Robotic Hand-Eye Motor Learning

Thesis by

Carl Frederick Ruoff

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1993

(Submitted on January 4, 1993)

Acknowledgements

I am deeply indebted to Fred Culick, my advisor, Arden Albee, the Dean of Graduate Studies, and the management of the Jet Propulsion Laboratory, including Lew Allen, the former director, Clarence Gates, the former associate director, Moustafa Chahine, the chief scientist, Rody Stephenson, deputy assistant laboratory director, and Giulio Varsi, deputy technology program manager, for making it possible for me to attend Caltech. I would also like to express my gratitude to the members of my thesis, committee Allan Acosta, Joel Burdick, Christof Koch, and Carver Mead, who also served as my advisor in Computer Science, for their continued academic support and friendship.

In addition, I gratefully acknowledge the financial support provided by NASA through JPL, and the supercomputing support provided by the JPL/Caltech Concurrent Supercomputing Consortium, and would like to acknowledge the assistance of Robert Mackin, who helped me focus my research goals, Larry Matthies and Don Gennery, who helped in modeling computer vision error sources, and Abhi Jain, who helped me navigate the pitfalls of Latex.

Finally, I would like to thank my wife, Nancy, and my children Gina, Melinda, Andrew, and Nick, for being tolerant and understanding.

Abstract

This thesis investigates the use of neural networks and nonlinear estimation in robotic motor learning.

It presents a detailed experimental investigation of the performance and parametric sensitivity of resource-allocating neural networks along with a new learning algorithm that offers rapid adaptation and excellent accuracy. It also includes an appendix that relates feedforward neural networks to familiar mathematical ideas.

In addition, it presents two learning hand-eye calibration systems, one based on neural networks and the other on nonlinear estimation. The network-based system learns to correct robot positioning errors arising from the use of nominal system kinematics, while the estimation-based system identifies the robot's kinematic parameters. Both systems employ the same two-link robot with stereo vision, and include noise and various other error sources. The network-based system is robust to all error sources considered, though noise naturally limits performance. The estimation-based system has significantly better performance when the robot and vision systems are well modeled, but is extremely sensitive to unmodeled error sources and noise.

Finally, it presents a robot control system based on neural networks that learns to catch balls perfectly without requiring explicit programming or conventional controllers. It uses only feedforward pursuit motions learned through practice, and is initially incapable of even moving its arm in response to external stimuli. It learns to identify and control its pursuit movements, to identify and predict ball behavior, and,

with the aid of advice from a critic, to modify its movement commands to improve catching success. The system, which incorporates information from visual, arm state, and drive force sensors, characterizes control situations using input/response pairs. This allows it to learn and respond to plant variations without requiring parametric models or parameter identification. It achieves robust execution by comparing predicted and observed behavior, using inconsistencies to trigger learning and behavioral change. The architectural approach, which involves both declarative and analog knowledge as well as short- and long-term memory, can be extended to learning other sensor-motor skills like mechanical assembly and synchronizing motor actions with external processes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives and Motivation

In recent years there has been an explosion of interest in the computational aspects of behavior: How do organisms acquire, represent and process the information necessary to control and regulate their activities, and how do they learn to modify their behavior to improve or maintain performance in response to internal and external changes? This interest has been stimulated in part by scientific curiosity and in part by a widespread desire to improve the performance of robots and other automated systems [Churchland (1986), Sejnowski, Koch and Churchland (1988), Arbib (1987), Churchland and Sejnowski (1988), Lisberger (1988), Wise and Desimone (1988), Pomerleau (1990)].

Since it is generally held that biological computation is accomplished primarily

in animals' nervous systems,[1] there is great interest in exploring and modeling this computational activity at various levels of detail. There has been particular interest in the computational, dynamical, and learning properties of networks of simplified neurons [Grossberg (1982), Denker (1986), Rumelhart and McClelland (1986), Cronin (1987), Amit (1989), Koch and Segev (1989), Touretzky (1989)]. These are often called connectionist models because they consist of interconnected networks of relatively simple computing elements. They have promising capabilities, and a theoretical understanding of their properties is developing rapidly. Investigators have begun implementing neural networks in hardware [Mead (1989)], and have explored their application to a variety of problems in motor learning, including robot hand-eye coordination and the learning of simple skills [Kuperstein (1987b), Kuperstein (1988), Miller (1987), Goldberg and Pearlmutter (1988), Mel (1989)].

The overall objective of this research, part of a long-term investigation into robotics, is to explore robotic motor learning and control using computational and control elements based on neural networks and other biological ideas. It is motivated by a personal fascination with motor learning mechanisms and the desire to build robots that can learn to exhibit interesting, skilled behavior. It is inspired by biological models of motor learning and neural function. The intent is to explore a framework that will allow robots to learn and maintain motor skills automatically without requiring that they be explicitly programmed.

Specific objectives of this work are to design, simulate, and test a robot hand-eye coordination system that learns to correct kinematic positioning errors, and to design and simulate a robot control system capable of learning primitive behaviors as well as sensor-motor tasks, evaluating its performance using ball catching as a model task.

In hand-eye coordination the problem is to be able to use visual information to

[1]Systemic biochemistry is also involved, establishing a context for neural computation.

generate accurate feedforward commands that position the chosen point on a robot hand at the point of visual attention. This is a nontrivial problem because of errors and nonlinearities in the robot, the vision system, and the camera pan-tilt mechanism that are functions of loading, temperature, and machine wear. It has historically been an irritating issue in robot applications [Ruoff (1980), Gennery et al (1987)]. On-line learning algorithms have shown great promise in addressing this issue because, by their nature, they generate corrections based on actual performance errors, and they can accommodate slow drifts in system characteristics.

In motor skill learning, the problem is to learn to perform primitive motor behaviors (pursue, track, push, pull, comply, etc.) and to learn to generate the sequences of motor commands and expected sensory events (sensor-motor sequences) necessary to accomplish particular tasks. Learning and executing a motor skill involves: learning to recognize, verify, and predict significant events and the behavior of external objects; synchronizing internal and external activity; recognizing task situations so the appropriate sequences are invoked; and correcting command sequences and their timing so the task is accomplished.

In addition to being scientifically interesting, the study of motor learning has important practical applications. Giving robots and other man-made systems such a capability can significantly improve their capacity for autonomous operation, enabling a much wider variety of applications than is now possible. It has the potential to improve the currently rather poor performance of robots in dynamic situations, and to allow compensating for drifts in system characteristics. Motor skill learning can also make robots substantially easier to use since the programming burden can be reduced.

This work has involved developing neural representations and computational mechanisms for sensing, memory, and control elements; investigating the structure

and performance of feedforward neural networks and learning algorithms; and developing detailed simulations of realistic physical plants.

An attempt has been made to develop learning and control structures that have some biological plausibility. The understanding of brain and neural function is still limited, however, and since the main objective of this work is to investigate mechanisms that can improve the performance of man-made systems, serial and connectionist computational paradigms have been freely mixed.

1.2 Results and Contributions of This Thesis

The principal contributions of this thesis are:

1. A detailed experimental investigation of the structure, performance and parametric sensitivity of feedforward resource-allocating neural networks, the development of a new learning algorithm that offers rapid adaptation and can be implemented relatively easily in hardware, and a description of feedforward neural networks that relates their capabilities to familiar mathematical ideas.

2. The design, simulation and evaluation of two learning hand-eye calibration systems, one based on neural networks and the other on nonlinear estimation, that can form the basis for practical hand-eye calibration in real robots. The neural system learns to correct robot positioning errors that result from using the nominal system kinematics, while the system based on nonlinear estimation identifies the robot's kinematic parameters. System simulations involving a two-link robot with stereo vision included the effects of parameter drifts, visual measurement noise, kinematic parameter errors, encoder offsets, and nonlinearities. The neural system is robust to all of the error sources considered, though

noise naturally limits performance. The estimator-based system is faster and significantly more accurate than the neural system where the robot and vision system are well modeled, but is extremely sensitive to unmodeled error sources and noise.

3. The design, simulation, and evaluation of a robot motor learning system that is extremely successful at learning to catch balls without requiring explicit programming or conventional controllers. It learns to identify and control its pursuit movements, to identify and predict ball behavior, and to modify its movement commands through trial and error to improve catching success. The system, which employs realistic, detailed physical models of the arm, the ball, and their physical interaction, employs an architectural approach involving both declarative and analog knowledge as well as short- and long-term memory that can be extended to learning sensor-motor skills like mechanical assembly and synchronizing motor actions with external processes. The short-term memory, which stores both predictions and observations, is temporally ordered. The system characterizes control situations using input/response pairs, which allows it to learn and respond to plant variations without requiring parametric models and parameter identification. It achieves robust execution by comparing predicted and observed behavior, using inconsistencies to trigger learning and, as a consequence, behavioral change.

1.3 Previous Work

Previous work on robot hand-eye calibration has focused either on learning the entire hand-eye map, or on learning the geometrical coordinate transformation from visual to manipulator coordinates. Learning the entire hand-eye map is computation-

ally intensive and requires many learning cycles to achieve reasonable performance [Kuperstein (1987b), Kuperstein (1988), Mel (1989)]. Learning a linear transformation matrix is relatively easy to implement, but does not handle local anomalies. The approaches reported here learn to correct deviations from nominal robot kinematics, which is known from the structure of the robot. They therefore start with reasonable positioning accuracy and improve as the robot system operates.

Much of the previous work on robot motor learning has addressed the problem of learning to compensate for manipulator dynamics [Albus (1972), Albus (1975b), Albus (1975a), Atkeson (1986), Goldberg and Pearlmutter (1988), Miller (1987), Raibert (1977), Raibert and Horn (1978)]. More recent work has sought to extend learning from motion control to the task domain [Handelman, Lane and Gelfand (1989)]. That work has also used declarative knowledge to guide the learning process, and has tended to focus on particular motions and on well-defined repetitive events such as learning to hit a ball that is dropped on a particular point.

The research reported here investigates a more global framework for sensory perception and motor learning, based, in part, on learning by trial and error, that can be used for a variety of sensor-motor tasks including those in which the system must interact with external objects. It includes a memory of the motor commands and observed behavior in the current training episode and can learn to identify, predict, and respond to control situations, and the behavior of external objects. Basic design principles are that robustness is achieved by comparing predicted with observed behavior and that, at least from a motor control standpoint, a system comprehends a situation when it can reliably predict how it will evolve.

1.4 Thesis Overview

Chapter 2 addresses the motivation for learning motor control, describing current robotic capabilities and limitations as well as general features needed by competent robotic systems. It introduces the motor learning problem and describes the detailed scope of this research along with the approach taken to learning control.

Chapters 3 and 4 address hand-eye calibration and robot motor learning respectively. They describe the learning approach, representations, controllers, system architecture, and other system elements and evaluate system performance.

Chapter 5 concludes by comparing this work with previous work, drawing technical conclusions, and identifying extensions and future work.

Appendices A and B respectively give a brief introduction to neural networks and discuss the VLSI implementation of adaptation algorithms studied in this work. Appendix A also provides an intuitive motivation for feedforward neural networks, relating them to familiar mathematical ideas.

1.5 Notation

In this thesis vectors are in bold-faced type and vector magnitudes are in plain type. Vector components are indicated in plain type with subscripts.

Chapter 2

Background

2.1 Introduction

This chapter discusses learning control in robotics. The motivation for learning control and a more biologically-inspired approach to robotics is outlined, followed by a description of the motor learning and control problem.

2.2 Robotics

Robotics is becoming increasingly important in automating the production of goods and services and as an enabling technology for activities such as unmanned planetary surface exploration and operations in hazardous environments [Varsi et al (1992)]. Robotics is also extremely important as a military technology [Ruoff (1984)], and

has an enormous potential consumer market.

Industrial robots are extremely effective at repetitive manufacturing tasks involving positioning tools and workpieces. As a consequence, thousands are in daily use worldwide performing tasks like stuffing circuit boards and welding automobile bodies. If properly programmed, they can respond to inputs from various types of sensors including limit switches, force sensors, and computer vision systems, but they offer little in the way of graceful force and fine motion control or the ability to sense and adapt to unexpected circumstances. They are therefore limited to well-structured situations, and perform poorly on dynamic tasks unless both the control system and task are carefully engineered [Andersson (1988)].

A significant body of recent research has been devoted to improving robotic intelligence [Waldrop (1988), Laird et al (1991)], planning capabilities [Hutchinson and Kak (1990), Popplestone, Liu and Weiss (1990)], and control [Slotine and Li (1986), Craig (1986a), Larkin (1993)]. This research has had positive results, yet robot performance remains limited, and setting up a robot to perform a task is an elaborate process involving designing and fabricating tools and fixtures, configuring the robot and its work environment, calibrating the robot and sensor systems, and programming [Engelberger (1980)].

Configuring a robot and its work environment requires positioning tools and fixtures, calibrating the robot sensor systems, and so on. Vision calibration, for example, generates the hand-eye transformations that are necessary to compute arm coordinates corresponding to the positions of objects in the visual field and vice-versa. Generating these transformations involves positioning an object held in the robot hand at several, perhaps many, positions that can be seen by the vision system, and calculating both the visual and arm coordinates of the object at each position. These coordinates are used to generate the required transformations

[Ruoff (1980), Hayati (1990), Gennery et al (1987)]. This process, which usually requires an accurate or precalibrated arm and fixtures of various sorts, is cumbersome and yields transformations that may have local inaccuracies due to nonlinearities in the vision system and other elements. It is also difficult to make work well in systems that have kinematically complex camera platforms and arms. Finally, calibration parameters are sensitive to thermal and mechanical drift, requiring frequent recalibration.

Task programming is usually done in one of the following ways:

1. The robot is led through the task sequence, which is recorded and then repeated slavishly with, perhaps, some coordinate adjustment. This approach is called teach by showing. It is not suited for robots that must respond to changing circumstances and will not be discussed further.

2. The robot program, including branch points, calculations, sensor input requests, and control parameters, is explicitly coded using macros, subroutines, and other constructs, perhaps using the arm itself to acquire necessary coordinate frames [Ruoff (1980), Backes and Tso (1990), Backes (1991), Backes (1992a), Backes (1992b)]. Programs at this level usually involve sequences of commands that invoke pre-coded primitive robot behaviors like "push" and "move." Robots programmed in this way can perform useful tasks reliably, but explicit coding can be awkward, since robust explicit programs, that must necessarily involve extensive sensing and force control, can be extremely complex.

3. The robot program is generated using interactive planning programs that concatenate known primitive behaviors and routines, including sensor and perception routines, into task sequences. This is a powerful approach, used in the JPL Telerobot Testbed, that allows for very high-level task specification [Wilcox et al (1989)]. Planners, however, do not yet model dynamic or contact

interactions well, so detailed tuning of the resulting program is required, and low-level sensor and arm calibration must still be performed. In some cases, a robot system can generate its own plans if it can adequately identify the task situation. Autonomous vehicles such as Mars rovers are programmed in this way [Wilcox and Gennery (1987)]. High-level commands such as position targets are given; the system decomposes them in to motion commands that avoid sensed obstacles and other hazards.

After configuration and programming are complete, the robot application must be debugged and tuned. This involves modifying the program to accommodate unforeseen problems or modeling errors and adjusting control parameters, including stiffnesses and forces, for the various steps. If the program involves dynamic interactions such as tracking and capturing objects, tuning can be time consuming because it is necessary to ensure that the robot motion is synchronized with the moving object.

The setup process just outlined is cumbersome. It can be tolerated in manufacturing situations where setup costs can be amortized over large production runs, but it is unacceptable in situations such as maintenance, where runs are small or highly variable. It is totally unsuitable for situations, such as exploring rough natural terrain or making emergency repairs, where the ability to comprehend and respond quickly and adroitly to situations is critical.

In addition to being cumbersome to program and set up, robots also perform rather awkwardly, both at dynamic tasks like tracking and capturing moving objects, and in tasks involving dexterous manipulation and the gentle application of forces. Robots have been programmed to perform dynamic tasks like tracking and capturing spinning satellites [Wilcox et al (1989)] and playing ping-pong [Andersson (1988)], but there are significant constraints, the programs are not really robust, and they cannot be easily generalized. Current robots simply lack the agility, grace, and adapt-

ability required for versatile robust behavior in novel and dynamic situations. This stems largely from an inability to perform the perceptual and control computations necessary to comprehend new situations and to generate and apply good real-time dynamic models.

