The method used to develop the profit function has been described in Section 3.2 on page 51.

6.2 A Review of Direct Numerical Optimisation

Direct Numerical Optimisation is an approach to finding the solution to a problem that has the highest profit value. The label *direct* comes from the fact that any implementation neither performs or requires any analysis about the relationship between the parameters and the error function. This has the advantage of freeing the developer from producing and implementing pre-specified models of the activity which then have

to be analysed to determine the optimum parameter settings. However, this gain in generality has a corresponding loss in performance in situations when such models are available. In the context of this thesis optimal parameter settings means the parameter setting (or optima) that produces a minimum value (or 0) from the profit function.

Direct Numerical Optimisation is a well understood technology with a long history of development, with Gauss considered to have laid the foundations ([Pierre 86]). Over time a wide variety of methods and implementations of those methods have been developed, *e.g.* [Press *et al.* 90, Fletcher 87]. The principal features present in all of these are that:

- Explicit knowledge of the function to be optimised is not required.
- It must be possible to evaluate the function to be optimised at points in the parameter space selected by the method. Some direct numerical methods simply require a single numeric value to be returned as the result of evaluating the function at that point; others require the gradient of the function at the point as well. Whilst the latter class generally outperform the former, the gradient is not always available, especially when the function is unknown as is the case here.

Every direct numerical optimisation mechanism requires, in some form an initial starting point (and gradient at that point if relevant). Some mechanisms also require an initial direction in which to begin searching for the optima. This initial point is taken to be the initial guess of the location of the optima (or location of the optimum) point. From this starting location every mechanism then performs a similar sequence of steps:-

1. The initial estimate is assumed not to be the optima. The mechanism then estimates in what direction the true optima might lie using the locally available information, and the maximum distance along that direction to search for. As a minimum the locally available information consists of an estimate of the local gradient around the current optima. In more sophisticated methods this information can include second order gradient estimates.
2. The mechanism then conducts a search. If it finds a better estimate, it takes that location to be the new optima and starts again. If it does not find a better

estimate, then the initial estimate is reported as being the true optima and the searching ends.

Unless care is taken, the searching often fails if the error surface in the vicinity of the current point is uniform, as there is no information at this point to suggest a promising subsequent search direction, or if the mechanism gets trapped in a local optimum.

The feedback loop evident in the operation of the mechanism is similar to that operating in the QUEASY servoing algorithm, and in servo controllers in general. Such loops are a very powerful yet very simple tool which are capable of reaching a stationary minimum if one is available, or tracking a dynamic minimum. If the desired goal of an activity can be expressed as a minimum of some function, then a mechanism based around a feedback loop is likely to be a robust and effective way to reach that goal.

As has been mentioned, there are two major classes of optimisation methods: those which require to be supplied gradient information, and those that do not. Within both of these classes there are a number of different algorithms, and for each of those algorithms there are usually a number of separately developed implementations, some of which are optimised for particular applications. Rather than develop new implementations to test the idea of using direct numerical optimisation, and so save time and reduce the chance of error, published implementations of the selected algorithm have been used as a starting point for the work described here. The selection of the particular implementation is described in the next section, along with a description of how the algorithm was enhanced to improve the mechanism's ability at avoiding plateaus and local optima, and the other problems specified in Section 6.1, page 122.

6.3 A Review of Alternative Approaches

There have been other technologies developed for finding suitable parameterisations of procedures, most notably from classical planning research. Approaches such as planning using geometric reasoning over a model of the environment, task and robot to parameterise strategy skeletons, or admittance matrices have been reviewed in Section 2.2.3 on page 17. However, three technologies have not so far been discussed: *Case Based Reasoning*, *Explanation Based Learning*, and *Genetic Algorithms*. Both of the

first two of these technologies are approaches to automatic development of programs to solve problems. The third is based on an entirely different, biologically inspired approach. However, within each technology there has been considerable variation.

In the remainder of this section each technology will be reviewed in turn. These reviews are orientated around the amount of presupplied knowledge that they required.

6.3.1 Case Based Reasoning

Case Based Reasoning (CBR) (see [Kolodner 94] for more extensive coverage) originates from Schank's work on dynamic memory [Schank 82] looking at how prior experience, even if not directly related to the problem at hand, can be a useful resource when planning solutions to novel problems. Practical systems that have been developed such as CHEF [Hammond 89], or the real time system described in [Kopeikina *et al.* 88], or more pertinently, ACBARR [Ram *et al.* 92] store cases, or scenarios, in a database. To apply a case, this database is searched to find the one which is the closet match to the problem, which is then used by the planner in its deliberations. The issue of deciding when a search is appropriate, and the problems of matching and then applying the case cannot be solved without embedding within the system a model of the environment that can be used to evaluate the probable effects of using that case. How cases are represented has turned out to affect the difficulty of their retrieval and application. In all of the three cited cases, the representation, the model and the planning process were developed together.

The ACBARR system is most relevant to the discussion here. Its cases consist of parameterisations of activities used to guide a simulated mobile robot around a domain which has a variable number of objects that have to be avoided. The underlying operation of ACBARR is similar to that of the Planner-Reactor systems described in Section 2.2.6 (page 35): it is used to modify a behaviour system controller in a way which improves performance. Cases are indexed by a set of environmental feature values. Matching then took place by comparing these values to the values of those features currently sensed by the robot, the closest matching 'winning', though there are some special cases that are selected when certain hardwired conditions come true. The cases are applied by modifying the current parameterisation of the robot towards the

parameterisation given in the case. If this results in an improvement in performance, the modification is repeated until either the values are equal to those in the case, or the situation has changed to make another case more suitable. As long as there is an appropriate case, then the performance of ACBARR is approximately an order of magnitude better than that of an unmodified reactive system in the same environment. Apart from the requirement that the appropriate cases be available, this appears to be a useful system. The problems are the need to identify and pre-specify all relevant cases, and more fundamentally, the necessity to construct the system controller, the model and the cases in a manner which allows CBR to be applied. The assumptions behind this are that it is always possible to identify when a new case is needed, and that is possible to separately index different cases. The validity of these assumptions depends on the problem and how that can be represented. However, as was described in Section 2.2.5 (page 24), there can be situations where arranging for these assumptions to be valid is not physically possible.

However, it is the more immediate problem of developing the cases which makes this approach difficult to apply. In the context of the ill-defined functions that are to be approximated, identification of the cases would consist of discovering beforehand all of the points where a different parameterisation was required for a given task. This task requires identification of the partitions (and their causes) in the parameter space of the activity. This represents an increase in the work required by the developer, which for complex applications where each activity has a large parameter space that could be finely tessellated, could be never ending. If the methods proposed here can adapt the parameterisation to match local conditions, then they will be more effective than any CBR approach in those circumstances.

6.3.2 Explanation Based Learning

This approach to the selection of suitable parameters for activities has developed out of GOFAI planning techniques that use domain specific information, including models of the task being performed, to first generalise and then to customise an example plan to suite a new application. Not surprisingly, Explanation Based Learning (EBL) can be proved to be formally equivalent to planning [Harmelen & Bundy 88]. In practice,

EBL should not need to generate novel plans for itself, instead being able to customise the closest example. Dufay and Latombe developed the first notable application of EBL to robotics [Dufay & Latombe 84]. Segre has developed a robotic EBL system that acts as an apprentice to a human operator performing an assembly task [Segre 88].

Once these systems have been taught how to achieve one task, such as peg in hole, they can then perform as many variants of that task as can be handled by the knowledge they have been supplied with. During the apprenticeship phase of the system, the commands used by the operator to achieve the task are chunked into the activities used by the system to represent and implement subtasks. It is this learning that distinguishes EBL from CBR. In effect EBL systems have the ability to add to their database of cases.

To do the generalisation a full geometric model of the robot, environment, and the parts involved in the assembly, plus knowledge of the effects and preconditions of the various activities used are required. When faced with a new task the EBL mechanism must identify which of its available generalised plans is most suitable, a process that is rarely straightforward. Customisation of the selected generalised plan to the new task consists of extracting from the task specification, using knowledge of the task, the appropriate parameter settings for the different stages of the general plan. All the knowledge that has to be presupplied to the system for the generalisation and teaching to be performed is specific to the task, and to the teacher, or source of initial plans.

In robotic assembly applications where one robot is to perform similar tasks for all of its working life, and where the ways of teaching the robot are restricted, the EBL seems a workable prospect, but only if the necessary knowledge is available, and there is a source of example plans. Apart from the problems in where all the knowledge that is required is supplied from, the most significant weakness of EBL systems is the assumption that the customisation can be performed used the combination of domain knowledge and information in the task specification. In the last chapter when developing an example of using learning to find a model parameterisation it was evident that the best parameterisation could not be established without tests, as not all the necessary information was available *a priori*. If EBL had been used on that example, or on any of the examples described in this chapter it is unlikely that a successful

parameterisation would have been found.

A weaker form of EBL that has been successfully applied to situations where not all the necessary information for customisation is available is simple imitation, *e.g.* [Hayes & Demiris 94], or [Pook & Ballard 93]. Instead of trying to generalise the activities performed by a teacher, the apprentice simply repeats the observed activities whenever its environmental conditions are perceived to be similar to those encountered by the teacher. Imitation still requires the apprentice to have some model of what the teacher is doing to at least (though not necessarily understand) identify the separate activities used by the teacher. Like the strong form of EBL, imitation requires a teacher, and the ability to detect and link the situation of an action's performance and the action. If, as is the case for some activities such as reaching there is neither a teacher nor an easily established link, EBL in whatever form is not suitable.

6.3.3 Genetic Algorithms

The third alternative technology to direct numerical optimisation is the class of software called *Genetic Algorithms* (GAs). Although these might have been inspired by the processes of evolution, the details and results of their operation are in no way analogous to their biological sources [Holland 75]. Holland, regarded by most as the founding father of GAs, first suggested them as part of his ongoing research program into the study of natural adaptive systems [Holland 75]. They are widely used as function optimisers, *e.g.* [Gordon & Whitely 93].

In the language of AI, a GA is a weak search engine that iteratively combines representations of problem solutions according to the relative merits of the separate solutions to produce new solutions which might be closer to the desired goal. The genetic label comes from the representation of solutions being strings (*chromosomes*) made up of bits (*genes*), the evaluation producing a *fitness* measure, and the combination operators being variations on the crossover and mutation mechanisms found in biological genetics. Any particular chromosome and its genes encode a particular parameter setting for an agent, and it is the performance of this agent on the particular task that is encoded by the fitness value. Choosing how to encode a parameter setting as a chromosome and genes is intimately linked to selecting the operators (and their parameters),

which together with the creation of the evaluation function, will take up the bulk of a developer's time in applying a GA to a problem. Once completed, the remaining decisions to be taken before execution are the size of population of chromosomes and the number of generations to be developed during any one run of the GA.

Determination of the chromosome structure and subsequently the relevant operators and their parameter setting usually precedes the work on the evaluation function, but in more complex applications it can be necessary to perform a number of cycles of development in order to get the desired behaviour. The design of the chromosome structure defines the solution space for the problem. In addition to containing the desired goal, this space should be one which supports the kinds of search trajectories generated by the GA, as implemented by the operators used. If the space does not support the search, such as when the distance in parameter space does not relate to the corresponding differences in fitness, then progress by the algorithm is going to be slow at best.

The fact that adequate search spaces can easily be designed is one of the great strengths of GAs. Although there are a lot of decisions and a lot of options for those decisions, it is easy to avoid the bad ones. Performance of the resulting algorithm is relatively robust with respect to its parameter settings. The usual penalty for a poor parameter set is that more generations are required to achieve convergence. One operational detail that has to be carefully handled is the selection of the initial population. If the population is too similar, yet does not contain any examples close to the goal, then there is a high chance of premature convergence. If this occurs then the time required to reach the best solution will increase greatly as the GA becomes dependent on mutation supplying the necessary variation.

Any GA is inherently parallelisable, and much work has gone into developing parallel applications, *e.g.* see the extensive summary in [Stender 93]. Parallelisation consists of distributing the evaluations onto multiple processors. As more processors become available to handle evaluations then a near linear speedup proportional to the number of processors can be achieved. Perfect linear speedups are not possible because of the extra communications overhead involved in dispatching chromosomes for evaluation and waiting for their results, and if the time taken to communicate is greater than the

time taken to evaluate an individual, then overall performance will decrease.

From the basic idea of a GA many variants have been developed. The most common is to have multiple populations (the *Island* model), each population seeded and evolving independently but with periodic breeding between populations which can result in a faster convergence time to the goal for the number of individuals involved than if all were concentrated into a single population, *e.g.* [Goldberg 89b, Starkweather *et al.* 91]. Variants involving multiple chromosomes have been explored, but do not seem to offer significant advantages for current applications [Goldberg 89a].

As noted above, a poor choice of GA variant usually results in an increased number of evaluations being required. However, finding a good variant for the application appears from the literature to require both some knowledge of the application and a number of separate experiments, *e.g.* [Davidor 89, Yuret 94]. Often specific operators and chromosome designs are developed for the application being studied, *e.g.* [Kosak *et al.* 91]. This suggests that to obtain adequate performance from a GA applied to the problem of reaching being considered here, then an appreciable amount of effort would be needed to design and implement the best chromosome design and associated operators. Without undertaking this work, this opinion cannot be substantiated. However, the body of published evidence suggests that within the limited time available, this is not an issue to be considered here.

This requirement for appreciable knowledge about the application would on its own be sufficient to disqualify GAs from consideration here. However, even when the chromosome design and operators, and all other design issues, are optimised, it would appear that GAs have, from the perspective of this thesis, a problem; they require a lot of evaluations to find the optimal value. An example that typifies this which is relevant to this work can be found in [Davidor 89]. This paper describes the use of a Lamarckian inspired genetic algorithm called *Analogous Crossover* to discover a straight line trajectory of 4.2m length for a six jointed robot arm. The parameter space for this problem contained 10^{30} separate values; the GA is reported to have required some 2000 generations to achieve an error of 1 cm over that trajectory. Results for random search and a hill climbing algorithm were also presented, but neither of these approaches achieved better than $10cm$ error over the trajectory. However, as no detail was pro-

vided on either the size of the population or the number of evaluations performed per generation, the only statement that can be made here about the performance of this approach in terms of the total number of evaluations is that at least 2000 were required. This work was conducted entirely in simulation. Simple arithmetic would suggest that the time this would require on a real robot that had to make multiple moves would be excessive, unless a mixture of simulation and occasional reality tests was used, *e.g.* [Cliff *et al.* 92b].

A more general survey of the literature supports this opinion, *e.g.* [Goldberg 89a, Cliff *et al.* 93, Gordon & Whitely 93, Starkweather *et al.* 91, Yuret 94]. Whilst as with the issue of the degree of knowledge required to develop a suitable GA experimentation would be required to substantiate the true relevance of these figures in the reaching application of concern to this thesis, the published evidence is overwhelmingly against the ability of GAs to meet the requirements given in Section 3.3 (page 56).

6.3.4 Summary

For all three of the categories reviewed here, especially Case Based Reasoning and Explanation Based Learning, a large amount of application specific knowledge is required. One of the arguments for using learning is that it should simplify the task of the developer, not add to it. For a robot system that is to be moved between different applications then unless that knowledge is invariant with respect to the specific system task, then it would be better if it were not a requirement for an operating system.

Establishing an acceptable and achievable means for the system to acquire that knowledge during development without the developer investing a large amount of effort has been one of the purposes of this research (See Section 3.1, page 44). The primary reason why these methods have not been used here is because they violate the developer effort constraint. In the descriptions a number of other problems were evident, the most notable of which was the inability for either of the methods to propose solutions that were outside their available knowledge base. When using direct optimisation, it is the environment, viewed through the profit function, which can suggest appropriate solutions.

The third category, Genetic Algorithms, has the same two features as direct numerical optimisation list on page 127. However, in addition to requiring an appreciable amount of knowledge to design a good variant, they require a high number of evaluations (*i.e.* a high number of robot trials), which is why this technology has not been considered here.

6.4 Optimising and Reaching —P_MOVE

QUEASY, described in the last chapter is an example of using learning in an action function where knowledge about a model used at the heart of the action function is integrated into the learning cycle. P_MOVE is an action function that implements reaching and uses learning, but uses no knowledge about the relationship between the sensor and the robot in the computation or learning. Instead a direct numerical optimisation method is used to select the best displacement vector from the gripper's initial location to the target. The main purpose of this experiment is to show that a learning mechanism does not need the more complex model of the type used in QUEASY.

Algorithm 4 describes the detail of P_MOVE, and follows the broad outline and terminology used in Algorithm 3 on page 101. The significant changes are the removal of the *do_initial_calibration* and the *update_calibration* functions, and the introduction of these new functions:-

profit(vx, vy): the profit function for P_MOVE. This profit function is a direction implementation of this cost function:

$$f_s = \frac{\sqrt{(vx - target\_x)^2 + (vy - target\_y)^2}}{max\_error} \tag{6.1}$$

i.e. , the cost is the square of the Euclidean distance (measured in pixels) between the current position and the target position, scaled to the closed interval $[0, 1]$ by dividing the Euclidean distance between the current and target position by the maximum possible Euclidean distance (in pixels) between any target and any position of the arm. This value is the length of the diagonal of the area viewed by the camera.

Algorithm 4 P_MOVE

```
seed_optimisation
get_rlocation(i_rx, i_ry)
rx := i_rx
ry := i_ry
get_slocation(i_sx, i_sy)
vx := i_sx
vy := i_sy
itcount := 1
while (abs(vx − target_x) > TOL ∧ abs(vy − target_y) > TOL) ∧ itcount ≤ ITMAX
do
  r := profit(vx, vy)
  get_optimised_displacement(r, vx, vy, δx, δy)
  abs_move(i_rx + δx, i_ry + δ)
  get_slocation(vx, vy)
  get_rlocation(rx, ry)
  itcount := itcount + 1
end while
```

get_optimised_displacement$(r, \delta x, \delta y)$: calculates the next move of the robot. This function, together with its supporting function *seed_optimisation*, are fully described in Section 6.4.1, page 137.

seed_optimisation: is used to set up the optimisation routines. The function is covered in Section 6.4.1, page 137.

6.4.1 Direct Numerical Optimisation within P_MOVE: Powell's Quadratically Convergent Method

The two functions within Algorithm 4 that implement the direct numerical optimisation that is used to estimate the displacement $(\delta x, \delta y)$ required to move the arm from its current position to the target are:-

seed_optimisation: which is called to initialise various internal data structures, and secondly,

get_optimised_displacement$(r, vx, vy, \delta x, \delta y)$: calculates the next move. It is within this function that the optimisation calculations take place, using a modified version of Powell's method [Powell 64, Press *et al.* 90].

To briefly describe[1] Powell's method we need to consider an N-dimensional space. The goal of the method is to optimise some function f of **N** variables for which gradient information is not available from a starting point **P** with an associated initial search direction $\vec{n}$ (a vector in **N**). The method works by performing a succession of *line optimisations* using a "black-box" algorithm, which in this case is the *linmin* function taken from [Press *et al.* 90] and defined as [Press *et al.* 90]:

linmin: Given as input the vectors **P** and $\vec{n}$ and the function f, find the scalar λ that optimises $f(\mathbf{P} + \lambda\vec{n})$. Replace **P** by $\mathbf{P} + \lambda\vec{n}$. Replace $\vec{n}$ by $\lambda\vec{n}$. Done.

This approach of performing multi-dimensional optimisation through successive line optimisations was first developed by Gauss (referenced in [Pierre 86]). To be really effective, this approach needs the initial, and then subsequent values of $\vec{n}$ to point towards the optima of f.

The approach used by Powell (and others) has been to perform line optimisation successively on each member of a *direction set*, comprising the unit vectors of **N**, *i.e.* $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_{\mathbf{N}}$, rather than just along $\vec{n}$. This step is repeated as many times as necessary until an optima of f is found.

To briefly describe this procedure, consider a function $f(x, y)$ whose contour map (shown in Figure 6.1 as a series of ellipses) defines a long narrow valley (optimisation in this case means minimisation) orientated at an angle of 45° to the coordinate direction vectors e_1 and e_2. From the starting point **P**, the only way for the optimisation algorithm to "move" down the valley is by a series of short steps (highlighted in red). Whilst this is effective, in this case it is not very efficient. A different direction set, *e.g.* one aligned to the orientation of the valley, might have been much faster.

In general, if the second derivatives of the space **N** vary in magnitude (as is usual) then multiple cycles through the direction set are required [Press *et al.* 90]. Within methods such as Powell's that use the direction set approach, much of the computation is directed at ensuring the direction set is updated as the optimisation proceeds so that either:

[1] This description follows largely that of [Press *et al.* 90] and has been included here for completeness.

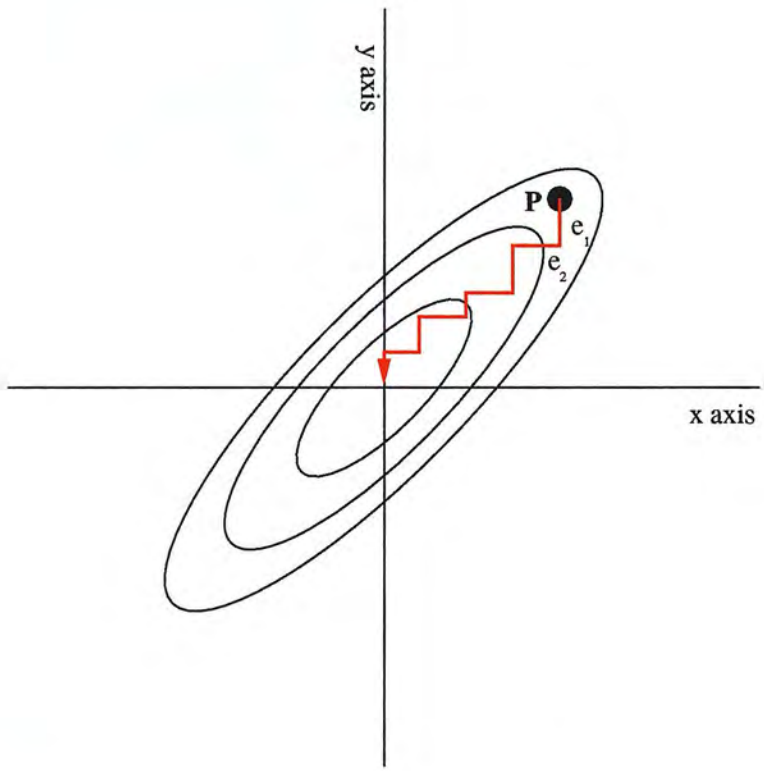

Figure 6.1: Example of optimisation through successive line optimisation along each member of the direction set for a function $f(x, y)$

- The direction set includes directions that will take the optimisation quickly along narrow valleys, or
- The direction set contains *conjugate directions* that have the property that results of line optimisation a long one are not fully, or partially, reversed by the line optimisation along a subsequent member of the direction set.

The value of the second approach is that it reduces, or prevents, high numbers of cycles through the direction set being required to successfully optimise the function. The best that a conjugate direction set optimisation method can do is to derive a direction set where each member is a linearly independent, mutually conjugate direction. One cycle through such a direction set from some starting point $\mathbf{P}$ would put the optimisation algorithm at the optima of f, providing f is a quadratic form. If f is not a quadratic form, then multiple cycles will be required, with each cycle moving quadratically closer to the optima.

Powell [Powell 64] was the first to produce a method for deriving a mutually independent conjugate direction set. He showed that this method would exactly optimise a quadratic function of dimension $\mathbf{N}$ in $\mathbf{N}(\mathbf{N}+1)$ steps (*q.v.* [Brent 73]). However, within Powell's basic method there was a tendency for the direction set to become linearly dependent, or to "fold up on each other" [Press *et al.* 90]. This resulted in the algorithm only finding the optima of a subspace of $\mathbf{N}$, *i.e.* to fail.

The approach adopted here to resolving this problem follows that of [Powell 64], as implemented in [Press *et al.* 90]. Instead of deriving a mutually independent direction set, a heuristic scheme is used that attempts to find a few good directions along which to optimise.

This scheme is simply to recalculate the value for the direction set element along which the last line minimisation cycle achieved the *largest* optimisation after each cycle. Although the value of that element will have been the most effective in the last cycle, it is also the value that is most likely to cause linear dependence to develop within the direction set. This scheme means that the optimisation algorithm is no longer quadratically convergent, and so will require a number of addition line optimisation stages compared to the pure version of Powell's method. This number will be high-

est when all the second derivatives of f are equal (as would occur when f takes an ellipsoidal shape). However, this reduced performance is traded off against the higher probability of successfully finding the optima of f.

The implementation of this variant of Powell's method that has been used here, together with all ancillary functions, *i.e.* implementation of the line optimisation algorithm have developed from [Press *et al.* 90]. This is a widely (though not publicly) available implementation that has been extensively tested though long exposure of the source code to scrutiny and use. However, in its source form, the implementation does not meet the requirements described in Section 6.1 (page 122). Accordingly, the implementation has been developed to include the following features:

Boredom If a mechanism reaches a plateau region of the function its chances of escaping from that plateau are low, as it loses the ability to effectively determine appropriate directions and magnitude because the differences between neighbouring parameter settings become zero. The boredom mechanism monitors the progress of a mechanism, and if there is no improvement for a specified number of iterations, then the optimisation mechanism is terminated.

Random Restarts An alternative to terminating the mechanism when the boredom threshold is reached is to restart the algorithm at a different, randomly selected location in parameter space. By specifying that the new point be at least some minimum distance from the point at which the restart mechanism was initiated, an attempt can be made to escape from the plateau or local minimum in which the optimisation method had become trapped. This restart can also be performed whenever the algorithm has claimed to have found a good parameter setting. Restarting a short distance away from that point and seeing if the algorithm returns there is a standard technique for verifying that the optimum found by the algorithm is good.

Limited Distances For both of the applications considered here, and many other possible robotics applications there are limits set on the ranges of the parameters, because of either physical limits *e.g.* robot positions cannot be inside a solid object, or user defined restrictions, such as a maximum speed. Each mechanism

has been extended so that at any one time the distance it will travel in any one direction will not exceed those limits. The initial limits are set to the boundaries of the search space.

Spanning Directions Initially 'limiting' the distance along a direction explored by the algorithm during the previous modification to the maximum possible had a useful side effect: the mechanism became better at finding the minimum whereas before it was frequently being trapped on the plateau. The directions and distances suggested by the supplied code limited the search to a very small neighbourhood around its starting point. By extending the distances to the limits of the search space the mechanism was able to explore a wider area, and so be more likely to escape from the plateau. The order of magnitude increase in success rate more than made up for the number of evaluations required growing by approximately 20%.

These extensions have been introduced to expand the range of possible profit functions that can be successfully optimised. Placing the robustness of the optimisation in the method rather than in the developer's crafting of the profit function lowers the developer's involvement with the system in an area which can require a great deal of effort. Despite these improvements, any optimisation method can still fail if the error surface is 'wrong' for that method, something which can only be established through experimentation.

6.4.2 Evaluating P_MOVE

Visual P_MOVE (the prefix *Visual* denotes the use of the vision system) has been evaluated following the experimental method described in Section 4.3 on page 84. The overall results of this evaluation are presented first in this section, and are followed by an analysis of the failures and then by an analysis of the performance of Visual P_MOVE.

Tolerance (pixels)	Position Error(pixels)			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	0.66	1.71	4	0.26	0.16	1	1	4.48	1.64	4
4	0.74	2.22	4	0.26	0.16	1	4	4.60	1.81	4
3	0.26	0.52	3	0.26	0.17	1	0	4.24	1.18	1
2	0.12	0.38	1	0.26	0.17	1	0	4.00	0.00	1
1	0.04	0.28	1	0.25	0.17	1	1	4.12	0.59	1

Table 6.1: Summary results of Visual P_MOVE batch testing

Tolerance (pixels)	Position Error(pixels)			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	10.00	0.00	3	0.24	0.00	2	1	10.00	0.00	2
4	8.00	0.81	2	0.31	0.31	2	4	10.00	0.00	2
3	—	—	—	—	—	—	0	—	—	—
2	—	—	—	—	—	—	0	—	—	—
1	2.00	0.00	1	0.05	0.00	1	1	8.00	0.00	1

Table 6.2: Negative of results of Visual P_MOVE batch testing

Tolerance (pixels)	Position Error(pixels)			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	0.47	1.07	5	0.26	0.16	1	49	4.36	1.45	5
4	0.12	0.51	2	0.25	0.15	1	46	4.13	0.88	1
3	0.26	0.52	2	0.26	0.17	1	50	4.24	1.18	1
2	0.12	0.38	2	0.26	0.17	1	50	4.00	0.00	1
1	0.00	0.00	1	0.26	0.17	1	49	4.04	0.19	1

Table 6.3: Positive results of Visual P_MOVE batch testing

Overall Results for Visual P_MOVE

The summary, negative, and positive results for these experiments are shown in Table 6.1, Table 6.2, and Table 6.3. Table 6.4 shows the numbers of the trials that failed, while Figure 6.2 shows the traces of two moves made with a tolerance of 2.

In comparing the tabulated results against those of the Visual QUEASY variants on page 106, page 108 and page 110, Visual P_MOVE appears to have better accuracy, though worse efficiency and requires a greater number of moves than any of the Visual QUEASY variants. For all the statistics, Visual P_MOVE is more consistent than the Visual QUEASY variants.

Tolerance (pixels)	Failed Trials
5	28
4	7, 9, 21, 42
3	No Failures
2	No Failures
1	26

Table 6.4: Indices of failed Visual P_MOVE trials

Analysis of Visual P_MOVE's Failures

The trials on which Visual P_MOVE failed, like those trials on which Visual QUEASY failed, provide information on the limits of the algorithm. Like the Visual QUEASY failures, all of the failed target points in these trials were at singularities in the robot control space. If a trace of a failure, shown in Figure 6.3, is examined, a clustering of points about the final position can be seen. This behaviour could be observed for each failure. It reflects the inability of the robot controller to produce a small enough end motion of the gripper during the large arm motions required as the gripper crosses and re-crosses the singularity. From the point of view of the optimisation algorithm, these failures represent optima that it tried to achieve, but which were beyond the resolution of the robot. This was a problem that the optimisation mechanism had not been equipped to detect.

Analysis of Visual P_MOVE's Performance

The times taken to perform a reach using Visual P_MOVE, shown in Table 6.5 are what might be expected given the number of moves Visual P_MOVE makes. The calculation of each move is more complex than for any of the Visual QUEASY variants, but the times that are shown in the table are a reflection of the bandwidth limitations in the experimental setup than of computational requirements. The variation in times reflects the different length moves required, and the effects of delays in the communications links.

As for the Visual QUEASY variants, a final statistic that can be produced on its performance is to calculate the mean and standard deviations of the Euclidean and Mahalanobis distances from the actual performance of Visual P_MOVE to the ideal per-

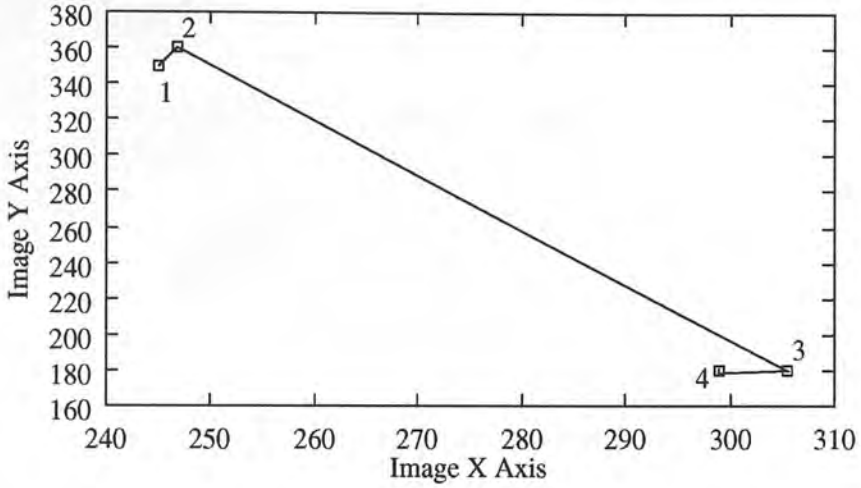

Figure 6.2: Movement Trace of Visual P_MOVE on Trial 17

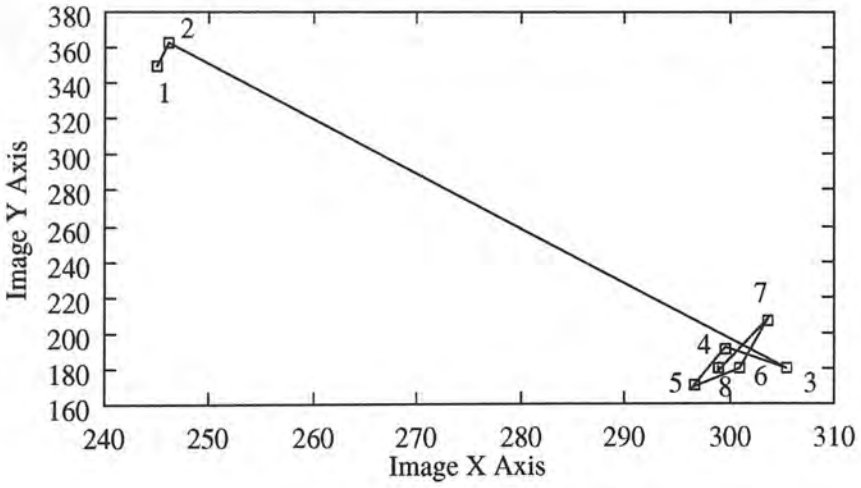

Figure 6.3: Movement Trace of Visual P_MOVE on Trial 21

Tolerance (pixels)	μ_t	σ_t
5	161.05	64.85
4	164.76	67.91
3	151.59	42.81
2	143.24	13.84
1	146.93	19.16

Table 6.5: Mean (μ_t) and standard deviation (σ_t) of the times required in seconds to complete Visual P_MOVE reaching

Tolerance (pixels)	Euclidean Distance		Mahalanobis Distance	
	μ_E	σ_E	μ_M	σ_M
5	161.09	64.88	46.15	7.32
4	164.81	67.94	63.28	7.47
3	151.63	42.83	88.72	7.29
2	143.28	13.84	44.81	7.10
1	146.97	19.16	126.02	7.28

Table 6.6: Distances of Visual P_MOVE performance from ideal

formance of zero position error, 1 step being taken and 100% efficiency. Section 4.4.2 (page 92) describes the rationale and methods for this processing in detail.

These results are shown in Table 6.6 for each of the tolerances, and when compared to the values shown in Table 5.11 and Table 5.12 for the QUEASY variants, the effects of having a lower efficiency and requiring more moves in each trial make themselves evident. Unlike all the Visual QUEASY variants, which have a minimum distance with a tolerance of 3, the optima occurs with a tolerance of 2 for Visual P_MOVE. When the tolerance decreased to 1, the higher number of moves that are required combine with the reduced efficiency to increase the distances reported. This result is connected to the repeatability of the robot being equivalent to a sensor resolution of between 1 and 2 pixels. A tolerance of 2 is within the robot's repeatability, while a tolerance of 1 is not.

6.5 Generalising P_MOVE

Like QUEASY, P_MOVE is not dependent on a visual sensor, and could be used with any sensory device that will locate arm and target in a common coordinate system. To

demonstrate this Force P_MOVE has been implemented using the force table described in Section 4.2.4 on page 79. This section describes how that was done and presents the verifying experimental results.

Algorithm 4, as described on page 137, has been used here. The only changes have been the *get_slocation*(x, y) procedure being replaced with the finger tapping approach described in Section 4.2.4, page 82. Even with the filtering that is used to refine the localisation, successive taps do not produce exactly the same estimated position. This means that the performance of this implementation will not be as good as for the visual implementation. The summary, negative, and positive results developed from the processed statistics (described at the start of Section 5.1.6, page 105) that are displayed in Table 6.7, Table 6.8 and Table 6.9 confirm this poorer performance.

Overall Results for Force P_MOVE

The overall performance of Force P_MOVE is better than that of Force CAUTIOUS_QUEASY, described in the tables on page 116, though this is due to Force P_MOVE suffering fewer failures. It is interesting that the efficiency of Force P_MOVE is higher than for the visual version, which is the opposite of the situation for CAUTIOUS_QUEASY. The closer correspondence between the size of the coordinate frames of the sensor and the robot in this case has made the problem simpler to solve for the optimisation mechanism, which no longer needs to approximate the scaling factor required in the transform as it is not needed. An example of a successful reach by Force P_MOVE is shown in Figure 6.4.

Analysis of Force P_MOVE's Failures

The use of fewer moves by Force P_MOVE to reach the target is one reason for the fewer failures (tabulated in Table 6.10), though when failures do occur they are for the same reason that failures occurred with Force CAUTIOUS_QUEASY. When a target point is at a singularity in the control space of the robot, the difficulty in reaching to that point that was encountered by Force P_MOVE is exacerbated by the low accuracy of the estimates for the robot's position produced from the Force table, a point demonstrated by the movement track shown in Figure 6.5, which like all the other movement tracks was taken from a trial using a tolerance of 2. In this plot, points 6 to 10 all were

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Failed	μ_i	σ_i	Rank
5	2.53	1.67	4	0.55	0.27	1	0	4.60	1.22	1
4	2.38	1.63	4	0.55	0.26	1	2	4.60	1.21	1
3	1.74	0.94	3	0.55	0.27	1	0	4.76	1.31	1
2	1.39	0.77	1	0.54	0.26	1	0	4.78	1.23	1
1	1.21	0.71	1	0.55	0.26	1	7	5.08	1.41	5

Table 6.7: Summary results of Force P_MOVE batch testing

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Number	μ_i	σ_i	Rank
5	—	—	—	—	—	—	0	—	—	—
4	7.38	0.95	2	0.71	0.28	1	2	7.5	0.71	2
3	—	—	—	—	—	—	0	—	—	—
2	—	—	—	—	—		0	—	—	—
1	2.42	0.67	1	0.53	0.26	2	7	6.42	2.51	1

Table 6.8: Negative results of Force P_MOVE batch testing

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Number	μ_i	σ_i	Rank
5	2.52	1.67	4	0.55	0.27	1	50	4.60	1.22	1
4	2.17	1.28	4	0.54	0.26	1	48	4.47	1.07	1
3	1.74	0.94	3	0.55	0.26	1	50	4.76	1.31	1
2	1.39	0.77	2	0.55	0.26	1	50	4.78	1.23	1
1	1.01	0.49	1	0.55	0.26	1	43	4.86	1.03	5

Table 6.9: Positive results of Force P_MOVE batch testing

Tolerance (mm)	Failed Trials
5	No Failures
4	22, 24
3	No Failures
2	No Failures
1	8, 16, 28, 29, 31, 42, 47

Table 6.10: Indices of failed Force P_MOVE trials

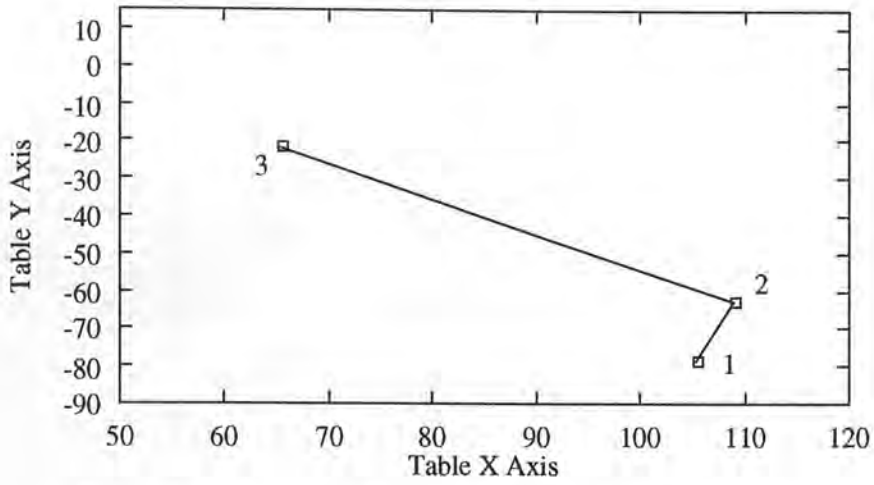

Figure 6.4: Movement Trace of Force P_MOVE on Trial 11

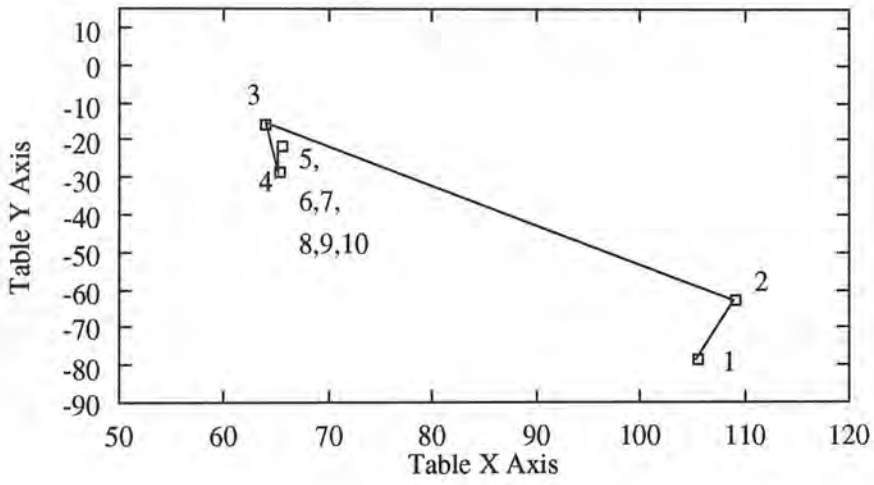

Figure 6.5: Movement Trace of Force P_MOVE on Trial 27

located within 3cm of point 5 and so are not shown as separate items.

Analysis of Force P_MOVE's Performance

The statistics of the times taken for Force P_MOVE to complete, shown in Table 6.11 and the distances of the performances from the ideal, shown in Table 6.12 contain no surprises given the previously tabulated results. That the distances are smaller, and more consistent than those for Force QUEASY listed in Table 5.18 on page 118 could be expected from the fewer failures suffered by Force P_MOVE.

Unlike the situation between the two visual reaching activities so far proposed, the

Tolerance (mm)	μ_t	σ_t
5	197.51	57.61
4	196.73	53.62
3	204.21	60.21
2	204.74	55.75
1	217.25	60.91

Table 6.11: Mean (μ_t) and standard deviation (σ_t) of the times required in seconds to complete Force P_MOVE reaching

Tolerance (mm)	Euclidean Distance		Mahalanobis Distance	
	μ_E	σ_E	μ_M	σ_M
5	197.56	57.62	35.31	7.29
4	196.78	53.62	38.73	7.25
3	204.26	60.21	44.40	7.32
2	204.78	55.76	39.43	6.93
1	217.30	60.93	39.76	7.50

Table 6.12: Distances of Force P_MOVE performance from ideal

differences between the two force reaching activities are more marked, with P_MOVE being significantly better. The closer matching between the coordinate frames of sensor and robot seems to suit the optimisation mechanism, with the consequence that fewer moves are required. This reduces the chances for error to accumulate. The failures of the optimisation mechanism when trying to reach an optima that was outside the effective resolution of the robot that occurred using both forms of P_MOVE indicates that even careful development following analysis of the possible problems cannot spot all problems, and that experimentation is required. It also indicates that there are problems that cannot be handled by a mechanism as they are outside of its 'sensory' abilities. Such problems are why it is necessary to structure activities using some scheme (See Section 2.3, page 29 for background).

6.6 Summary of Chapter

This chapter has described the background and experimental work involved in applying a direct numerical optimisation method based on Powell's method to selecting good parameters for a reaching activity. This work was developed through applying the

guidelines and ideas developed in Chapter 3 to the model based reaching activity described in Chapter 5.

The result has been a generalisation of QUEASY, with the development of an activity called P_MOVE that did not need any knowledge about the relationship between the robot and the sensor, but instead relied on the profit function to tell it how to modify the structure in light of experience. By linking sensing to action through a specified profit function and optimisation method in a feedback loop, the activity has been enabled to effectively approximate the explicitly computational mechanism that was required in QUEASY. The importance of this result is that this shows how a single activity might be used in a number of similar tasks (reaching in this case) without being developed for the specific sensors being used. The existence of such an activity allows the developer to concentrate on the things human developers are still best at; the production of profit functions and the structuring of activities.

During this research a development took place that has turned out to be very useful in its own right. To save computational effort during the tuning of P_MOVE the results of the profit function were cached for reuse as they were produced, indexed by the parameters of the activity. Once this development was in place it quickly became clear that this cache was a valuable resource that could be used actively to predict where good parameter settings might be found. This subject is explored in the next chapter.

Chapter 7

Active Memory

The behaviour of the P_MOVE activity used in the experimental verification of the parameter optimisation approach developed in the last chapter and that of the family of QUEASY algorithms presented in Chapter 5 is heavily dependent on the values of the supplied control and task parameters. These parameters for P_MOVE and QUEASY are respectively the target point and the error tolerance. For both mechanisms, a number of experiments were required to find the correct target points in the robot's coordinate frame on each trial. For both mechanisms the value of these parameters can be stored, and then reused to save on repeating a trial, *i.e.* so that when the robot is next instructed to go to a location defined in the sensor's frame of reference, it can use the now determined location. This type of storage will be called *passive memory.*

This chapter considers how this stored data can play an active role in the process of selecting parameter values for an activity, hence the term *active memory* (AM). The space defined by the parameters and the measured results of the activity will be referred to as the *parameter space.* The term *activity space* will be used to refer to the space defined by the separable activity results and the parameters that cause those results, *i.e.* parameter space is the inverse of the action space. A memory, or mapping of an activity, is the parameter, and possibly action space for that activity. Experimental results are used to define the two spaces. These can then be probed to either recall, or to estimate, activity results or parameters.

The value of this approach is that it can be used to create action-sensor couplings in activities that exactly match the physical transform that exists between the coordinate

frame of the agent (in this case the RTX robot) and sensor. Unlike P_MOVE, which required that the transform could be approximated by the general mechanism provided in the optimisation method, an AM mechanism would not require such assumptions to be made. Because the transform learned by an AM mechanism is the exact transform, any implementation will be more time efficient than an implementation that uses successive approximations. Even if the relevant part of the transform has to be iteratively estimated it will not require the local calibration necessary that is required even by P_MOVE.

As more results are entered into memory the correspondence between the true relationship and that represented by mapping will increase. However, as the accuracy improves, the space required to store the mapping will increase. For example, a full 512 by 512 mapping from the positions of a 6 degree of freedom robot to image space would require at least 6Mb of memory[1]. If several dense mappings were to be maintained then overall memory requirements would be high. Even if the available computer memory and off-line storage were sufficient, obtaining and then maintaining a complete map of the results of an activity indexed by the parameter values will not always be easy, or even a possible option. Unless the physical relationship represented by the mapping is effectively static, *i.e.* does not change in a way that would change the mapping during a number of uses of the mechanism, there is no point in beginning to consider AM. It is also the case, as will be described later, that AM can only be applied if the relationship between input and output is not one to many, as any procedure for estimation will not work under such circumstances.

Complete mappings are not necessary for two reasons:-

- A sparse map, which contains sufficient information to accurately estimate (especially if those estimates can be iteratively improved) every possible desired output will not improve with extra results. When such a state has been reached, the mapping can be frozen.
- Many of the activities used in an application will only be used in a few ways. For example, rather than have to repeat the servoing process for every stage in

[1] 512*512 image locations *6 positions * 4 bytes memory

the sequence it would seem sensible to use some method of storing the results of the servoing for reuse when the system is next required to move the arm to that location. Storing these results as an explicit map offers a number of advantages:-

- With appropriate indexing for the mapping, recalling a previous position is a cheap and fast operation, and unlike other methods of approximation, is guaranteed to produce the same answer for every identical recall.
- Once the global mapping has been updated with a certain number of moves, then it becomes possible to actively use the mapping to predict arm locations for a given visual target, possibly speeding up the change over time from one product to another.

AM mechanisms will require the developer to provide a means for the system to initialise a mapping, and to provide feedback to allow the AM to improve an estimate. In return, the developer gains access to a generic action function which does not need *a priori* specification of the input output function, as used in Chapter 5 (page 94). Nor do AM mechanisms require that the true relationship can be approximated by some class of functions. This was an assumption (viable within certain limits) made to use the parameter optimisation of the last chapter. AM is a form of weak learning which provides an initial approximation that is gradually refined as more results are supplied into a exact correspondence from the defined and undefined input parameters to the output parameters in the activity where it is applied.

In the rest of this chapter the detail of AM mechanisms, and the alternatives are explored. In separate sections, beginning with a review of the alternatives, the mechanisms for storing and updating data to create the mapping and the mechanisms for estimating from a mapping are considered. To validate the idea and compare the effectiveness of AM compared to the P_MOVE and QUEASY the penultimate section describes and reviews the testing of I_INTERP, a reaching activity using AM. Following this the work done in this and the last two chapters on using learning in action-sensor couplings is reviewed and discussed. The chapter ends with a short summary.

7.1 Alternative Approaches

Before going on, it will be useful to examine and assess relevant existing work for its usefulness in the kinds of situation which AM might be used. There have been a number of different ideas proposed and explored which can be classed as either traditional, *e.g.* [Chongstitvatana & Conkie 90b], or biologically inspired, *e.g.* [van der Smagt *et al.* 95].

Examples of the traditional approaches towards establishing dense mappings have already been reviewed in Section 5.1.1 (page 95). Here a number of references to existing work on using a dense mapping between image system and robot hand (or some other target) were noted. In all of these cases the researchers had invested the time in acquiring such mappings (usually automatically) because of the increased accuracy provided during localisation of object features relative to each other and global coordinate frames. However in all of this work, especially that related to robotics, this mapping had to be acquired before any useful work could be done. Here it is proposed that AM should be used with another activity which is capable of achieving the goal. This activity will act as a teacher for AM, as well as providing a degree of redundancy. For now the issue of how to sequence the AM and another other method in such a way as to achieve the best performance, will be ignored until Chapter 8 starting on page 180.

Using AM working over a dynamically acquired sparse mapping has two advantages over traditional approaches:-

Early availability AM will only need a low density before interpolation, etc. can be effective. Combined with the coupling to an existing module of equivalent ability and some kind of decision mechanism this means that AM will avoid a lengthy pre-operational training period.

On line reactivity The continual updating to the memory can be linked with a monitoring capacity to detect changes in the physical relationship through corresponding changes in the mapping, allowing reporting of changes plus the opportunity for remedial action. This would be more robust than in a fixed system where a small change in the physical relationship can render a mapping invalid.

More modern, often biologically inspired, systems such as MURPHY [Mel 88], INFANT [Kuperstein & Rubenstein 89], or those of van der Smagt [van der Smagt *et al.* 95], are typical of the recent applications of Artificial Neural Networks (ANN). Each, like the traditional approaches discussed above, requires a separate training period before becoming operational. Each uses a layered feed-forward architecture to store the associations between input, which is the location of the robot joints as seen and processed by a vision system, and the output which is the vector of joint positions required to move the arm to the designated point in the image by learning an approximation of the true computational function.

This works best when the function to be approximated can easily be represented by the combination of simpler functions supported by the implemented ANN architecture. MURPHY requires a predefined training set. The others examples use randomly generated training sets.

In all cases a large number of cycles through the training set were required before the error over a separate test set is small enough to allow practical application. This result is made worse by the fact that there have been few attempts to apply these methods to a real six degree of freedom arm, [Kawato 90] being a rare exception.

It would appear that these methods do not scale well, as learning times and training set size increase dramatically as the size of the problem becomes larger. Whilst these studies can only be loosely related to biological truth, it is interesting to note that neurophysiological studies of visual and touch systems have frequently argued that the neural architecture they have established is a mapping between the inputs and the outputs of the relevant brain region, *e.g.* [Gross & Graziano 90, Churchland & Sejnowski 92], rather than that architecture establishing a functional, or a model based relationship of the kind used in QUEASY.

Compared to the other developments, *e.g.* CMAC [Albus 75], that have most in common with what is proposed here, AM might seem to be a poor relation. However, these methods do not actually use an explicit mapping between inputs and outputs. Instead, they use a smooth approximation of the mapping, the parameters of which are adjusted to warp the mapping into a suitable form. These methods work as long as

the range of approximations possible spans those necessary, and the application is able to withstand a certain amount of error. If the mapping contains severe nonlinearities, or the samples, which are used in training, do not allow the appropriate warping to be performed, then these methods fail. The point of AM is that its successful use is not dependent on the application being of the right type, or the training information being of the right distribution. As the memory becomes more complete, then the accuracy of AM can be expected to increase up to the limit of the resolution of the indices to the memory. Unlike CMAC or similar methods, it does not have to suffer the errors that are a consequence of having the property of being generalised.

In this review of alternative ways to using AM, a number of suggestions for what it must be able to do have been made. These include:-

- operation over sparse mappings
- detection and response to change in the underlying physical relationship.

Earlier on at the start of this chapter an outline was provided for how AM would work, with separate components to handle the storage and the estimation. In the next two sections the detail of these components will be reviewed.

7.2 The Components of Active Memory

An AM mechanism has three components. The first is the structure of the memory, the second is the updating mechanism, and the third is the mechanism for estimating from the mapping the outputs that will cause a specified, but as yet unknown result. This section describes each of these in turn.

7.2.1 The Memory

This is the only passive component of an AM mechanism, but it is the most important. In an implementation, it will be an area of memory, or perhaps disk space, in which the results are entered or updated and over which estimations will be made. The structure of this memory is a direct reflection of the relationship between the defined inputs

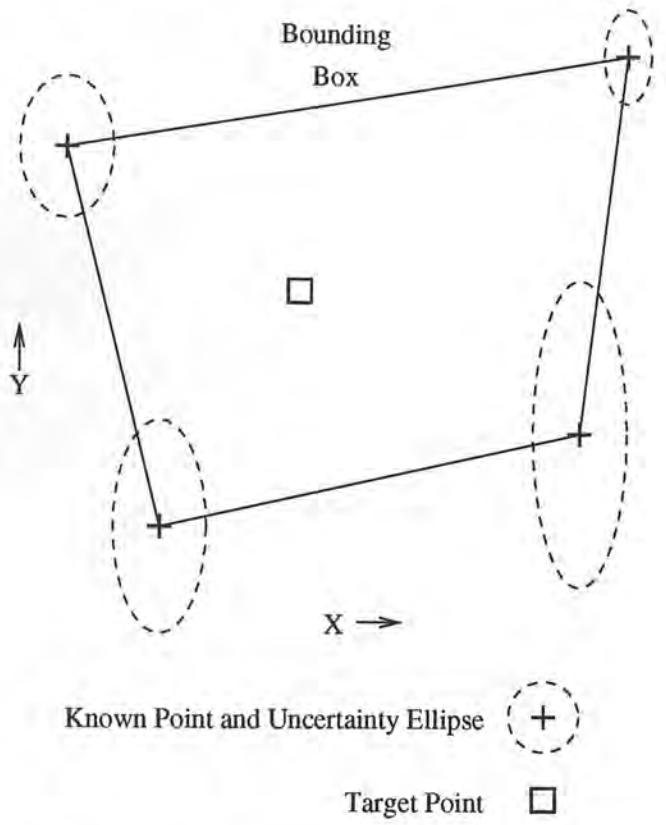

Figure 7.1: Structure of the global mapping

and outputs of the function being approximated. Figure 7.1 depicts the contents of this memory when there are 2 input and 1 output parameters. This structure also determines if, and how many times the estimation component of the AM mechanism can be applied. If there are i input and o defined output parameters, then unless $i \geq o$ and there is a one to one or one to many relationship between the values these parameters can take in the desired direction of approximation, it will not be possible to estimate for a given input what the output of the function might be. This condition must hold for AM to be used.

7.2.2 The Updating Mechanism

Each result presented to the updating mechanism will fall into one of two types. Either it is a repetition of an existing result, or it is a new one. If it is a new one then updating is easy as it can simply be inserted into the appropriate memory location. If there is already and example of the result, then things become more complex.

There is a range of options available here, and considering the reasons why the new result could be different from the old will assist in their identification. If the relationship between input and output parameter settings is one to one, *i.e.* for each input parameterisation there will only ever be a single output value (or vector) then a different result to an existing can indicate two things: The underlying relationship being approximated has appreciably shifted, or there is some noise in the system that is causing an error to be detected. If the situation between input and output parameters is one to many, *i.e.* there can be a range of results for a single parameter setting, an example of which would be the mapping from image positions to gripper locations, then the reasons listed above for a result being different for an identical input parameter setting can be expanded further.

This cause, and the cause of there being noise in the system can be handled in the same manner. When estimates are made using interpolation or extrapolation (see next section) then this variation can be handled by making estimates using the mean and standard deviation of the different results. To ensure that the estimation process does not then develop inertia with respect to the number of samples obtained, the determination of the mean and the standard deviation can be performed using a Kalman Filter such as that used in Section C on page 247 and described in Algorithm 9 on page 249. The measurement variance used would depend on the noise in the system and how many outputs there were in the one to many relationship, though the latter can be established *a priori*.

The handling of a shift in the underlying relationship is more complex, and is an area where methods such as CMAC [Albus 75] that use parameterised models have an advantage. The first issue that has to be handled is how the shift is detected. If a Gated Kalman Filter, *e.g.* (Algorithm 10, page 251), were used, then the number of gating failures could be counted. If this number exceeds a threshold, then a shift might be happening. Apart from introducing a need to set an application specific threshold, there is a possible flaw in using such a detection mechanism. Unless the coordinate frames of the sensor and the agent were a 180o reflection (possibly with some scaling) of each other, and the physical relationship between the two changed along the collinear axes, then the pattern of relative position errors following an alteration

in the physical relationship between sensor and agent would not be isotropic. By using a serial correlation test, it would be possible to say when, with a particular level of confidence, such a situation arose.

From the developer's point of view, this approach is preferable as selection of a confidence level is easier to do and justify than determination of an application specific threshold. Whilst the first test proposed should not be ignored, a serial correlation of relative position errors is a more robust detection mechanism. The question of how best to deal with the drift depends on the rate of drift, the number of results collected, and the number required. A still implicit assumption behind the idea of AM is that the relationship remains static. Change is expected to be one of two things:

- A slow drift due to slippage, and incremental errors.
- A catastrophic change as a result of complete failure, radical reorientation of sensors by external agents.