As a consequence, most of the work in robotics to date has assumed stiff quasi-static environments, and has relied on geometrical precision and kinematic positioning accuracy. Relying on kinematic precision is appropriate for creating geometry and for handling rigid manufactured objects, but it is a limited strategy. It is very difficult, for example, using geometrical and mechanical models, to predict and control the motion of an oddly-shaped object accurately as it is manipulated within multiple compliant fingers. Current robots are, in fact, primarily programmable positioning devices, and are very much like machine tools. To achieve positioning accuracy and stability with the simple fixed-gain controllers that are typically employed, robots must be stiff. Hence they are massive as well.

Animals have apparently approached the control problem in a very different way. Instead of relying on kinematic precision, which would surely be an irrelevant and disastrous survival strategy, evolution has traded off precision for speed and agility and have evolved the ability to process large amounts of sensory and control information. They rely upon being able to make quick situation assessments while rapidly modifying their behavior in response. The ability to learn about situations and predict behavior is critical for this. Rather than employing fixed gains, animals employ variable stiffnesses carefully matched to the situation [Brooks (1986)], and use muscle-tendon actuation which is backdrivable. This means they are less susceptible to collision damage and can use their limbs and extremities as fast, active sensing devices. Stiffnesses and other control parameters are learned and maintained automatically by low-level systems [Brooks (1986)]. Since they process a great deal of sensory information and must have some kinematic precision because of the need

to locate stimuli in space, animals also must deal with the sensor and limb calibration problem. They do so, however, with on-line adaptive systems that perform calibration automatically in a background mode as the animal behaves [Brooks (1986)]. It would be extremely useful to develop robot control systems that have similar capabilities for automatically acquiring primitive behaviors, maintaining their own sensor calibration, automatically learning motor tasks given high-level instructions, and generalizing previously learned capabilities to new situations.

2.3 Hand-Eye Calibration

Hand-eye calibration[1] is the process of determining the feedforward[2] arm axis position commands[3] that correspond to a robot's (or animal's) positioning its hand at (accessible) points it observes visually in its workspace. Accurate feedforward position commands are important in static tasks like grasping and for dynamic tasks like catching objects. This is true even when the hand position loop is closed using visual and tactile feedback because system bandwidth limitations may make it impossible to achieve adequate performance if feedforward position commands are inaccurate.

---

[1] Limb-eye or appendage-eye coordination might be more appropriate terms because the same process must occur for all appendages that must be positioned to visually-perceived locations in space.

[2] Here the term feedforward means that the hand is positioned by position servos without modifying the position *command* with visual, tactile, or force feedback. It does not imply that arm axis position feedback is not used in the axis position servo loops.

[3] Position commands are input to an axis position control system to cause the axis to assume the commanded position as measured by an axis position transducer.

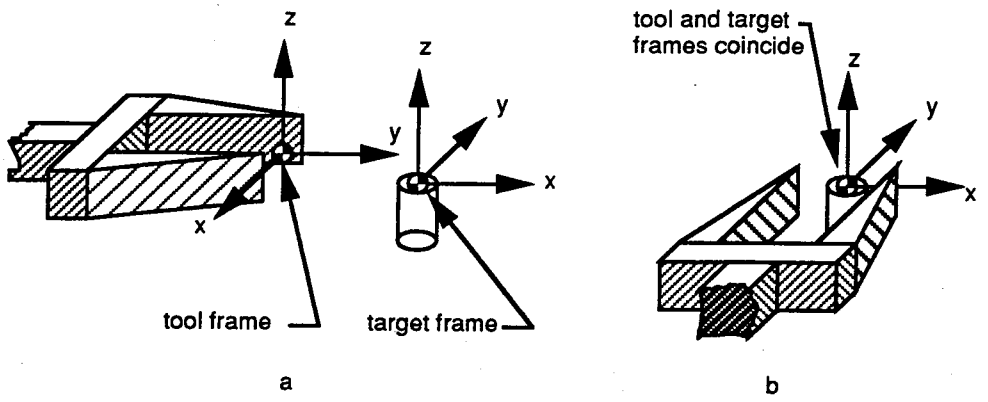

Figure 2.1: Robot hand positioning. In a) the robot hand is not positioned correctly relative to the cylindrical object. Positioning is correct in b) because the tool and target frames coincide.

2.3.1 The Hand-Eye Calibration Problem

Positioning a hand with respect to an object is equivalent to bringing a coordinate frame, fixed with respect to the hand (called the *tool* frame in robotics), into (or near) coincidence with a coordinate frame (the *target* frame) fixed with respect to the object. This is illustrated in figure 2.1.

Actually bringing the frames into coincidence can be accomplished by using accurate feedforward axis position servo commands or by closing a position command generation loop using a combination of visual, tactile, and force feedback. Feedforward position commands provide faster response, but their generation requires accurate knowledge of positions and kinematic parameters. Accurate feedforward commands improve performance even when feedback is used during the terminal phases of motions because uncertainty zones can be reduced.

Generating the axis position commands that bring the tool and target frames into coincidence involves transforming the position description of the target frame into a position description relative to the arm base frame and then applying the

arm's inverse kinematics [Ruoff (1980), Paul (1981), Craig (1986b)] to obtain the arm axis positions in a kinematically idealized joint space representation. The actual servo position commands are obtained from the idealized representation by including the effects of encoder offsets, scaling, and other characteristics. Positioning is then effected by sending the position commands as reference inputs to the position servos. Animals must accomplish something similar, but the details are not known [Brooks (1986)].

If a robot is reaching for a target on an object it sees rather than one for which the position is known, obtaining the target frame relative to the arm base frame will involve a (possibly nonlinear) transformation from visual coordinates. Since the visual frame is attached to the vision system, and the vision system is attached, in turn, to the vision pointing system, transforming object locations from visual coordinates to the arm base frame will involve the pointing system's kinematics and its geometrical relationship to the arm base frame. If the arm is redundant or has multiple solutions, the control system must provide an input that selects the appropriate solution. Noise arising from retinal image quantization is involved as well.

Position and kinematic errors seriously degrade robot performance since visual servoing or long guarded moves[4] are then required to avoid unexpected collisions and to reach target positions with sufficient accuracy.

In order to minimize feedforward position command errors, robot systems are calibrated both kinematically and visually. Kinematic calibration makes it possible for the arm to reach commmanded positions in space accurately, while visual calibration ensures that visual range and position estimates are accurate. Hand-eye calibration establishes the transformation from vision coordinates to the arm base frame. Together, the three types of calibration ensure that a robot can accurately

[4]Guarded moves are regions of slow compliant motion [Craig (1986b)].

reach accessible coordinate frames that it observes visually.

In current practice, robot hand-eye calibration is an off-line parameter identification technique that requires special fixturing [Ruoff (1980), Hayati (1990), Gennery et al (1987)]. It does not handle unmodeled local distortions like astigmatism, and it cannot accommodate system drifts due to wear, mechanical stress, and temperature changes unless the system is recalibrated, which involves taking it off-line. It has also been found to have limitations where active vision pointing platforms are employed because it is difficult to handle the entire kinematic chain from the visual system through the vision platform and the arm base to the tool frame. This forces the use of pre-stored and calibrated vision positions [Hayati (1990)].

It would be useful, then, to develop an on-line calibration system that observes the behavior of the hand-eye system during operation and learns to generate accurate feedforward axis position commands

$$\theta = \theta(s, t, p, v), \tag{2.1}$$

for s, the particular arm solution class, t, the relevant tool frame, p, the vision pointing system configuration, and v, the observed target frame in visual coordinates. An on-line hand-eye calibration system should be able to handle the kinds of robot system drifts mentioned above without being taken off-line.

Two related, but distinct, problems are 1) predicting the vision pointing coordinates corresponding to a location in space and 2) estimating an object's size and Cartesian coordinates. These are important problems that involve the internal representation of space and its relationship to motor command generation. The approaches described in this work can be applied to these problems as well, but they are not addressed here.

2.4 Motor Learning and Control

How do we learn to do things, and what happens when we do so? What is involved, for example, in learning to do things that are not obvious, like wiggling our ears? How does a young violinist learn to turn early screeches into mellow tones? How do we learn to touch an object that we can see?

Typically, when we desire to learn a particular task, we have an idea of what constitutes doing it successfully and what performance improvements are, but we don't know exactly what steps to take or what commands to issue. That is, we have some mental model of the task [Norman (1982)]. If we already know some related skills, we try them and we watch how they work. If things look good, we vary the commands we already know and remember them, continuing to do so until we are successful at performing the task. If we don't know related skills, or if nothing seems to work, we may resort to flailing, or trying random things until we notice that something we tried almost worked. We then try variations of that repeatedly until it works or until we decide it is hopeless. This kind of learning is conscious: We think about what we're doing and how to improve it. In effect, we become our own internal teacher or coach, watching and modifying our performance. Learning is much the same even if we have an actual coach. A coach makes thing easier by pointing out deficiencies and suggesting improvements, but we have to learn to incorporate the suggestions ourselves.

In learning to catch balls, for example, we usually already know how to reach for objects, what it means to catch something, how to modify position and timing of body commands to arrive earlier or later or at different places, and so on. A ball is thrown, we estimate its trajectory, generate an interception point, and move to try to catch it. If we miss, next time we start earlier or move to a slightly different point,

or make a better estimate of the ball's behavior. If the ball is not actively trying to evade us, and we are reasonably adept, this procedure will converge to a stable, robust skill if we are persistent.

If we do not already know how to perform basic or primitive behaviors, like reaching for objects, it is necessary to learn them. Developmentally, of course, learning basic behaviors occurs before learning complex ones. This learning, which is a form of adaptation, occurs automatically, that is, without conscious effort, as we move about in our normal activities [Brooks (1986)]. Hand- eye coordination and the vestibulo-ocular reflex, which causes head motion to be subtracted from eye motion so eye pointing is maintained under head disturbances are examples. Internal mechanisms always strive to keep these processes trimmed.

This work addresses the problem of learning basic motor skills of which large tasks, like building cities or space stations, are ultimately composed. Learning is considered here to be a relatively permanent change in system behavior leading to performance that is improved in some sense. Learning implies using previous experience to improve future performance in similar situations. In this context, improved motor skills will be taken to be [Norman (1982), Brooks (1986)] motor skills that are smoother, faster, and more precise, with less irrelevant activity.

2.4.1 Motor Tasks

A motor task is a hierarchical collection of actions that can be accomplished by invoking primitive skills. Invoking skills means accessing them, which means, in turn that the skills must have internal names or addresses.

If the world can be broken down into recognizable elements or situations, then

once a context is established, performing a task involves recognizing situations and applying the named skills.

Events and behaviors have measurable effects. It is the occurrence of these effects that comprises the occurrence of the event or behavior. If a task is proceeding as expected, or an external agent is behaving as expected, the effects will be as predicted. If not, the situation has changed, and the robot's behavior and knowledge must be adjusted accordingly.

Knowing how to do a sensor-motor task involves knowing, at a reflexive level:

1. what information is needed;
2. how to associate information with the task in the correct way (parameter binding);
3. where and when to look for the information during task execution;
4. what commands to issue;
5. how to associate and phase commands with the situation;
6. how the task is expected to evolve;
7. when it is impossible.

A real motor task is a concatenation of skills. Transitions between skills are skills themselves.

Learning motor skills requires trying [Brooks (1986)]. Trying is needed to build the association between the situation and the appropriate action.

2.5 Scope of This Work

This work is concerned with the problem of learning and executing motor behavior sequences in a rich sensory environment within the context established by higher-level systems. More specifically, it is concerned with learning and maintaining primitive behaviors and learning and performing motor task segments once the need for learning or performing a particular segment has been identified by a higher-level system such as a planner. It is not concerned with high-level planning itself, nor with identifying the task to be performed. This work considers the generation, synchronization and control of particular primitive motions, not how the motion contributes to a global plan.

This work addresses motor learning as opposed to declarative learning. The distinction is somewhat fuzzy, but here motor learning entails learning how to perform actual movements and other motor behaviors that synchronize correctly with internal and external processes, while declarative learning is the learning of specific rules or facts. Motor learning involves learning analog values for control parameters and learning about the behavior of external systems. It also involves learning about events.

We are considering learning by systems that have a significant amount of internal structure and already have some competence. That is, they understand space in the sense that they understand distance orderings, how to move to targets, how to move to correct position errors, and how to recognize correct behavior at a high level, though not necessarily at the sensory level. This work models learning by trial and error with an internal teacher: The system already knows about learning to some extent, and has been told by higher-level systems what task is to be considered. Tasks involve primitive motor behaviors and synchronizing with the behavior of external objects. In analogy with a human teaching himself or herself a task, the teacher does not

know explicitly the precise motor commands necessary to accomplish the task, but recognizes good and improved behavior. The teacher can also give advice on the types of behavior to try and whether to perform an action more or less intensely or earlier or later, but does not know explicitly what should be done.

In order to concentrate on basic issues without the distraction of too much complexity, simple robots are used. Positioning commands for smooth motions are learned and executed, but no posture, stance, stiffness or force learning is considered. The work is extensible, however, to those regimes. Because processing time delays can be significant in versatile robots, emphasis in this work is on feedforward control at the primitive level. Feedback is incorporated at the cognitive level: If the feedforward command is inappropriate the cognitive system adjusts it. This is consistent with motor control in animals [Brooks (1986)].

Motor control and perceptual elements are modeled as learning neural networks. Control networks generate input voltages for amplifiers that drive conventional dc servomotors. The only conventional controller employed is the default damper that slows the arm when it has no active command.

Chapter 3

Learning and Maintaining Hand-Eye Calibration Corrections

3.1 Introduction

This chapter describes and compares two approaches to the hand-eye calibration problem introduced in chapters 1 and 2. One is based on resource-allocating neural networks; the other, which serves as a comparison, is based on conventional nonlinear (Levenberg-Marquardt) estimation. Both approaches have been implemented as computer simulations and consider the effects of kinematic errors and arm solution degeneracy. In addition, thermal/mechanical drift, visual measurement noise, and nonlinear visual and drive system perturbations are considered for the system based on neural networks. A new virtual-neuron learning algorithm for networks with a single layer of adaptable processing units, or neurons, is presented, and it is shown

that the step and frequency-response of such networks is surprisingly linear, making it possible to predict their plant-tracking performance.

3.2 Previous Work

A number of workers have addressed issues relating to hand-eye calibration. Ruoff [Ruoff (1980)] discussed off-line hand-eye calibration for an industrial robot that was assumed to be kinematically accurate. To effect calibration, the robot repeatedly placed a disc on a vision stage at known increments along a user-designated line segment and used a least-squares fit to identify the vision magnification factor and the vision coordinate frame with respect to the manipulator base frame. Since the manipulator was calibrated, and therefore provided a distance scale, the system could return both Cartesian coordinates and metrical information. The system was capable of accuracies approaching 0.5 mm, but, as is typical of industrial robots, was subject to thermal drifts that forced recalibration as the ambient temperature varied during the day.

Gennery and co-workers [Gennery et al (1987)] use off-line nonlinear estimation to calculate best-fit vision calibration parameters for laboratory robots under development for the United States space program. Their procedure uses a large, accurately machined and located fixture consisting of a regular array of light discs machined in a dark anodized background. The manipulator itself is calibrated separately. The procedure involves visually observing the disc pattern and inserting a special locating probe, accurately held by the manipulator, into precisely machined holes at each disc location. The known relationship between each disc position and the corresponding position of the locating probe allows the model parameters for each camera and their relationship to the manipulator base frame to be estimated in an overall least-squares

sense. This procedure leaves residual errors that may amount to several centimeters at some locations.

Lokshin and Kan implemented a system for accommodating these residual errors interactively [Lokshin (1990)]. An operator designates corresponding actual and computed feature locations in a stereo visual display. The system uses this information to calculate a local frame correction that is applied to nearby robot motions. Motions corrected in this way are usually sufficiently accurate that robot tasks can be completed if terminal motion force feedback and control are employed.

The systems just described assume a Cartesian internal representation, and visual and arm computations are done in that context. Kuperstein [Kuperstein (1987b), Kuperstein (1988)] and Mel [Mel (1989)] have addressed the issue of hand-eye calibration from a biologically-inspired perspective. They have considered the problem of reaching visible targets and have not attempted to provide explicit metrical information. The approaches they employ are based on idealized neural architectures that associate visual target position representations and arm position representations. The mappings are locally rather than globally optimized, and do not explicitly employ kinematics.

Kuperstein [Kuperstein (1987b)] has developed a simulated hand-eye system that generates feedforward commands for positioning an arm to a visually-sensed target by learning to map camera pointing information into appropriate arm axis commands. His system, which assumes no a priori kinematic information, employs a three DOF revolute arm and two cameras. Each camera is capable of independently pointing to the visual target. The system borrows heavily from anatomical models, using pairs of simplified antagonistic "muscles" to position each arm axis and three pairs of equally-spaced antagonistic "muscles" to point each camera. Antagonistic pairs are used since muscles are unidirectional and because antagonistic processing is thought

to be important in biology.