In the case of the latter, there is really only one thing to do, namely wipe the memory and start again unless a few trials can be performed and from these a global transformation calculated. Once such a transformation had been performed then the mechanism used to correct slow drift could be employed to complete the reorientation. A Gated Kalman Filter contains within it the mechanism to do this last correction. If the new results are within its gate distance then the estimated result for that parameter setting will be gradually changed. If enough new results are outside the gate, then the filter will reset itself and start again. The slow drift might also be corrected in the same way as catastrophic change, through estimating an appropriate transformation. As in this case the transformation would not be as large, it is less likely to require extensive further corrections.

Whilst the update procedure is an important part of an AM mechanism it needs to be coupled with the memory structuring discussed in the last section and the estimation component described in the next section. In this section, it has come out that the updating of a single memory element needs to be done with a Gated Kalman Filter and that any errors need to be serially correlated to detect drift. At the same time an element is being updated, any global transformations that are required to correct changes

in the mapping following changes in the underlying relationship can be calculated and applied. All of these activities should execute in constant time.

7.2.3 Estimation from Memory

The simple (*not* trivial) use of a memory of stored results produced by the mechanisms proposed in the last section is to recall old results whenever a parameter setting is repeated. However, as the number of remembered results from experiments made with the same system grows, then it becomes possible to apply the idea of AM and to make guesses as to the outcome of future experiments from the memory. This section is concerned with the second step of AM, the estimation of a parameter's settings (or output values) that will produce a specified output value (or result from specified parameter settings). If the target point is at a location in memory surrounded by existing points then interpolation can be used, otherwise the estimation will have to be done by extrapolation. These two different methods are dealt with in the rest of this section.

Interpolation

The basic procedure for any interpolation mechanism (see [Gerald & Wheatley 89] for details and theory of interpolation and [Press *et al.* 90] for code) is to first locate bracketing points for the target. In the 2d case here this necessitates finding the bounding box depicted in Figure 7.1. The number of bracketing points available (or used) affects the order of the interpolation, as given n points then an interpolation of an order up to $n-1$ can be performed. For 2d (or higher dimension) interpolation the method used is to perform the interpolation for each dimension in turn, using the results of the last dimension as the input to the interpolation (which need not be the same type or order of interpolation) for the next dimension, a procedure identical to that used during multi-dimensional optimisation. If the data is irregularly scattered, then a divided difference calculation is necessary to calculate the coefficients of the interpolating polynomial. No special measures need be used if the edges of the bounding box are not aligned with the index axes.

As the number of entries into the memory rises, then the order of interpolation can be

increased, and more usefully, the size of the bounding box can be reduced. Unless some detail is known about the underlying function, and in particular what its derivatives look like, increasing the order of interpolation beyond a certain limit that is determined by the application can result in increasing errors in the estimate. In practice greater accuracy can be obtained by using a constant, low order interpolation with a bounding box of minimum area. It is also the case that as more entries are entered, the pattern of distribution becomes more regular, allowing other forms of interpolation which require such regularity like Polynomial or Lagrangian Interpolation to be used, though a divided difference mechanism operating over regular data is equivalent to Lagrangian interpolation. These types of interpolation usually produce more accurate estimates, though requiring greater computation time [Gerald & Wheatley 89].

In the case of AM, as it cannot be predicted when a method other than divided differences could be used, the estimation procedure would have to include tests to identify which method were most suitable for any given estimation. Whilst this can sometimes increase the accuracy of the estimation, during some early experimentation this idea was discarded from the implementation described in Section 7.3, page 164, as by the time a sufficient number of results had been established in the memory for there to be a regular pattern, it was possible to specify very small bounding boxes over which a low order divided difference approach performed as well as the more complicated methods just discussed.

All of these types of interpolation, divided difference, Lagrangian, and polynomial, do not produce continuous gradients between separate memory results. If such continuity is important, then a spline method should be applied, as these produce a continuously smooth curve between and through the points on which the interpolation is based.

Whatever interpolation method is used, the time and resources required are constant for its order (and the number of results in memory), and as the number of results in memory rises the accuracy of the estimate will improve until the estimate is the actual answer. The time taken to reach such a point is dependent on the application and the state of the robot and environment during operation. Experimental results presented in Section 7.4, page 167, indicate that convergence is reasonably fast. When these properties are compared against the list in Section 3.3.2 on page 60 and Section 3.3.3

on page 62 it would appear that interpolation displays all of those operational and learning properties that have been argued for. The existence of the implementation properties will be established as part of the descriptions provided in Section 7.3 on page 164.

Extrapolation

The difference between extrapolation and interpolation is that one or more of the vertices of the bounding box used during interpolation is estimated. The most general way to describe such an estimation is as the fitting of a preconceived model to the existing data from which as many as yet unreported results as needed can be produced. These results can include the actual one being sought, removing the need for a second interpolation step. However, unless the model and interpolation are very similar, the estimates produced are unlikely to be similar. The quality of the extrapolated estimate is dependent on the parity between the fitted model and the underlying function. This in turn is dependent on how well the fitting procedure is performed. That is a process that for non-linear models can be heavily dependent on the initial estimate of the fitting.

AM requires there to be an activity that can supply it with parameter result pairs that form the memory from which estimation can be performed. Unless the details of the function that is being approximated are sufficiently well known to allow a suitable model to be selected and fitted, it is unlikely that any estimation process could produce results close to those produced by the alternative activity. Extrapolation will not be used, as any alternative activities used will be competent and there is no intention here to determine which model is most suited, though this could be a topic for learning. If the experimental system used here were being applied to a real application, then requiring that the developer establish appropriate models, or that the system learn those models would increase the effort required to produce a working system without adding extra worth to that system. Extrapolation over memory is an example of where the benefits of the procedure are outweighed by the costs of its development.

7.2.4 Summary

In this review of the components of an AM mechanism it has been clear that as long as the constraint of the input to output relationship not being one to many is not violated, then many of the issues such as detection of drift and handling uncertainty in the values of the input or output are application specific.

The combination of memory, updating and estimation components defines AM. The next section presents a verifying experiment. Collectively these three mechanisms, if implemented properly, will give the developer a generic action function similar in scope to that used in P_MOVE that need only have its interfaces and details of storage specified and then coupled to a decision function and alternative module to be useful. This last property is present because of the ability of the resulting action function to be instantly applicable and constantly adaptable without requiring further intervention. At the same time as adapting useful information can be made available on what changes are occurring in the underlying physical relationship. All this is brought out in the next section.

7.3 Visual Interpolation — I_INTERP

The visual servoing using the QUEASY algorithms described in Section 5.1, page 95, and the P_MOVE (Section 6.4, page 136) action function are not the only way of moving a robot gripper to a location defined in the visual space of the system. The alternative approach explored here is to determine through experimentation a mapping between the visual space and the control space of the robot, which could be in terms of the robot joint angles, or in terms of the positions of the robot gripper. Moving to a goal becomes a matter of table look up, and if the mapping is correct, then the robot will move directly to the target position. The algorithm which combines the storage, update, and lookup functions to do reaching is called I_INTERP. This section is devoted to describing the algorithm.

Algorithm 5 contains the details of I_INTERP. It is similar to Algorithm 3 on page 101 Algorithm 4 on page 137. The new functions in this algorithm are:-

Algorithm 5 I_INTERP

```
get_rlocation(i_rx, i_ry)
rx := i_rx
ry := i_ry
get_slocation(i_sx, i_sy)
vx := i_sx
vy := i_sy
update_memory(rx, ry, x, y)
itcount := 1
while (abs(vx − target_x) > TOL ∧ abs(vy − target_y) > TOL) ∧ itcount ≤ ITMAX
do
  estimate_robot_location(vx, vy, nx, ny)
  abs_move(nx, ny)
  get_slocation(vx, vy)
  get_rlocation(rx, ry)
  update_memory(rx, ry, x, y)
  itcount := itcount + 1
end while
```

estimate_robot_location(vx, vy, nx, ny): calculates a new absolute position, nx and ny, for the arm. This is done by interpolating over the memory of previous moves using a 2 dimensional divided difference approach, as described in Section 7.2.3, page 161

update_memory(rx, ry, x, y): adds new results to the memory, checking for consistency as it does so. If memory drift is found to be significant then the entire memory is wiped. This simple scheme loses hard won information, but for this work it is effective. The development of a more sophisticated approach that preserves the information, such as determining the amount of drift and then shifting the mapping by that amount, has been left for future projects, as the point of the experimental work here is to show that AM is a useful thing to have available.

Like CAUTIOUS_QUEASY and P_MOVE, unless the arm can be moved to within TOL distance of the target within $ITMAX$ steps, then that reach will be considered a failure.

Strictly, and unlike the other two algorithms, no initialisation is required. However, if I_INTERP were to be used without any results in its memory (a minimum of four are needed), it would fail as the interpolation mechanism could not calculate the appro-

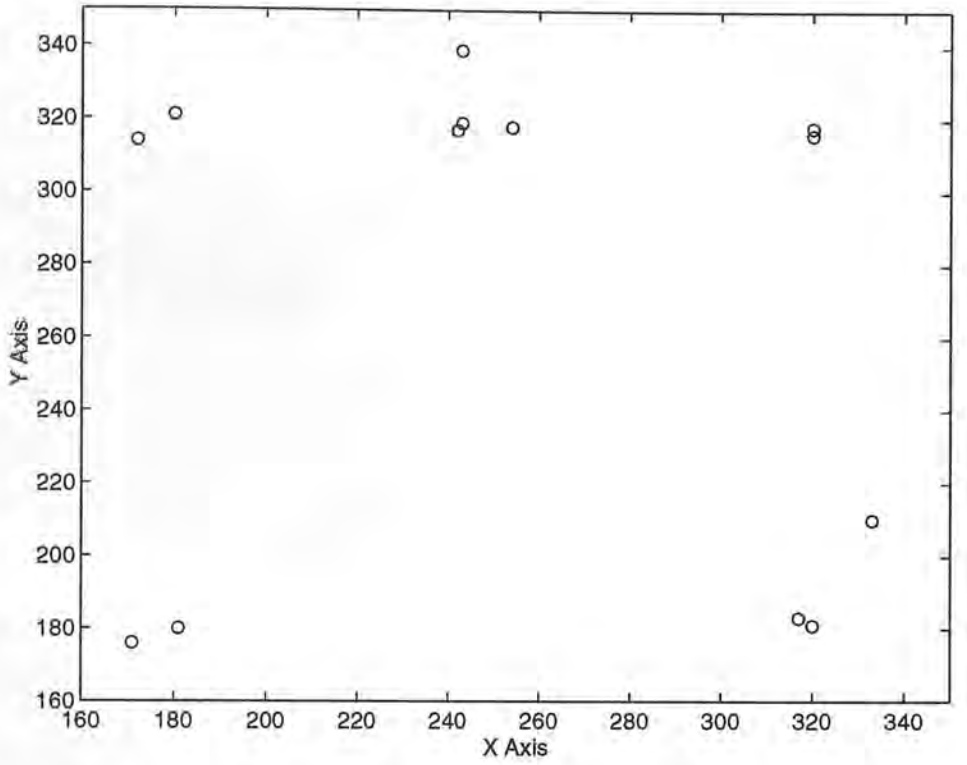

Figure 7.2: Initial Visual interpolation points

priate robot move. In Chapter 8 an approach is presented that could be used to work around this difficulty. Here though, the approach taken is to 'seed' the memory before each experiment. This was felt to be appropriate because it would allow a thorough testing of I_INTERP.

The seeding procedure is straightforward. The arm is moved in turn to the four corners of its work envelope, as defined by the sensor system being used, using the appropriate version of P_MOVE. At each corner the sensor system is used to get the location of the arm. This result, together with the arm's location is entered into the memory. After moving to each corner, the arm is returned to its starting location before being moved to the next corner. The same starting point, and ordering of target points and the same parameter settings for P_MOVE have been used for all experiments. Figure 7.2 shows the results of an iteration of the seeding procedure. In this diagram the hollow circles mark the locations, as seen by the camera, of the actual points reached by the arm.

In the next section the experimental results from applying I_INTERP in the robot system

and task described in Chapter 4 are described.

7.4 Experiments

The pattern and methodology of the experiments used to validate and assess Visual I_INTERP(the prefix *Visual* denotes the use of the Vision system) is identical to that used with Visual QUEASY, described in Section 5.1.6 (page 105) and with Visual P_MOVE, described in Section 6.4.2 (page 142), *i.e.* as given in Section 4.3 (page 84). The only significant procedural difference, apart from the mechanisms used, is the seeding of the memory prior to each data gathering run using the procedure described at the end of the preceding section. The statistics gathered during each run are as described in Section 4.4 on page 90.

Turning to the results tabulated in Table 7.1, Table 7.2 and Table 7.3, and comparing them to the results listed for Visual P_MOVE in Section 6.4.2, page 143 and Visual QUEASY in Section 5.1.6 on page 106, it can be seen that when Visual I_INTERP works, its accuracy is comparable to any of the Visual QUEASY variants, but not as good as that of P_MOVE, though its efficiency is higher and number of moves required lower than for either Visual P_MOVE or the Visual QUEASY variants.

The poor overall performance shown in Table 7.1 is a reflection of when Visual I_INTERP fails, it does so very badly as the negative results in Table 7.2 indicate. These results do not include the failures due to the interpolation algorithm being unable to find a suitable bounding box, though this latter case does have an impact on the summary results. If the indices of the failed trials where reaching was attempted, shown in Table 7.4 are compared to the indices of failures for Visual P_MOVE or the Visual QUEASY variants then the reason for the failures can be deduced. Like the two reaching activities developed, when the target point is at a singularity in the control space of the robot, then it is difficult to move the robot to that location if it is not reached quickly. Figure 7.3 and Figure 7.4 show this, where in the first case the reach is successful in a single move, and the second where the first reach was off target, and in subsequent moves the large motions of the robot that were necessary to produce the small gripper motions required introduced further position errors. In this second plot, points 5 to 10

Tolerance (pixels)	Position Error			Efficiency			Failed	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	33.07	43.86	1	0.96	0.07	1	21	2.06	2.59	3
4	37.41	48.85	1	0.98	0.05	1	20	1.23	0.56	1
3	38.47	47.49	1	0.98	0.06	1	22	1.48	1.71	1
2	34.95	46.51	1	0.97	0.06	1	21	1.81	2.37	3
1	31.16	43.96	1	0.96	0.09	1	22	2.97	3.21	5

Table 7.1: Summary results of Visual I_INTERP batch testing

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	16.89	13.12	3	0.92	0.09	2	2	10.0	0.0	1
4	–	–	–	–	–	–	0	–	–	–
3	12.08	0.0	2	0.95	0.0	1	1	10.0	0.0	1
2	7.39	5.36	1	0.87	0.11	4	2	10.0	0.0	1
1	17.01	13.99	3	0.93	0.07	2	5	10.0	0.0	1

Table 7.2: Negative results of Visual I_INTERP batch testing

Tolerance (pixels)	Position Error			Efficiency			Number	Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank		μ_i	σ_i	Rank
5	2.19	1.65	5	0.97	0.07	1	29	1.52	1.55	1
4	1.43	1.08	2	0.98	0.05	1	30	1.23	0.56	1
3	1.34	0.82	2	0.98	0.06	1	28	1.17	0.47	1
2	1.45	0.73	2	0.99	0.05	1	29	1.24	0.95	1
1	0.93	0.53	2	0.96	0.09	1	28	1.71	1.21	4

Table 7.3: Positive results of Visual I_INTERP batch testing

were within 3cm of point 4, and have not been discretely identified on a figure of this resolution.

Table 7.4 lists the indices of the trials where Visual I_INTERPwas used, but failed.

As with P_MOVE and QUEASY, the times taken by Visual I_INTERP, shown in Table 7.5 are as could be expected from the results tabulated above, with the low mean but high standard deviation produced by a combination of successful applications which are faster than Visual QUEASY or Visual P_MOVE, but where failures, though taking up no time, increase the variation in performance. These times are better than those of either of the other two variants due to the fewer moves and much smaller amount of computation required.

The low distances shown in Table 7.6 compared to those of the other visual reach-

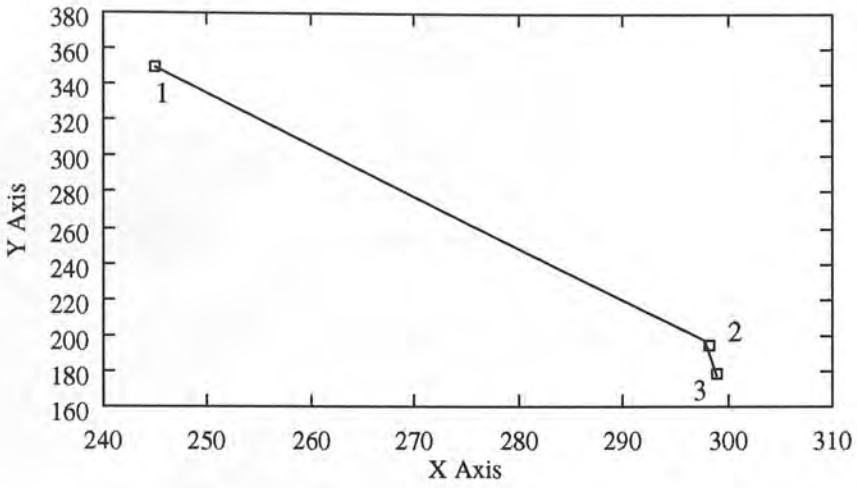

Figure 7.3: Movement Trace of Visual I_INTERP on Trial 17

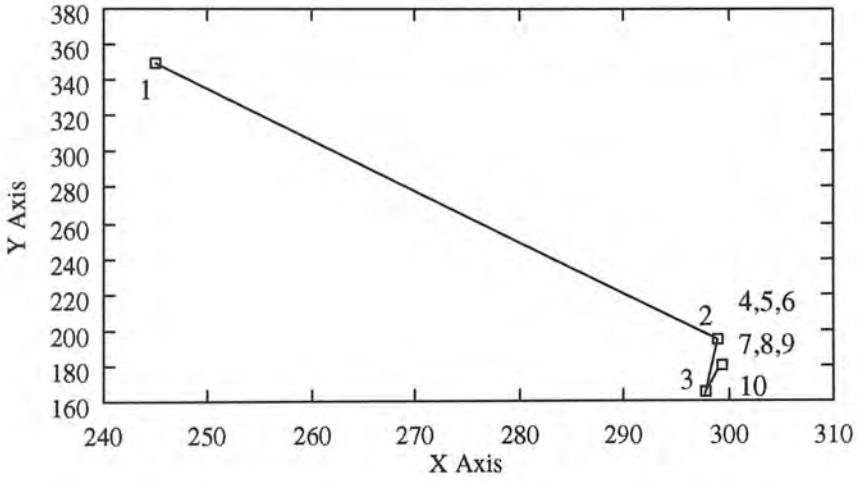

Figure 7.4: Movement Trace of Visual I_INTERP on Trial 21

Tolerance (pixels)	Failed Trials
5	21, 24
4	No Failures
3	26
2	21, 28
1	21, 24, 26, 43, 46

Table 7.4: Indices of failed Visual I_INTERP trials

Tolerance (pixels)	μ_t	σ_t
5	282.77	264.33
4	294.28	238.15
3	150.75	106.96
2	244.31	205.72
1	102.66	63.27

Table 7.5: Mean (μ_t) and standard deviation (σ_t) of times required in seconds to complete Vision I_INTERP reaching

Tables (pixels)	Euclidean Distance		Mahalanobis Distance	
	μ_E	σ_E	μ_M	σ_M
5	68.04	41.03	13.89	7.34
4	59.76	29.88	15.64	6.51
3	59.81	28.21	16.78	6.01
2	65.88	29.12	16.72	7.23
1	72.14	43.04	13.58	7.49

Table 7.6: Distances of Visual I_INTERP performance from ideal

ing activities, shown in Table 5.11 and Table 5.12 on page 114, and Table 6.6 on page 146 reflect that although the accuracy is not the best, in the three other performance metrics Visual I_INTERP consistently exceeds the performance of the other activities. The closeness of the distances reflects the independence of Visual I_INTERP's performance from the specified tolerance level. If anything, performance improves as tolerance increases, though this observation can not be quantitatively verified.

The explanation for Visual I_INTERP's constant performance with respect to decreasing tolerance lies in that as the tolerance increases, more moves are required to reach each target, moves which increase the results stored in memory, which allows for more accurate interpolation to occur. Given this, it could be expected that over time, as the density of the memory becomes higher, then in addition to the number of failures decreasing, the performance of Visual I_INTERP would increase.

7.4.1 Summary

The experimental results indicate that Visual I_INTERP is an effective visual reaching activity, especially as the density of the memory increases. As the necessary density for

interpolation is relatively low, the costs of having to regenerate the memory following changes in the system are acceptable. During the experiments the amount of memory required never exceeded 2k.

As with the Visual QUEASY variants, the principal limitations of using AM are the accuracy of the robot proprioception and controller, and the accuracy of the gripper detection mechanism. This last component can be a significant cause of errors, but as with Visual QUEASY, developing high accuracy gripper recognition modules is not of prime concern here. Again, like the Visual QUEASY variants, this approach can be generalised to other sensory modalities, as is shown in the next section where the force table is used. It could also be generalised to other domains. As long as the principal requirement of being able to determine agent and target position in a global manner is met, and the control space of the agent is well defined, then it should be possible to use AM to initially supplement, and then replace servoing.

7.5 Generalising Interpolation

As described in Section 5.2, page 115, for the visual servoing using the Visual QUEASY algorithms, it is possible to use these interpolation algorithms with other sensory modalities. Not surprisingly, the constraints that apply are identical, namely that the localisation of gripper (or active agent) and target be in the same measurement space, and that motion trajectories in the measurement space can be converted into the real motion trajectories for the arm. Again, as in Section 5.2 (page 115) interpolation using the force-tray device described in Section 4.2.4 (page 79) has been used to experimentally verify this generality.

Apart from the differences in gripper localisation, there have been no changes made to the experimental approach used for the visual interpolation experiments, including the use of seeding.

The statistics of the results of the experiments are shown in Table 7.7, Table 7.8 and Table 7.9. As with Visual I_INTERP, when this version, called Force I_INTERP to denote the use of the Force table, works its accuracy is as good as that of either Force P_MOVE or Force CAUTIOUS_QUEASY, but when it fails, it fails badly. Like Visual I_INTERP, the

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Failed	μ_i	σ_i	Rank
5	3.11	1.49	3	0.97	0.07	1	18	2.43	0.80	1
4	5.06	5.85	5	0.89	0.12	2	22	4.42	3.41	3
3	2.23	0.96	1	0.91	0.12	2	15	3.08	1.26	2
2	3.42	3.87	3	0.88	0.13	4	23	5.18	3.31	3
1	2.88	3.41	1	0.85	0.07	4	28	7.12	3.07	5

Table 7.7: Summary results of Force I_INTERP batch testing

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Number	μ_i	σ_i	Rank
5	0.0	0.0	1	0.0	0.0	4	0	0.0	0.0	1
4	13.11	9.5	4	0.79	0.15	1	7	11.0	0	3
3	0.0	0.0	1	0.0	0.0	4	0	0.0	0.0	1
2	11.83	2.93	4	0.72	0.11	3	5	11.0	0.0	3
1	7.26	3.72	3	0.82	0.07	1	9	11.0	0	3

Table 7.8: Negative results of Force I_INTERP batch testing

Tolerance (mm)	Position Error			Efficiency				Iterations		
	μ_{er}	σ_{er}	Rank	μ_{ef}	σ_{ef}	Rank	Number	μ_i	σ_i	Rank
5	3.11	1.48	4	0.97	0.07	1	32	2.43	0.8	1
4	3.05	1.43	4	0.92	0.11	2	28	2.78	0.83	2
3	2.23	0.96	3	0.91	0.12	2	35	3.08	1.26	2
2	1.86	0.63	2	0.91	0.11	2	27	4.11	2.31	4
1	0.89	0.41	1	0.86	0.08	2	22	5.54	2.11	5

Table 7.9: Positive results of Force I_INTERP batch testing

efficiency of the reaching, and the number of moves required are superior to either of the other two force reaching activities.

The failures of Force I_INTERP are tabulated Table 7.10. These failures occurred during the application of interpolation when the memory did not contain enough quality readings to allow the correct estimation of the robot position. The poor performance that then resulted would often be worsened further if the target point was a singularity in the control space of the robot. Although the trace provided in Figure 7.6, which like the trace in Figure 7.5 was taken from the trials with a tolerance of 2, does not show an isotropic pattern, the overall distribution of relative position errors is isotropic. Like the poor trials that occurred with Force CAUTIOUS_QUEASY and Force P_MOVE, the uneven distribution of errors along the measuring coordinate frame of the sensor produces the

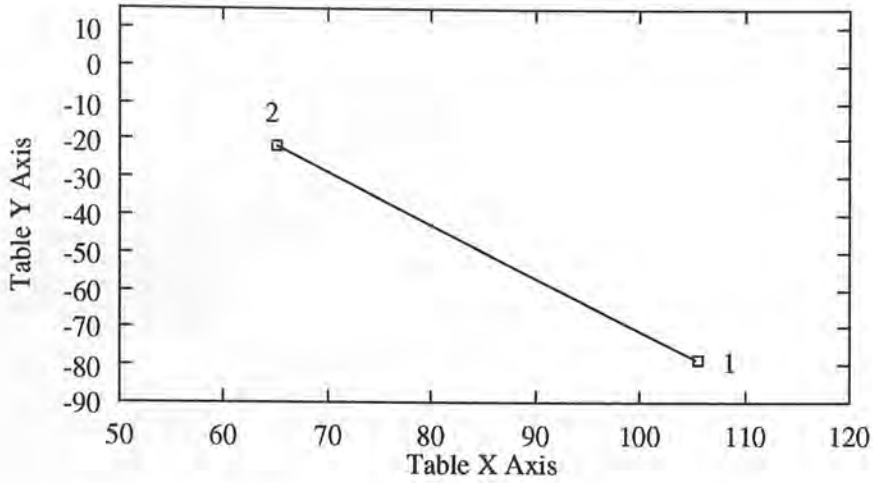

Figure 7.5: Movement Trace of Force I_INTERP on Trial 11

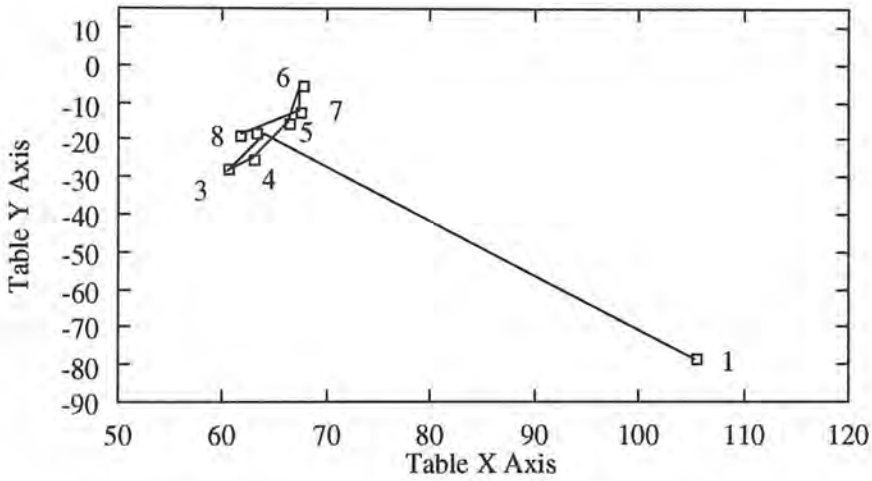

Figure 7.6: Movement Trace of Force I_INTERP on Trial 27

robot motion along an axis parallel to a diagonal of the table as a consequence of the distribution of estimates of the point of contact.

As for Force P_MOVE, the statistics of the time taken to complete each trial tabulated in Table 7.11 and the distance of the actual from the ideal performance reported in Table 7.12 follow from the results so far presented. Both the times and the distances are on the whole as good as or better than those for Force P_MOVE (page 150) or Force CAUTIOUS_QUEASY (page 118). Where the results are worse, it is because the interpolation algorithm has been unable to be applied, consequently reducing overall performance. The pattern of improvement evident in the performance of Visual I_INTERP does not seem to be present here, with the errors in the position estimates

Tolerance (mm)	Failed Trials
5	No Failures
4	8, 12, 13, 21, 31, 38, 42
3	No Failures
2	10, 12, 34, 43, 47
1	8, 11, 13, 18, 21, 34, 37, 42, 46

Table 7.10: Indices of failed Force I_INTERP trials

Tolerance (mm)	μ_t	σ_t
5	87.03	41.57
4	151.92	146.63
3	123.25	85.07
2	166.45	154.99
1	212.93	169.43

Table 7.11: Mean (μ_t) and standard deviation (σ_t) of time in seconds to complete Force I_INTERP reaching

never permitting the same degree of accuracy and consistency.

Like with Visual I_INTERP, if the system 'knew' when the force interpolation could or could not be used, then there would not be the overhead of first attempting to use an interpolating reach, or suffer from trying to use Force I_INTERP before sufficient results had been obtained. This experiment, like all the other experiments that have been described, has allowed identification and characterisation of problems with the mechanism that could not have otherwise be easily identified, or which could be easily overlooked during theoretical and practical development. In particular it, like the other force reaching experiments, has shown how although the force table might appear to be a more suitable sensor as the relation between its coordinate frame and that of the

Tolerance (mm)	Euclidean Distance		Mahalanobis Distance	
	μ_E	σ_E	μ_M	σ_M
5	109.95	35.12	977.27	5.66
4	197.87	154.26	32.42	6.62
3	156.91	80.81	20.83	6.77
2	235.91	155.53	16.42	7.00
1	316.18	133.91	41.08	6.20

Table 7.12: Distances of Force I_INTERP performance from ideal

robot is one simple translation and rotation without the addition spherical distortion of pixel quantisation present with the camera, the low quality of the position estimates that can be obtained even using filtering actually renders it less suitable.

The experiments have also shown, like in the other reaching experiments, that whenever the target point is at, or close to a singularity in the control space of the robot, then the large motions of the arm that result from trying to move the gripper a small distance induce position errors. This is a feature of the robot and its controller that can be added to the list described in Section 4.2.2 on page 73 of ways in which the details of the equipment affect how a particular functionality is implemented in the system controller.

7.6 Review of Learning in Action-Sensor Couplings

This chapter is the last of the three in this thesis where learning has been applied to the action-sensor couplings that are the heart of most activities. The details of these couplings are specific to the mechatronics, the environment, and the task of the system. In this and the preceding two chapters the development of action-sensor couplings has progressed through the following stages:

1. A model of the relationship between sensing and action is tuned through learning to produce the best performance (Chapter 5, page 94).

2. Parameter optimisation methods are used to learn approximations to the models of the first stage, (Chapter 6, page 121).

3. Parameter settings are stored in a structured memory. This is used by an interpolation mechanism to produce an approximation to the model. (This chapter).

In this progression there was a declining requirement for presupplied knowledge about the sensor action relationship. Instead, the emphasis has been placed on the establishment of profit functions (Section 3.2, page 51) that report on the difference between the desired outcome and the actual outcome of an action function. Such profit functions represent a higher level interface for a developer wishing to produce robot systems

where sensor mediated activities are necessary for that system to be useful. This is because instead of being concerned with the detail of a relationship (*i.e.* writing an action function), and the procedures for managing change in a relationship, they need only be concerned with linking in a generic action function and stating how from the system's point of view the performance of that action function differs from the ideal necessary to achieve the task of an activity.

Although two different kinds of learning mechanism were identified as being suitable for this kind of learning in Section 3.1, page 44, it has turned out that in practice they can both be described as learning optimal parameter settings. In the case of learning model parameterisations (Chapter 5), the learning mechanism has to be derived from the model. Whilst this improves the performance of the activity, it does not make life easier in the short term for the developer. Combining iteration with the powerful function approximators of direct numerical optimisation methods makes for activities that have equal performance to model based ones but which require far less development work.

Such generic activities, as used to produce P_MOVE and I_INTERP, are not the silver bullets of robotics. Apart from needing linkages and profit functions to be established, they have to be structure appropriately. This structuring performs two functions: allowing error recovery and providing a high level interface to the functionality of the system controller.

The first kind of functionality is essential, as when an activity fails, then unless the system is able to notice that it has failed, and is able to take remedial action, then the whole system will fail. In the learning mechanisms described here, each activity has the ability to notify the system when it has failed. This ability was implemented through a combination of timeouts and tolerance, *i.e.* the mechanism would indicate to the calling mechanism when it had completed more than the allowed number of iterations without reaching within the set tolerance. The recovery ability has been provided by having a number of different activities, each of which can achieve the same result. In reaching, when failure occurs there are no intermediate stages that have to be dismantled to let recovery take place, so when one of these learning mechanisms fails, another can instantly be tried. Handling the dismantling of intermediate stages is an essential ability for a complete system. In classical systems the error detection and recovery

structures are created during the production of the customising controller for the task (see Section 2.2.6, page 25). The error recovery structures of behavioural systems are built in to the activities from the start. In both cases, construction of these structures has to be preceded by identification of where errors might occur.

In the customised controllers produced under the Classical approach there is no need for any kind of on-line reasoning. The general behavioural system on the other hand does need some on-line reasoning, or planning to provide appropriate scheduling of activities. In Chapter 2 where this was explained, it was noted several times that for the planning to be tractable, it must be done in a world from which as many of the complexities of reality have been removed as possible. The kind of structuring used in a behavioural system to provide error recovery can be built in such a way as to present a series of hierarchical virtual machines, *e.g.* SOMASS(page 38), or CARS (page 39), on top of which a classical AI planning system can be made to work. At the highest level this kind of behavioural controller provides a set of abilities about whose implementation the planner can remain ignorant. These abilities, such as 'pick up this object', or 'put the third part into this location', can be used as operators by the planning system in a world that is free of uncertainty and variation. From these planners the appropriate schedules of activities can be produced.

In the structuring of the controller there are often decision points where the system has to select one activity from a candidate pool of equally applicable activities. Each activity can use a different kind of action-sensor coupling, or even different sensors and robots to achieve the same task. The task of deciding which to use in any given application was identified in Section 3.1, page 44, as being one where a weak learning mechanism could be useful, as this would increase the reliability of the system at the same time as removing a difficult chore from the developer. From the work on I_INTERP described in this chapter, it is clear that if such a learning mechanism can handle a non-stationary world and provide error recovery, then activities such as I_INTERPwould become even more useful. The constant improvement of an AM mechanism with every result it is supplied with means that activities such as I_INTERP get better. However, as has been seen here, they only become useful after a minimum number of results have been supplied. Until that stage is reached I_INTERP will fail, though with decreasing

frequency, and when it does an alternative needs to be used. The pattern of its usage over time should reflect that it is improving. In the next chapter a mechanism that exhibits both of these properties, as well as all of the other essential properties described in Section 3.3, page 56, is presented.

7.7 Summary

The subject that has been covered in this chapter is the active use of a memory of previous results. The idea for this stemmed from realising that the cached intermediate results from the parameter optimisation experiments could be used as the basis for interpolation (or extrapolation) to estimate motor output that achieves a particular sensor result, providing the estimation was embedded within a feedback loop that attempts to minimise the resulting error. This is another way to approximate the model that would otherwise be required at the heart of an action-sensor coupling.

In the experiments presented in Section 7.4, page 167, the maximum density of any mapping was less than 1%, yet successful performances were as good as, or better than those of the QUEASY algorithms. Yet, these performances were obtained without making assumptions about the underlying relationship, as was done in Chapter 5, or could be approximated by (Chapter 6). Although storing the complete actual relationship in the would require a lot of memory, unlike a model based approach, it would be guaranteed to be correct for that application, as long as the physical relationship remained effectively stable. In practice only a spares mapping would be required as the iterated estimation would be able to fill in the gaps, and many areas of the mapping would be never visited in a single application.

This kind of learning is a form of weak explanation based learning (See Section 6.3.2, page 130) that includes the ability to generalise from examples. These examples are provided initially from the other, comparable, activities of the system. As the skill of the learning mechanism increases, it became able to provide examples for itself. At all times the other activities remained available to provide backup in the event of failure or a task being beyond the scope of the learning mechanism's experience. The interpolation mechanism gives AM a useful advantage over EBL and CBR: it is not

limited to only being useful when it 'knows' for certain what to do. Instead, it can make a guess, and use the results of that guess to make a better guess, repeating the process until either it achieves the goal, or decides it has failed. If this happens, then under the architecture proposed here, another activity will be used to try and achieve that goal.

Chapter 8

Activity Selection

So far in this thesis three different approaches to moving a robot gripper to a target point defined in the coordinate frame of a sensor have been presented:-

- The Visual and Force QUEASY activities (Section 5.1, page 95, Section 5.2, page 115)
- The Visual and Force P_MOVE activities, (Section 6.4, page 136, Section 6.5, page 146)
- The Visual and Force I_INTERP activities (Section 7.3, page 164, Section 7.5, page 171)

Each approach has utilised a different form of learning. In the last two approaches the learning mechanisms were used to produce a general purpose activity that contained action-sensor couplings implemented without knowledge of the specific sensors, or the robot, or the relationship between them. Both of these approaches used a user supplied profit function to customise the learning mechanism to a task. Unlike the model based learning presented in Chapter 5, here only a profit function has to be specified. This leaves the developer with the tasks of customising and structuring activities, so reducing their workload to the things computers are still bad at. This reduction could be continued further by using learning inside the decision functions which are present in many activities.

In Chapter 1, page 5, activities were described as structured entities, composed from other (in this case subsidiary) activities. Although these subsidiary activities might be used to achieve separate sub-goals, it might be the case that for one, or more of the sub-goals, there is more than one activity that could be used. Of these, some will be better than the others in any given application. However, experimentation is usually required to identify the best activity, especially when, as with learning, the comparative quality of the behaviour of the activities changes over time.

The task of the learning mechanism described in this chapter is to choose the best of a set of comparable activities. This learning will be done through physical experiments. During the discussion of what kinds of things learning should be used for in robotics in Section 3.1 (page 44) it was argued that as current learning mechanisms are in effect automated experiment performers and evaluators, the best tasks to which they can be applied are those where they will remove the necessity of experimentation from the developer. This is what the mechanism described in this chapter does. It, like the parameter tuning methods used in Chapter 6, does not need knowledge about the activities it is selecting. It is a general purpose decision function that is customised through the profit function which informs it of the quality of a particular activity's behaviour.

Evidence to support this this idea of using this kind of decision function can be found in a number of ethological studies, *e.g.* [Herrnstein 61, Harder & Real 87, Real 91, Grundel 92, Anderson *et al.* 92]. In each study the animal in question was found to choose the activity which gave it the most benefit, from its perspective. This latter feature is supported by [Anderson *et al.* 92], where rats with electrodes embedded near their hypothalamus quickly choose to prefer activating the electrodes over eating, even when they had been starved and food was made freely available.

It is important to stress that the mechanism proposed here chooses how to achieve a task. It does not determine what task to achieve. Most applications of reinforcement learning in robotics, *e.g.* [Kaelbling 93, Salganicoff *et al.* 94] attempt to associate activities with particular environmental states, so that when fully trained, the robot will be able to achieve the goal no matter what environmental state it starts from. In doing this, they are learning how to link subgoals together to achieve the

overall goal. In effect they are learning the plan of how to achieve the task, *e.g.* [Sutton 90, Whitehead 91, Connell & Mahadevan 93]. In this thesis a different view has been taken, with the assumption being that this kind of activity, *i.e.* planning, will be performed by an appropriate GOFAI planner, as is the case with the SOMASS system [Malcolm 87] or the CARS system [Hallam 91b]. This view has been based on the review presented in Chapter 2 (page 11).

This chapter follows the pattern that has been established in the preceding chapters by beginning with the properties of the mechanism. It then goes on to describe the detail and reason of that mechanism and presents the experimental results which validate those abilities. This mechanism, like the mechanisms introduced in the previous chapters, is intended for use in sensor-rich behaviourally redundant systems where flexibility and robustness are products of the software.

8.1 Desired Properties

Section 3.3, page 56, lists the various properties that any learning mechanism used in a robotics application should display. This section adds to that list those properties that are specific to the kind of learning described in this chapter. The purpose in listing these properties is to guide the selection, development, implementation and testing of the learning mechanism.

Architectural Properties and Requirements

The role of the mechanism in the computational component will be to select the best activity to achieve a particular goal. In the context of the behavioural approach, this means that the mechanism will be inside each activity. As such, the consequences of its selections will be largely hidden from it. Activities are not always going to be independent, especially in the structured activities typical of dextrous manipulation. The mechanism should allow for an external expert to be used which can indicate if one, or more, of the candidate activities should not be used because of their future consequences. Three alternatives to this approach are:-

- To provide the system with the ability to recover from 'dead-ends'.
- To tolerate the inefficiencies that purely local selection will incur.
- To make the mechanism aware of the effects of the consequences by only updating the internal state after completing the entire activity in proportion to the effect that decision had on the outcome.

The last alternative is more commonly known as credit assignment [Sutton 84]. This approach will be avoided here because reported evidence, *e.g.* [Sutton 90] indicates that it is likely to make learning which action is best slow, requiring an exponential amount of time with respect to the number of activities used to achieve the final outcome [Whitehead 91]. Instead it will be assumed that a model of the interactions of the activities in relation to a task can be constructed. This will mean that on each call to the learning mechanism the selection process can be modified to avoid the activities that will cause future conflict. This kind of modelling, and the instructions to use, or not use, an activity can be handled by the scheduling and planning systems (See Chapter 2, page 11).

Operational Properties

In addition to those operational properties described in Section 3.3.2 (page 60) the mechanism needs to include as much error recovery as possible. As noted in Section 2.3.1 (page 30) even otherwise very good activities will occasionally fail. Evidence of this can be seen in the results of testing the reaching activities. When they fail, and do so in a manner which allows the other candidate activities to be performed, the mechanism should try to use those other activities in order of decreasing quality until it finds one which succeeds, or until it runs out of candidates. When an activity has caused irreversible changes before failing, then error recovery will need to be more complex.

Summary

The presence of some of these properties, such as having few parameters, can be determined from inspecting the implementation of the mechanism. Others, such as cor-

rectness and tractability can be established by analysing the underlying algorithm, but it is always necessary to verify their existence experimentally. These analytically derivable properties can be used to identify, or build the most promising mechanism. If there are two mechanisms that are otherwise equivalent in terms of the properties they display, then the deciding factor will be the relative complexity of the methods. The remaining properties can only be demonstrated through experimentation.

8.2 ANNIE

ANNIE[1] is the outcome of a development process to adapt Kaelbling's IE [Kaelbling 93] algorithm to cope with non-stationary worlds and real valued results. At the heart of the learning mechanism is the selection of the activity with the largest upper confidence limit at the time of the decision. The question of why this is a good thing is considered first before the details of the algorithm and its experimental evaluation.

8.2.1 Decision Making under Uncertainty

Here the learning mechanism has to select the best candidate activity to achieve a goal. It has to do this only using the history of the previous performances of those activities. When making a decision about which activity to use it has to balance selecting to produce what it knows will be a reasonable performance, or selecting to find out if one of the other activities is better.

As activities can fail, the decision of which activity to use can be considered a decision on which lottery to select, each lottery having two outcomes, success or failure. Formally, for activity a the associated lottery $\mathcal{L}^a$ is

$$\mathcal{L}^a = [< p_s^a, o_s^a >, < p_f^a, o_f^a >] \tag{8.1}$$

where $0 \leq o_s^a \leq 1$ and $-1 \leq o_f^a \leq 0$ are respectively the values of successful or failed outcomes (as described in Section 3.2, page 51), and which respectively have a chance

[1] Algorithm for Normal, Natural, Interval Estimation

of p_s^a or p_f^a of occurring, with $p_f^a = (1 - p_s^a)$. The *expected payoff* $EP(\mathcal{L}^a)$that can be expected from such a lottery is

$$EP(\mathcal{L}^a) = p_s^a o_s^a + p_f^a o_f^a. \tag{8.2}$$

For any one expected payoff there can be a number of combinations of p_s^a, o_s^a and o_f^a that will produce that value. It can also be the case that even though two lotteries have same the expected value might be the same, the actual values produced could vary widely, *e.g.* activities a and b have the same expected payoff of 0.36, but their lotteries are:

$$\mathcal{L}^a = [< 0.7, 0.9 >, < 0.3, -0.9 >] \tag{8.3}$$

and

$$\mathcal{L}^b = [< 0.9, 0.4 >, < 0.1, 0 >] \tag{8.4}$$

Activity b has a lower payoff than activity a. However, it has a higher chance of succeeding, and when it does fail the consequences will not be as painful. It is not difficult to see that for two lotteries, even if one of the lotteries has a higher pay out for success, then because its chances of having that payoff are lower, or that it has worse consequences of failure, then its expected payoff is lower.

The mechanism described in Section 3.2, page 51, for determining the profit value of a trial comprises a series of assumptions and mechanisms that limit the values of the outcomes to $[-1, 1]$. Using profit functions constructed by this approach constrains the possible expected payoffs to the closed interval $[-1, 1]$. If the possible combinations of payoffs and probabilities with these constraints are computed then a number of observations can be made:-

- Activities which are likely to fail generally have negative expected payoffs.
- Activities with low success payoffs will not have large expected payoffs.

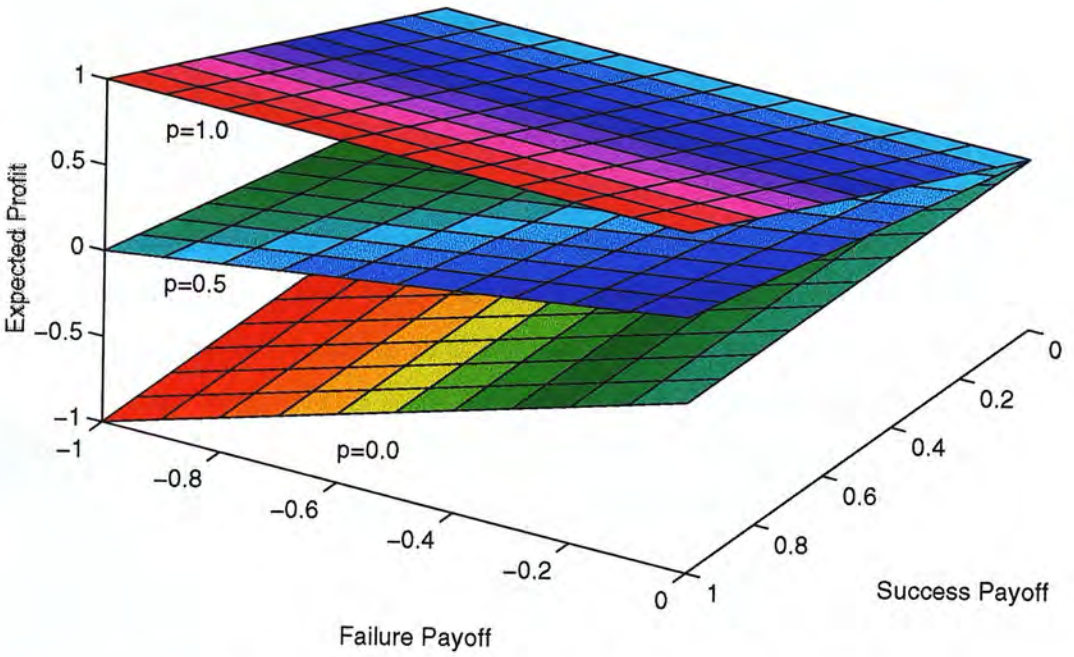

Figure 8.1: Expected payoff for different probabilities and outcomes

- Activities with large expected payoffs have high success payoffs, and a high probability of success.
- The expected payoff for activities with low success and failure payoffs is relatively indifferent to the probability of those activities succeeding.
- For a given probability of succeeding the expected payoff will be less than any possible expected payoff's for activities which have a higher chance of succeeding.

Examples of the possible payoff's for three different probabilities of success (1, 0.5, and 0) are shown in Figure 8.1.

If the choosing mechanism follows a strategy of picking the activity with highest expected payoff, then the mechanism will pick the activity which will is most likely to succeed, and which will be the best at achieving the goal of the activity. This strategy does not guarantee to pick the activity with the lowest chance of failing. For robot

arms, where the cost and chance of a failure can be difficult to predict, this strategy is a good one.

Unfortunately, the situation is not this simple. The system cannot be supplied *a priori* with the lottery of each activity, but has to work it out for itself. This problem is an example of the 'k armed bandit' problem, where k is the number of activities in the candidate pool. Initially the system has no information to distinguish which of the arms will provide the most payoff, or for the case here, which activity produces the most profit. Every time it performs an activity, it learns the payoff for that activity, and as it performs more and more trials it is able to construct a picture of how the activities compare. Whenever it chooses an activity to perform it has to make a decision based on these past results, which might be misleading, and cannot be fully trusted. The dilemma between choosing the known best activity, or an apparently worse activity to find out if it is better, is called the *explore/exploit* dilemma.

There are a number of strategies for dealing with this dilemma. The simplest of these, and so the baseline against which all strategies for making decisions in the face of uncertainty are compared, is to randomly select an action.

Before going on to describe the strategies it is important to note that none of them learn to associate states of the robot and environment with what activity be used, *i.e.* they are not *associative*. Instead, they learn which action produces the highest profit irrespective of state, *i.e.* they are *non-associative*. Behind this decision to use non-associative methods is an assumption that at a particular decision point during a task the environment and robot are going to be in similar states whenever that decision point is reached. This assumption depends on the environment and robot not significantly changing between successive decisions. This issue has also been reviewed in Section 8.1, page 182.

Direct strategies make decisions based only on the actual information gathered after each decision. They can be simple and effective, such as the expedient strategy of 'pick the best until it fails'. A good direct strategy (see [Berry & Fristedt 85] for a summary of different approaches) is to use the worst activity on an exponentially decreasing frequency [Holland 92]. If a discount rate on the profit values can be specified then

the optimal learning and performance strategy is to calculate the *Gittins' indices* for each action [Gittins 88], and select the maximum, an approach that has been applied to robotics [Salganicoff & Ungar 95]. However, these strategies do not handle learning which is not a Bernoulli process, as the metric from which decisions are made have to be calculated at every decision point for every expected eventuality. As the computations required in some of these strategies, such as Gittins indices, are large, applying them to non-stationary environments would require having access to powerful computing resources.

Among the first approaches proposed that were capable of handling non-stationary environments are the *probability-vector* strategies [Bush & Estes 59]. In these the profit value measured after each trial is used to update an internal state vector that describes the probability that an activity will produce the most profit. Activities are chosen randomly, with the chance of being selected equal to the probability that it will be the best. Like the direct strategies probability vector strategies make no attempt to take account of the state of the world when they make a decision, and in practice they can be as good as or slightly better than the direct strategies. Their performance is better if they use a nonlinear update to the state vector [Thathachar & Sastry 85], to obtain a high convergence rate. However, obtaining good performance is dependent on getting the parameters, which control the change of the state vector, right. Like all of the other approaches to learning which can be applied to non-Bernoulli learning environments, each probability-vector strategy embodies a heuristic that tells it how to select activities. The better that heuristic is, the better the results, which can be further enhanced if more information is used. Probability-vector strategies do not increase their inertia as they perform more activities, and most do not have absorbing states, so they are suitable for use in the type of learning mechanisms described here, but there are other types of strategy which out perform them.

One of these is Sutton's *reinforcement-comparison method* (RC) [Sutton 84]. This, like the probability vector approaches uses an internal state which has a constant inertia with respect to the number of trials performed. This strategy however updates the internal state in proportion to the difference between the expected and the actual pay-off. Actions are selected by seeing if the internal state associated with each action,

perturbed by some random amount, exceeds a threshold. This perturbation allows exploration by an otherwise purely exploitative algorithm. Sutton has been able to perform extensive analysis on RC, showing that it is correct, capable of learning the optimal answer, and in a stationary environment is absolutely expedient [Sutton 84]. In experimental comparison reinforcement comparison methods perform as well as, or better than the best the direct or probability-vector strategies, with a faster convergence rate and lower regret [Kaelbling 93]. However, there is something even better.

It has already been suggested that if the lottery of an activity were known *a priori* a good strategy is to pick the activity with the highest expected profit. This value is not initially known, but if the upper confidence limit for that value is calculated from statistics of prior performances using:

$$\mu_a + t_p \frac{s}{\sqrt{n}} \tag{8.5}$$

where μ_a is the sample mean, s the sample standard deviation of the n profit value samples, and t_p is the value of Student T distribution at the set confidence level for $n-1$ degrees of freedom. If then the activity with the largest upper confidence limit is selected, an interesting thing happens. Either the activity with the actual highest expected payoff or the activity for which the mechanism is least certain of its expected payoff is selected.

This is an elegant handling of the explore/exploit dilemma coupled with an implementation of a near optimal learning time decision making strategy for robotics. In relying on the environment to indicate when the system does not know enough to make a decision it removes the risk that exists with other systems, *e.g.* RC, that depend on chance to decide when to explore, that it might be necessary to wait a long time before gaining enough information to be able to act in the best way. It is clear from Equation 8.5 that as the number of samples grows then for a constant confidence level the difference between the upper confidence limit and the true mean will decrease. This convergence to the true mean can be formally proved as a consequence of the Central Limit Theorem, providing that in the limit the variation about the true mean converges to the normal distribution [Whittle 92]. This approach is computationally simple, having bounded

time and memory requirements. It is also guaranteed to be correct, and to learn the optimal answer as defined by the profit function that is being used.

Kaelbling has developed a decision making mechanism that uses confidence limits, IE which in her comparative tests performs as well as, if not usually better, than any of the other methods mentioned here [Kaelbling 93]. However, it cannot be used here as it was designed for a stationary, binary valued payoff environment. Salganicoff and Ungar have used IE and ID-3 [Quinlan 86] in a system which learns the best orientation and elevation angles to grasp a variety of objects [Salganicoff *et al.* 94]. The latter algorithm is used to associate a vector of object properties with a number of grasp parameterisations, with IE then used to select the best of the contending grasp parameterisations. Without this last stage system performance declines from having a 75.8% to a 38.83% chance of success [Salganicoff *et al.* 94]. Their use of IE is identical to what is being proposing here: learning which applicable activity is best to achieve a specific goal. The main thrust of their work is to learn the association between an activity and preconditions with IE used to enhance learning performance, and they make no attempt to increase the robustness of IE with respect to change.

The rest of this section describes the development of IE into a mechanism which meets all the criteria given in the first section of this chapter, and in the next section provide some verifying experimental results.

8.2.2 Details and Development of the Algorithm

The process of changing of IE into ANNIE had three stages:-

1. The basic algorithm was modified to use real profit values.
2. The modified algorithm was extended to allow it to operate in a non-stationary environment.
3. The facility was introduced to allow the user to be able to select which activities were candidates at each selection.

The resulting algorithm was then wrapped in a loop that once entered will force repeated selections to be made until either an activity succeeds, or there are no untried

Algorithm 6 ANNIE

```
1:  x = 0
2:  x2 = 0
3:  n = 0
4:  c = RATIO_LIMIT
5:  p = DEFAULT_PROBABILITY
6:  repeat
7:    a=select(s)
8:    if a ≠ failed then
9:      r = evaluate(a)
10:     s = update(s, a, r)
11:     if action_succeeded(a) =false then
12:       inadmissible(a)
13:     end if
14:   end if
15: until action_succeeded(a) ∨ num_candidates = 0
```

candidate activities. The result was ANNIE, which is fully described in Algorithm 6.

The various functions and variables used within Algorithm 6 are:-

evaluate(): is a function used to evaluate, *i.e.* call the selected activity. It returns the results of the evaluation.

select(s): is used to return the candidate activity that should be evaluated next. The function takes in the current and returns the index number of the selected activity. From within *select*, two other functions are called:-

random_choose is a function which picks one of the activities presented and returns the index to that option. The selection is performed randomly, with each activity having an equal chance of selection.

max is a function which returns the indices of those admissible actions which have the largest, or equal to the largest, upper confidence limit.

update(s, a, r): updates the state information following the evaluation of the outcome of an experiment. The update function is detailed in Algorithm 8 on page 193. Within this algorithm are three mathematical functions, the definitions of which are:

$s(n, x, x2) \equiv \sqrt{\frac{n.x2-x^2}{n(n-1)}}$

$nub(n, p, x, x2) \equiv \frac{x}{n} + t_p^{(n-1)} \frac{s(n,x,x2)}{\sqrt{n}}$

$t_p^{(n-1)} \equiv$ Student's t value for n-1 degrees of freedom at the p confidence level

admissible is a function that returns true if the activity being tested is one that has been designated as a usable candidate in this selection process. Reasons that an activity might be excluded are that it might not be suitable for the current task, or because that activity has failed in a way which might indicate equipment problems, in which case that activity is useless until the equipment is repaired.

inadmissible is a function which removes the specified activity from the list of candidates under selection.

action_succeeded is a function that returns true if the selected activity achieves its goal, otherwise it returns false. This can be implemented by comparing the profit function, or the cost function, or some element of the cost function against a specified target, or the activity itself might explicitly report its success or failure.

num_candidates is a function to count the number of candidate activities for consideration on the next inner loop of ANNIE.

s: is a state description s that is maintained for each activity in the candidate pool. It comprises the sum of the previous profit values(x), the sum of the previous profit values squared (x^2), the number of times an activity has been called (n), the number of decisions that have taken place since an activity was last used (c) and the confidence level for that activity (p). The state is updated using the *update* function, whilst the *select* function returns the index number of the best activity.

a: is a variable used to hold the index number of the activity currently being evaluated by the algorithm. The mapping between index numbers and activities is arbitrary, but must remain constant, otherwise it is necessary to reinitialise the mechanism. By using index numbers the mechanism is prevented from knowing anything of the details of the candidate activities.