The coordinates of a unit vector lying along each camera's optical axis are calculated from its six muscle activation values. Each camera position is then transformed into a unimodal activity distribution in a two-dimensional matrix of 20×20 overlapping neural processing units. A disparity map, which represents the disparity between the right and left camera positions, is generated from the two position matrices. The disparity map is also a 20×20 matrix, and the activity of each unit is calculated from the corresponding units of the left and right camera position matrices.

Arm positions corresponding to camera positions are generated by using the camera position and disparity matrices to calculate the activation of each arm muscle, which is a weighted sum of the activities of each unit in the two camera position arrays and the disparity map. Learning is accomplished by adjusting these weights.

The system first operates in a learning mode in which muscle activation commands corresponding to target positions are associated with camera position and disparity signals corresponding to the target positions. After learning has converged, the system enters the execution mode. In the execution mode the target is positioned and the system responds by generating the muscle activation commands appropriate for the target position.

In the learning mode, the arm, with the target attached, is positioned by randomly activating its muscles (this is akin to an infant's babbling [Mel (1989)]). Once the arm is positioned, each camera is pointed to the target's visual contrast center by an independent control system that is not modeled. Pointing causes activation of the three antagonistic muscle pairs that point each camera. This causes activity in the camera position and disparity maps described above. This activity, in turn, is used to generate the weighted sums that correspond to arm muscle activations.

Learning is accomplished by adjusting the weights using an incremental learning rule that minimizes the error between the randomly-commanded activations and the weighted sums. Learning converges in three to five thousand trials, depending upon the learning rate, with an accuracy of about two percent [Kuperstein (1987a)]. Accuracy is taken to be the average difference between the randomly-excited and generated activations over all arm axes as a percentage of the activation range. Given that there are three arm axes, each with two opposing muscles, and three 20×20 matrices, there are 7200 weights.

In more recent work, Kuperstein [Kuperstein (1988)] uses an arm with five DOF to address the problem of learning to generate feedforward commands for grasping a randomly positioned and oriented cylindrical object. This work requires processing stereo retinal images because of the need to determine the orientation of the cylinder's axis. Muscle activations are weighted sums of eye pointing and retinal matrix units. Again, the system learns the correct arm response by associating sensory signals with arm muscle commands.

Kuperstein's work employs a number of signal transformations thought to mimic processes found in animals, but it uses just a single layer of adjustable weights. Noise and arm degeneracy are not considered.

The hand-eye calibration system embedded in MURPHY [Mel (1989)] also learns to reach for objects by associating visual stimuli with arm positions. Instead of learning the joint angles corresponding to grasp positions, however, MURPHY learns to direct its arm to a target by learning forward kinematic and inverse differential kinematic maps. The forward kinematic map is retinotopic, mapping the joint angles to a virtual image of the arm joints, hand and fingers (which are bright dots) on the image plane, which is a representation of the space accessible to the arm. By mentally trying different arm positions, MURPHY can determine if a set of joint angles will

make the arm collide with visible obstacles (which form real images on the image plane).

The inverse kinematic map generates a set of incremental joint angles corresponding to an incremental movement by associating the arm joint angle configuration and the desired hand motion increment in polar coordinates. Each arm angle or angle increment is represented in a separate coarsely- coded (Gaussian) population of overlapping neural processing units in which the order of a unit in the population corresponds to the joint angle range it dominantly represents, and the unit's activation peak corresponds to the center of its range.

Reaching is effected either by flailing toward the target in a search strategy using the forward kinematic map or by making incremental movements toward the target based on the current arm configuration and the target position. Desired hand movement increments are determined by an external serial controller.

This approach, rather than the straightforward association of target positions with joint angles, was taken to allow exploring reaching in the presence of obstacles. It involves visual servoing, however, since the hand is guided to the target by calculating incremental positions in a loop.

MURPHY has a three DOF arm that moves in a plane in the view of a camera system. MURPHY's architecture, which employs many sigma-pi (conjunctive) neural processing units with coarsely-coded inputs and many connections, is similar to table lookup. In effect, an input vector becomes an address code that activates the appropriate units for computing the result, which is generated by thresholding a weighted sum of the outputs of the active units. Associations are learned using a Hebbian approach in which connections are formed for all input-output pairs that are sufficiently active. This allows the forming of associations in just one trial. Because the units use

coarse coding and are consequently able to interpolate, MURPHY requires just one uniformly-subsampled pass to learn its forward kinematic map. This pass, however, requires several hours, and generates over two million connections.

The inverse differential kinematic map is learned by randomly moving the arm through small increments and associating the resulting hand position increments and the active neural processing units representing the arm joint angles with the observed joint angle increments.

MURPHY, which has actually been implemented in hardware (the neural processing is simulated on a serial computer), is kinematically redundant because it has three DOF, but is restricted to move in a plane. The issue of redundancy is finessed by learning the "average" inverse differential kinematic map. MURPHY, as implemented, cannot easily deal with drifting plants since there is no gradual way to modify connection strengths.

3.3 An Engineering Approach to Hand-Eye Calibration

The biological approaches of Mel and Kuperstein described above attempt to model the *style* of computation thought to exist in animals. They assume little is known *ab initio* about system kinematics. In contrast, both of the approaches described here have an engineering flavor, even though one employs a neural network as the adaptive element.

It is assumed that the nominal kinematic structure and parameters of the system are known, and that the actual system deviates somewhat from the nominal system

because of manufacturing variations and system drifts. It is therefore possible, using nominal system kinematics, to calculate reasonably accurate feedforward joint angles corresponding to the locations of visually-sensed targets.

The approach based on neural networks performs hand-eye calibration by learning to *correct* the feedforward position commands generated by the nominal system kinematics. A resource-allocating neural network (RANN) [Platt (1990)] learns the correction either by adding neurons or by modifying the neurons that already exist as described in section 3.5.1 and appendix A. Actual command angles are generated by summing the nominal angles and the corrections. It is simpler, faster, and more accurate to learn a correction than to learn the entire kinematic map, as will be seen below in section 3.5.9.

The approach based on nonlinear estimation identifies the kinematic parameters using the nominal parameters as initial values. Feedforward angles are generated by the system's kinematics routines using the parameters identified by the estimator.

In both cases, initial performance is good because of the explicit use of engineering knowledge. This is in contrast to the more biologically-oriented approaches that perform poorly at first. Arm solution degeneracy is also incorporated in a natural way. This work addresses hand-eye calibration for direct reaching using visual information. Tactile sensing is not considered.

The remainder of this chapter describes the hand-eye system, including the arm, stereo vision system, and error sources, and develops the neural-network and nonlinear estimation approaches to hand-eye calibration. It examines the effects of noise on resource-allocating neural networks and explores the performance of neuron allocation and adaptation algorithms. Those with the best performance are selected for further tests. These tests include evaluating the ability of networks with the selected algo-

rithms to track step and periodic plant variations, compensate for kinematic errors, and learn hand-eye mappings without using knowledge about the kinematic structure of the system. The mathematical relationships required for nonlinear estimation are given and, where possible, the performance of the estimator-based system is examined in the same situations as the system based on neural networks. Finally, the performance of the network-based and estimation-based approaches is compared.

3.4 The Hand-Eye System

This section describes the hand-eye system, which is used for both the neural network-based and estimation-based calibration approaches.

3.4.1 System Layout and Kinematics

The system layout is shown in figure 3.1. The underlying system geometry and operation are identical for both the neural and estimation approaches. The system, which is planar, includes an arm, a stereo vision system, and a vision pointing system. The vision and pointing system origins coincide.

The arm has two angular degrees of freedom, α and β, and links $\mathbf{a}$ and $\mathbf{b}$, which are represented by vectors as shown. The angle of link $\mathbf{a}$ relative to the base frame x-axis is α, while β is the angle of link $\mathbf{b}$ with respect to link $\mathbf{a}$. Both α and β can be independently given arbitrary values within their individual ranges of motion. The stereo vision system measures the range r of objects from the vision origin, and their angular eccentricity, ϵ, from the visual pointing axis $\hat{\mathbf{p}}$, which is aimed by the pointing system. The pointing system has one degree of freedom, γ, the angle of

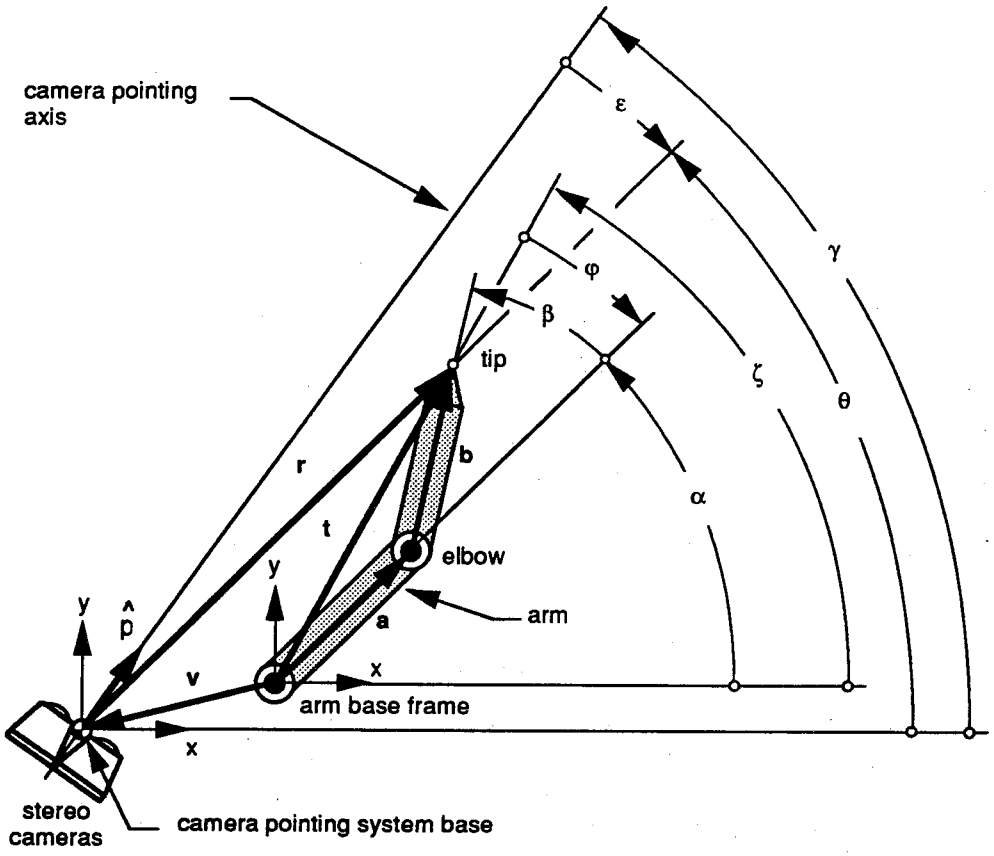

Figure 3.1: Hand-eye system layout. Small circles indicate the datum from which an angle is measured. Angles increase counterclockwise.

$\hat{\mathbf{p}}$ with respect to the x-axis of the pointing system base frame, which is nominally parallel to the arm base frame. The vector $\mathbf{v}$ is the location of the vision pointing system base relative to the arm base frame. The angular positions of the arm and visual pointing axes are measured by encoders.

Encoders are assumed to give exact readings and to have stable offsets except when mechanical or thermal drifts are involved (see section 3.5.7). This is a reasonable assumption given the quality of encoders available commercially, though position quantization will cause a small amount of uncertainty in actual systems. An encoder's offset is the difference between the actual axis angle and the encoder reading when the axis is at a defined standard position, which is taken to be zero. The relationships

between encoder readings and axis angles are not assumed to be linear, however, as described below.

Because of the various error sources, the actual angles and the angles measured by the encoders will differ. The relationships between measured and actual quantities are defined as follows:

$$\alpha_{act} = \alpha_{meas} + O_\alpha + \delta\alpha \tag{3.1}$$

$$\beta_{act} = \beta_{meas} + O_\beta + \delta\beta \tag{3.2}$$

$$\gamma_{meas} = \gamma_{act} - O_\gamma + \delta\gamma \tag{3.3}$$

$$\epsilon_{meas} = \epsilon_{act} + \delta\epsilon + n_\epsilon \tag{3.4}$$

$$r_{meas} = r_{act} + \delta r + n_r, \tag{3.5}$$

where *meas* and *act* mean measured and actual, respectively; O_α, O_β, and O_γ are the respective encoder offsets; $\delta\alpha$, $\delta\beta$, and $\delta\gamma$ are respective errors due to nonlinear perturbations (see section 3.4.4); and n_ϵ and n_r are normally-distributed visual noise sources with zero mean that will be discussed below.

The x and y coordinates of $\mathbf{t}$, the tip of the arm, are nominally given in terms of the joint angles by

$$t_x = a\cos(\alpha) + b\cos(\alpha + \beta) \tag{3.6}$$

$$t_y = a\sin(\alpha) + b\sin(\alpha + \beta). \tag{3.7}$$

The tip position relative to the vision base is given by

$$r_x = t_x - v_x \tag{3.8}$$

$$r_y = t_y - v_y, \tag{3.9}$$

and the angle of the tip as a function of the pointing angle and angular eccentricity is given by

$$\theta = \gamma + \epsilon. \tag{3.10}$$

The tip position relative to the arm base is given in terms of the vision angles by:

$$t_x = r\cos(\theta) + v_x = r\cos(\gamma + \epsilon) + v_x \tag{3.11}$$

$$t_y = r\sin(\theta) + v_y = r\sin(\gamma + \epsilon) + v_y, \tag{3.12}$$

while the tip polar angle, ζ, is given in terms of the tip position by

$$\zeta = \arctan(t_y, t_x). \tag{3.13}$$

Since $\mathbf{a} + \mathbf{b} = \mathbf{t}$, we have, taking the dot product of each side with itself and solving for β,

$$\beta = \arccos(\frac{t^2 - a^2 - b^2}{2ab}). \tag{3.14}$$

Similarly, since $\mathbf{b} = \mathbf{t} - \mathbf{a}$, we have

$$\phi = \arccos(\frac{t^2 + a^2 - b^2}{2at}). \tag{3.15}$$

The elbow can be either to the left or the right of $\mathbf{t}$, so the arm solution is not unique. From the fact that a, b, and t are sides of a triangle and figure 3.1 we see that

$$\mathrm{sgn}(\phi) = -\mathrm{sgn}(\beta) \tag{3.16}$$

$$\alpha = \zeta + \phi. \tag{3.17}$$

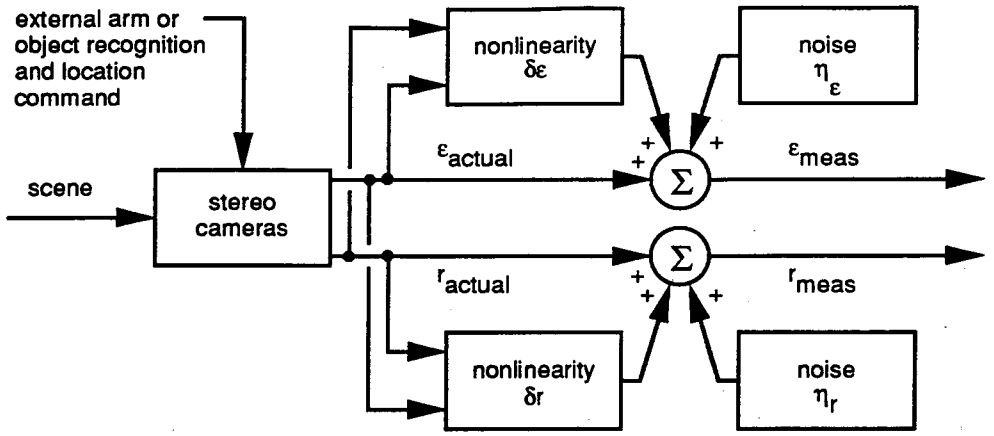

Figure 3.2: Schematic representation of stereo vision processing with error sources. The system measures the range and eccentricity, subjecting the measurements to additive noise, nonlinear perturbations, and range scale factor errors. Object or arm tip recognition and location commands are supplied by an unmodeled external system.

3.4.2 The Stereo Vision System

The vision system, which is not modeled in detail, is illustrated in figures 3.2 and 3.3. It is assumed to use two separated cameras, each with a regular array of constant-size pixels. As described above, it measures the range and angular eccentricity of features within its visual field.