Algorithm 7 The Select Function

select(**s**)
if $\exists a : n_a < 2 \vee c_a = 0$ **then**
  random_choose($\{a | admissible(a) \wedge (n_a < 2 \vee c_a = 0)\}$)
else if *admissible*(a) **then**
  random_choose($\{a | admissible(a) \wedge max(nub(n_a, p_a, x_a, x2_a))\}$)
else
  fail
end if

Algorithm 8 The Update Function

update(**s**, a, r)
$x_a := x_a + r$
$x2_a := x2_a + r^2$
$n_a := n_a + 1$
for all $i | i \neq a$ **do**
  $c_i := c_i - 1$
  $p_i := (p_i + DECAY * (1 - p_i))$
end for
$c_a := RATIO\_LIMIT$
$p_a := DEFAULT\_PROBABILITY$

r: is a variable used to hold the result of the evaluation of an activity. This is a value between $[-1, 1]$, as noted in Section 3.2, page 51.

As noted in Section 4.3.3, page 86, ANNIE has different control parameters to the other activities:-

RATIO_LIMIT controls the frequency with which activities are tested. If the number of decisions since an activity was last selected exceeds this, then if that activity is admissible, it will be chosen. Although not stated, if this value is set to less than or equal to zero the forcing mechanism is disabled.

DEFAULT_PROBABILITY is the initial confidence level that the true mean is less than the upper confidence limit.

DECAY is the amount to reduce the current difference between the confidence level of the true mean of an activity being less than the upper confidence limit and the maximum confidence level. As implemented this decay to the highest confidence level is exponential for reasons described later.

The parameters described in Section 4.3.3, on page 86, are still required by the candidate activities.

ANNIE can be, and has been, implemented as a re-entrant procedure with the state and control data and the index maps for each application supplied as parameters. This provides for economies of space as only a single copy of the implementation is needed, unless parallel decision making is required. This property is one that was stated in Section 3.3, page 56, as being desirable because of the reduced space requirement and simpler system development it implies. If parallel decision making is required then the appropriate number of copies of the same procedure, parameterised appropriately, can be used.

ANNIE has a two stage initialisation process. Before the mechanism can be used its state must be reset to the initial value. This has to be done when the system is powered up for the first time, but should not have to be done subsequently unless the system is powered down without saving the current state. The user can reset the algorithm when the system is applied to a novel task in which the accrued expertise of ANNIE is considered an encumbrance. This would be the case only when the task is so very different that until the mechanism had learned which activity was now best, performance would be very poor. Following initialisation each activity must be attempted twice to provide enough results for the upper confidence limit to be calculated. This a consequence of using an estimate of the standard deviation in the calculation which cannot be determined from fewer than two results. This stage of the initialisation is commanded internally from within the implementation of ANNIE. If new activities are added on the fly, or existing ones modified and reintroduced then once their state vectors have been initialised ,the mechanism will perform the two essential selections immediately. This ability to change the pool of candidate actions on the fly is a necessary part of a flexible system as new abilities can be introduced and evaluated without removing the system from profitable use.

Admissibility

When a decision is required, ANNIE must be supplied with information about which activities are the candidates for selection. The mechanism then selects the best, and

initiates it. If the activity fails ANNIE will remove that activity from the candidate pool and select the next best, repeating the process until it exhausts the pool or finds a successful activity. Here the initial information on which activities are admissible is that all candidate activities are usable. If all the activities of a system are truly independent then this approach will not be a problem. For dextrous manipulation this is rarely the case, as has been noted in Section 8.1 on page 182.

The admissibility mechanism is a device to allow the candidate pool to be dynamically changed through the inclusion or exclusion of activities by an activity external to ANNIE. Three examples of how the mechanism might be used are:

- An external planning agent might change the pool to exclude activities, that although might be locally suitable, in the long term would result in a reduced performance over the whole task. This might be implemented using a scheduling system such as that used in SOMASS [Malcolm 87]. If coupled with a model of the interactions of activities in relation to the task at hand the agent could enforce global optimality over the system, and it could do so before any single activity had been performed.

- A developer might wish to evaluate a new activity and so wants to ensure that it is used by the system, regardless of its performance.

- A system supervisor (human or automatic) might deselect an otherwise perfectly acceptable activity to allow for maintenance.

- A system supervisor (human or automatic) might detect that an activity is repeatedly failing, indicating some problem requiring attention. This might be detected by setting an upper limit on the length of run of failures, or by calculating and monitoring a confidence level. Once categorised as requiring attention, the problem activity could be removed from the candidate pool to prevent loss of performance.

All of these processes require some kind of external decision maker. However, these can be specialised and tailored to the situation and activity. This would add to the development work, but this might be automated. None of these mechanisms have

been addressed here as they are not central to the topic under discussion. That this mechanism supports these is a point in its favour and an avenue for future research.

Summary

In this technical description of ANNIE it has been pointed out how and why the mechanism has the properties that were laid down as being desirable. On its own, this description is not sufficient to allow full confidence in the mechanism. That can only be achieved by experimental verification, which is provided in the next section.

8.3 Evaluating ANNIE

This section describes two experiments that were performed to test the ability of the mechanism to work in real situations:-

1. An evaluation of ANNIE and a comparison of the forcing and the decay mechanisms.
2. An assessment of how the performance of ANNIE changes when the candidate pool increases in size.

In the first experiment the candidate activities were the Visual QUEASY and Visual I_INTERP activities. In the second experiment Visual P_MOVE was added to the candidate pool. The goal for the mechanism in each experiment was to pick the activity that was best at moving the robot arm to a selected point. Each experiment was repeated five times, and in each repeat 50 trials were performed, with the same profit function used in both experiments to assess the decisions. This profit function is described in Section 8.3.1, page 197.

During a repeat of an experiment a record was kept for each trial of which activity was chosen, its costs, its profit value, the upper confidence limits at the time of selection and the number of iterations the choosing algorithm had to go through to find an activity that succeeded. The resulting data files were evaluated to produce the action, confidence and profit plots that are shown in each of the experimental write ups. Prior

to these experiments the performance results of the various reaching modules were used to estimate the likely outcome of selection by calculating the expected profit values of the different activities. These are tabulated in Table 8.1. Comparison of the actual choices to this result, and the performance with respect to the criteria given in Section 8.1 on page 182 will form the basis of the assessment of how good ANNIE is in practice.

8.3.1 The Profit Function

From the ideal performance of a reaching activity that was defined in Section 4.3.1 (page 84) a set of metrics has been used in this thesis to analyse and compare the different reaching activities. These have been defined in Section 4.4.1 on page 90. These metrics are in turn based on four measurements that are collected at the end of each reach. It is these measurements that will be used to determine the profit function for ANNIE:-

- The final Euclidean distance between the final observed position of the robot and the target position.
- The efficiency of the reaching, calculated as

$$\frac{\sqrt{(\vec{r}_{final} - \vec{r}_{initial})^2}}{\sum_{i=1}^{n} \sqrt{(\vec{r}_i - \vec{r}_{i-1})^2}} \tag{8.6}$$

  where $\vec{r_{final}}$ is the robot's final location, $\vec{r_{initial}} = \vec{r_0}$ is the initial point of the robot, and $\vec{r_i}$ is the robot's location after iteration step i. This measure tends to 1 as the route taken by the robot approaches a straight line.
- The number of moves taken to reach the target.
- The time taken to reach the target.

These measurements have been converted into costs (except for the efficiency measurement) by first, dividing the value returned following a reach by the maximum possible value. The efficiency measurement is converted into a cost by subtracting the returned

value from 1. The modified costs are then squared. This process is encoded in the following cost function:

$$f^s = (\frac{error}{max\_error})^2 + (1 - efficiency)^2 + (\frac{time}{max\_time})^2 + (\frac{steps}{max\_steps})^2 \quad (8.7)$$

The justifications for using these measurements are:

Position Error: The further away the gripper is from the target at the end of a reach, then the harder it will be to perform any activity about that target point.

Efficiency: This is included to force the system to pick the reaching method that makes the most direct moves to the target, so even if it makes a lot of moves, the arm will have been driven through excessive displacements en route.

Iterations: Although the efficiency element should ensure the mechanism that chooses the most direct route is selected, this element is aimed at finding the activity which requires fewest iterations to achieve the goal. This partially overlaps the efficiency and the time costs (next).

Time: Including the time taken into the cost calculation will encourage selection of the activity which completes in the shortest time, improving the efficiency of the system as a whole. However, there might be a conflict between this cost and the efficiency and iterations costs.

Applying the profit function resulting from combining Equation 8.7 with Algorithm 1 (page 56) to the data produced during the various proving experiments for the Visual QUEASY, Visual P_MOVE and Visual I_INTERP reaching activities presented in Section 5.1.6 (page 105), Section 6.4 (page 136), and Section 7.4 (page 167), produces the expected payoffs for success of each activity shown in Table 8.1.

The values in this table will act as comparative benchmarks against which the decisions of ANNIE can be compared. From this table a qualitative prediction of ANNIE's performance can be made: the mechanism should select Visual I_INTERP most frequently, followed by Visual CAUTIOUS_QUEASY and then Visual P_MOVE as this is the ordering given by the profit function defined here. On to the experiments.

Module	Expected Payoff
QUEASYC	0.85
P_MOVE	0.82
I_INTERP	0.95

Table 8.1: Expected payoffs for success

8.3.2 Choosing between 2 activities —- Servo or Interpolation

The experiment described here considers which of the two variants of ANNIE is best to use to decide which of Visual CAUTIOUS_QUEASY or the Visual I_INTERP reaching mechanisms is best under the profit function described above. The test of each variant was repeated five times, each test having 50 trials. In each repeat a different sequence of target positions was used. The same target positions are used to test both variations. These experiments will show ANNIE's ability to choose the best activity at any one time and also show how ANNIE copes with activities whose performances gradually improve.

In each test the initial mapping between the vision system and arm locations used by Visual I_INTERP is empty. As more samples are acquired then Visual I_INTERP will become more and more useful. If ANNIE is able to change its patterns of decisions in response to changes in the relative abilities of the activities, then a more frequent use of Visual I_INTERP should be evident.

In the first experiment $DECAY$ is set to 0 and the forcing interval to 5; in the second $DECAY$ is set to 0.3 and the forcing to 50. Doing this will allow comparison of the two different methods for ensuring that ANNIE will not ignore a good activity by chance.

The reinforcement values, confidence limits and action plots from the five repeats of two experiments are shown respectively in Figure 8.2, Figure 8.3 and Figure 8.4. In each figure identical repeats of the experiment are horizontally paired, with the first repeat at the top of the page and the fifth at the bottom.

The results indicate that the mean of the decay repeats is generally less than or equal to the mean of the forcing repeats, though the smaller standard deviations of the profit of the decay variant repeats, shown in Table 8.2, suggests that it is more consistent. The reason for the lower mean values can be explained with the plots of profit value per time step in Figure 8.2, where there are more failures in each decay experiment than

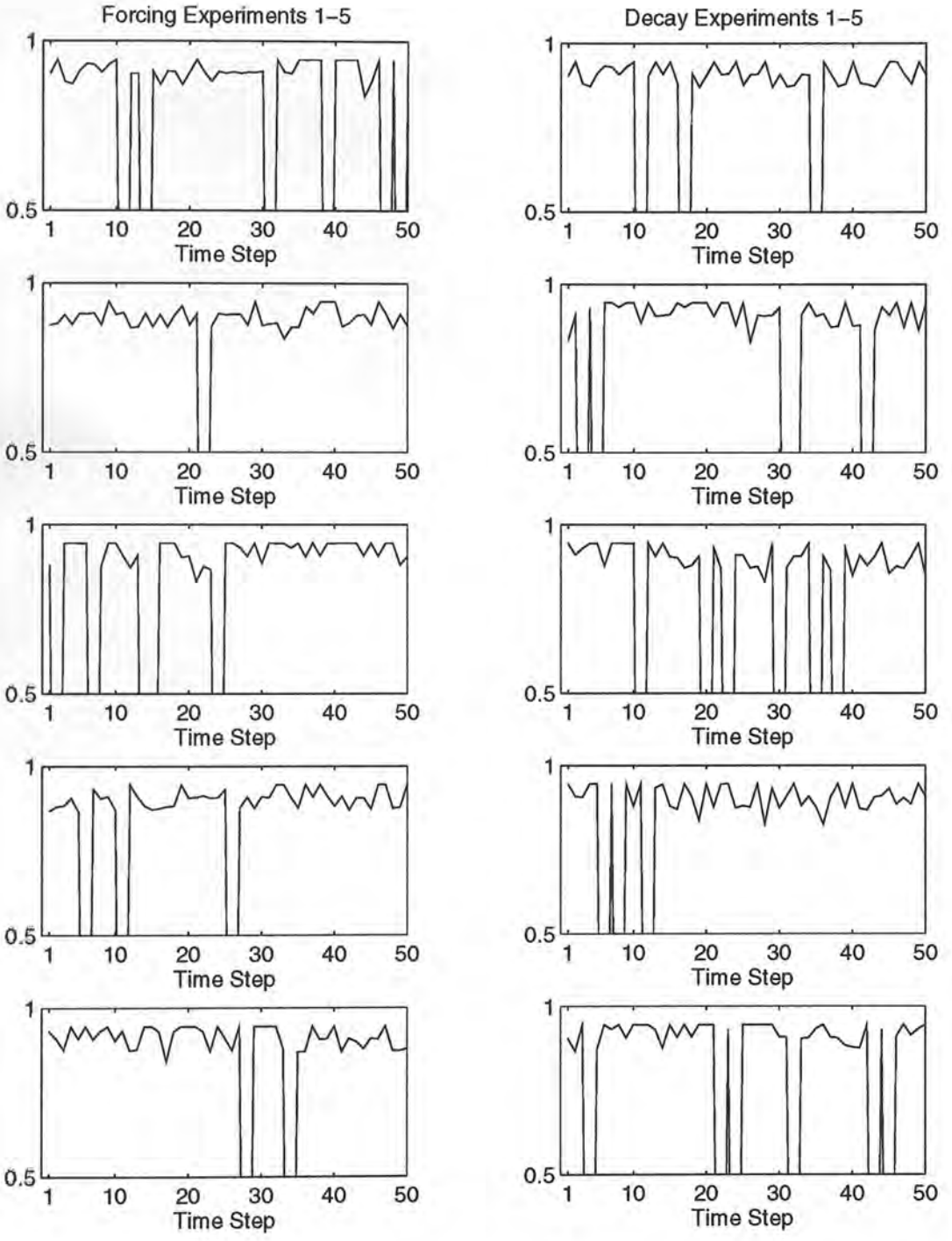

Figure 8.2: Profit values for 2 candidate Visual reaching

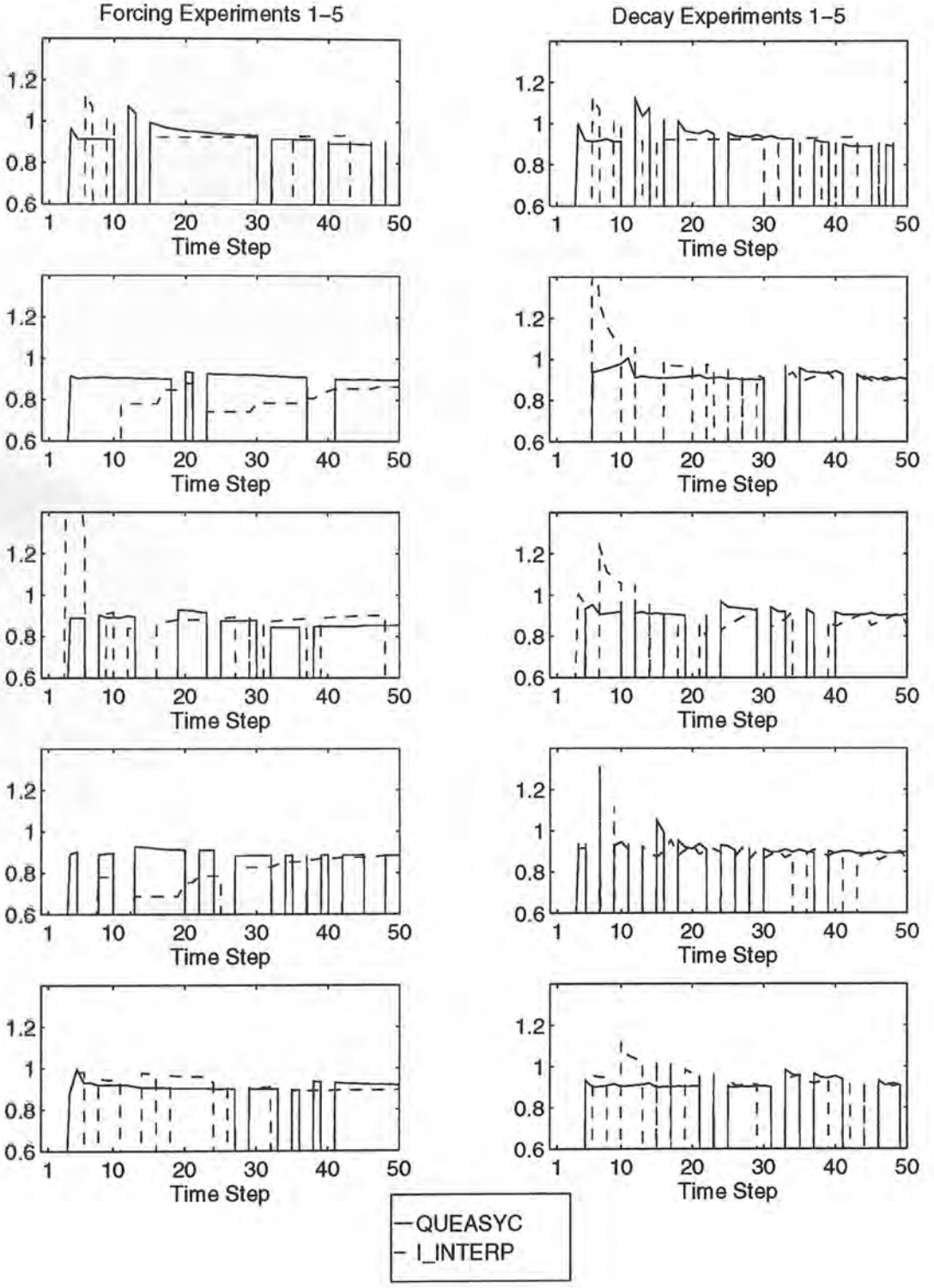

Figure 8.3: Upper Confidence Limits for 2 candidate Visual reaching

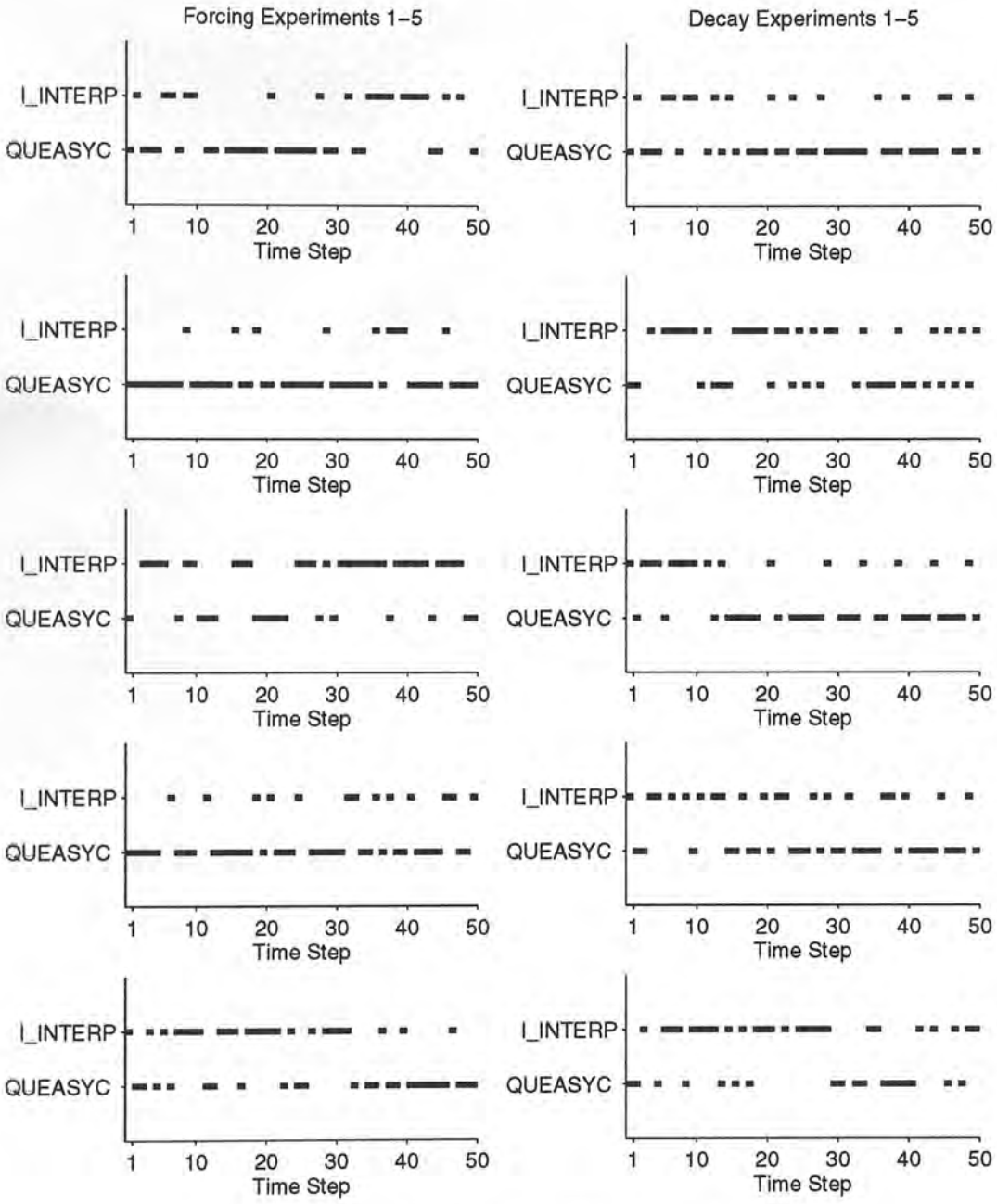

Figure 8.4: Action plots for 2 candidate Visual reaching

	Forcing Experiment		Decay Experiment	
Experiment	μ_f	σ_f	μ_d	σ_d
1	0.69	0.63	0.80	0.25
2	0.86	0.27	0.72	0.18
3	0.73	0.58	0.68	0.32
4	0.79	0.46	0.79	0.25
5	0.84	0.38	0.69	0.23

Table 8.2: Statistics of profit values produced by Forcing and Decay Variants of ANNIE

there are in the equivalent forcing experiment. It is these failures that have reduced the mean considerably. From this, one possible observation is that with an equal number of failures, it could be expected that the decay variant would outperform the forcing variant of ANNIE. The plots of the upper confidence limit on the estimated mean shown in Figure 8.3 reinforce this view.

The overall results of using either variant are far worse than the values predicted from using just one of the two activities. If the number of failures occurring with both variants are taken into account, then the difference is not surprising, as these failures reduce the overall performance averaged across the different trials. The prediction that was made from Table 8.1 about the relative frequencies of selection has turned out to be true, as Visual I_INTERP is selected more frequently, though only as the number or results entered into memory rises.

If the repeats of the variants are compared statistically with the other repeats of that variant using two sided t-test and F-test, then the results are as shown in Table 8.3 and Table 8.4. In each table the numbers indicate the likelihood that the profit values of one repeat of the variant is similar to another repeat of the same variant, with values above 0.95 indicating significant differences. These tables provide quantitative confirmation of the observation made from Table 8.2 that the decay variant is more consistent. If the repeats of each variant are compared against all the repeats of the other variant this time using one-sided t-test and F-tests, the results produced are as shown in Table 8.5. In this table the numbers indicate the chance that the mean of the decay variant is greater than the mean of the forcing variant, or that the variance of the decay variant is less than that of the forcing variant. Whilst only a value of 0.95 or greater can be taken as indicating significant difference between two repeats, given

Forcing Experiment Number	Forcing Experiment Number									
	t Test					F Test				
	1	2	3	4	5	1	2	3	4	5
1	—	0.92	0.28	0.65	0.84	—	0.99	0.40	0.97	0.99
2	0.92	—	0.83	0.64	0.27	0.99	—	0.99	0.99	0.99
3	0.28	0.83	—	0.42	0.72	0.40	0.99	—	0.90	0.99
4	0.65	0.64	0.42	—	0.42	0.97	0.99	0.90	—	0.81
5	0.84	0.27	0.72	0.42	—	0.99	0.99	0.99	0.81	—

Table 8.3: Results of cross comparing forcing using Pooled t Values and F values

Decay Experiment Number	Decay Experiment Number									
	t Test					F Test				
	1	2	3	4	5	1	2	3	4	5
1	—	0.51	0.71	0.21	0.64	—	0.89	0.97	0.01	0.99
2	0.51	—	0.29	0.49	0.19	0.89	—	0.41	0.89	0.44
3	0.71	0.29	—	0.96	0.09	0.97	0.41	—	0.97	0.04
4	0.21	0.49	0.70	—	0.63	0.01	0.89	0.97	—	0.98
5	0.64	0.19	0.09	0.63	—	0.99	0.44	0.04	0.98	—

Table 8.4: Results of cross comparing decay using Pooled t Values and F values

Decay Experiment Number	Forcing Experiment Number									
	t Test					F Test				
	1	2	3	4	5	1	2	3	4	5
1	0.83	0.80	0.27	0.52	0.68	0.99	0.99	0.95	0.50	0.91
2	0.62	0.93	0.52	0.73	0.87	0.71	0.99	0.51	0.96	0.99
3	0.52	0.96	0.67	0.84	0.93	0.51	0.99	0.69	0.98	0.99
4	0.83	0.81	0.72	0.51	0.69	0.98	0.99	0.95	0.50	0.90
5	0.52	0.96	0.62	0.81	0.92	0.51	0.99	0.71	0.98	0.99

Table 8.5: Pooled t Values and F values for Forcing versus Decay

that most of the values for the mean and all of the values for variance are greater than 0.5, it would be fair to state that it is more likely, given this evidence, that the decay variant is better than the forcing variant.

These two tests have together shown three things:-

- ANNIE can learn which activity is best.
- ANNIE can cope with a non-stationary world, as in most of the experiments the pattern of selection changes for the better as Visual I_INTERP reaching becomes better.

- That using a decay rather than a forcing mechanism to encourage periodic re-sampling is more effective in that it produces higher rewards, as well as being application independent.

8.3.3 A Test of Scaling Ability

The experiment described here has been designed to explore the algorithm's abilities to handle choosing between more than two choices. If the performance, in terms of the time before the mechanism selects the best activity, of ANNIE does not change linearly with respect to an increase in the size of the candidate pool, then using ANNIE to make decisions will force a change in the activities used in a system, as explained in Section 3.3.2, page 60.

The evidence for ANNIE not being able to scale linearly would be significantly more samples than expected being taken before a correct decision was made. For, say three candidate activities, then if ANNIE scales linearly, the time taken before a clear decision is reached about which is the best observable convergence performance should be no longer than one and a half times the time required for a two candidate pool. From inspecting Algorithm 6 on page 191, it is clear that for three activities the initialisation period will be longer but after this the time taken by the algorithm to decide which activity to use should be only one and a half times the period required to decide between two activities. What is not clear from the algorithm is how having three different activities, one of which will be improving over time, will interact in terms of their relative profits and upper confidence limits.

The method used to get the data necessary to decide if the scaling is linear is identical to that used to test the decay variant in the last experiment, except there are three candidate activities: Visual CAUTIOUS_QUEASY, Visual I_INTERP and Visual P_MOVE. As before the experiment is repeated five times, each repeat using an identical sequence of target positions to that used in the repeats of the experiments assessing whether decay or forcing was better. Like in the last experiment the memory of Visual I_INTERP was initially empty.

The reinforcement values, confidence limits and action plots from the five repeats of

Experiment	μ_p	σ_p
1	0.94	0.05
2	0.93	0.06
3	0.92	0.06
4	0.94	0.05
5	0.92	0.06

Table 8.6: Mean and Standard Deviation of profit values for 3 candidate Visual reaching

the experiment are shown in Figure 8.5, Figure 8.6 and Figure 8.7. As with the plots for the 2 candidate experiments, gaps in the action plots, with corresponding low profit and confidence limit values are indicative of all the candidate activities having failed on that trial.

A visual comparison of Figure 8.4 and Figure 8.7 suggests that in these experiments the decision making performance of ANNIE scales linearly. In Figure 8.4 approximately 10 steps are required before the algorithm begins to favour the selection of one activity over another, whereas in Figure 8.7 about 15 steps are required.

The higher, and more consistent profit values obtained using a three candidate pool that are shown in Table 8.6 are a consequence of the greater reliability of the system, and through that greater reliability, more entries being made into the memory of Visual I_INTERP, which improves its performance faster than in the 2 pool experiments. These values are closer to the expected profit values given in Table 8.2, though the prediction made on page 203 that Visual P_MOVE would be selected less frequently than Visual CAUTIOUS_QUEASY is only true for the fifth repeat of this experiment. If the Euclidean and Mahalanobis distances of Visual P_MOVE and CAUTIOUS_QUEASY, given in Table 6.6 on page 146, Table 5.11 and Table 5.12 on page 114 are compared, it can be seen that although Visual P_MOVE's Euclidean distance is smaller than that of VisualCAUTIOUS_QUEASY, its Mahalanobis distance is larger, a contradiction that indicates that the comparative performance of these two mechanisms are application dependent.