The vision system is assumed to be pre-calibrated in the sense that it returns accurate range and angular values. This seems like a strong assumption, but we will see below and in section 3.5.8 that range scale factor errors, angular deviations, and kinematic errors are naturally corrected in the same way. Range scale errors, which might result from camera misalignments and lens focal length errors, just establish a different distance scale that is incorrect, but is rendered internally consistent for hand-eye positioning by the learning or estimation process. As shown in figure 3.2, the range and eccentricity measurements are assumed to be corrupted by independent additive zero-mean Gaussian noise, by nonlinear perturbations, and by the systematic range scale factor error just mentioned. The visual noise sources are assumed to apply

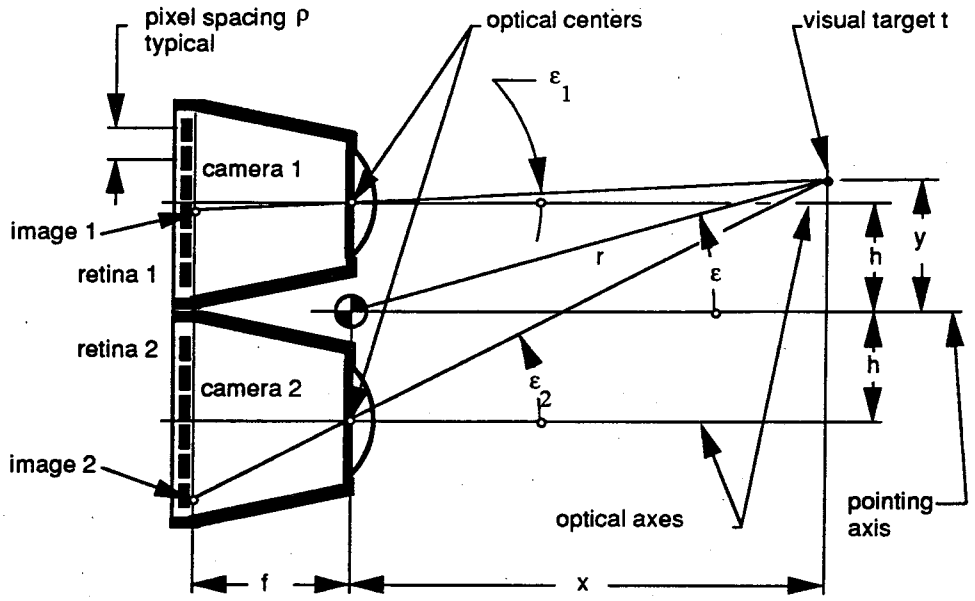

Figure 3.3: Schematic representation of vision system geometry. The visual target is located at point t, which has Cartesian coordinates x, y and polar coordinates r, ϵ, where ϵ is the angular eccentricity. The cameras are separated by $2h$. The eccentricity measured by camera one is ϵ_1; that measured by camera two is ϵ_2.

wherever the image point falls on the retina. That is, effects due to the locations of pixels in space are not modeled. Noise standard deviations are estimated using uniform distributions as described below.

In addition, the visual pointing system is subject to a constant angular offset that models encoder positioning errors, and to nonlinear perturbations similar to those that are often present in gear and traction drive systems. Nonlinearities are described below in section 3.4.4.

For purposes of estimating the noise source standard deviations, the cameras are assumed to have a uniform pixel spacing, ρ, of 0.0275 mm, a focal length, f, of 8.0 mm, and a sixty-degree field of view. They are assumed to be separated by 50 cm. The range measurement noise standard deviation is estimated as follows: Referring

to figure 3.3, and defining

$$m_1 \stackrel{\text{def}}{=} \tan(\epsilon_1) = (y-h)/x \tag{3.18}$$

$$m_2 \stackrel{\text{def}}{=} \tan(\epsilon_2) = (y+h)/x, \tag{3.19}$$

we find that

$$r = \frac{h}{m_2 - m_1}\sqrt{4 + (m_1+m_2)^2}. \tag{3.20}$$

Then, linearizing to find the change in r due to small changes in m_1 and m_2, we obtain

$$dr = (\frac{r^2\cos(\epsilon)}{2h} + r\cos(\epsilon)\sin(\epsilon))dm_1 + (r\cos(\epsilon)\sin(\epsilon) - \frac{r^2\cos(\epsilon)}{2h})dm_2. \tag{3.21}$$

We assume that the angles ϵ_1 and ϵ_2 are uniformly distributed over the camera's field of view. Since ρ, the pixel width, is very much smaller than the focal length of the cameras, and the field of view is just sixty degrees, it is reasonable to consider the corresponding small changes in slope, dm_1 and dm_2, as random variables that are uniformly distributed over the pixel width. This implies that $\langle dm_1\rangle = \langle dm_2\rangle = 0$, where the angle brackets $\langle\rangle$ indicate the average, or mean, value. The standard deviations, $\sigma(dm_1)$ and $\sigma(dm_2)$, are given by:

$$\sigma(dm_1) = \sigma(dm_2) = \frac{1}{\sqrt{12}}\frac{\rho}{f}. \tag{3.22}$$

Using this result and letting n_r correspond to dr, the mean, $\langle n_r\rangle$, and the standard deviation, $\sigma(n_r)$, of n_r are given by:

$$\begin{aligned}\langle n_r\rangle &= (\frac{r^2\cos(\epsilon)}{2h} + r\cos(\epsilon)\sin(\epsilon))\langle dm_1\rangle + \\ &\quad (r\cos(\epsilon)\sin(\epsilon) - \frac{r^2\cos(\epsilon)}{2h})\langle dm_2\rangle = 0 \qquad (3.23)\\ \sigma(n_r) &= ((\frac{r^2\cos(\epsilon)}{2h})^2 + r^2\cos^2(\epsilon)\sin^2(\epsilon))^{1/2}\frac{\rho}{\sqrt{6}f}. \qquad (3.24)\end{aligned}$$

Similarly, noting that

$$(m_1 + m_2)/2 = \tan(\epsilon) \stackrel{\text{def}}{=} m, \tag{3.25}$$

we verify that the eccentricity measurement noise mean, $\langle n_\epsilon \rangle$, is zero, and calculate its standard deviation, $\sigma(n_\epsilon)$:

$$\begin{aligned} \langle n_\epsilon \rangle &= \langle d\epsilon \rangle = \langle d(\arctan(m)) \rangle = \langle \frac{dm}{1+m^2} \rangle \\ &= \cos^2(\epsilon) \frac{\langle dm_1 \rangle + \langle dm_2 \rangle}{2} = 0 \end{aligned} \tag{3.26}$$

$$\sigma(n_\epsilon) = \sqrt{\langle (d\epsilon)^2 \rangle} = \frac{\cos^2(\epsilon)}{2} \sqrt{\langle dm_1^2 \rangle + \langle dm_2^2 \rangle}. \tag{3.27}$$

Assuming that $\langle dm_1^2 \rangle = \langle dm_2^2 \rangle$, this becomes:

$$\sigma(n_\epsilon) = \frac{\cos^2(\epsilon)}{\sqrt{2}} \sqrt{\langle dm2 \rangle} = \frac{\cos^2(\epsilon)}{\sqrt{2}} \sigma(dm) = \frac{\cos^2(\epsilon)}{2\sqrt{6}} \frac{\rho}{f}. \tag{3.28}$$

The expression for $\sigma(dm)$ is given by equation 3.22.

3.4.3 System Operation

Hand-eye calibration is achieved by randomly[1] positioning the arm[2] at many different locations and using the discrepancy between the measured and calculated arm joint angles to drive the learning-based or identification-based adaptation processes. This is represented schematically in figure 3.4a. In the neural network-based adaptation process, shown in figure 3.5, the stereo vision system, which is mounted on the pointing system, observes the arm tip (the tool frame), measuring its range r relative to the pointing system base, and angular eccentricity ϵ relative to the optical axis $\hat{\mathbf{p}}$, as shown in figure 3.1. The range and eccentricity are used along with the measured

[1] A grid could also be used.

[2] The visual eccentricity ϵ may be selected as well. See below.

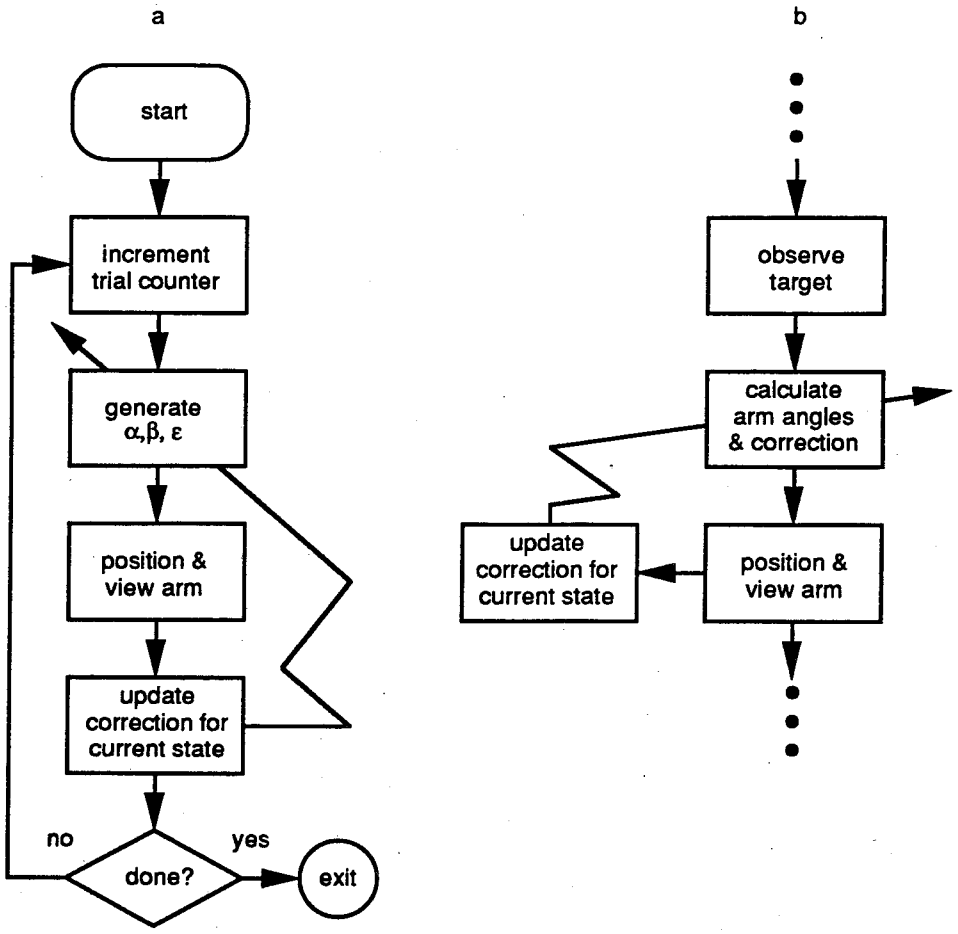

Figure 3.4: Hand-eye calibration. In a) the adaptation process is embedded in a program specifically designed to perform hand-eye calibration. In b) the adaptation process is run in the background to maintain hand-eye calibration during normal operation.

pointing angle γ_{meas}, and the nominal system kinematics to calculate the nominal arm angles α_{nom} and β_{nom}. The nominal arm angles, the measured pointing angle, and the measured vision parameters r_{meas} and ϵ_{meas} are used by the neural network to generate the arm angle corrections $\Delta\alpha$ and $\Delta\beta$. These are added to the nominal arm angles to predict the arm angles. These predicted angles are compared with the angles measured by the arm joint encoders. Discrepancies are used to drive the network learning procedure to improve the arm angle corrections. Depending upon the error size and the availability of neurons, the learning procedure either adjusts

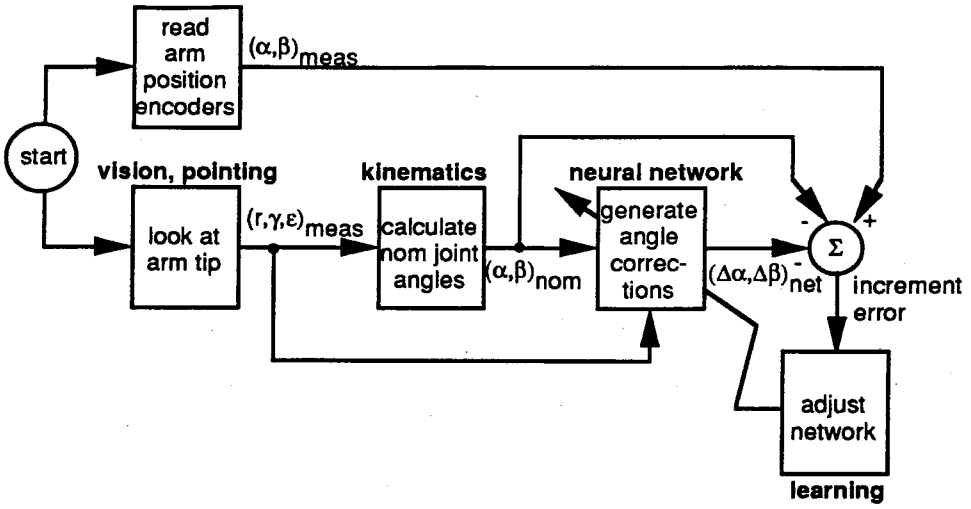

Figure 3.5: Neural network-based adaptation process.

the network parameters using a adaptation algorithm (see section 3.5.3), or allocates an additional neuron to reduce the error as described below in section 3.5.1. The estimation-based adaptation process, which is shown in figure 3.50, is similar except that the kinematics calculations use the current kinematic parameter estimates rather than the nominal kinematic parameters and the estimator replaces the neural network and learning algorithms. Either adaptation process can be run in the background of a robot control system to maintain calibration during operation as illustrated in figure 3.4b. Doing so just requires that the arm be visually observed at appropriate times and that the relevant data be made available.

The stereo vision pointing system may be commanded either to foveate on the arm tip (align its pointing axis $\hat{\mathbf{p}}$ with vector $\mathbf{r}$ in figure 3.1) or to point to the vicinity of the tip so the target lies within the angle (ϵ) from the pointing axis. The latter case simulates the current practice in robotics, which typically employs cameras that lack the high-resolution foveas found in vertebrate eyes, and does not attempt to keep objects centered in the visual field. Vision system pointing is handled by an unmodeled servo system represented schematically in figure 3.6.

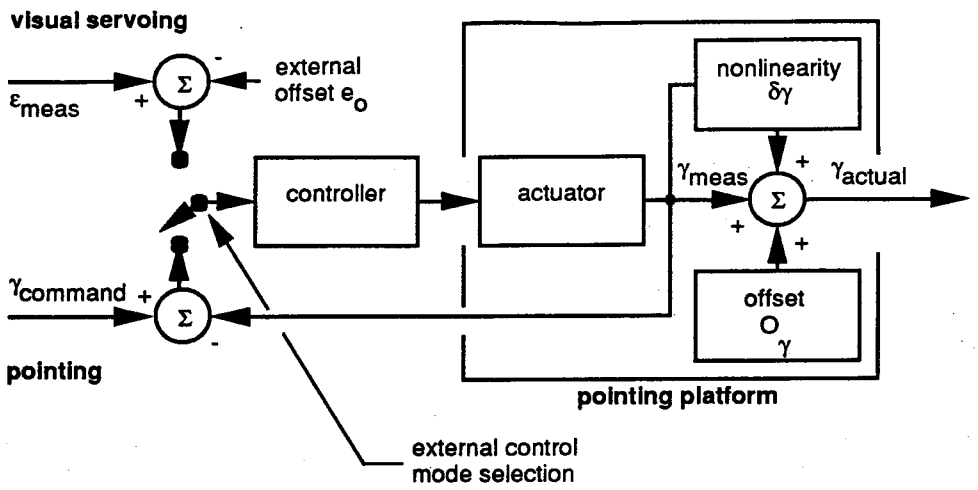

Figure 3.6: Vision pointing system with error sources. When the switch is in the lower position, the cameras are aimed to a specified angle, as when looking for an object. The pointing loop is closed around the γ position encoder. When the switch is in the upper position, cameras are aimed at a visually-sensed object, and the pointing loop is closed around vision system output.

Arm positions are generated by selecting joint angle commands within the ranges of joint motion. Joint angle travel limits may be set to allow both elbow-left and elbow-right solutions,[3] thereby introducing arm solution degeneracies. Actual arm positioning is handled by an unmodeled servo system represented schematically, along with sources of error, in figure 3.7.

It is assumed that the measurements of range, eccentricity, pointing angle, and arm angles corresponding to a particular arm position are effectively simultaneous. It is also assumed that each of the axes α, β, γ, incorporates a position control loop that is closed around an axis position encoder, and that the position loop is capable of asymptotically perfect postioning performance with respect to the encoder. That is, that the position command and the encoder reading (e.g., $\alpha_{command}$ and α_{meas}) will asymptotically coincide. Instead, it could be assumed that the *commanded* rather than the encoder axis position is measured, but that approach has not been taken here.