The observation that using a three candidate pool gives a more consistent performance is supported by the quantitative results of cross comparing the profit values produced on each trial in one repeat with those produced in the other repeats which are shown in Table 8.7. Only the fourth and fifth repeats have significantly different mean results,

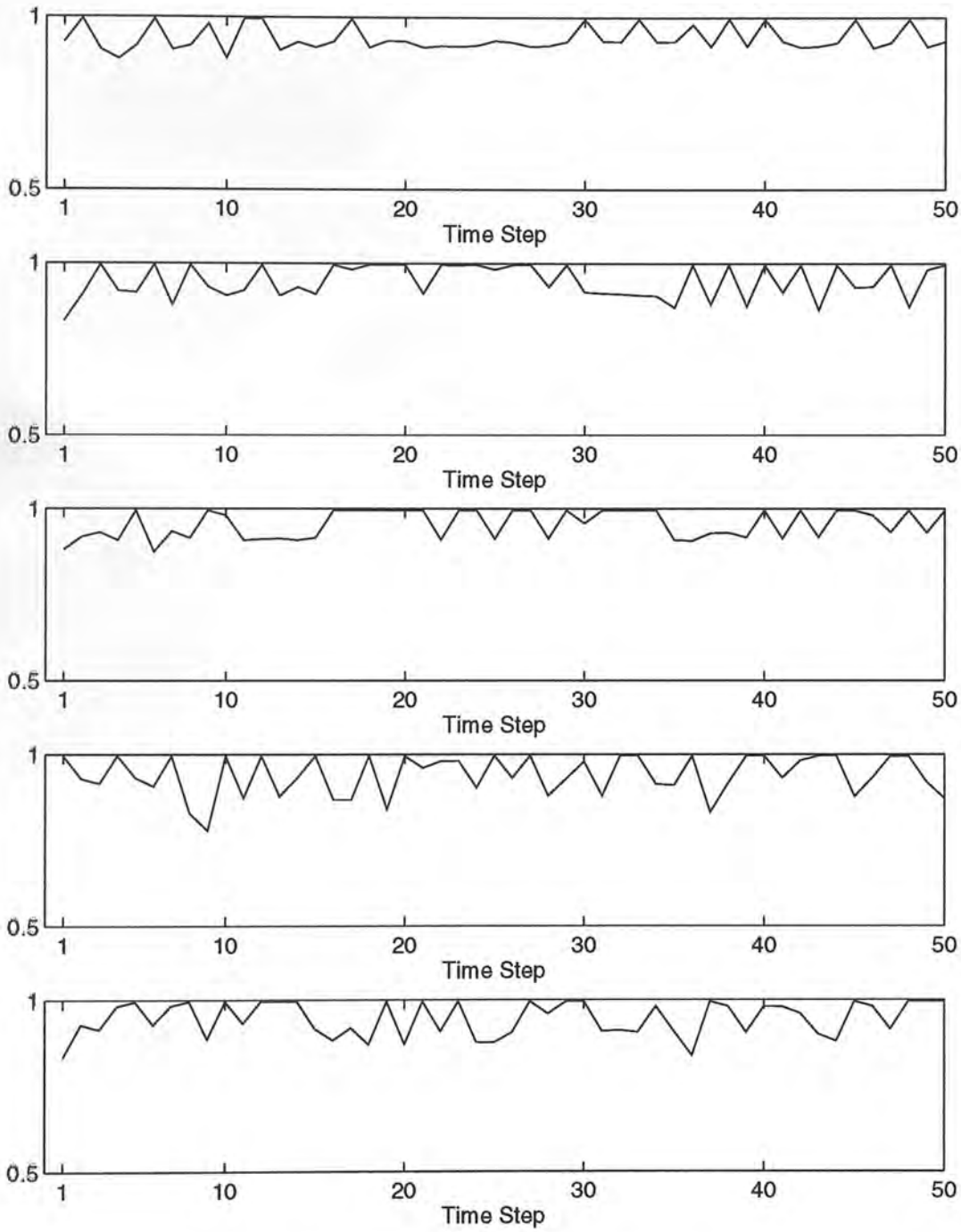

Figure 8.5: Profit Values for 3 candidate Visual reaching

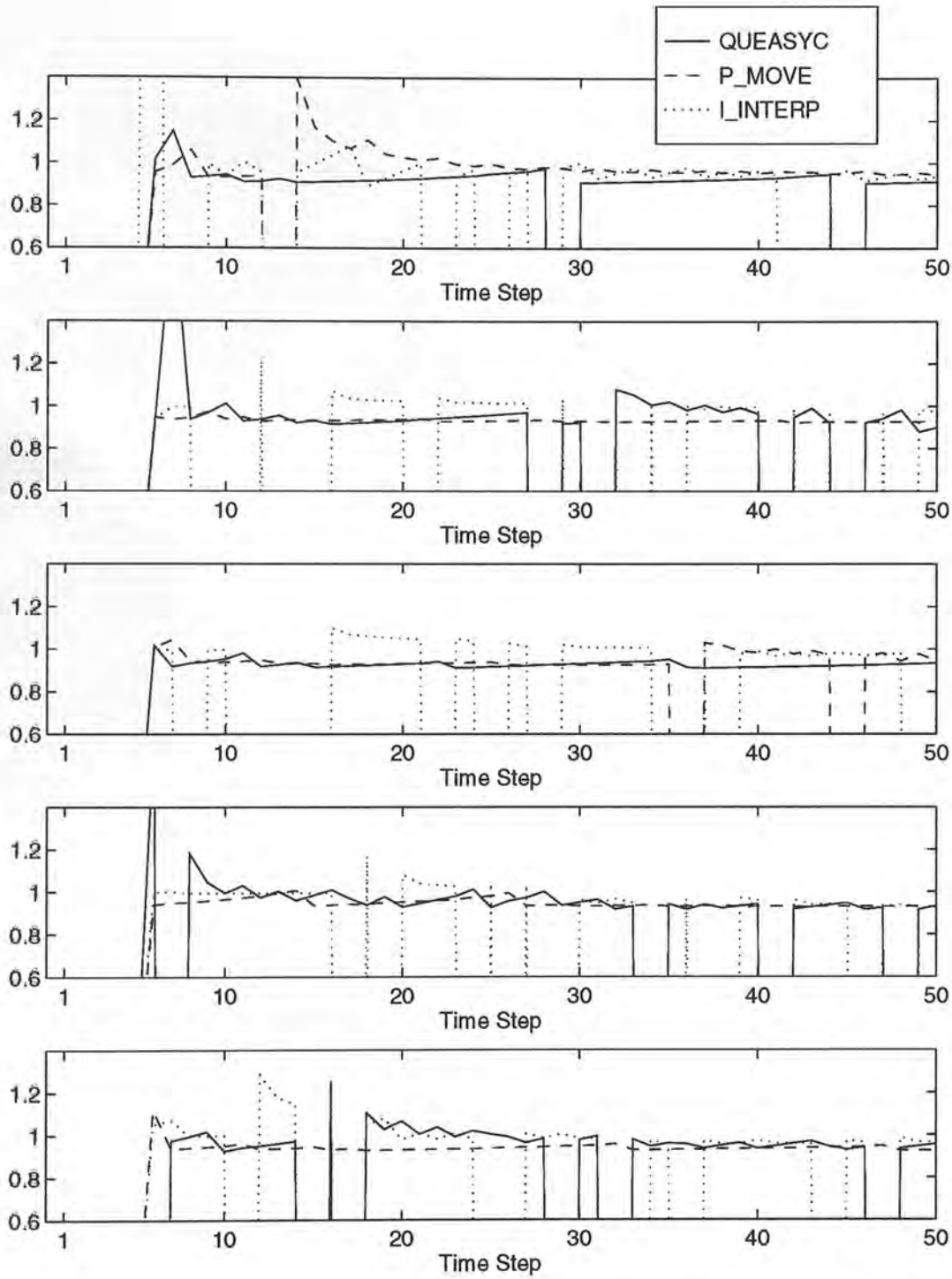

Figure 8.6: Upper Confidence Limits for 3 candidate Visual reaching

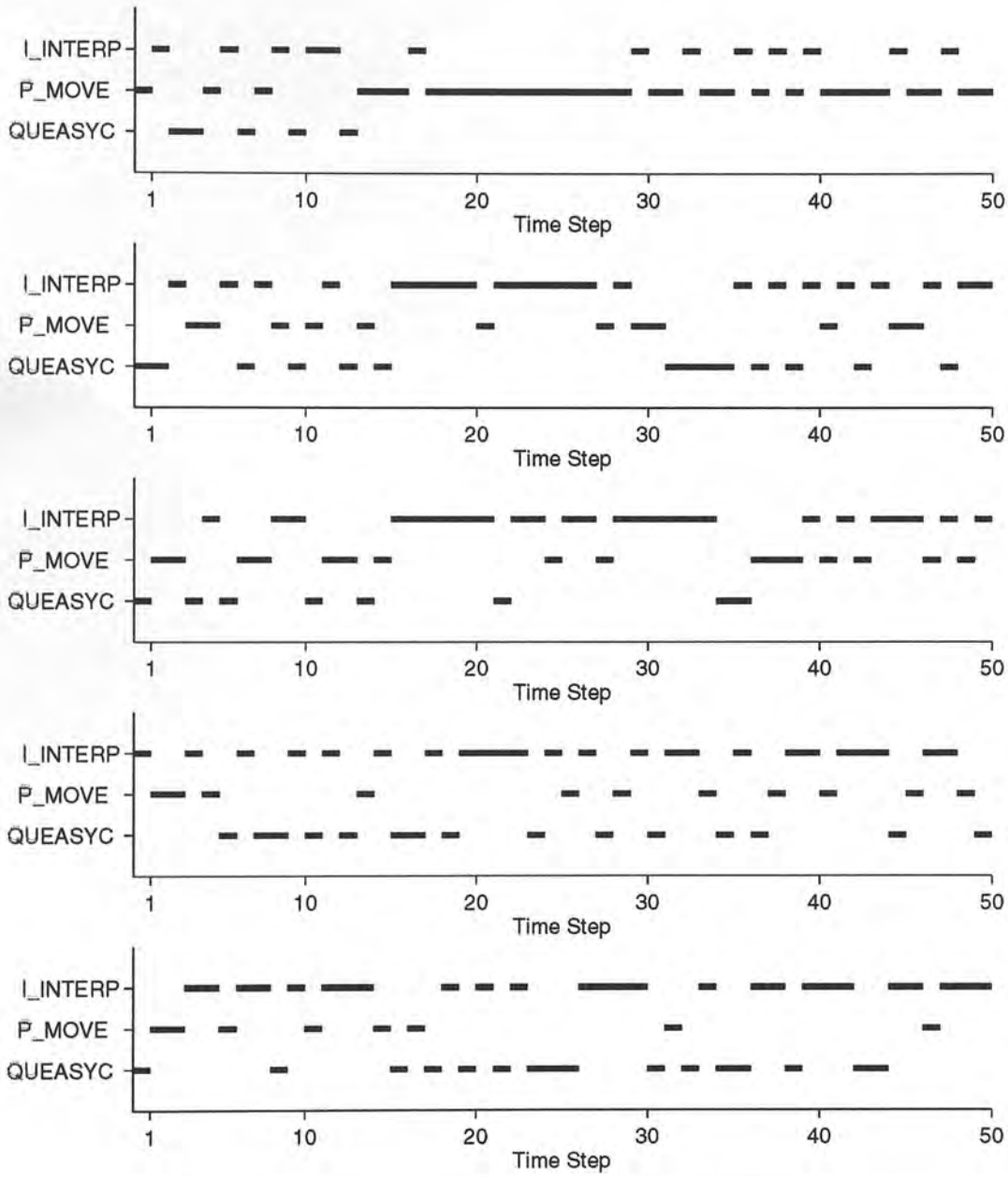

Figure 8.7: Action Plot for 3 candidate Visual Reaching

Experiment	Experiment									
	t Test					F Test				
	1	2	3	4	5	1	2	3	4	5
1	—	0.27	0.78	0.34	0.88	—	0.62	0.46	0.02	0.56
2	0.27	—	0.59	0.55	0.74	0.62	—	0.21	0.61	0.09
3	0.78	0.59	—	0.91	0.25	0.46	0.21	—	0.44	0.13
4	0.34	0.55	0.91	—	0.95	0.02	0.61	0.44	—	0.55
5	0.88	0.74	0.25	0.95	—	0.56	0.09	0.13	0.55	—

Table 8.7: Results of cross comparing using Pooled t Values and F values

though only just exceeding the 95% confidence level. Again, this is a reflection of the greater reliability provided by a three candidate pool evidenced by the lack of failures on any trial of any repeat, though individual activities did fail on separate trials.

8.3.4 Summary

From these experiments two conclusions can be drawn. Firstly, ANNIE works, and does so effectively, and secondly, those properties (specified in Section 3.3, page 56 and Section 8.1, page 182) that were attributed to the mechanism by virtue of its design can be shown to be present in practice. In short, the claim that ANNIE can automate the identification of the best activity to select without knowing the details of the candidate activities, or the task, is valid. It can only do this because of the profit function which has to be supplied by the developer. Producing that is not easy, but the formulation that has been using (described in Section 3.2, page 51) seems to provide an effective handle on to this problem.

8.4 Summary of Chapter

This penultimate chapter has contained the theory, description and verification of a novel mechanism called ANNIE. This has been derived from Kaelbling's IE algorithm [Kaelbling 93] and can learn to choose which activity in a pool of comparable activities is best. It does this by collecting and maintaining statistics on performance from which the upper confidence limit on the estimated profit values can be calculated. The selected activity is the one which has the highest value. This approach is provably optimally correct and has a near optimal learning time. Importantly, it can scale

linearly with respect to the number of candidate activities. In making decisions on evaluated performance alone ANNIE does not require any detail about the activities, or the activity it which it is embedded, a goal which has been constantly aimed for.

ANNIE is last of the three different learning mechanisms that was identified as being useful in activities in Section 3.1 (page 45) on the grounds that they can automate the parts of the development of a robot system that would otherwise require possibly a great deal of effort from the developer, and at the same time they can improve the reliability and flexibility of the system. The next, concluding, chapter summarises the various developments described in this thesis and presents some suggestions for future research into the ideas that have driven this work.

Chapter 9

Conclusion

The experimental work described in the latter half of this thesis contains five technical innovations:-

- The Visual and Force QUEASY activities (Section 5.1, page 95, Section 5.2, page 115).
- The Visual and Force P_MOVE activities (Section 6.4, page 136, Section 6.5, page 146).
- The AM concept (Section 7.2, page 157).
- The Visual and Force I_INTERP activities (Section 7.3, page 164, Section 7.5, page 171).
- The ANNIE action selection algorithm (Section 8.2, page 184).

These innovations are answers to the question posed on page 6 about how learning can be used to improve the performance of a behavioural assembly robotic system whilst simplifying its development. Individually, each of the answers is good (see Section 9.1, page 215 for details). Each increases the robustness of the system, though not all of them reduce the workload of the developer.

Their development began from matching what goes on in a system controller with the kinds of thing weak learning mechanisms are good at: learning optimal parameter settings, learning action-sensor couplings and learning which activity is best. These

kinds of learning all automate some of the test and analyse cycles that are an essential part of the development of computer programs that control systems which cannot be completely modelled, or even fully defined in the program. The remaining parts of such systems are determined by the environment, the mechatronics of the system and the task to which it is being applied. The mechatronics of the system also determines how the activities of the system have to be implemented. The answers listed above were developed in the listed sequence from this starting point.

Each of the first four steps along the sequence resulted in an activity that required less knowledge about the specifics of the task to which it was applied as it had become more general. This was because the reduction in built in knowledge was matched by an increase in the approximating power of the learning mechanism so that by the time I_INTERP was produced, it was a generic activity (the active memory mechanism) that could be used to instantiate any action-sensor coupling. This generality of I_INTERP is dependent on a teacher to initiate I_INTERP, *e.g.* P_MOVE. Unlike the weak EBL mechanisms to which I_INTERP is related, this teacher is within the system nor is I_INTERP restricted to simple repetition of the demonstrated tasks.

The final learning mechanism developed was the ANNIE activity selection mechanism. This was initially identified back in the list on page 45 in Section 3.1. The need for it was made concrete at the end of the description of the experimental results of I_INTERP on page 174 in Section 7.5. There it was noted that the performance of I_INTERP was only acceptable after sufficient results had been entered into memory. It was also noted that as the number of samples entered into memory increased, the performance improved rapidly. The ANNIE mechanism learns to select the best activity and can handle the non-stationary world that results from incorporating mechanisms such as I_INTERP into the controller. ANNIE has the same requirement for knowing about what it is selecting as active memory does about the relationship between robot and sensor, *i.e.* none.

There are clear benefits to using capable activities that have little or no built in information concerning their application such as P_MOVE, or I_INTERP or ANNIE, and that only need appropriate linkages and profit functions to be specified to be used in system controllers that have to handle complex tasks such as assembly. For such systems

failure can have a very high cost there is a risk that any failure will cause significant damage the robot, or its environment. As the task, or the environment, as perceived by the robot[1] becomes more complex, the controller needs to become more structured to provide the necessary error recovery abilities and to include some form of reasoning ability to handle activity scheduling on any given task. When introducing learning into such controllers work on complex tasks, of which reaching with a kinematically redundant robot is an often overlooked example, it should support the controller rather than dictate how that controller should be constructed without being dependent on that construction. This is something that has been demonstrated with the weak learning mechanisms that can be slotted into a controller whenever the developer would otherwise have to perform and analyse experiments to determine the best parameterisation or activity to use.

The result of all of this work can be summarised into a statement that expresses the single most significant contribution made here:

> Appropriate weak learning mechanisms appear to be useful components of the system controller of an assembly robot system.

Here appropriate means a mechanism that requires the minimum of pre-specified knowledge to perform its task, instead using that available in the profit function and in its history to determine the best thing to do. This statement is the answer to the 'why' question that has defined this whole thesis that was originally given on page 6. The 'how' part of this question has been addressed through the various learning mechanisms that have been developed.

These learning mechanisms are not silver bullets. The specification of the profit function needs care, even though the method proposed and used for doing this is effective. The mechanisms can also fail, though this is more due to the robot than the mechanism, and if faster processing hardware and more sophisticated tracking were used, many of these failures could have been avoided. In doing such development it would be interesting to see how the availability of such equipment would change the mechanisms.

[1] Many successful robots have had their environments 'simplified' through limited, local sensing modalities and the minimum of motor abilities [Holland 95]

Along the way, a few other interesting things have come out. One of these is the realisation that the Behavioural and the Classical approaches actually have a great deal in common in terms of how working versions are implemented, though the Behavioural approach has the advantages of being more explicit and intending to be more general. Another thing was that the poor performance of many of the reinforcement learning methods that have been proposed and applied to date is due to their attempting to learn the sequence of subgoals that have to be achieved. This is a problem that grows exponentially with the number of goals. It is also a problem that has been considered within this thesis to be better dealt with by the classical planning technology, or exploration methods, which can effectively consider options before attempting them. Neither of these have been widely reported, being buried in technical articles that do not mention them explicitly in the title or abstract. Without knowledge of these facts, especially the latter, it would have not been as easy to obtain the results that have been reported here.

The rest of this final chapter begins with a review of the separate technological innovations that have occurred during the research that has led me to make these conclusions and comments. Following this is a description of some of the directions in which the research described here could be continued. After this chapter is the bibliography and then three appendices where various relevant technical matters are described.

9.1 Technical Innovations

The technical innovations described in this section are a mixture of things that had to be done to produce a suitable experimental system, and things that were developed to test out the ideas on using learning. Developing the latter actually served as inspiration for other ideas, or provided a vehicle for improving the understanding of the issues involved. It is very likely that had the type of experimental apparatus described in Section 4.2 or using the problem described in Section 4.1 not been used, then the research would have taken a different course, as mobile robots, or a problem such as part mating, would have had very different operational requirements and constraints, *e.g.* if a mobile robot were used then a failure would be far less important than for an arm. Collectively these innovations, and the experimental apparatus, represent part

of a complete robot system. The learning mechanisms that are described in the last two thirds of this list are the most important products that have arisen from this work. They represent one, possibly useful, answer to how learning can be used in a system controller, by reviewing them collectively it has been possible to answer why they should be used. This answer to why, given on page 214 is in many respects more important than the how.

The Force Table

By mounting a wooden tray onto a force torque sensor and performing the appropriate processing and filtering (described in Section 4.2.4) a sensor has been produced which is as useful as mounting a force sensor into the robot itself, but without the resulting payload reduction. The main usage of the table here has been to estimate the location of objects placed on the tray by their centre of mass, though their orientation cannot be distinguished. The filtering described in Appendix C is an essential part of the sensor, as the estimates otherwise produced would vary with every instance of mechanical noise in the environment. The table allows the force sensor to be shared between multiple robots, even if they are not cooperating on a task. An addition to the table would be a tactile array mounted on the table surface. This would allow localisation of multiple objects, possibly with identification of their orientation, and if the resolution of the array were sufficient, then some steps towards object identification could be made. Being able to sample at a higher rate might allow more accurate estimation, as long as the level of noise remained constant.

The Profit Function Development Method

Developing error functions for learning mechanisms based on a conception of the mechanism being a producer rather than a consumer is an approach which offers a different perspective on what can be a very difficult task. The methodology that was proposed for developing profit functions in Section 3.2 and subsequently used whenever a profit function has been required leaves only the initial identification of the costs and their relationships as problems to be solved by the developer. During use the first statement of costs and relationships can be refined until the mechanism using that profit func-

tion produces the desired behaviour. Such refinement can be a useful tool for users to identify their actual wishes for the mechanism's behaviour (*q.v.* [Papert 80]).

The QUEASY Algorithms

The three QUEASY variants, (described in Section 5.1) which can guide a robot to a point in either a camera image or on the table surface to within the resolution of that sensor do not require global calibration of the relationship between the sensor and the robot. They can withstand disturbances to the relationship between the sensor and robot during reaching. Their low efficiency reported here can be explained by the low frequency of control and simple robot tracking mechanisms, both of which are areas that could be improved. Two of the QUEASY variants use learning to improve the parameterisation of the model; that learning being derived from, and specific to, the model. This is an example of how learning has often been used in robotics, and whilst effective, the experimental results show, it does require extra work by the developer. In general this work is worth doing as the overall result is a system with a higher degree of robustness, and so a greater reliability. Better things are possible by using more general models which can approximate the relationship between sensor and robot through appropriate parameterisation, as typified by the parameter optimisation and the active memory described in the next two sections.

The Parameter Optimisation Approach

Learning optimal parameter settings and learning action-sensor couplings were identified as separate activities on page 46 in Section 3.1, but the work with QUEASY suggested that in fact they are very similar. If the action-sensor coupling uses a model that needs to be appropriately parameterised, then a learning mechanism can (though not always) be derived from the model. That learning mechanism increases the reliability by correcting bad parameterisations that would otherwise cause the coupling to fail. In QUEASY a very simple model was used which provided a weak approximation of the relationship between sensor and robot that needed to be embedded within a feedback loop as successive attempts were required, each reducing the error.

By implementing a direct numerical optimisation method into P_MOVE (Algorithm 4,

page 137), the explicit model of QUEASY was removed. Instead, the optimisation mechanism approximates the action-sensor coupling implicitly. The performance of P_MOVE was comparable to that of QUEASY, but without the development effort that was required for that module.

Active Memory

Active memory, described in Chapter 7, consists of estimating by interpolating over a memory of previously established results the robot commands that will cause a particular sensor result. This mechanism can only be used when the mapping between results and parameters is not one to many, and successful use requires the active memory mechanisms to handle noise in the sensors and robot, as well as drift or change in the relationship robot sensor relationship. Embedding active memory within a feedback loop allows for initial, poor estimates to be gradually refined through repeated attempts. The I_INTERP algorithm presented on page 165, and its subsequent experimental verification acted as proof of the usefulness of this development. Like the learning mechanisms that use an optimisation technique to approximate the function, an active memory activity can achieve any instance of a task once linkages have been established, though another competent activity is required for the times when the estimation process fails due to insufficient results. Active memory is a weak form of explanation based learning that does not require an external teacher, and which can generalise from the results it has learned whenever a result that is outside its experience is required.

The ANNIE Algorithm

Usually, a specific task can be completed in a number of different ways. The decision about which way is best can only be made after some experimentation, a process for which learning mechanisms are well suited. What is novel about the learning mechanism that was developed to do this, called ANNIE and described in Chapter 8, is that it can learn to select the best activity from a candidate pool solely on the basis of reported performance. Importantly, it learns in a near optimal time to select the optimally correct activity. Apart from the profit values, no other information need be

supplied to the algorithm, though it is important that the activities that are in the candidate pool are independent, both of each other and of activities that will be used at a later time. In this respect this learning mechanism is like one produced using an optimisation technique or active memory: it is an activity that can achieve any instance of a task, and only requires a profit function plus linkages to be established before being applicable.

9.2 Now What?

Although this piece of research is for now completed, there are two classes of avenues for future exploration that might produce interesting results:-

- Technical improvements to the implementations described here
- Conceptual developments that further the ideas described in this thesis.

These two classes of development are discussed in the next two sections, beginning with the technical improvements.

9.2.1 Technical Improvements

There are two areas in which the present system might be improved:-

- The first area covers the developments to the technical innovations that have been made, and which have been dealt with in their descriptions or earlier in this chapter during the summary.
- The second, more involving area is the development and implementation of a full system. As it stands the experimental system is only a small fraction of what is necessary to perform a task such as assembly. Extending the current work to have the necessary functionality to perform assembly would require developing the activities and behaviours which perform part acquisition and mating, and linking the system to a planner that could convert a user supplied task specification into a sequence of high level activities.

Both of these activities are still undergoing active and widespread research. During this development it would be interesting to see how far the idea explored and justified in this thesis could be taken. In finding the limits of the idea a complete understanding of its usefulness would be obtained.

Although not offering the possibility of discovering the limits of the idea, it would be interesting to apply the individual learning mechanisms described in this thesis and the ideas on system architecture to a mobile application. This would be a test of the presented developments, but it would also provide another testbed for the longer term conceptual development described in the next section.

9.2.2 Conceptual Development

The one area where learning has been identified (Section 3.1) as being useful, but has deliberately not been explored, is the invention of novel activities. It would be at this single problem that future research would be directed, as progress in this area will affect the issues of how robots are programmed. This research would include the various technical extensions described in the last section.

For a robot to be able to invent and implement new behaviours is for it to have a level of controller beyond the five fold category described in Section 2.1, as in addition to providing a task level interface, it can make structural modifications to its controller in response to changes in task, situation, or its body. These modifications would include changing the learning mechanisms used to achieve a coupling, adjusting the structuring on activities, or inventing novel activities.

There are two reasons why researching this area would be useful:-

- There are a number of research areas that have an interest in how novel behaviours are generated, *e.g.* Genetic Programming (*e.g.* [Koza 92]), psychology, (*e.g.* [Drescher 91]). Attempts by others to implement biologically theories on robots, *e.g.* [Hallam & Halperin 92, Webb 93] have often found both limitations and opportunities that the unimplemented theories had not expected or found. Accordingly, attempting to implement the technologies and ideas suggested by different research areas would serve both to validate them, and to provide ideas

for further work within assembly robotics.

- In purely assembly robotics terms, the inclusion of the ability to generate novel activities within a controller would reduce the need for users to develop the controller beyond the initial competencies which are necessary for activity invention to take place. Apart from the extra flexibility the system would exhibit, novel activities would be expected to arise which can take advantage of regularities in the world which human developers would miss. Such invention would become important when robots are applied to domains in which the human developers have no direct experience, such as deep water mining, or planetary exploration.

This proposed programme of development is supported by the work described in this thesis is two ways:

- As noted in Section 3.1 the strong learning mechanisms that are needed in the invention of novel activity need to be backed up by the kinds of weak learning mechanism that have been used and proved here. This is because the kinds of thing strong mechanisms are good at are not the kinds of thing that need to be achieved when applying novel modules, which is where the abilities of weak learning mechanisms are most useful.

- The progression of learning mechanisms described in this thesis is a progression of increasingly generic mechanisms that are shaped to match the needs of the particular application. If this pattern can be applied to other areas during the development of novel activities, then the problem might be simplified. This is because instead of having to specify a unique model for each action-sensor coupling, or decision function that is needed, it becomes sufficient to just select the kind of generic activity and then specify its linkages and profit function, the definition of which is related to the linkages that are established. This is something that strong learning mechanisms such as Case-Based Reasoning, or Genetic Programming are good at.

In addition to the issue of deriving a new activity, there are a number of other issues that should also be considered as part of this programme, including how to identify

when novel activities are required, and finding suitable linkages. Both of these become harder when the structural framework of the system controller has also to be learned. These are not the only issues, and like the research that has been achieved so far, it is unlikely that all of the issues or their interaction will be known until practical experiments have been developed, performed and analysed. This will require a lot of work.