[3]The elbow may be either to the left or right of the radial line drawn from the robot base to the arm tip.

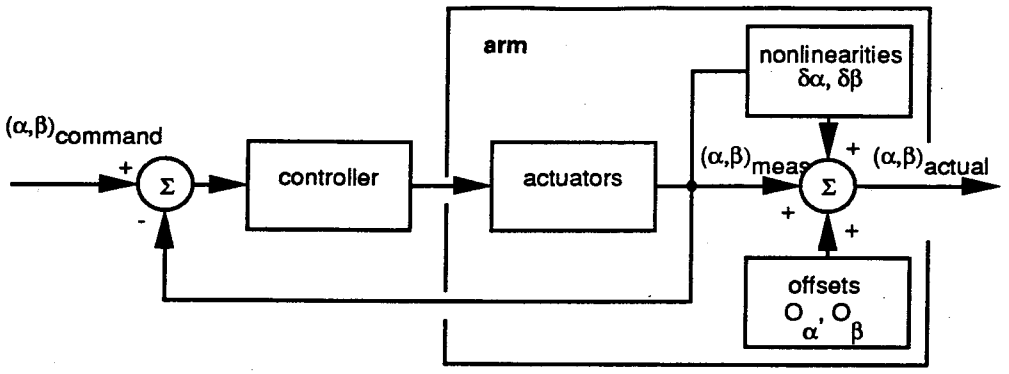

Figure 3.7: Arm positioning system with error sources.

In that case compensation for poor (but consistent) servo positioning performance would be included in the learned angle command correction and deviations from consistency would be noise sources.

Finally, it is assumed that the hand-eye system is embedded in a higher-level control system that provides crucial planning, sequencing, and timing functions that are not addressed in this work. The higher-level control system determines which of the multiple arm solutions to use in a particular instance, when the position has stabilized, when and how to use visual information, when to issue commands, and so on.

3.4.4 Kinematic Parameters and Modelling Errors

Kinematic parameters

Nominal and actual values of the kinematic parameters, the link lengths, pointing system position, and encoder offsets, are given in table 3.1. Refer also to figure 3.1.

Each of these kinematic parameters is assumed to be in error except for the α

Parameter	Nominal Value		Actual Value	
$\|\mathbf{a}\|$	1.00	Meter	1.005	Meter
$\|\mathbf{b}\|$	1.00	Meter	1.003	Meter
$\mathbf{v}_x$	-1.00	Meter	-1.01	Meter
$\mathbf{v}_y$	0.00	Meter	-0.02	Meter
O_α	0.00	Radian	0.00	Radian
O_β	0.00	Radian	0.018	Radian
O_γ	0.00	Radian	0.035	Radian

Table 3.1: Nominal and actual kinematic parameters.

encoder offset, O_α, which is fixed at zero for convenience. This amounts to defining the x-axis as corresponding to the angle $\alpha = 0$. Other values could have been used as well. Setting $\alpha = 0$ is reasonable because a base x-axis has to be defined in some way. In manufacturing practice this is typically accomplished by positioning the arm at a standard, physically-defined position corresponding to a desired encoder reading. The observed difference (offset) between the desired and actual encoder readings is recorded. This offset is then added to subsequent observed readings as a correction.

It is not necessary to fix O_α, the alpha encoder offset, but it simplifies debugging to do so. Both the network and nonlinear estimation approaches are able to generate internally-consistent kinematics when O_α is estimated along with the other parameters.

The nominal arm joint travel limits are (in radians):

$$-0.8\pi \leq \alpha \leq 0.8\pi \tag{3.29}$$

$$-0.8\pi \leq \beta \leq 0.8\pi. \tag{3.30}$$

The β range permits multiple arm solutions. Because of solution singularities and associated computational problems that occur when the arm is completely extended

($\beta = 0$), the region around $\beta = 0$ is excluded by restricting β to the regions:

$$-0.8\pi \leq \beta \leq -0.1\pi \tag{3.31}$$

$$0.1\pi \leq \beta \leq 0.8\pi. \tag{3.32}$$

This reduces the accessible radius by less that two percent, and the accessible work area by less than three percent over the nominal region.

Nonlinear Perturbations

The hand-eye system includes nonlinear position perturbations for the vision pointing system, the arm, and the stereo vision system itself. These perturbations, which model manufacturing, positioning, and alignment errors, can be independently selected. Parameters were chosen to cause a maximum tangential perturbation of about one centimeter at a radius of two meters for each angular degree of freedom selected and a maximum radial perturbation of two centimeters at a radius of three meters. These were selected as being representative of significant errors that might be encountered in actual robot systems.

In the following s_c, s_a, and s_v are selection parameters for the vision pointing system, the arm, and the stereo vision system, respectively. A selection parameter equals one if the corresponding nonlinearity is selected and is zero otherwise. In what follows, the subscript *meas* means the measured value, while the subscript *act* means the actual value as before.

The vision pointing perturbation $\delta\gamma$ is given by

$$\delta\gamma = s_c \nu\gamma \sin(3\gamma + 0.7), \tag{3.33}$$

where

$$\gamma_{meas} = \gamma_{act} + \delta\gamma - O_\gamma, \tag{3.34}$$

ν is an adjustable parameter, and O_γ is the vision pointing system encoder offset defined above. This form was chosen to mimic periodic and scale factor errors that can result from using gears and traction drives. A value of $\nu = 0.002$ gives a maximum perturbation of 1 cm at a radius of 2 meters.

Arm angle perturbations $\delta\alpha$ and $\delta\beta$ are given by

$$\delta\alpha = s_a \xi \alpha_{meas} \sin(\alpha_{meas} + 2) \tag{3.35}$$

$$\delta\beta = s_a \kappa \beta_{meas} \sin(\beta_{meas} - 1), \tag{3.36}$$

where, as defined before,

$$\alpha_{act} = \alpha_{meas} + \delta\alpha + O_\alpha \tag{3.37}$$

$$\beta_{act} = \beta_{meas} + \delta\beta + O_\beta, \tag{3.38}$$

ξ and κ, are adjustable parameters and O_α, O_β are the α and β encoder offsets respectively. $\xi = -0.0017$ and $\kappa = 0.0018$ yield the desired maximum perturbations of about 1 centimeter at 2 meters for their respective axes.

Recalling that ϵ and r are the angular eccentricity from the visual axis and the range from the origin of the vision pointing system base frame respectively, the stereo vision system perturbations are defined to be

$$\delta\epsilon = s_v \eta \sin^2(\epsilon)(r_{act} + 2/r_{act} + 1) \tag{3.39}$$

$$\delta r = s_v \zeta(\delta\epsilon + (r_{act} - 1)^2, \tag{3.40}$$

Parameter	Value
Vision Perturbation Parameters	
η	1.5×10^{-2}
ζ	5.0×10^{-3}
Vision Pointing Perturbation Parameter	
ν	2.0×10^{-3}
Arm Perturbation Parameters	
ξ	-1.7×10^{-3}
κ	1.8×10^{-3}

Table 3.2: Nonlinear perturbation parameters.

where, ignoring noise for the moment,

$$\epsilon_{meas} = \epsilon_{act} + \delta\epsilon \tag{3.41}$$

$$r_{meas} = r_{act} + \delta r. \tag{3.42}$$

The effect of the eccentricity perturbation is to model astigmatism by compressing angular measurements to the left of the visual axis while expanding them on the right. With the assumed sixty-degree field of view, a range of 2 meters, and $\eta = 0.015$, the perturbation moves a point at the left edge of the nominal field of view one centimeter toward the visual axis, while a point at the right edge of the field of view is moved one centimeter away. A plot of distorted and undistorted circles with $\eta = 0.015$ is shown in figure 3.8. With $\zeta = 0.005$, the maximum range perturbation is 2 cm at an actual range of 3 meters. Perturbation parameters are summarized in table 3.2. Perturbations for the given parameter values are plotted in figure 3.9.

The effect of these perturbations on system performance is shown in section 3.5.8.

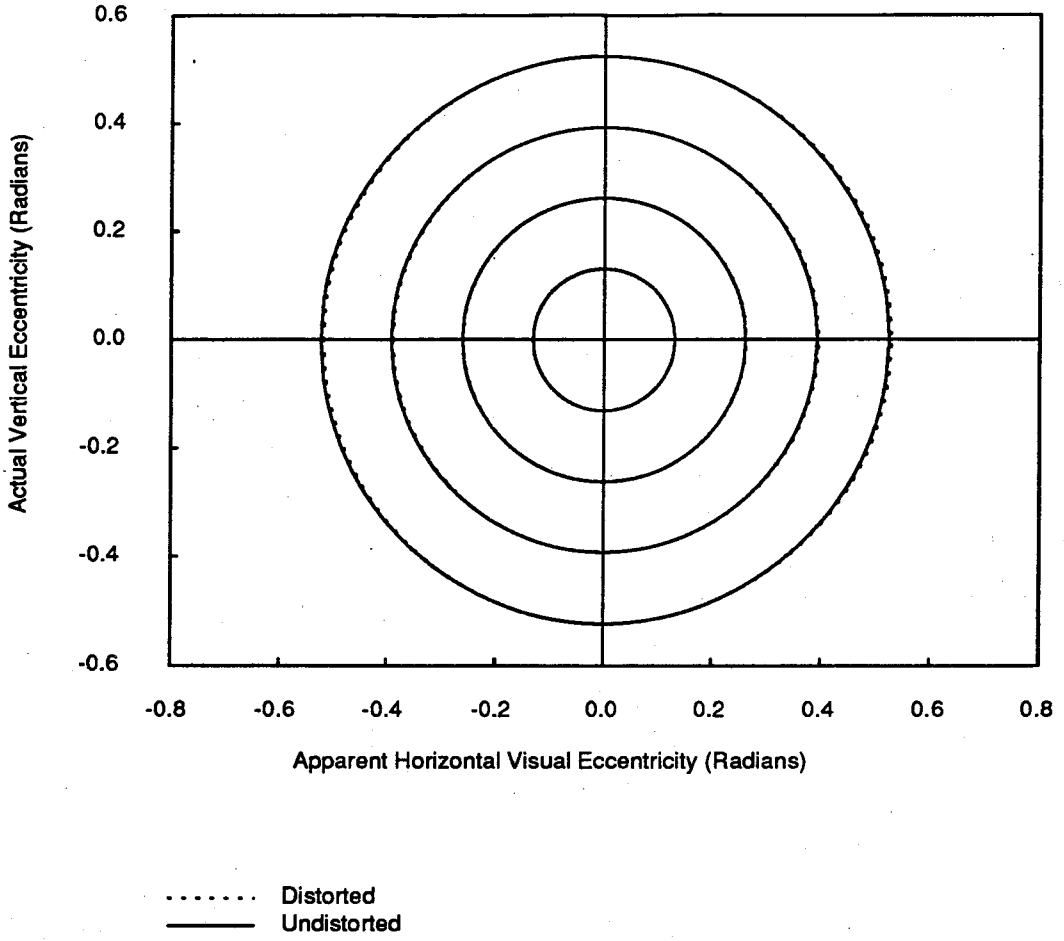

Figure 3.8: Nonlinear visual angular eccentricity perturbation. One circle in each of the plotted circle pairs is distorted by the eccentricity perturbation. The other is not. In the left half plane the distorted circle is closer to the origin, while in the right half plane it is farther away. The distortion is zero on the y-axis. Units are in radians.

3.4.5 System Implementation

As part of this research, the hand-eye calibration system has been implemented as a simulation written in "C." The program is table-driven to allow easy parameter modification. It includes routines that handle the neural networks and identification routines as well as intermediate disc storage and system parameter management. Experiments were run under UNICOS on the JPL Cray X-MP/18 and under UNIX on Sun Sparcstations.

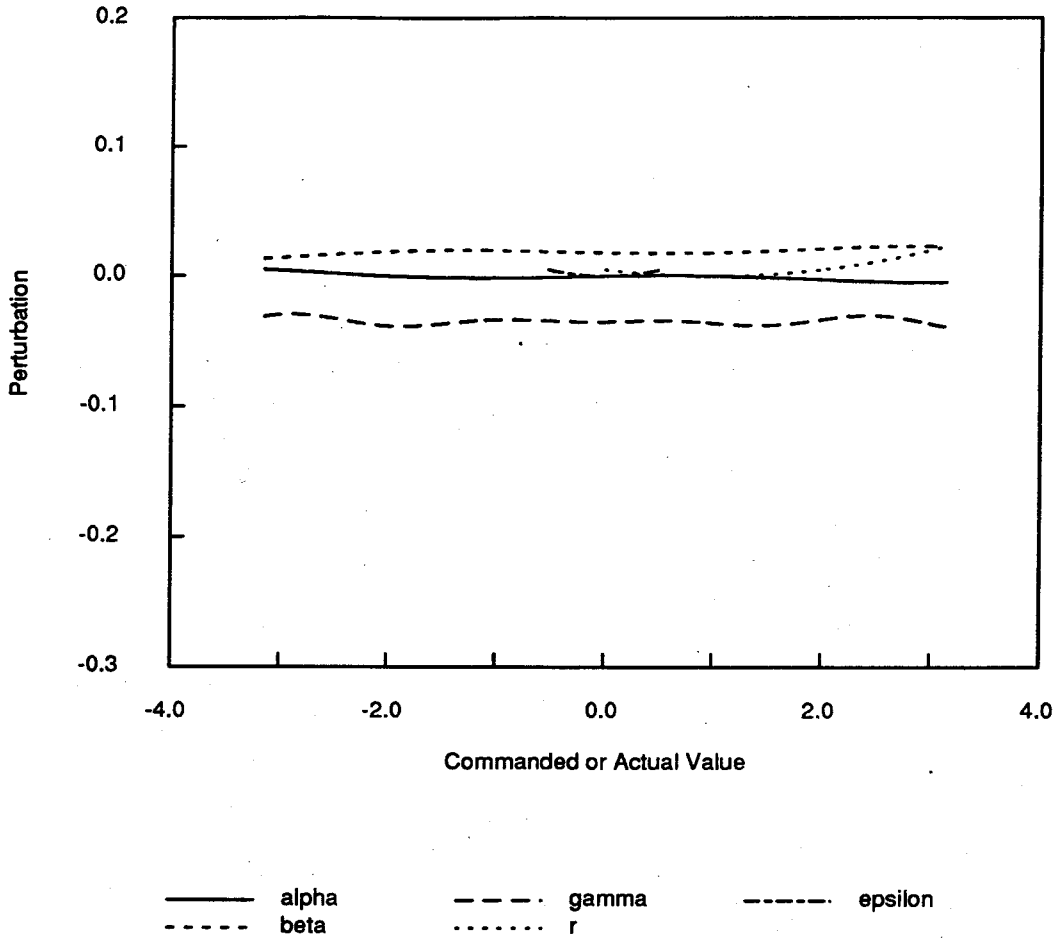

Figure 3.9: Plots of the $\delta\alpha$, $\delta\beta$, $\delta\gamma$, $\delta\epsilon$, and δr perturbations over their accessible domains. Offsets O_α, O_β, and O_γ are added to the $\delta\alpha$, $\delta\beta$, and $\delta\gamma$ plots respectively. Angular units are radians; linear units are meters. Note that in all cases $r \geq 0$. Axes and scale are identical with those of figure 3.47 to allow comparison.

3.4.6 Performance Testing

Performance testing involves alternating sets of training and measurement trials. The system is first initialized with the nominal kinematic parameters.[4] In neural network learning, this means the network initially has no neurons, and hence that the angle corrections $\Delta\alpha$ and $\Delta\beta$ are exactly zero. In nonlinear estimation, the initial kinematic

[4]The parameter values are defined in figure 3.3 and in tables 3.1 and 3.2. See also the plots in figure 3.9.

parameter estimates are set to the nominal parameter values.

After initialization, learning or estimation is enabled, and the system begins a set of training trials as described in section 3.4.3. Each new arm position cycle is considered a trial.

When a set number of training trials is completed, learning or estimation is disabled and position error data are gathered over a fixed number t of testing trials for each iteration. When the t testing trials are complete, learning or estimation is re-enabled and the training trials continue. This process repeats until a preset total number of training and measurement trials is completed.

When the training and measurement trials are finished, performance statistics are computed for each set of t testing trials and plotted as a function of the number of training trials. These statistics include the average length of the positioning error vector $\langle \delta p \rangle$, the *sample* standard deviation [Alexander (1961)] of the length of the positioning error vector, and the number of neurons.

The average Euclidean length δp_j of the error vectors $\delta \mathbf{p}_j$ is defined as

$$\langle \delta p \rangle = \frac{1}{t} \sum_{j=1}^{t} \| \delta \mathbf{p}_j \|, \tag{3.43}$$

and the sample standard deviation is defined as:

$$\sigma_{\delta p,\text{sample}} = \sqrt{\frac{\sum_{j=1}^{t} (\delta p_j - \langle \delta p \rangle)^2}{t-1}}. \tag{3.44}$$

Accuracy is evaluated with position error because it is more meaningful for robotic applications than joint angular error.

For many of the tests, networks were restricted to 200 neurons. Using more

neurons in noisy situations gives minimal accuracy improvement as discussed below. In selected tests more neurons were used, and in some the number of neurons was not limited. Performance evaluation runs typically involved running 1000 testing trials after every 100 learning trials.

As initial tests to verify system function, both the neural network and nonlinear estimation approaches were evaluated with perfect kinematics[5] and without nonlinear perturbations or additive noise. In these tests, the neural network remained empty[6] (no neurons were allocated) and kinematic parameter values computed by the estimation approach were either exactly the nominal values or were within 10^{-11} of the nominal value. Exact values were obtained when an error threshold was used to invoke the estimator. With exact kinematics, there was no error and the estimator was never invoked. The 10^{-11} error was a numerical artifact that arose when the error threshold was set to zero to force using the estimator.

3.5 Hand-Eye Calibration Correction Using Resource-Allocating Neural Networks

This section discusses the design, implementation and performance of the hand-eye calibration system that employs resource-allocating neural networks (RANN's) as learning elements. Such networks learn an input-output mapping either by allocating additional neurons or by adjusting[7] the parameters of those that already exist. The network implementation is described and its calibration performance using different

[5]Nominal and actual values of kinematic parameters coincided.

[6]Recall from section 3.3 that in this work the network learns a *correction* either by allocating additional neurons or by adjusting the parameters of those that already exist. This is discussed in appendix A and section 3.5.1.

[7]In the present work, the parameter adjustment process is called adaptation.

neuron allocation and adaptation schemes in noisy and noise-free situations is analyzed and evaluated. Based on the performance tests, preferred network allocation and adaptation schemes are selected.

3.5.1 The Resource-Allocating Neural Network

Resource-allocating neural networks [Platt (1990)] are described in appendix A. Through suitable choices of the neuron response function, receptive field widths, and neuron allocation and adaptation algorithms, a RANN is capable of rapid learning and excellent performance, and can generalize (interpolate) to input vectors that have not been observed.

Neurons, or Processing Units

The neurons principally used in this work have Gaussian response functions,[8] though cosine and parabolic units are considered as well (see section 3.5.9). A neuron's response is a function of the Euclidean distance, $d = \sqrt{(\sum_{i=1}^{n}(c_i - x_i)^2)}$, of the n-dimensional real input vector $\mathbf{x}$ from $\mathbf{c}$, the center of the unit's receptive field. The response functions are radially symmetric, and take their maximum value of one at the centers of their receptive fields. In the current context processing units are equivalent to the radial basis functions of approximation theory [Poggio and Girosi (1989)].

Each neuron has the following adjustable parameters:

c_i, the n components of the vector $\mathbf{c}$ that positions the center of the neuron's

[8] A neuron's response function specifies its (scalar) output, or response, as a function of its input. Response functions are also called *activation* functions, in which case the response is called the *activation* or *activity*.

receptive field in the input space.

R, the radial width parameter that controls the size of the neuron's receptive field. This is the fraction of the distance to the nearest neighbor at which the neuron's output drops to one-half.

w_k, the m weights that couple the neuron's output to the components of the network output vector.

These parameters are given initial values when the neuron is allocated. They may be modified during operation by the adaptation algorithms discussed in section 3.5.3.

If we let $z = d^2 = (\sum_{i=1}^{n}(c_i - x_i)^2)$, the Gaussian response function is given by:

$$o = e^{-z/R}, \tag{3.45}$$

where o is the output or response for the input z, and R is the width parameter. The response drops to $1/e$ at a distance of $\sqrt{R}$ from the maximum. The Gaussian response function is nonzero for all finite arguments.

Network Inputs

Network (neuron) inputs in this work, except as noted, are analog values representing the two nominal arm angles α and β, the visual pointing angle γ, the visual radius, R, and the visual eccentricity ϵ. All neurons are connected to the same inputs. Input representation is an issue, as discussed in appendix A, and hardware network implementations will involve mechanisms, such as multiple input lines and decoders, to compensate for limited available dynamic ranges. In a possible input representation, not all neurons would be connected to the same inputs. Instead, different input lines would correspond to different neighborhoods of the input space, and would map, in a

topology-preserving manner, onto corresponding sets of contiguous neurons. Analog voltage values would be used within the sets for input distance calculations. Similar representations are ubiquitous in biology.

3.5.2 Unit Allocation

The performance of a RANN in a particular application depends critically upon both the algorithm for allocating new neurons and initializing their parameters as well as the way that training examples are presented. As described above, a new neuron is allocated if the network error for an input vector exceeds a given error threshold θ_e *and* the distance from the current input vector to the center of the nearest receptive field is greater than a distance threshold θ_d. The new neuron's receptive field is centered at the input vector. The weights $w_{k,\text{new}}$ coupling the output of the new neuron to the network output components O_k are chosen so the new neuron's contribution to each network output component for the current input is equal to the output component's current error:

$$w_{k,\text{new}} = (T_k - O_k) \quad 1 \leq k \leq m. \tag{3.46}$$

Since the new neuron's activity (output) is identically one for the current input vector, adding the new neuron completely corrects the network output error for the current input.

The new neuron's width parameter, R, is chosen so the neuron's output drops to one-half at a prescribed fraction, h, (the half-maximum fraction) of the distance to the center of the nearest existing neuron's receptive field. The fraction h critically affects the ability of the network to generalize. If h is too large, the effect of the new neuron will not be localized; if h is too small network output will be lumpy and

the network will not interpolate well between neurons, as described further in section 3.5.6 below.

The error threshold, θ_e, can be given *a priori* to establish a desired network accuracy. The distance threshold, θ_d, is more subtle. According to the Platt algorithm [Platt (1990)], θ_d usually decreases according to some schedule in an apparent effort to force the network to learn large-scale structure before committing its resources to fine structure. This can be important to conserve resources in situations in which the hand-eye domain is initially sampled only locally during learning, but if θ_d decreases too slowly, the network will learn sluggishly. Furthermore, a lower bound on θ_d (≥ 0) and constraints on the number of neurons limit the network's ability to learn fine structure.

Using identical setups, several allocation algorithms were tried to assess their effects on network performance. These algorithms, which are functions of t, the number of training trials, included:

$$\theta_d = kD/t \tag{3.47}$$

$$\theta_d = kD/\sqrt{t} \tag{3.48}$$

$$\theta_d = kD/t^2 \tag{3.49}$$

$$\theta_d = De^{-t^2/x^2} \tag{3.50}$$

$$\theta_d = kD/\sqrt[4]{t} \tag{3.51}$$

$$\theta_d = kD/\ln(t+1) \tag{3.52}$$

$$\theta_d = kD/(t)^{1/n_{eff}} \tag{3.53}$$

$$\theta_d = 0.0 \tag{3.54}$$

$$\theta_d = 0.1 \tag{3.55}$$

$$\theta_d = 0.5, \tag{3.56}$$

where k is a constant fraction (0.5), D is the diameter (diagonal) of the network input space (treated as a Euclidean space), and n_{eff} is the effective dimension (5 in this case) of the network input space. In eq 3.50, χ, the Gaussian width constant, is 1000.

Arm angles and visual eccentricities for both learning and accuracy assessment were selected randomly from a uniform distribution over the input region formed by the Cartesian product of the visual field and accessible arm angles, except that the singular region around $\beta = 0$ was excluded as described in section 3.4.4. Assessment runs included the nonlinear perturbations and kinematic errors as well as visual noise, but did not include parameter adaptation. The sequence of learning and evaluation inputs was the same in all runs, and the average uncorrected position error was about 4.8 cm. The half-maximum fraction, h, used in determining neuron receptive field width was set to 0.8.

The position error performance and neuron count for threshold allocation algorithms 3.47–3.51 in the presence of visual noise are plotted in figures 3.10 and 3.11 respectively, while the position error performance and neuron count of algorithms 3.52–3.56 with noise, again respectively, are plotted in figures 3.12 and 3.13.

The figures show that the accuracy of the networks, as a function of the number of trials, varies widely, and that 11 mm average positioning error seems to be a lower bound. From the plots of neuron counts, however, it is apparent that some networks have far fewer neurons than others, and that accuracy is correlated with the number of neurons allocated, as we might expect. Figure 3.14, which plots the position error as a function of the number of neurons rather than the number of trials shows the striking result that, for uniform random input presentations, accuracy depends predominantly upon the number of neurons, not upon the way the allocation threshold is varied. The figures also show that the most responsive, or fastest-learning, allocation algorithms, those of eqs. 3.47, 3.48, 3.49, 3.54, and 3.55, have nearly identical performance and

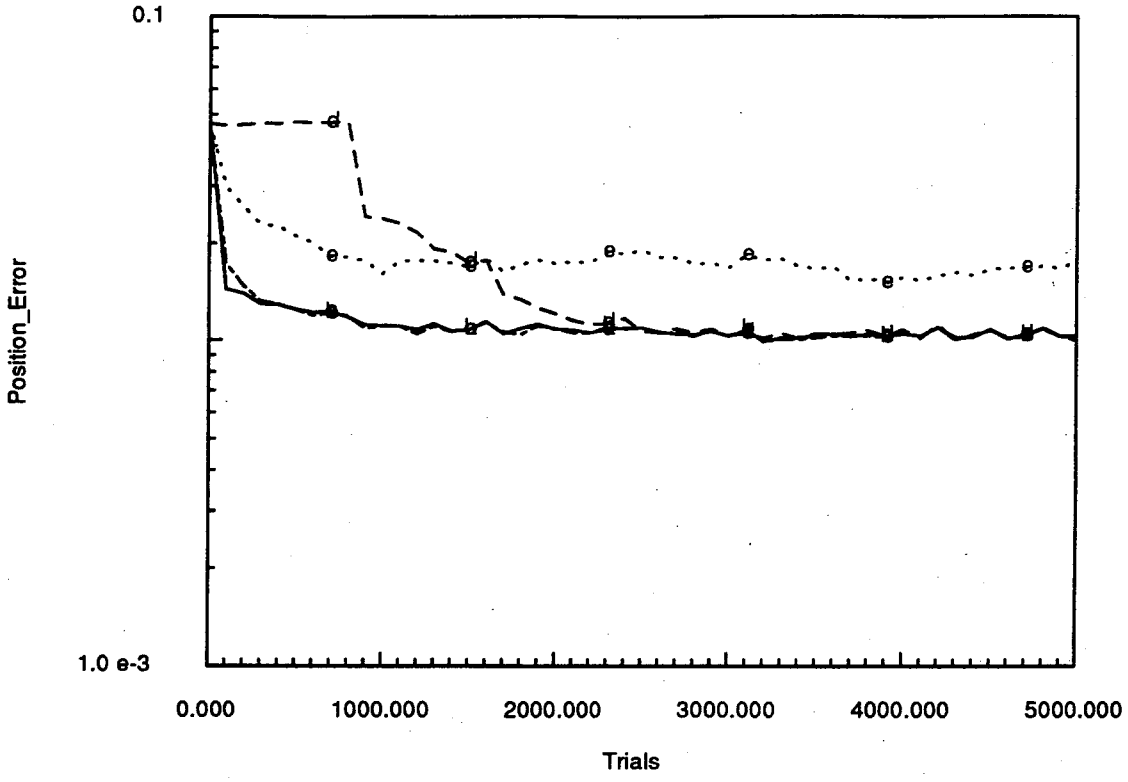

Figure 3.10: Positioning error for allocation algorithms of eqs. 3.47–3.51 in the presence of visual noise. Graphs are labeled (a)–(e), with (a) corresponding to 3.47, (b) to 3.48, (c) to 3.49, (d) to 3.50, and (e) to 3.51. Graphs (a), (b), and (c) largely overlap. The Gaussian (graph d) allocates no neurons for nearly 800 trials due to the slowly decreasing allocation threshold.

are liberal at allocating neurons.

Grid-Based Allocation

Experiments in which neurons were allocated at grid points rather than at random locations were also performed. These included fixed-resolution grids with neurons spaced equally along the axes, as well as variable-resolution grids composed of a sum of successively finer subgrids in which subgrid $i+1$ had twice as many grid points per axis (and therefore 2^n many neurons) as subgrid i. The motivation was to force the network to learn large-scale behavior before allowing the input space to

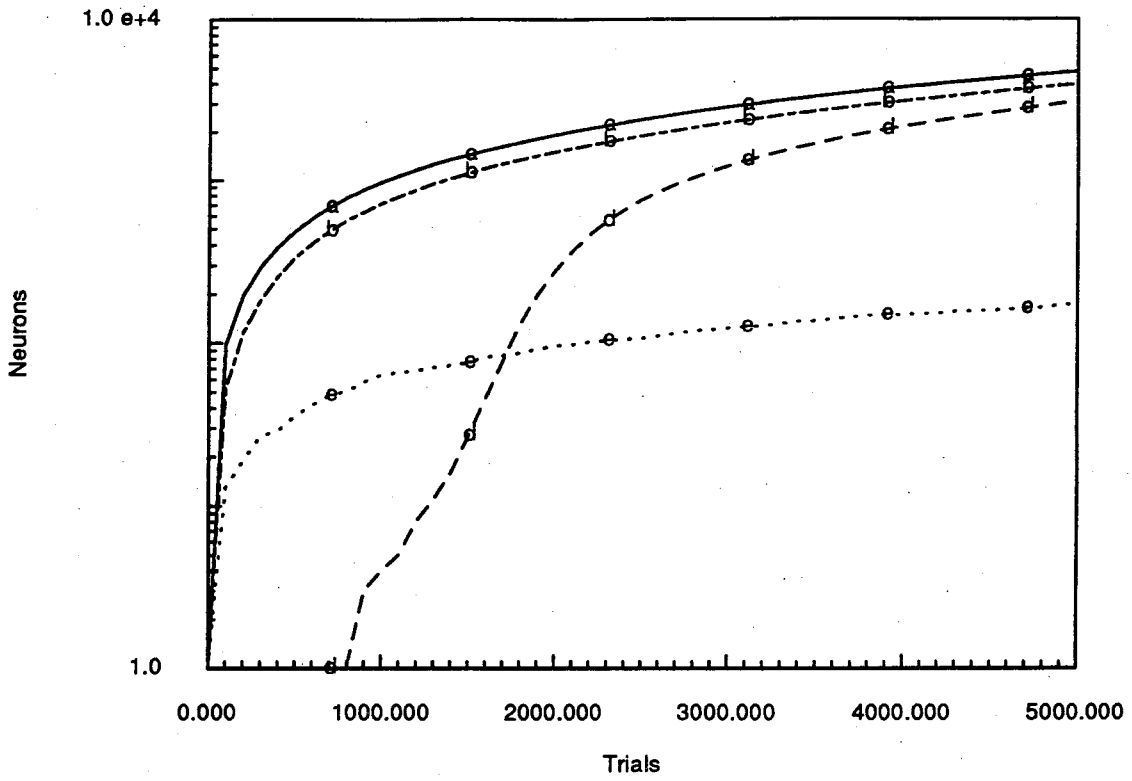

Figure 3.11: Neuron counts for allocation algorithms of eqs. 3.47–3.51 in the presence of visual noise. The graphs are labeled (a)–(e), with (a) corresponding to 3.47, (b) to 3.48, (c) to 3.49, (d) to 3.50, and (e) to 3.51. Graphs (a) and (c) overlap.

become fragmented. As can be seen in figure 3.15 which used a 568-neuron virtual neuron network, random allocation had the best performance, but variable-resolution grids also performed relatively well. In the noisy case, however, performance of the three approaches was comparable. This indicates that using electronic neural network components in which neuron locations are pre-allocated will not seriously degrade performance in real situations. Initial learning as portrayed in the plots for the variable- and fixed-resolution grid cases does not correctly represent network learning capabilities since network evaluation trials were only conducted upon grid or subgrid completion. The number 568 is the number of neurons that is actually allocated in a variable-resolution grid with four subgrids. Not all grid points receive neurons because singular points are rejected. All three allocation approaches used 568-neuron