9.3 Summary of Chapter

This final chapter has described the conceptual and technical innovations that have been covered in the preceding chapters, and covered some suggestions for how these innovations could be developed or utilised. This description has contained an overview of the research that has lead to this thesis, research that was directed by trying to find ways of making the system controller of a robotic system better able to handle the uncertainty that can otherwise cause failures. By using appropriate kinds of learning this goal can be achieved, and at the same time the tasks of the developer simplified.

During the work that led to this result a number of other results came to light. Some, like the profit functions, were the result of deliberate development. Others, like active memory were fortuitous accidents.

Looking back over the whole process of this research one feature which has repeated itself over and over is that rarely are the consequences of an idea all that can be expected before that idea is realistically implemented and tested. This feature of research is one that is rarely discussed or noted, with even the most complicated of results usually presented without discussion of how they were achieved. The work described here resulted from the evolution of ideas, techniques and technology from the starting point of learning the parameters of a model used to guide reaching. A different starting point, environment, or situation might have lead to a very different thesis.

Bibliography

[Aboaf 88] Eric W. Aboaf. *Task-Level Robot Learning.* Unpublished PhD thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, August 1988.

[Adv91] Advanced Manipulation Laboratory, The Robotics Institute, CMU, Pittsburgh, Pennsylvania 15213. *CHIMERA II Real-Time Programming Environment*, 2nd edition, November 10 1991.

[Albus & Evans, Jr. 76] James S. Albus and John M. Evans, Jr. Robot systems. *Scientific American*, February 1976.

[Albus 75] James S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Transactions of the ASME Journal of Dynamic Systems, measurement and Control*, (97):220–227, September 1975.

[Anderson *et al.* 92] C.D. Anderson, R.J. Freland, and M.D. Williams. Negative contrast associated with reinforcing stimulation of the brain. *Society for Neuroscience Abstracts*, (18):874, 1992.

[Angeline & Pollack 93] Peter J Angeline and Jordan Pollack. Evolutionary module acquisition. In *The Second Annual Conference on Evolutionary Programming*, 1993.

[Appleton 90] E. Appleton. Robotic assembly of aerospace fabrications - a case study approach. In *ARA 3rd National Conference on Robotics*, pages p34–44, June 1990.

[Arbib 81] M. Arbib. Perceptual structures and distributed motor control. In *Handbook of Physiology — The Nervous System.* Brooks, 1981.

[Arbib 87] M. Arbib. Levels of modelling mechanisms of visually guided behaviour. *Brain and Behavioural Sciences*, 10:407–465, 1987.

[Arkin 92] R.C. Arkin. Integrated control for mobile manipulation for intelligent materials handling. In *Intelligent*

Material Handling using Mobile Robots — Collected Papers from the Mobile Robot Laboratory. Georgia Tech Material Handling Research Centre Report OP-92-19, 1992.

[Arkin 98] R.C. Arkin. *Behaviour-Based Robotics.* The MIT Press, 1998.

[Asada 90] H. Asada. Teaching and learning of compliance using neural nets. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1237–1244, 1990.

[Balch 92a] Peter R. Balch. A review of the work reported in [Balch 92b]. Private email message, March 1992.

[Balch 92b] Peter R. Balch. A software architecture for robot assembly. DAI Working Paper 235, Department of Artificial Intelligence, University of Edinburgh, January 1992.

[Barto 90] A.G. Barto. Connectionist learning for control. In W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks For Control*, pages 5–58. MIT Press, 1990.

[Begg *et al.* 94] David Begg, Stanley Fischer, and Rudiger Dornbusch. *Economics.* McGraw-Hill Book Company, 4th edition edition, 1994.

[Beguelin *et al.* 91] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. A user's guide to PVM: Parallel virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[Berry & Fristedt 85] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments.* Chapman and HallByte, 1985.

[Bowen *et al.* 94] Jonathan Bowen, Jifeng He, and Ian Page. Hardware compilation. In J.P. Bowen, editor, *Towards Verified Systems*, chapter 10, pages 193–207. Elsevier, 1994.

[Braitenberg 84] Valentino Braitenberg. *Vehicles: Experiments In Synthetic Psychology.* MIT Press., 1984.

[Brent 73] R.P. Brent. *Algorithms for Minimization without Derivatives*, chapter 7. Prentice-Hall, 1973.

[Brooks & Mataric 93] Rodney A. Brooks and Maja J. Mataric. Real robots, real learning problems. In Jonathan H. Connell and

Sridhar Mahadevan, editors, *Robot Learning*, chapter 8, pages 193–213. Kluwer Academic Publishers, 1993.

[Brooks & Stein 93] Rodney A. Brooks and Lynn Andrea Stein. Building brains for bodies. Technical Report 1439, MIT AI Lab, August 1993.

[Brooks 82] Rodney A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(3):29–68, 1982. AIM-685, Stanford University.

[Brooks 85] Rodney A. Brooks. A robust layered control system for a mobile robot. A.I. Memo 864, MIT, September 1985.

[Brooks 86] Rodney A. Brooks. Achieving artificial intelligence through building robots. A.I. Memo 899, MIT, May 1986.

[Brooks 90a] Rodney A. Brooks. The behaviour language : user's guide. AI Memo 1227, MIT, Artificial Intelligence Laboratory, 1990.

[Brooks 90b] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[Brooks 91a] Rodney A. Brooks. Intelligence without reason. AI Memo 1293, MIT, 1991.

[Brooks 91b] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, 1991.

[Brost & Goldberg 94] Randy C. Brost and Kenneth Y. Goldberg. A complete algorithm for designing modular fixtures for polygonal parts. In Ken Goldberg, editor, *IEEE ICRA Workshop S-1: Design of Parts and Devices for Automation.* IEEE Robotics and Automation Society, May 8-13 1994.

[Brost 86] Randy C. Brost. Automatic grasp planning in the presence of uncertainty. In *Proceedings of IEEE International Conference on Robotics and Automation*, April 1986.

[Bush & Estes 59] Robert K. Bush and William K. Estes. *Studies in Mathematical Learning Theory.* Stanford University Press, 1959.

[Cameron *et al.* 93] Jonathan M. Cameron, Douglas C. MacKenzie, Keith R. Ward, Ronald C. Arkin, and Wayne J.

Book. Reactive control for mobile manipulation. In *Proc. International Conference on Robotics and Automation*, pages 228–235, May 1993.

[Carbonell *et al.* 90] J.G. Carbonell, C.A. Knoblock, and S. Minton. PRODIGY: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for intelligence*. Erlbaum, 1990.

[Chapman & Agre 87] David Chapman and Philip E. Agre. Abstract reasoning as emergent from concrete activity. In M. P. Georegeff and A. L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop at Timberline, Oregon*, pages 411–424. Morgan Kaufman, 1987.

[Chapman 91] David Chapman. *Vision, Instruction, and Action.* Artificial Intelligence. MIT Press, 1991.

[Chongstitvatana & Conkie 90a] P. Chongstitvatana and A.D. Conkie. Behaviour-based assembly experiments using vision sensing. DAI Research Paper 466, Department of Artificial Intelligence, University of Edinburgh, 1990.

[Chongstitvatana & Conkie 90b] P. Chongstitvatana and A.D. Conkie. Behaviour-based assembly experiments using vision sensing. DAI Research Paper 466, Department of Artificial Intelligence, University of Edinburgh, 1990.

[Chongstitvatana 92] P. Chongstitvatana. *The Design and Implementation of Vision-Based Behavioural Modules for a Robotic Assembly.* Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Churchland & Sejnowski 92] Patricia S. Churchland and Terrence J. Sejnowski. *The Computational Brain.* MIT Press, 1992.

[Cliff *et al.* 92a] D. Cliff, I. Harvey, and P. Husbands. Incremental evolution of neural network architectures for adaptive behaviour. CSRP 256, School of Cognitive and Computing Sciences, University of Sussex, December 1992.

[Cliff *et al.* 92b] D. Cliff, P. Husbands, and I.Harvey. Evolving visually guided robots. CSRP 220, School of Cognitive and Computing Sciences, University of Sussex, July 1992.

[Cliff *et al.* 93] D.T. Cliff, P. Husbands, and I. Harvey. Evolving recurrent dynamical networks for robot control. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*

Proceedings of the International Conference at Innsbruck, Austria, Wien and New York, 1993. Springer.

[Connell & Mahadevan 93] Jonathan H. Connell and Sridhar Mahadevan. Rapid task learning for real robots. In Jonathan H. Connell and Sridhar Mahadevan, editors, *Robot Learning*, chapter 5, pages 105–139. Kluwer Academic Publishers, 1993.

[Connell 89] J.H. Connell. A behaviour-based arm controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, 1989.

[Connolly *et al.* 95] C.I. Connolly, R.A. Grupen, and K. Souccar. A hamiltonian framework for kinodynamic planning. In *Proceedings of the 1995 IEEE Conference on Robotics and Automation*, 1995.

[Coulon & Nougaret 83] P.Y. Coulon and M. Nougaret. Use of a tv camera in closed-loop position control of mechanisms. In A. Pugh, editor, *Robot Vision*. Springer-Verlag, 1983.

[Craig 89] J.J. Craig. *Introduction to Robotics, Mechanics and Control*, pages 1–144. Addison Wesley, 2nd edition, 1989.

[Critchlow 85] A.J. Critchlow. *Introduction to Robotics*, chapter 8, pages 293–324. MacMillan, 1985.

[Davidor 89] Yuval Davidor. Analogous crossover. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1989.

[Deacon & Malcolm 94] Graham E. Deacon and Chris Malcolm. A robot system designed for task-level assembly. In *Int. Workshop on Advanced Robotics and Intelligent Machines*, March 1994.

[Deacon & Wright 95] Graham E. Deacon and Mark Wright. Orientating 2.5d objects in the plane. Unpublished Working Paper, May 1995.

[Deacon 92] Graham E. Deacon. An algorithm for orientating objects in a minimum number of robot sweeping motions. Unpublished Working Paper, 1992.

[Deacon 97] G. Deacon. *Accomplishing Task-Invariant Assembly Strategies by Means of an Inherently Accommodating Robot Arm*. Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, April 1997.

[Deley 91] David W. Deley. Computer generated random numbers. World Wide Web page, 1991. Available from: http://draco.centerline.com:8080/~franl/crypto/random-numbers.html.

[Dellaert & Beer 94] Frank Dellaert and Randall D. Beer. Toward and evolvable model of development for autonomous agent synthesis. In R.Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the 4th International Workshop on Synthesis and Simulation of Living Systems*, 1994.

[Drabble & Cox 85] B. Drabble and P. Cox. Error detection and recovery in an uncertain environment. Technical report, Department of Computer Science and Applied Mathematics, University of Aston, Birmingham, 1985.

[Drescher 86] Gary L. Drescher. Genetic ai: Translating piaget into lisp. A.I. Memo 890, MIT AI Laboratory, February 1986.

[Drescher 91] Gary L. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence.* MIT Press, 1st edition, 1991.

[Dufay & Latombe 84] Bruno Dufay and Jean-Claude Latombe. An approach to automatic robot programming based on inductive learning. *The International Journal of Robotics Research*, 3(4):3–20, 1984.

[Erdmann & Mason 88] M.A. Erdmann and M.T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, August 1988.

[Erdmann 84] M.A. Erdmann. On motion planning with uncertainty. Technical Report 810, MIT, 1984.

[Erdmann 94] M.A. Erdmann. Understanding action and sensing by designing action-based sensors. In Ken Goldberg, editor, *Design of Parts and Devices for Automation.* IEEE Robotics and Automation Society, May 8-13 1994.

[Ernst 61] H.A. Ernst. *MH-1: A Computer Operated Mechanical Hand.* Unpublished PhD thesis, Massachusetts Institute of Technology, 1961.

[Feigenbaum *et al.* 88] Edward Feigenbaum, Pamela McCorduck, and H. Penny Nii. *The rise of the expert company : how visionary companies are using artificial intelligence to achieve higher productivity and profits.* Times Books, 1988.

[Fleming 85a] A.D. Fleming. Analysis of uncertainties in a structure of parts 1. DAI Research Paper 271, Department of Artificial Intelligence, University of Edinburgh, 1985.

[Fleming 85b] A.D. Fleming. Analysis of uncertainties in a structure of parts 2. DAI Research Paper 272, Department of Artificial Intelligence, University of Edinburgh, 1985.

[Fleming 87] A.D. Fleming. *Analysis of Uncertainties and Geometric Tolerances in Assemblies of Parts.* Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1987.

[Fletcher 87] R. Fletcher. *Practical Methods of Optimization.* John Wiley, 2nd edition, 1987.

[Fredrick P. Brooks 82] Jr. Fredrick P. Brooks. *The Mythical Man Month.* Addison-Wesley, 1982. Reprint of the 1975 edition with corrections.

[Gardner 61] M. Gardner. *More Mathematical Puzzles and Diversions.* Penguin, 1966 edition, 1961.

[Gerald & Wheatley 89] Curts F. Gerald and Patrick O. Wheatley. *Applied Numerical Analysis.* Addison-Wesley, fourth edition, 1989.

[Gini & Gini 83] M. Gini and G. Gini. Towards automatic error recovery in robot programs. In *Proceedings of 8th International Joint Conference on Artificial Intelligence*, pages 821–823, 1983.

[Gini & Smith 86] M. Gini and R.E. Smith. Reliable real–time robot operation employing intelligent forward recovery. *Journal of Robotic Systems*, 3:281–300, 1986.

[Gittins 88] J.C. Gittins. *Multi-armed Bandit Allocation Indices.* John Wiley and Sons, 1988.

[Goldberg 89a] David E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison–Wesley, January 1989.

[Goldberg 89b] David E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, 1989.

[Gordon & Whitely 93] V.S. Gordon and D. Whitely. Serial and parallel genetic algorithms as function optimisers. Technical Report CS-93-114, Department of Computer Science, Colorado State University, September 1993.

[Grey Walter 50] W. Grey Walter. An imitation of life. *Scientific American*, pages 42–45, May 1950.

[Gross & Graziano 90] Charles G. Gross and Micheal S. Graziano. Bimodal visual-tactile responses in the macaque putamen. *Society for Neuroscience Abstracts*, 16(10), 1990.

[Grundel 92] R. Grundel. How the mountain chickadee procures more food in less time for its nestlings. *Behavioural Ecology and Sociobiology*, (31):291–300, 1992.

[Gullapalli *et al.* 92] Vijaykumar Gullapalli, Roderic A Grupen, and Andrew G. Barto. Learning reactive admittance control. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1475–1480, 1992.

[Guo *et al.* 94] S. Guo, W. Luk, and P. Probert. Developing parallel architectures for ranging and image sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1994.

[Hallam & Halperin 92] Bridget Hallam and Janet Halperin. Halperin's neuro-connector model of learning and motivation. Working paper, University of Edinburgh, 1992. In preparation.

[Hallam 91a] John Hallam. Cars: an experiment in autonomous real-time control. Research Report LiTh-ISY-I-1236, Institutionen för Systemetknik, University of Linköping, 1991.

[Hallam 91b] John Hallam. Playing with toy cars: An experiment in real time control. *AISB Quarterly*, (77), Summer 1991.

[Hammond 89] K.J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, 1989.

[Harder & Real 87] H.F. Harder and L.A. Real. Why are bumble bees risk averse. *Ecology*, 68(4):1104–1108, 1987.

[Hardy *et al.* 89] N.W. Hardy, D.P. Barnes, and M.H. Lee. The automatic diagnosis of task faults in flexible manufacturing systems. *Robotica*, 7(1):25–35, 1989.

[Harmelen & Bundy 88] F. Van Harmelen and A. Bundy. Explanation-based generalisation =partial evaluation. *Artificial Intelligence*, 36:401–412, 1988.

[Harris & Conkie 92] M. Harris and A.D. Conkie. Vision guided part alignment with degraded data. In *Proceedings of IFAC*

Symposium on Intelligent Components and Instruments for Control Applications, Malaga, Spain, 1992. proceedings to be published.

[Haugeland 78] John Haugeland. The nature and plausibility of cognitivism. *The Brain and Behavioural Sciences*, 1:215–226, 1978.

[Hayes & Demiris 94] Gillian Hayes and John Demiris. A robot controller using learning by imitation. In *International Workshop on Intelligent Robotic Systems*, July 1994.

[Herrnstein 61] R.J. Herrnstein. Relative and absolute strength of response as a function of frequency of reinforcement. *Journal of the Experimental Analysis of Behaviour*, (4):267–272, 1961.

[Hirai & Iwata 92] S. Hirai and K. Iwata. A model-based generation of damping control law for part-mating. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 494–499, 1992.

[Hogg *et al.* 91] D. Hogg, F. Martin, and R. Resnick. Braitenberg creatures. Epistemology and Learning Memorandum 13, MIT, 1991.

[Holland 75] J.H. Holland. *Adaption in natural and artificial systems.* The University of Michigan Press, 1975.

[Holland 92] J.H. Holland. *Adaption in natural and artificial systems.* MIT Press, 2nd edition, 1992. Corrections made to the theory of Optimal Allocation from the 1st edition(1975).

[Holland 95] O. Holland. Personal Communication, March 1995.

[Hoyle 84] G. Hoyle. The scope of neuroethology. *The Behavioural and Brain Sciences*, 7:367–412, 1984.

[Hsia 86] T. C. Hsia. Adaptive control of robot manipulators – a review. In *IEEE 1986 International Conference on Robotics and Automation*, 1986.

[Huntingford 84] Felicity Huntingford. *The Study of Animal Behaviour.* Chapman and Hall, 1984.

[Iro88] Ironics Incorporated, Computer Systems Division, Ithaca, New York, USA. *IV-3220 VMEbus Single Board Computer and Multiprocessing Engine*, 1.1 edition, November 1988.

[Julier & Uhlmann 94] Simon Julier and Jeffrey Uhlmann. A general method for approximating nonlinear transformations of probability distributions. Technical report, Robotics Research Group, Department of Engineering Science, University of Oxford, December 1994.

[Kaelbling & Rosenschein 91] Leslie Pack Kaelbling and Stanley J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48. The MIT Press, 1991. Special Issue of Robotics and Autonomous Systems.

[Kaelbling 93] Leslie Pack Kaelbling. *Learning in Embedded Systems.* Bradford Books, 1993.

[Kalman 60] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82D:35–45, March 1960.

[Kalman 63] R.E. Kalman. Mathematical description of linear dynamical systems. *J.S.I.A.M Control*, series A(1):152–192, 1963.

[Kawato 90] Mitsuo Kawato. Computational schemes and neural network models for formation and control of multijoint arm trajectory. In W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks For Control*, pages 198–228. MIT Press, 1990.

[Khatib 95] O. Khatib. Real time obstacle avoidance for manipulators and mobile robotics. In *IEEE Conference on Robotics and Automation*, 1995.

[Kim 92] Taehee Kim. The development of vibrations sensors as event signature sensors in assembly. Unpublished M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Knuth 81] D.E. Knuth. *Seminumerical Algorithms: The Art of Computer Programming*, volume 2. Addison–Wesley, 2nd edition, 1981.

[Kolodner 94] J. Kolodner. *Case-Based Reasoning.* Morgan–Kaufmann, 1994.

[Kopeikina *et al.* 88] L. Kopeikina, R. Brandau, and A. Lemmon. Case-based reasoning for continuous control. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 250–259, May 1988.

[Kosak *et al.* 91] Corey Kosak, Joe Marks, and Stuart Shieber. A parallel genetic algorithm for network-diagram layout. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, 1991.

[Koza 92] J.R. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Mass., 1992.

[Kuperstein & Rubenstein 89] M. Kuperstein and J. Rubenstein. Implementation of an adaptive neural controller for sensory-motor coordination. In J. Denker, editor, *Proc. International Joint Conference on Neural Networks*, pages 265–70, 1989.

[Latombe 91] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[Law 94] Y.K. Law. *An Analytical tool with optimisation methods for the automation of fixture design.* Unpublished PhD thesis, Department of Engineering, University of Cambridge, 1994.

[Lee *et al.* 83] M.H. Lee, D.P. Barnes, and N.W. Hardy. Knowledge based error recovery in industrial robots. In *Proceedings of 8th International Joint Conference on Artificial Intelligence*, pages 824–826, 1983.

[Lieberman & Wesley 77] L.I. Lieberman and M.A. Wesley. Autopass: An automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development*, 21(4):321–333, 1977.

[Lin & Popplestone 95] Y. Lin and R.J. Popplestone. A group theoretic formalisation of surface contact. *International Journal of Robotics Research*, 1995. In Press.

[Lor87] Lord Corporation. *Installations and Operations Manual for F/T Series Force/Torque Sensing Systems*, 3rd edition, November 1987.

[Loughlin 92] C. Loughlin, editor. *InFACT: Project, Concepts, Machine.* MCB University Press Limited, Bradford, 1992.

[Low 92] Poh Lian Low. Sensorless soma part orientation. Unpublished M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Lozano-Pérez & Brooks 85] T. Lozano-Pérez and R.A. Brooks. An approach to automatic robot programming. AI Memo 842, MIT, April 1985.

[Lozano-Pérez & Wesley 79] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[Lozano-Pérez 76] T. Lozano-Pérez. The design of a mechanical assembly system. Technical Report 397, Artificial Intelligence laboratory, MIT, 1976.

[Lozano-Pérez *et al.* 84] T. Lozano-Pérez, M.T. Mason, and R.H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[Lozano-Pérez *et al.* 87] T. Lozano-Pérez, J.L. Jones, E. Mazer, P.A. O'Donnell, and W.E.L Grimson. Handey: A robot system that recognises, plans, and manipulates. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 843–849, 1987.

[Lozano-Pérez *et al.* 90] T. Lozano-Pérez, E. Mazer, J.L. Jones, and P.A. O'Donnell. Task level planning of pick-and-place robot motions. In P.H. Winston and S.A. Shellard, editors, *Artificial Intelligence at MIT : Expanding Frontiers*, volume 2, chapter 26, pages 40–59. MIT Press, 1990.

[Lozano-Pérez *et al.* 92] Tomas Lozano-Pérez, Joseph L. Jane, Emmanuel Mazer, and Patrick O'Donnell. *Handey: A Robot Task Planner.* MIT Press, 1992.

[Lynch & Mason 94] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. In A.K. Peters, editor, *The First Workshop on the Algorithmic Foundations of Robotics*, February 1994.

[Lyons & Hendriks 92] D.M. Lyons and A.J. Hendriks. Planning for reactive robot behaviour. In *IEEE International Conference on Robotics and Automation*, 1992.

[Lyons 85] D. Lyons. *Robot Schemas.* Unpublished PhD thesis, University of Massachusetts, Amehurst, USA, 1985.

[MacKenzie & Arkin 94] Douglas C. MacKenzie and Ronald C. Arkin. Formal specification for behaviour-based mobile robots. In *Proceedings of the SPIE conference on Mobile Robotics*, September 1994.

[Maddox 94] J. Maddox. The poor quality of random numbers. *Nature*, 372:403, 1994.

[Mahalanobis 48] P.C. Mahalanobis. Historical note on the d^2-statistic. *Sankhya*, 1948.

[Malcolm & Fothergill 86] C.A. Malcolm and A.P. Fothergill. Some architectural implications of the use of sensors. DAI Research Paper 294, Department of Artificial Intelligence, University of Edinburgh, 1986.

[Malcolm & Smithers 88] C.A. Malcolm and T. Smithers. Programming assembly robots in terms of tack achieving behavioural modules: First experimental results. DAI Research Paper 410, Department of Artificial Intelligence, University of Edinburgh, 1988.

[Malcolm 83] C.A. Malcolm. The outline corner filter. In B. Rooks, editor, *Proceedings of the 3rd International Conference on Robot Vision and Sensory Controls (ROVISEC)*, pages 61–68, Amsterdam, 1983. North Holland Publishing Company. Also available as University of Edinburgh DAI research paper No. 212.

[Malcolm 87] C.A. Malcolm. Planning and performing the robotic assembly of soma cube constructions. Unpublished M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh, 1987.

[Malcolm 92] C.A. Malcolm. Giving machines ideas. Dai teaching paper, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Malcolm 94] C.A. Malcolm. Language levels in assembly robot programming. Unpublished Teaching Paper, 1994.

[Mani *et al.* 85] Mani, Murali, and Wilson. A programmable orienting system for flat parts. In *Proceedings of NAMRII*, 1985.

[Manly 86] Bryan F.J. Manly. *Multivariate Statistical Methods.* Chapman and Hall, 1986.

[Mason 84] Matthew T. Mason. Automatic planning of fine motions : Correctness and completeness. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 492–503, March 1984.

[Mason 93] Matthew T. Mason. Kicking the sensing habit. *AI Magazine*, pages 58–59, Spring 1993.

[McConnell 96] Steve McConnell. *Rapid Application Development.* Microsoft Press, 1996.

[McFarland & Bösser 93] David McFarland and Thomas Bösser. *Intelligent Behaviour in Animals and Robots.* Bradford Books, 1993.

[McFarland 71] David McFarland. *Feedback mechanisms in Animals Behaviour.* Academic Press, 1971.

[McFarland 93] David McFarland. *Animal Behaviour: psychobiology, ethology and evolution.* Longman Scientific and Technical, 2nd edition, 1993.

[Mead & Conway 80] Carver Mead and Lynn Conway. *Introduction to VLSI systems.* Addison-Wesley, 1980.

[Mel 88] B. Mel. Murphy: A robot that learns by doing. In D. Anderson, editor, *Neural Information Processing Systems*, pages 543–553. Academic Press, 1988.

[Minsky & Papert 69] M. Minsky and S. Papert. *Perceptrons : an introduction to computational geometry.* MIT Press, 1969. Expanded Edition published 1988.

[Narendra & Thathachar 89] Kumpati Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, 1989.

[Nehmzow 92] Ulrich Nehmzow. *Experiments in Competence Acquisition for Autonomous Mobile Robots.* Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Nevins & Whitney 78] J.L. Nevins and D.E. Whitney. Computer controlled assembly. *Scientific American*, pages 62–74, February 1978.

[Nevins *et al.* 84] J.L. Nevins, M. Desai, E. Fogel, B.K. Walker, and D.E. Whitney. Adaptive control, learning, and cost effective sensor systems for robotic or advanced automation systems. In *The First International Symposium on Robotics Research*, pages 983–994, 1984.

[Nilsson 69] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.

[Ohmori 86] Takashi Ohmori. Binocular stereo vision system using an eye and neuron model. *Journal of Robotic Systems*, 3(2):149–163, 1986.

[Orr 92] Mark J.L. Orr. A Mickey Mouse Guide to Kalman Filtering. Departmental Working Paper 232, Department of Artificial Intelligence, January 1992.

[Papert 80] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, New York, 1980.

[Parma 92] George F. Parma. Development of a truss joint for robotic assembly of space structures. In Alan A. Desrochers, editor, *Intelligent robotic systems for space exploration*, chapter 3, pages 111–139. Kluwer Academic Publishers, 1992.

[Penrose 53] L.W. Penrose. Distance, size and shape. *Annals of Eugenics*, 18:337–43, 1953.

[Peshkin & Sanderson 87] M.A. Peshkin and A.C. Sanderson. Planning sensorless robot manipulation of sliding objects, 1987.

[Peshkin & Sanderson 88] M.A. Peshkin and A.C. Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6), December 1988.

[Peshkin 90] M.A. Peshkin. Programmed compliance for error corrective assembly. *IEEE Transactions on Robotics and Automation*, 6(4), August 1990.

[Pettinaro & Malcolm 95] G.C. Pettinaro and C.A. Malcolm. A program for describing two similar assemblies. DAI Technical Paper 35, Department of Artificial Intelligence, University of Edinburgh, 1995.

[Pierre 86] Donald Pierre. *Optimization Theory with Applications.* Dover Publications, 1986.

[Pook & Ballard 93] P.K. Pook and D.H. Ballard. Recognizing teleoperated manipulations. In *Proc. IEEE International Conference on Robotics and Automation*, pages 578–585, 1993.

[Popplestone 83] R.J. Popplestone. Group theory and robotics. DAI Research Paper 196, Department of Artificial Intelligence, University of Edinburgh, 1983.

[Popplestone *et al.* 78] R.J. Popplestone, A.P. Ambler, and I. Bellos. RAPT, a language for describing assemblies. *Industrial Robot*, 3:131–137, 1978.

[Powell 64] M.J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, (7):155–162, July 1964.

[Press *et al.* 90] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1990.

[Quinlan 86] J.R. Quinlan. *Learning efficient classification procedures and their application to chess end games*, pages 463–481. Morgan-Kauffman, 1986.

[Ram & Santamaria 93] Ashwin Ram and Juan Carlos Santamaria. Multistrategy learning in reactive control systems for autonomous robotic navigation. *Informatica*, 17(4):347–369, 1993.

[Ram *et al.* 92] Ashwin Ram, Ronald C. Arkin, Kenneth Moorman, and Russell J.Clark. Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems. GIT-CC-92-57, College of Computing, Georgia Institute of Technology, Atlanta, 1992.

[Rao & Goldberg 93] Anil S. Rao and Keneth Y. Goldberg. Manipulating algebraic parts in the plane. Technical report, Institute of Robotics and Intelligent Systems, Department of Computer Science, University of Southern California, December 1993.

[Real 91] L.A. Real. Animal choice behaviour and the evolution of cognitive architecture. *Science*, (253):980–986, 1991.