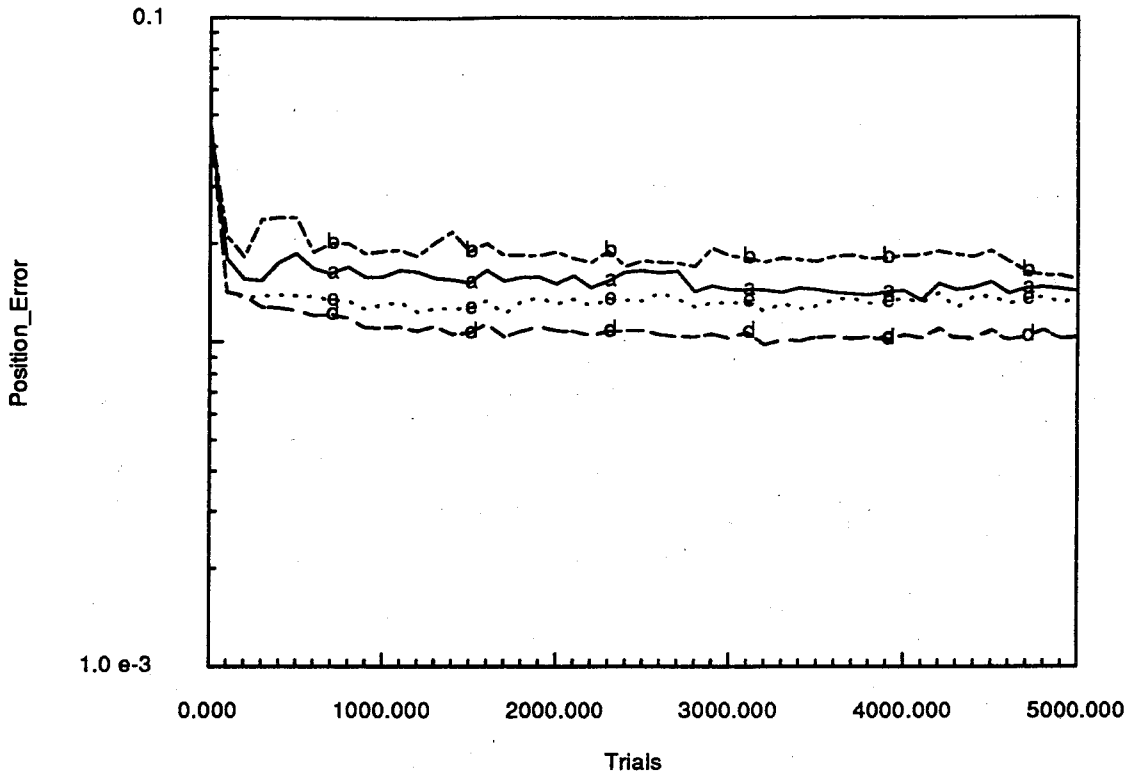

Figure 3.12: Positioning error for allocation algorithms of eqs. 3.52–3.56 in the presence of visual noise. Graphs are labeled (a)–(e), with (a) corresponding to 3.52, (b) to 3.53, (c) to 3.54, (d) to 3.55, and (e) to 3.56. Graphs (c) and (d) overlap.

networks to ensure fair comparison. Extra neurons necessary in the case of the fixed-resolution network were allocated randomly.

3.5.3 Adaptation

When network accuracy is not acceptable, but it is impossible or undesirable to allocate new neurons, adaptation algorithms are employed in an attempt to improve accuracy by adjusting neuron parameters. Adaptation also smooths errors induced by noise during allocation. Even though neuron allocation is important in terms of setup and initial learning speed, only a finite number of neurons can be allocated, and

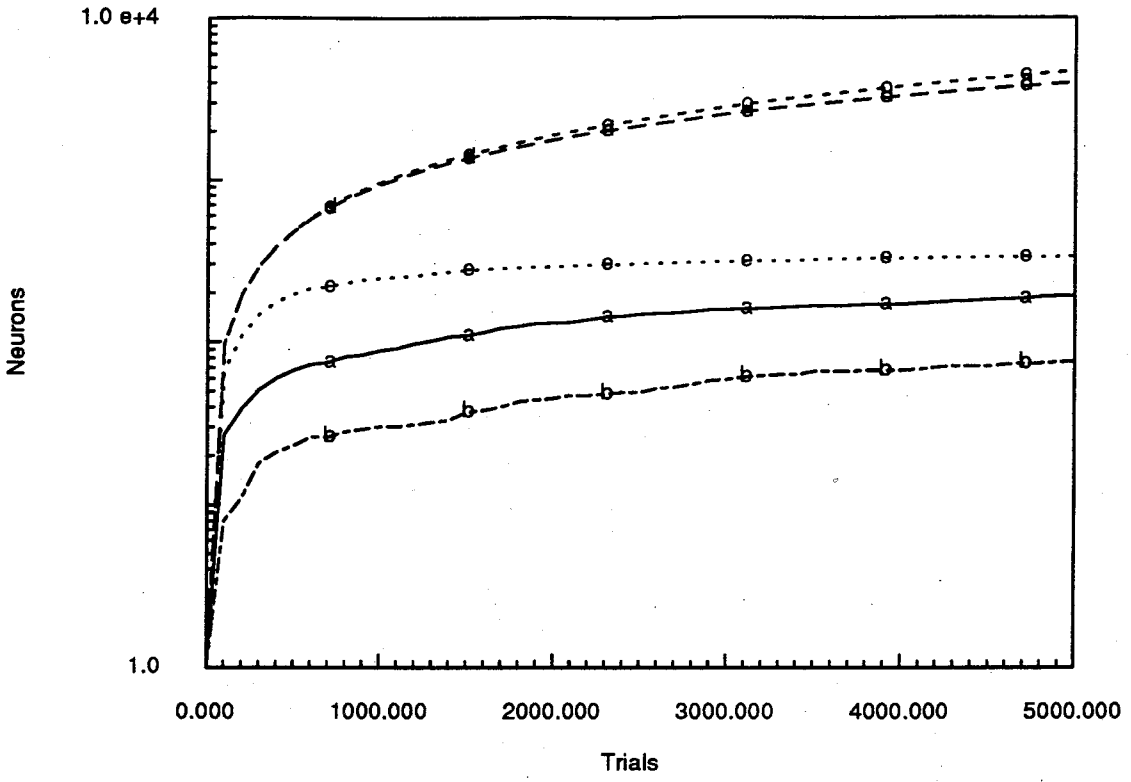

Figure 3.13: Neuron counts for allocation algorithms of eqs. 3.52–3.56 in the presence of visual noise. The graphs are labeled (a)–(e), with (a) corresponding to 3.52, (b) to 3.53, (c) to 3.54, (d) to 3.55, and (e) to 3.56. Graphs (c) and (d) nearly overlap.

adaptation algorithms play the most important role in long-term system performance.

Three adaptation approaches are described here. The first is based on gradient descent. It adjusts the output weights, the neuron center locations, and the neuron width parameters. The second is a novel learning algorithm that adjusts the weights of the currently active neurons according to the activity of a virtual neuron at the sites of the active neurons. The virtual neuron is centered at the input pattern. The last is an implementation of the CMAC learning algorithm [Albus (1975b), Albus (1975a), Albus (1981), Miller (1987)]. It also adjusts just the output weights. CMAC itself is based on a model of the cerebellum [Albus (1972)].

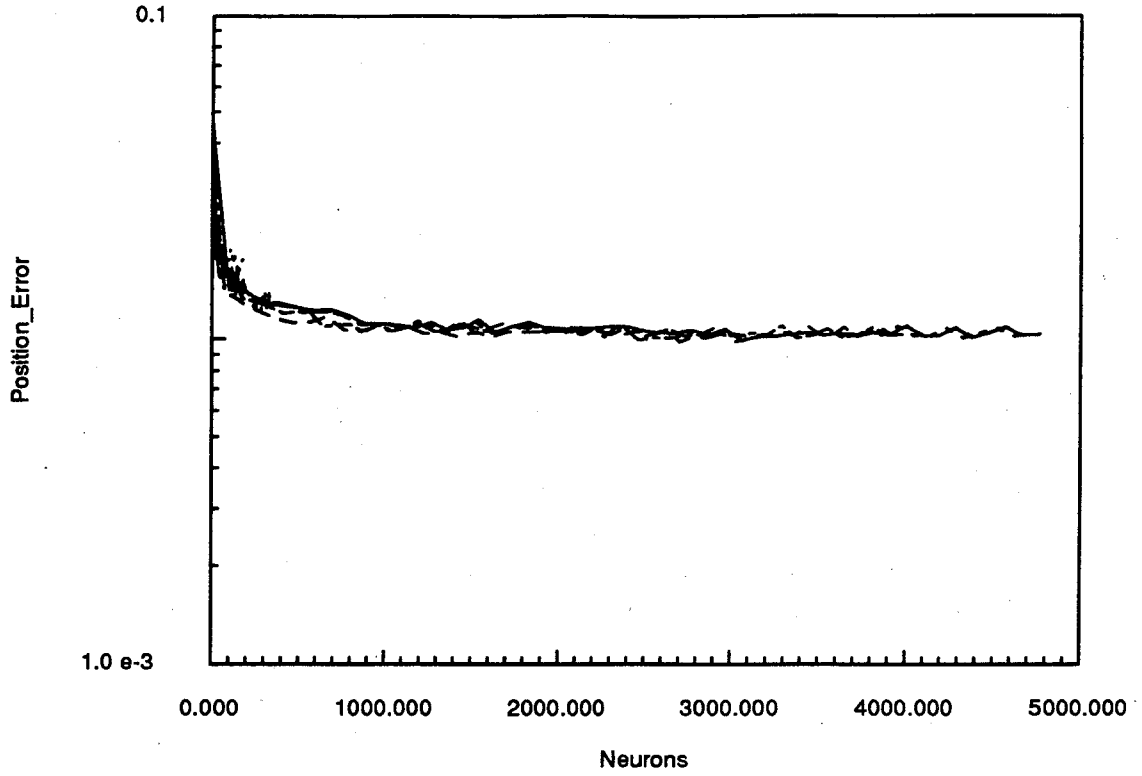

Figure 3.14: Positioning error as a function of neuron count for allocation algorithms of eqs. 3.47–3.56 with visual noise. Noisy behavior at low neuron counts is caused by error fluctuations arising from random position sampling while the neuron count for some allocation algorithms is changing slowly. Labels are omitted for clarity.

Gradient Descent Adaptation

The network output, hence the output error, is a known function of the network weights and neuron parameters. In gradient descent adaptation, the gradient descent method [Ralston and Rabinowitz (1978), Press et al (1988)] is applied to reduce the output error by adjusting the network weights and parameters.

If, for an input vector $\mathbf{x}$, an "energy," E, is defined as

$$E = \frac{1}{2}\sum_{k=1}^{m}(T_k - O_k)^2, \tag{3.57}$$

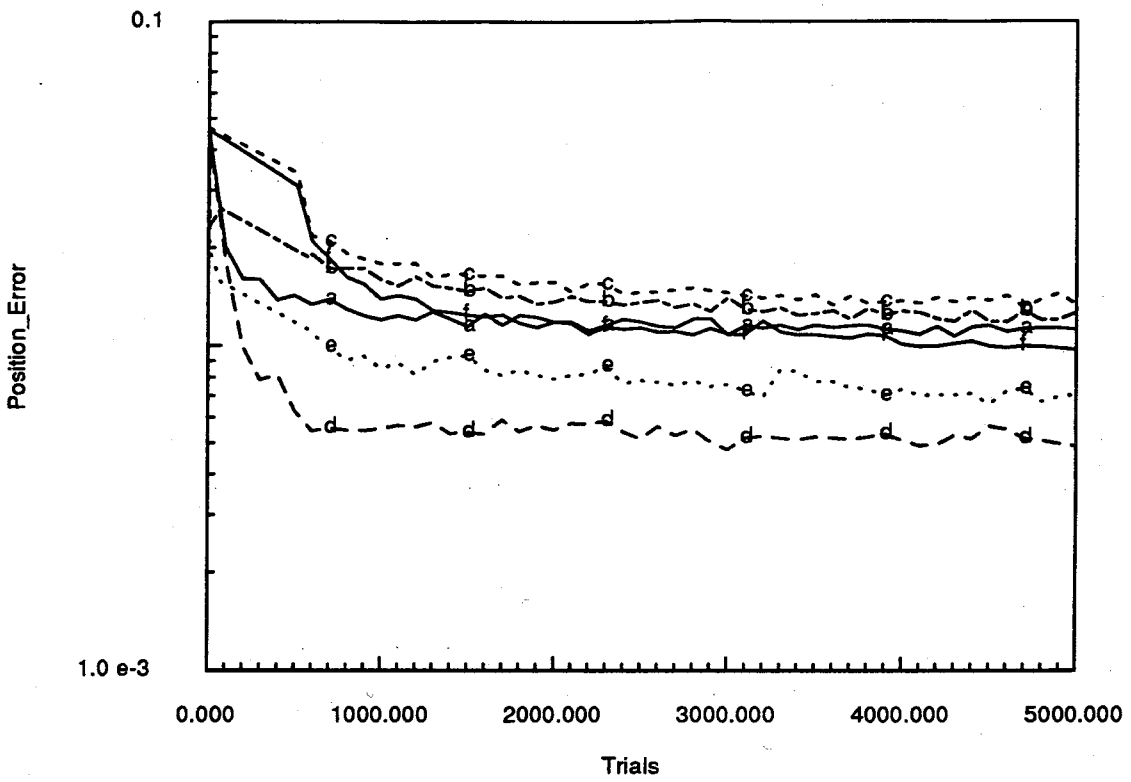

Figure 3.15: Error-correction performance of networks allocating neurons at random locations and at grid points. Plots (a), (b), and (c) are for random, variable-resolution grid, and fixed-grid neuron locations, respectively, in the presence of noise. Plots (d), (e), and (f) are for the same respective neuron location approaches except that noise is absent.

with $\mathbf{T}$ the desired or target m-dimensional network output vector and $\mathbf{O}$ the observed output vector with components given by

$$O_k = \sum_{\text{all neurons } l} (w_{kl} o_l), \tag{3.58}$$

then the (additive) parameter adjustments for neuron j are obtained by using the chain rule:

$$\Delta w_{kj} = -\lambda \frac{\partial E}{\partial w_{kj}} = \lambda o_j(z_j)(T_k - O_k) \tag{3.59}$$

$$\Delta c_{ji} = -\lambda \frac{\partial E}{\partial c_{ji}} = -2\lambda o_j'(z_j)(x_i - c_{ji}) \sum_{k=1}^{m} w_{kj}(T_k - O_k) \tag{3.60}$$

$$\Delta R_j = -\lambda \frac{\partial E}{\partial R_j} = \lambda \frac{\partial o_j(z_j)}{\partial R_j} \sum_{k=1}^{m} w_{kj}(T_k - O_k). \tag{3.61}$$

Here $o_j(z_j)$ is the response of neuron j to $z_j = \|\mathbf{x} - \mathbf{c}_j\|^2$, which is the squared distance of the input vector $\mathbf{x}$ from its receptive field center $\mathbf{c}_j$; $o_j'(z_j)$ is the derivative of neuron j's response to z_j; w_{kj} is the weight coupling the response of neuron j to component k of the network output vector $\mathbf{O}$; c_{ji} is component i of $\mathbf{c}_j$; R_j is the receptive field width parameter for neuron j; λ is the learning rate.

The factors $o_j'(z_j)$ and $\partial o_j(z_j)/\partial R_j$ are specific to the particular neuron response function. For the Gaussian we have:

$$o_j'(z_j) = -\frac{1}{R_j} e^{-z_j/R_j} \tag{3.62}$$

$$\frac{\partial o_j(z_j)}{\partial R_j} = \frac{z_j}{R_j^2} e^{-z_j/R_j}. \tag{3.63}$$

In operation, parameter changes are made to the network after each trial. Waiting for many trials to update the network tends to make the network unstable, but empirically there is little difference between small waiting intervals.

Network performance is sensitive to the learning rate λ: too low a rate causes slow convergence while too high a rate causes the network to tend toward instability. A learning rate of $\lambda = 0.02$ gave the best performance. System error correction performance with various learning rates for a 200-neuron network initialized to small output weights is shown in figure 3.16.