[Requicha & Tilove 83] A.A.G. Requicha and R.B. Tilove. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4), 1983.

[Rizzi & Koditschek 94] A.A. Rizzi and D.E. Koditschek. An active visual estimator for dextrous manipulation. In *1994 IEEE International Conference on Robotics and Automation Workshop on Visual Servoing*, 1994.

[Roeder 70] K.D. Roeder. Episodes in insect brains. *Scientific American*, 58:378–389, 1970.

[Rosenschein & Kaelbling 94] Stanley J. Rosenschein and Leslie Pack Kaelbling. A situated view of representation and control. Technical report, Teleos Research, June 1994.

[Rumelhart *et al.* 86a] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[Rumelhart *et al.* 86b] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing*, volume 2. MIT Press, 1986.

[Salganicoff & Ungar 95] Marcos Salganicoff and Lyle H. Ungar. Active exploration and learning in real-valued spaces using multi-armed bandit allocation indices. In *11th International Machine Learning Conference: Workshop on Robot Learning*, 1995.

[Salganicoff *et al.* 94] Marcos Salganicoff, Len G. Kunin, and Lyle H. Ungar. Active exploration based ID-3 learning for robot grasping. In *11th International Machine Learning Conference: Workshop on Robot Learning*, 1994.

[Schank 82] R.C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press, 1982.

[Segre & Turney 93] Alberto Segre and Jennifer Turney. Planning, acting and learning in a dynamic domain. In Steve Minton, editor, *Machine Learning Methods for Planning*, chapter 5, pages 125–158. Morgan Kaufmann, 1993.

[Segre 88] Alberto Segre. *Machine Learning of Robot Assembly Plans.* Kluwer Academic Press, 1988.

[Srinivas 76] S. Srinivas. *Error Recovery in Robot Systems.* Unpublished PhD thesis, California Institute of Technology, 1976.

[Starkweather *et al.* 91] T. Starkweather, D. Whitely, and K. Mathias. Optimization using distributed genetic algorithms. In *Parallel Problem Solving from Nature.* Springer Verlag, 1991.

[Stender 93] J. Stender, editor. *Parallel Genetic Algorithms.* IOS Publishing, 1993.

[Sutton 84] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* Unpublished PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1984.

[Sutton 90] R.S. Sutton. First results with dyna, an integrated architecture for learning, planning and reacting. In W.T. Miller, R.S. Sutton, and P. J. Werbos, editors, *Neural Networks For Control*, pages 179–189. MIT Press, 1990.

[Taylor 76] R.H. Taylor. The synthesis of manipulator control programs from task-level specifications. Report AIM 282, Stanford University, 1976.

[Thathachar & Sastry 85] M.A.L. Thathachar and P.S. Sastry. A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):168–175, 1985.

[Thrun 92] Sebastian Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University, 1992.

[Udupa 77] S.M. Udupa. *Collision Detection and Avoidance in Computer Controlled Manipulators.* Unpublished PhD thesis, California Institute of Technology, 1977.

[van der Smagt *et al.* 95] P. van der Smagt, F. Groen, and B. Kröse. A monocular robot arm can be neurally positioned. Technical report, University of Amsterdam, Department of Computer Science, 1995.

[Webb 93] Barbara Helen Webb. *Perception in Real and Artificial Insects: A Robotic Investigation of Cricket Phonotaxis.* Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1993.

[Whitehead 91] Steven D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action.* Unpublished PhD thesis, Department of Computer Science, University of Rochester, November 1991.

[Whitney 94] D.E. Whitney. A short primer on feature-based design for assembly. In Ken Goldberg, editor, *Design of Parts and Devices for Automation.* IEEE Robotics and Automation Society, May 8-13 1994.

[Whittle 92] Peter Whittle. *Probability via Expectation.* Springer-Verlag, 3rd edition, 1992.

[Wilson 92] Myra Scott Wilson. *Achieving Reliability using Behavioural Modules in a Robotic Assembly System.* Unpublished PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Yin *et al.* 84] B. Yin, A.P. Ambler, and R.J. Popplestone. Combining vision verification with a high level robot programming language. In *Proceedings of the 4th International Conference, ROVISEC*, pages 371–379, 1984.

[Yuret 94] Deniz Yuret. From genetic algorithms to efficient optimization. Unpublished M.Sc. thesis, Department of Electrical Engineering and Computer Science, MIT, May 1994.

Appendix A

Random Numbers

At various points in the work that has led to this thesis computer generated pseudo-random numbers have been used to generate simulated data and as part of operational software. In both types of application it was important that the generated values be as random as possible, and have a long, if not infinite, cycle time. It is widely accepted that the standard generators provided with the C compilers used (SunOS 4.1 cc and GCC 2.4) during the work described in this thesis are not particularly strong with respect to these criteria [Press *et al.* 90, Maddox 94]. Despite von Neumann's comment that "Anyone who considers arithmetical methods of producing random numbers, is of course in a state of sin" (quoted from [Knuth 81]), a variety of pseudo-random number generators have been developed that can effectively produce pseudo-random numbers with the desired statistical characteristics and operational properties (see [Knuth 81] and [Deley 91] for descriptions of tests to determine randomness). The four generators that have been used are:-

ran0 A linear congruential generator with additional randomising shuffling, based on the algorithm of Bays and Durham, implementation from Numerical Recipes [Press *et al.* 90].

ran1 Returns a value derived from combining the output of three separate linear congruential generators through a shuffling routine. The implementation is a Numerical Recipes [Press *et al.* 90] program based on an algorithm by Knuth [Knuth 81].

ran3 A generator based on a subtractive method algorithm developed by Knuth [Knuth 81]. Although this algorithm is similar to **kn_random**, the implementation, like that of **ran0** and **ran1** is taken from Numerical Recipes [Press *et al.* 90].

kn_random An implementation of portable subtractive pseudo-random number generator taken from [Goldberg 89a], and based on an algorithm described by Knuth [Knuth 81].

These four algorithms all generate uniform random deviates in the range 0.0 to 1.0. They can all can be initialised under user control. Where other distributions have

been required, then the output of one (or more, depending on the nature of the target distribution) of the generators was transformed appropriately.

In addition to the four generators, one further source of random numbers was occasionally used. If no microphone is connected to the relevant port on a Sun Sparc workstation, then the output from the port, read with the appropriate device driver, is random noise. By recording that noise over a period of time and then compressing the resultant file a sequence of random bytes is generated. This file is then read to obtain random numbers by loading the appropriate number of bytes required for the representation of that number.

The outputs of all of the generators and the audio source have been tested using appropriate statistical measures to ensure that they conform to the expected distribution. All of the generators have been available publicly for a number of years without any significant problems being reported.

Appendix B

Contact Position from Force Readings

This appendix provides the mathematical detail of how to calculate the position of an object resting on the surface of the force table (Section 4.2.4, page 79) from the forces and torques that result.

This 6 element vector containing the forces and torques measured by the sensor at time i is $\mathcal{F}_i^m = [fx, fy, fz, tx, ty, tz]^T$, where fx, fy and fz are the forces, tx, ty and tz are the torques and ^{T} indicates the transpose operator. A superscript m or a on a component of a vector, or the vector indicates whether that component or vector is a measured or applied force. The position in space where those forces are being exerted at time i is $\mathcal{P}_i = [x, y, z, rx, ry, rz]^T$, with x, y, and z being the position, and rx, ry and rz being the rotation, with $\mathcal{F}_i^a$ being the actual force exerted at $\mathcal{P}_i$. The point of exertion and the point of measurement can be assumed to be rigidly linked. The measured force and the exerted force are related by(From [Craig 89]):

$$\mathcal{F}_i^a = \mathbf{J}_a^m \mathcal{F}_i^m \tag{B.1}$$

where $\mathbf{J}_a^m$ is the *force-moment transformation*:

$$\mathbf{J}_a^m = \begin{bmatrix} R_a^m & \mathbf{0} \\ P_a^m \times R_a^m & R_a^m \end{bmatrix} \tag{B.2}$$

where R_a^m is a 3 by 3 matrix describing the rotation between the measurement point, and P_a^m being the 3 by 3 matrix describing the translation between the measurement points and $\mathcal{P}$:

$$P_a^m = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{B.3}$$

where x, y, and z are the first 3 components of $\mathcal{P}$.

The problem of localisation is determining the values of x, y and z from $\mathcal{F}^m$. If the following simplifying assumptions about position and the force are made:-

- During localisation there will be no force or torque exerted on the object.
- During localisation the object will be sitting on the bed of the tray, which is at a known height above the measurement point of the FT sensor.
- The measurement coordinate system of the FT sensor and the coordinate system of the tray are not rotated relative to one another about any axis.

then it can be stated *a priori* that z is known and constant, that rx, ry and rz are equal to 0, that tx^a and ty^a are equal to 0, and that

$$R_a^m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{B.4}$$

Equation B.2 now simplifies to:

$$\begin{bmatrix} fx \\ fy \\ fz \\ 0 \\ 0 \\ tz \end{bmatrix}_i^a = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -z & y & 1 & 0 & 0 \\ z & 0 & -x & 0 & 1 & 0 \\ -y & x & 0 & 0 & 0 & 1 \end{bmatrix}_a^m \begin{bmatrix} fx \\ fy \\ fz \\ tx \\ ty \\ tz \end{bmatrix}_i^m \tag{B.5}$$

Subsequent rearrangement provides us with the desired solution, namely estimates of the x and y positions of the current point of application:

$$x_i^{e\,1} = \frac{fx_i^m z_i + ty_i^m}{fz_i^m} \tag{B.6}$$

$$y_i^e = \frac{fy_i^m z_i - tx_i^m}{fz_i^m} \tag{B.7}$$

These two equations have been used to convert the sensed forces into a contact position throughout this thesis. It should be noted that the results are only valid for as long as the assumptions hold; should a non zero x and/or y torque be introduced into the applied force, then the position can no longer be accepted. It should also be noted that these equations, and workings from which they were derived, are independent of the units in which force is measured.

---

[1] the e superscript indicate that a quanity is an estimate

Appendix C

Filtering Noise

The two subjects covered in this appendix are the two different methods by which the estimates of contact point produced from the Force Table (Section 4.2.4, page 79) are improved. All the techniques described here are standard.

Handling Noise 1 — Using an Averaging Filter

The Lord FT sensor incorporates a moving-window averaging filter of adjustable length in the processing electronics. This filter, and the size of the window can be selected under software control. The output of the sensor after filtering is now the result of averaging the current reading with the previous $n-1$ readings, where n[1] is the window size. This has the obvious advantage of being very simple to implement, with no extra code having to be developed. The only problems are that using the filter reduces the sample rate to around 50Hz, and more significantly, it introduces a lag into the system. If $n = 16$, then this lag is around one third of a second. For this application this is not significant as only static objects are being used. For a more time critical applications, such as surface following then this lag would introduce serious problems.

Figure C.2 shows the results of repeating the experiments that generated Figure 4.5, but this time using an averaging filter with a window length of 16. Visual comparison to Figure 4.5 provides an indication of the effectiveness of this technique. Table C.1 summarises a more exhaustive series of experiments where position estimates were calculated for all of the window sizes. This data was gathered in a number of identical experiments in which the same object was placed into one of the corners of the table, and the same number of samples (101) gathered over approximately the same time. As well as the mean Euclidean error (μ_d) and standard deviation (σ_d), the mean errors(μ_x and μ_y) and standard deviations (σ_x and σ_y) were calculated separately for each axis. It is these results which are displayed in the table.

This experiment was repeated at a number of locations and with the results being compared by calculating their relative rankings using a one tailed T test at the are shown in Table C.2, with rank 1 being the best.

[1] Admissible values of n are 0(no filter), 1, 2, 4, 8 and 16

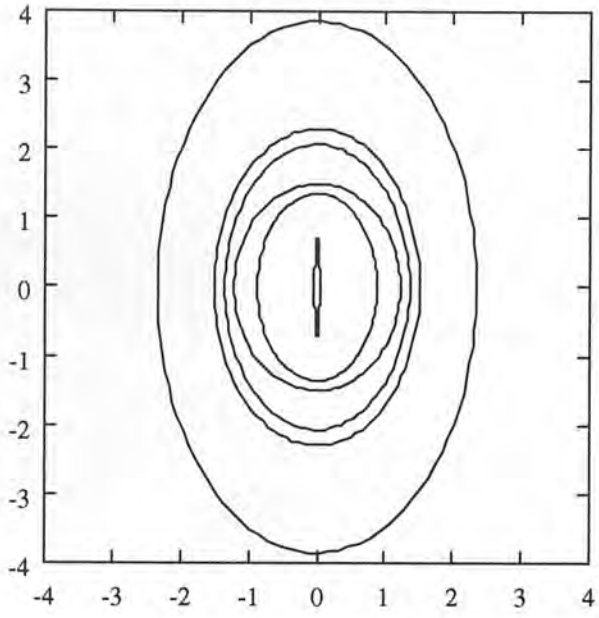

Figure C.1: Performance of contact point estimation using Averaging Filter

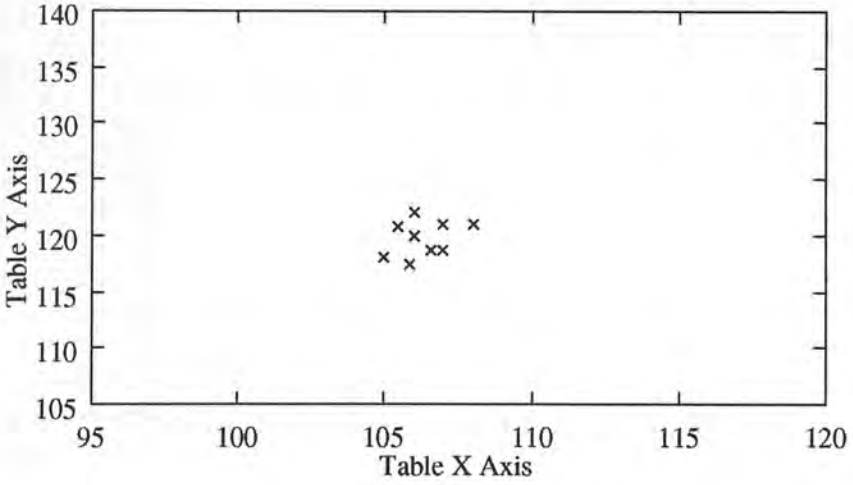

Figure C.2: Position estimates over 1s for a stationary object using an averaging filter with a window length of 16

	Statistic					
n	μ_x	σ_x	μ_y	σ_y	μ_d	σ_d
0	0.02	5.16	0.09	2.58	3.98	81.04
1	0.02	3.83	0.09	2.52	3.14	40.25
2	0.02	3.46	0.08	2.41	2.33	29.02
4	-0.01	1.65	0.07	1.86	2.26	27.51
8	-0.01	1.21	0.07	1.42	1.32	21.63
16	0.0	0.47	0.04	1.37	0.63	8.31

Table C.1: Statistics on position estimates for various sizes of moving-window average filter

n	Experiment								Mean Rank
	1	2	3	4	5	6	7	8	
0	6	6	5	2	3	6	6	3	4.625
1	5	5	4	3	6	5	1	6	4.327
2	3	3	6	6	4	4	2	4	4
4	3	2	3	4	1	3	3	1	2.5
8	2	3	2	5	5	2	5	5	3.625
16	1	1	1	1	2	1	4	2	1.625

Table C.2: Rankings of effectiveness of moving-window average filter for various experiments

From Table C.1 and Table C.2 it is readily apparent that, as could be expected, a larger window results in a more accurate estimate. However, this approach is not infallible. Comparing the different experiments in Table C.2 shows that the performance ranking is not fixed, with poor performances caused by disturbances to the table during an experiment. These are caused by environmental noise, typically mechanical vibration, affecting the table, a frequent occurrence when the table is placed within the workcell because of active robots, etc. At present the slab of polystyrene below the sensor provides some mechanical isolation. Further isolation might be achieved, but it would not be possible to guarantee that for each experiment then averaging filter would provide high quality estimates.

The low values of μ_x and μ_y compared to μ_d in Table C.1 suggests that the position estimates are normally distributed around the true position of the object. If so, then one method that could be used to obtain the true position might be to simply take the mean of a number of estimates. A number of experiments with this approach suggested that held promise, especially if it were extended to using a moving average rather than a simple running mean, as the latter could be unduly influenced by the occasional very incorrect estimate. This introduces an extra memory overhead to the system, and requires the user to select an appropriate window size. Since the ideal window size is application dependent, and so unknown without experimentation, a Kalman filter has been used to refine the position estimates. This approach is discussed further in the next section.

Handling Noise 2 — The Kalman Filter

To describe Kalman Filters as "a glorified weighted averaging technique"[Orr 92] is to neatly summarise a provably optimal estimation mechanism that provides an effective solution in a wide range of applications, many of which violate the underlying assumptions of the filter. The filter is a numerical tool developed in the early 1960's[Kalman 60, Kalman 63], which calculates how to change the parameterisation of a user supplied model of some process so as to minimise the expected innovation, or the difference between the expected next sample of the process and the estimate produced by the model of that sample. The closer the model matches to the real process, then the better the results of the filter. Following an appropriate initialisation, and given enough observations, the filter will be able to deliver an estimate of the true state

of the real process, though it must be presupplied with the variance (and covariance) of the noise of the measurement process. The filter mechanism assumes that the noise is independent and normally distributed, but in practice these last two assumptions have turned out to be relatively unimportant. If they are violated, the filters still work, but with the final results becoming dependent on the initialisation procedures.

Here a constant position model of the localisation process has been used, with the measurement equation for the differences between the estimate and the measured position at a particular time being the vector function:

$$\mathbf{f}(\mathbf{x}, \mathbf{a}) = (\mathbf{x} - \mathbf{a})^n \tag{C.1}$$

where $\mathbf{x} = [x^{s2}, y^s]^T$ is the sampled position and $\mathbf{a} = [x^e, y^te]^T$ is the estimated position. The Jacobians of Equation C.1, which will be required later on, are:-

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} n\,(x^s - x^e)^{n-1} & n\,(y^s - y^e)^{n-1} \end{bmatrix} \tag{C.2}$$

and

$$\frac{\partial f}{\partial \mathbf{a}} = \begin{bmatrix} -n\,(x^s - x^e)^{n-1} & -n\,(y^s - y^e)^{n-1} \end{bmatrix}. \tag{C.3}$$

Increasing the order of Equation C.1 will reduce the filter's sensitivity to noise, as the higher order models correspond to a reduced certainty about the accuracy of any measurement. For any $n > 1$, an *Extended Iterated Kalman Filter* (IEKF) mechanism must be used. The resulting filter, shown in Algorithm 9, is derived from a linearisation of Equation C.1, and in the event that $n = 1$, it takes the form of a standard Kalman filter. In this algorithm *get_estimate*($\mathbf{x}$) is a procedure which obtains the current estimated contact position, and the $INITIAL\_STATE\_COVARIANCE$ and the $MEASURMENT\_COVARIANCES$ are matrices describing the initial state error to be expected when the filter is initialised with a sample, and the error associated with each measurement. The latter value remains static throughout the filter's life. In practice the filter is run for a set number of iterations and the estimated contact point at the end of the processing is taken as the true point of contact. The actual implementation is more complicated than shown in Algorithm 9, with many of the operations expanded out and combined to reduce the processing overheads.

The effects of using the filter can be seen in Figure C.3 where the dark labelled line shows the track of the basic Kalman filter when applied to the sequence of position estimates depicted in Figure 4.5. These estimates were obtained from a data log generated at time of the experiment.

The convergence of the filter onto the target point can be seen in Figure C.4, where circles representing the state covariance at 1, 2, 4, 8, and 16 iterations are shown, each iteration producing a smaller circle. Compared to the equivalent representations of the

[2] an s superscript indicates that a quantity is sampled from the environment

Algorithm 9 Extended Iterated Kalman Filter

$\mathbf{S} := INITIAL\_STATE\_COVARIANCE$
$\mathbf{\Lambda} := MEASURMENT\_COVARIANCE$
$get\_estimate(\mathbf{x})$
$\mathbf{a} := \mathbf{x}$
while $TRUE$ **do**
  $get\_estimate(\mathbf{x})$
  $\mathbf{W} := \frac{\partial f}{\partial \mathbf{x}} \Lambda \left(\frac{\partial f}{\partial \mathbf{x}}\right)^T$
  $\mathbf{M} := \frac{\partial f}{\partial \mathbf{a}}$
  $\mathbf{K} := \mathbf{S}\mathbf{M}^T(W - \mathbf{M}\mathbf{S}\mathbf{M}^T)$
  $\mathbf{a} := \mathbf{a} - \mathbf{K}f(\mathbf{x}, \mathbf{a})$
  $\mathbf{S} := (\mathbf{I} - \mathbf{K}\mathbf{M})\mathbf{S}$
end while

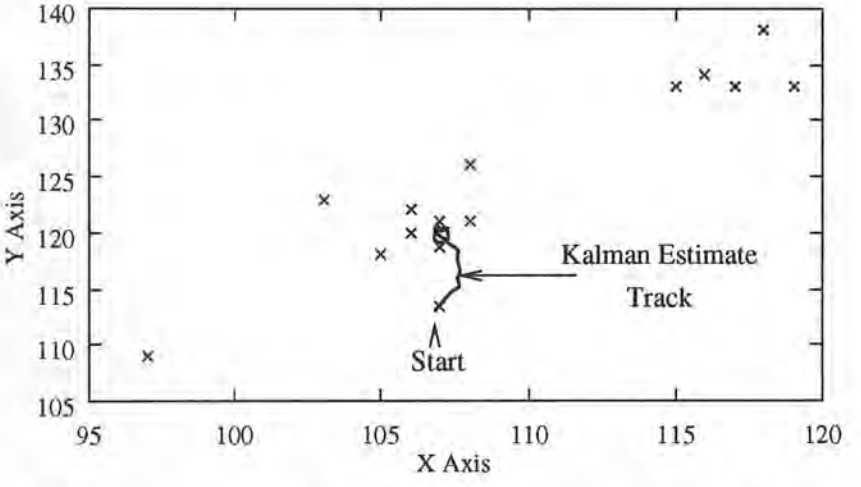

Figure C.3: Position estimates over 1s for a stationary object using the basic Kalman Filter

accuracy of the inbuilt filter in Figure C.1 for the equivalent number of samples it is possible to be more certain about the location of the actual point of contact.

The operation of the Kalman filter can be viewed as first deciding on the position of its embedded model in the space defined by the measurements (in this case the space is that of the possible positions on the table), and then fine tuning that initial position so that the filter's output rapidly converges towards the true state, *i.e.* the actual position of the object. During this convergence there are occasional very noisy localisation estimates given to the filter. The effect of these is to drag the Kalman estimate temporarily away from its convergence to the true state. Such measurements are in excesses of the level of noise expected by the filter. The rest of this section describes how the *Mahalanobis Distance*[Mahalanobis 48] has been used determine the *gate size* for the filter at any point in time. Only if a measurement is inside the gate then is it used to update the current Kalman estimate. Mahalanobis distances are a widely used metric for determining gate size in Kalman filtering. Unlike alternative measurements such as the Penrose distance[Penrose 53], in using the covariance the

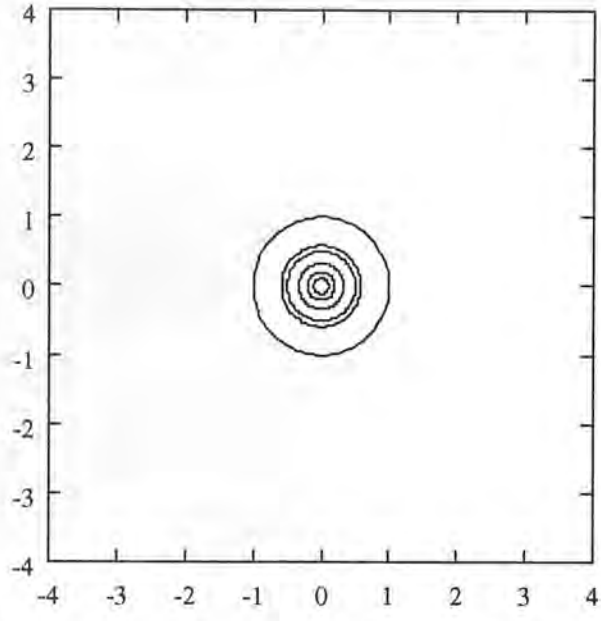

Figure C.4: Performance of contact point estimation using a Kalman Filter

Mahalanobis distance takes into account the independence (or lack of) of the component variables, so result of the calculation can be assessed directly against the relevant statistical distribution for assessment.

The gate size is calculated by determining the Mahalanobis distance between the current sensed value and the predicted value. If that distance is not less than ϵ, the value of a χ^2-distribution with p degrees of freedom at the specified confidence level where p equals the number of components in the sensed value, then that sensed value is considered to be outside the gate. The Mahalanobis distance is calculated using[Manly 86]:

$$d_i^m = \sqrt{\mathbf{e}^T \mathbf{C}^{-1} \mathbf{e}} \tag{C.4}$$

where **e**, the innovation (or prediction error), is the difference between the current estimate and current reading and **C** is the variance of **e**:-

$$\mathbf{C} = \frac{\partial f_i}{\partial x_i} \Lambda_i (\frac{\partial f_i}{\partial x_i})^T + \frac{\partial f_i}{\partial a_i} \mathbf{S}_{i-1} (\frac{\partial f_i}{\delta a_i})^T \tag{C.5}$$

The modified Kalman Filter algorithm using Mahalanobis distance gating is shown in Algorithm 10. In addition to the described algorithm, the implementation was extended to monitor the frequency of gating failures. When this frequency exceeds a user defined threshold then the filter is re-initialised and restarted. Introduction of gating and the associated reset has introduced two extra control parameters to the filter; the confidence level, and the reset threshold. In practice

There are two situations where this facility is an advantage:-

- The object on the table has moved.

Algorithm 10 Extended Iterated Kalman Filter with Mahalanobis Distance Gating

$\mathbf{S} := INITIAL\_STATE\_COVARIANCE$
$\mathbf{\Lambda} := MEASURMENT\_COVARIANCE$
$get\_estimate(\mathbf{x})$
$\mathbf{a} := \mathbf{x}$
while $TRUE$ **do**
  $get\_estimate(\mathbf{x})$
  if $d_i^m < \epsilon$ **then**
    $\mathbf{W} := \frac{\partial f}{\partial \mathbf{x}} \Lambda \left(\frac{\partial f}{\partial \mathbf{x}}\right)^T$
    $\mathbf{M} := \frac{\partial f}{\partial \mathbf{a}}$
    $\mathbf{K} := \mathbf{S}\mathbf{M}^T(W - \mathbf{M}\mathbf{S}\mathbf{M}^T)$
    $\mathbf{a} := \mathbf{a} - \mathbf{K}f(\mathbf{x}, \mathbf{a})$
    $\mathbf{S} := (\mathbf{I} - \mathbf{K}\mathbf{M})\mathbf{S}$
  end if
end while

- The filter has been incorrectly initialised.

If a gate is not being used, then over time the filter would converge to the correct value, regardless of which of the two above error conditions held. However, this might take a long time! By using a gate to discard inappropriate measurements, and a threshold on the number of discarded measurements tolerable before resetting the filter we can rapidly recover from either of these two errors.

In Figure C.5 two different gates are shown for a particular Kalman estimate, along with a number of localisation estimates. If the estimates are within the selected gate, then the filter is updated using that estimate. If they are outside the gate they are ignored as being irrelevant to this particular instantiation of the embedded model. The values attached to the two gates refer to the respective statistical certainty that when deciding to accept a localisation estimate you are not making a Type 1 error, rejecting the null hypothesis that the current sample is close enough to the current estimate.

Figure 4.6, back on page 82, shows the track of the new Kalman Filter when applied to the position estimates depicted in Figure 4.5. As with Figure C.3, these estimates were obtained from the data log generated when the relevant experiment was performed. This particular track was developed with a confidence level of 95% and a reset threshold of 10. When compared to the track in Figure C.3, the new filter produces a far more stable output. In operation analogies can be made between the new filter and visual aspect graphs. In the early stages the filter decides which of a small range of aspects best suits its particular model. Once decided then it generally ignores data that might suggest different aspects, though if the evidence for it being in the wrong aspect becomes overwhelming, it will restart itself.

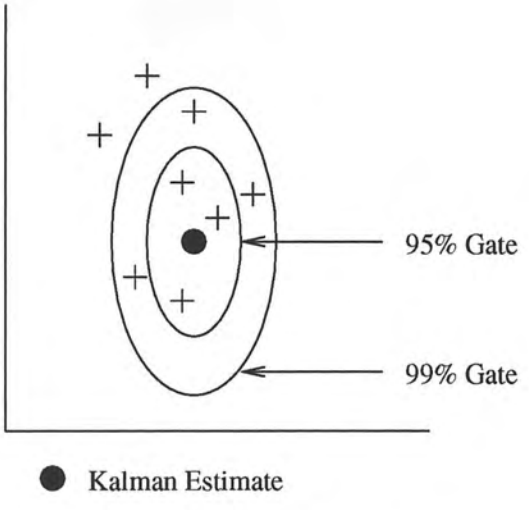

Figure C.5: Example of different gate sizes