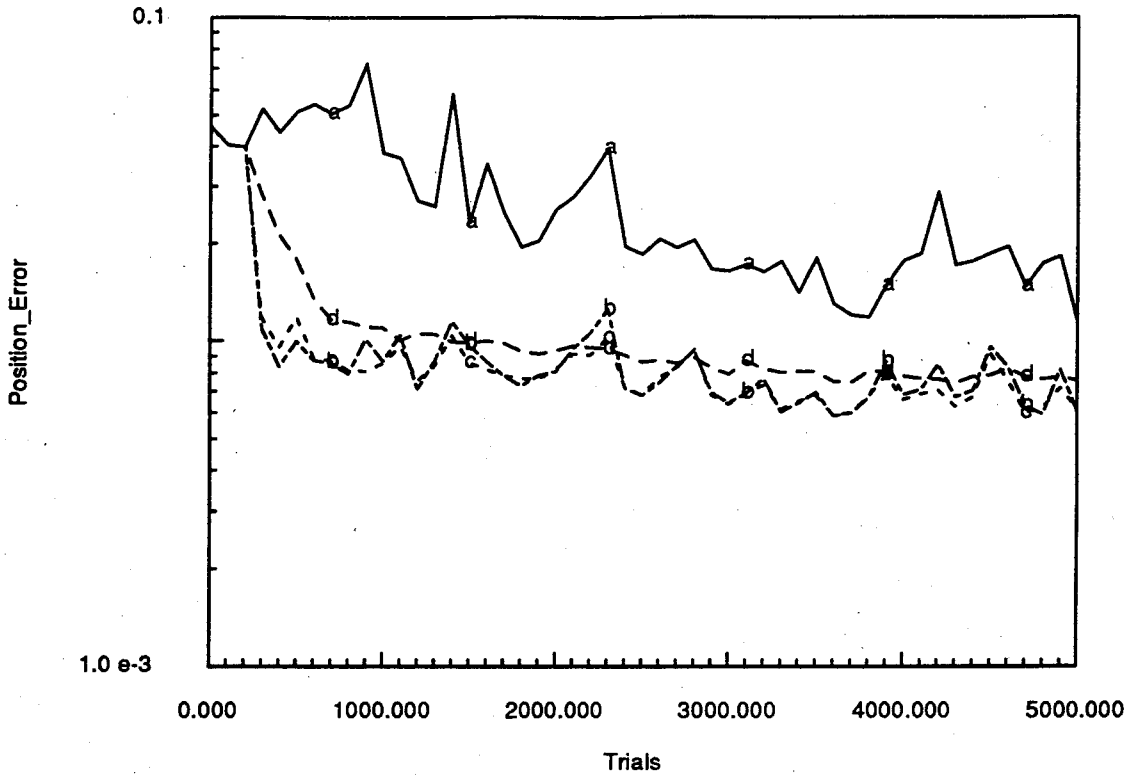

Figure 3.16: Error correction performance of gradient descent adaptation for different learning rates. The learning rates are 1.0, 0.1, 0.06, and 0.01 in plots (a) through (e) respectively. The network was limited to 200 randomly-positioned neurons. All output weights were initialized to 0.001.

Virtual Neuron Adaptation

In virtual neuron adaptation, which is novel to this work, network accuracy is improved at each trial by adjusting the output weights, w_{kj}, of the *active*[9] neurons j in such a way that the network output error for the input is reduced and the effect of the changes is localized to a region around the input. Receptive field widths and neuron center locations are not adjusted. This is reminiscent of the Platt neuron allocation scheme and is similar to the CMAC learning algorithm discussed below.

Weight adjustments are computed by centering a virtual neuron, with width pa-

[9]A neuron is active for an input if its activity is above a specified threshold.

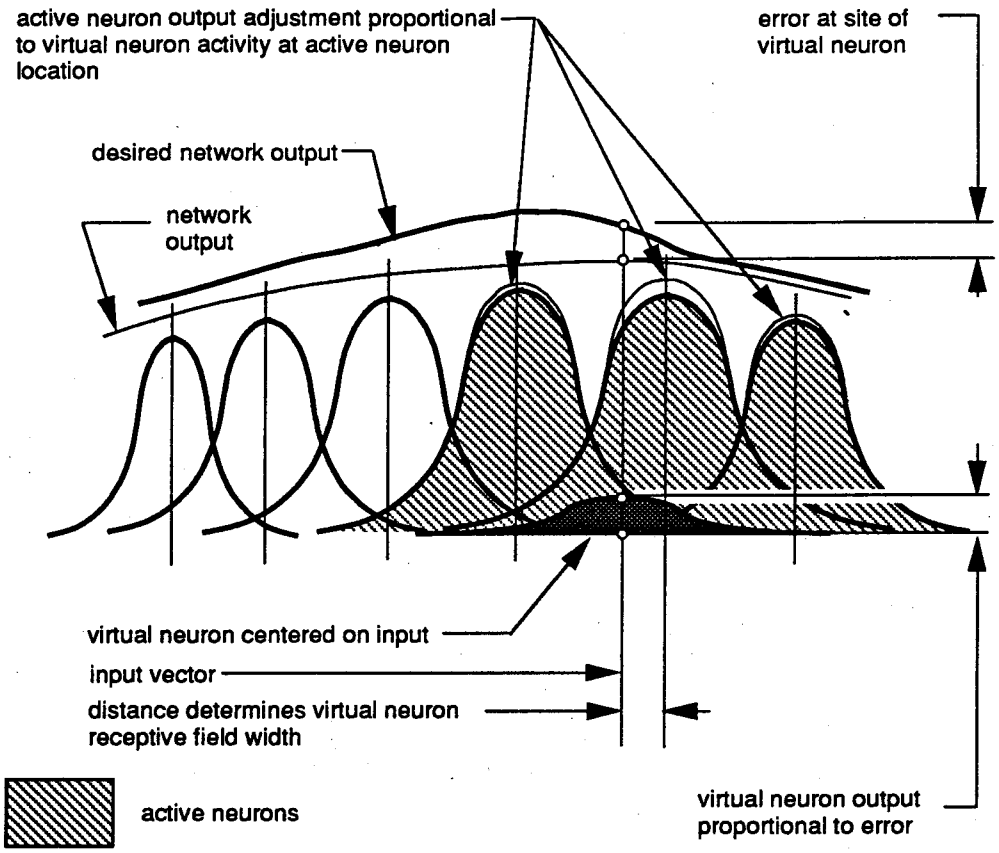

Figure 3.17: Adaptation using virtual neurons.

rameter as described below, at the current input vector. The change in the output weight, Δw_{kj} coupling active neuron j to network output component k is proportional to the output error of the component and to the activity of the virtual neuron due to an input centered at the location of neuron j. This is shown schematically in figure 3.17. Since neuron activation functions are localized, the effect of changes is localized. Neurons near the input vector have the largest weight changes, while weight changes become smaller as the neurons become more distant. Adopting the notation used in the gradient descent above, and writing δw_{kl} for the change in weight w_{kl} necessary (with the changes in the other weights) to correct ϵ_k completely, and letting $\mathcal{A}$ be the

set of active neurons, we can write

$$T_k = \sum_{l \epsilon \mathcal{A}} (w_{kl} + \delta w_{kl}) o_l, \tag{3.64}$$

where o_l is the output of neuron l due to the input vector $\mathbf{x}$, so that

$$\epsilon_k = \sum_{l \epsilon \mathcal{A}} \delta w_{kl} o_l. \tag{3.65}$$

If a virtual neuron $\mathcal{V}$ with width parameter $R_\mathcal{V}$ is centered at $\mathbf{x}$, and if δw_{kj} is made proportional to $\mathcal{V}(\mathbf{c}_j - \mathbf{x}; R_\mathcal{V})$, the virtual neuron's activity at $\mathbf{c}_j$, the location of neuron j, then δw_{kj} becomes

$$\delta w_{kj} = C_k \mathcal{V}(\mathbf{c}_j - \mathbf{x}; R_\mathcal{V}), \tag{3.66}$$

where C_k is a constant for network output component k. Then solving for ϵ_k,

$$\epsilon_k = \sum_{l \epsilon \mathcal{A}} C_k \mathcal{V}(\mathbf{c}_l - \mathbf{x}; R_\mathcal{V}) o_l, \tag{3.67}$$

so

$$C_k = \frac{\epsilon_k}{\sum_{l \epsilon \mathcal{A}} \mathcal{V}(\mathbf{c}_l - \mathbf{x}; R_\mathcal{V}) o_l}. \tag{3.68}$$

Then, letting $\Delta wkj = \lambda \delta w_{kj}$, with λ the learning rate,

$$\Delta w_{kj} = \lambda C_k \mathcal{V}(\mathbf{c}_j - \mathbf{x}; R_\mathcal{V}) = [\frac{\lambda}{\sum_{l \epsilon \mathcal{A}} \mathcal{V}(\mathbf{c}_l - \mathbf{x}; R_\mathcal{V}) o_l}] \epsilon_k \mathcal{V}(\mathbf{c}_j - \mathbf{x}; R_\mathcal{V}). \tag{3.69}$$

The term in square brackets is a constant for all active neurons and for all components of the output vector, so it has to be computed only once per learning trial. For non-negative virtual neuron activation functions with sufficient width, the denominator is positive if there are active neurons. The virtual neuron width parameter $R_\mathcal{V}$ can be

selected in various ways, including fixing its value at the outset. In this work, virtual neuron width parameters are selected in the same way as in neuron allocation: the width is chosen so the activation drops to one-half at a fraction h of the distance from the input vector $\mathbf{x}$ to the nearest active neuron. In addition, the virtual neuron activation function was chosen to have the same form as the network neurons. This means that the response of the virtual neuron at the site of neuron j can be calculated by neuron j if its width is temporarily set to that of the virtual neuron. This is possible because of the radial symmetry of the response functions considered here.

Again, algorithm performance is sensitive to the learning rate. Error-correction performance with various learning rates is shown in figure 3.18. The network uses 200 randomly-positioned neurons initialized to small output weights

Cerebellar Model Articulation Controller (CMAC) Adaptation

As discussed in appendix A, in the CMAC adaptation algorithm [Albus (1975b), Albus (1975a)] the output weights of all neurons activated by the input and connected to the same (vector) component of the output are adjusted by the same amount, which is proportional to the output error for the component.[10]

This is shown schematically in figure 3.19. If we let $\mathcal{V}$ be a constant function over the set of active neurons $\mathcal{A}$, then, absorbing $\mathcal{V}$ into C_k, eq. 3.66 becomes:

$$\delta w_{kj} = C_k, \tag{3.70}$$

[10]It is the CMAC adaptation algorithm that is being evaluated here. CMAC itself is implemented using linear discriminant functions rather than Gaussian neurons, and so has somewhat different interpolation properties.

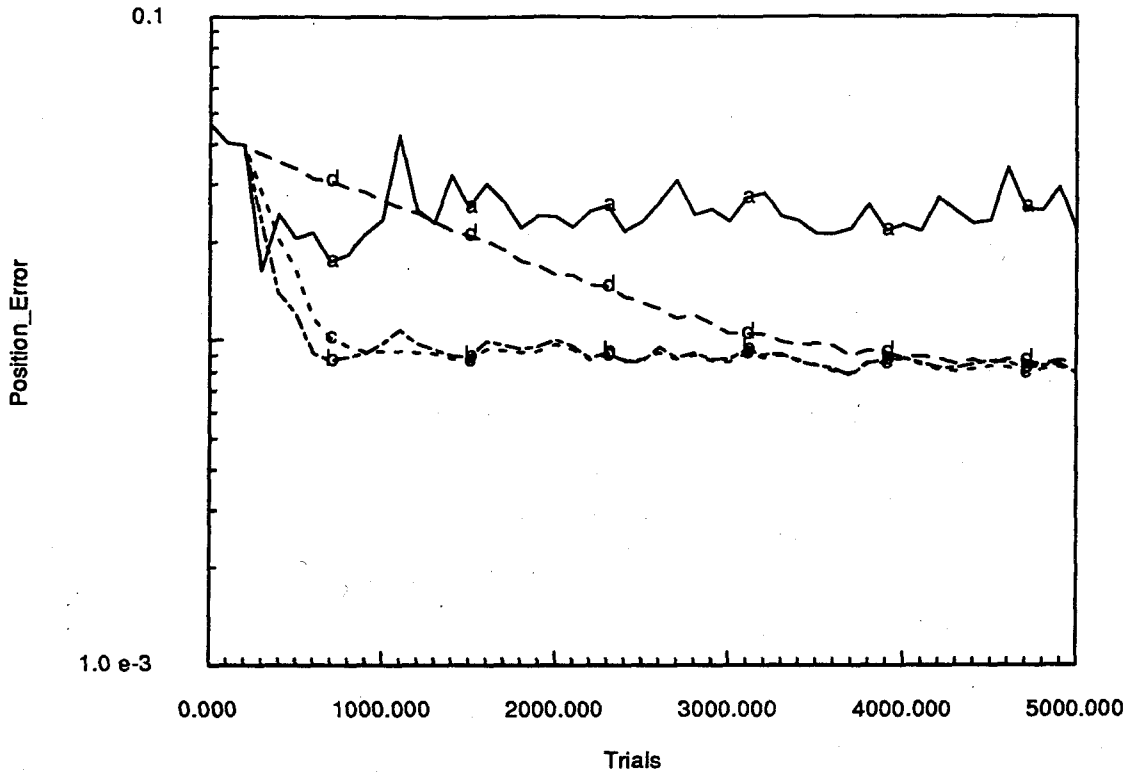

Figure 3.18: Calibration error correction performance of the virtual neuron adaptation algorithm for different learning rates. The learning rates are 1.0, 0.1, 0.06, and 0.01 in plots (a) through (e) respectively. The network was limited to 200 randomly-positioned neurons. All output weights were initialized to 0.001.

where C_k is a constant as before. Then

$$\epsilon_k = \sum_{l \epsilon \mathcal{A}} \delta w_{kl} o_l = C_k \sum_{l \epsilon \mathcal{A}} o_l, \tag{3.71}$$

so that

$$C_k = \frac{\epsilon_k}{\sum_{l \epsilon \mathcal{A}} o_l}. \tag{3.72}$$

Then, letting $\Delta wkj = \lambda \delta w_{kj}$, with λ again the learning rate,

$$\Delta w_{kj} = [\frac{\lambda}{\sum_{l \epsilon \mathcal{A}} o_l}] \epsilon_k. \tag{3.73}$$

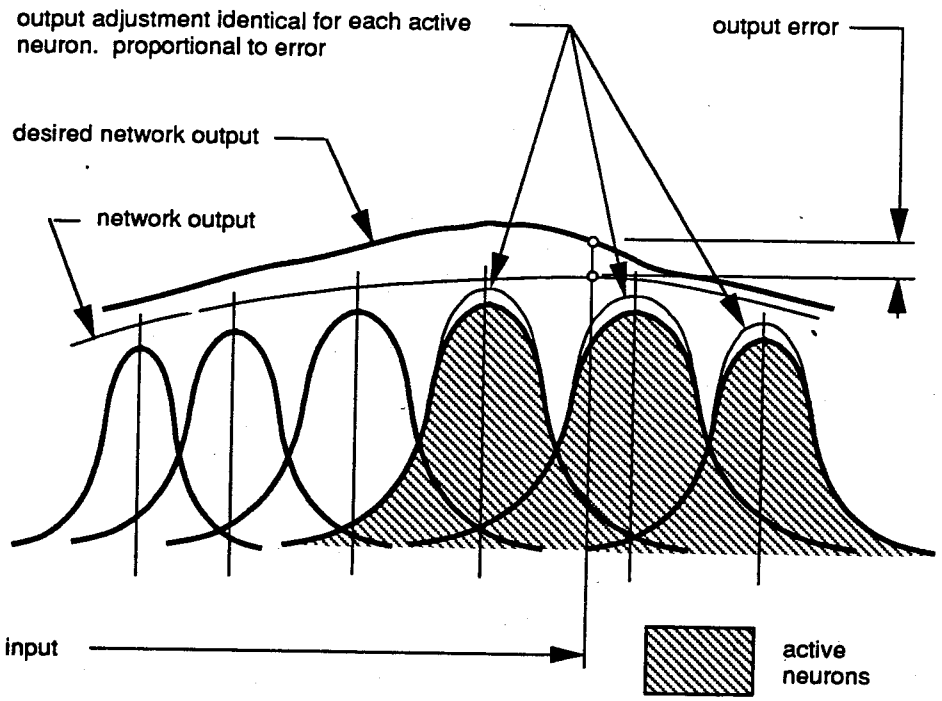

Figure 3.19: CMAC adaptation.

Again, the term in square brackets is a constant for all active neurons and all output components, so it need to be computed just once per learning trial. The denominator is nonzero if any neurons are above threshold.

The CMAC learning algorithm is the most sensitive of the three to large learning rates, as can be seen in figure 3.20, which shows calibration error performance for several values of λ.

Adaptation Algorithm Performance Comparison

The virtual neuron algorithm outperformed both gradient descent and the CMAC learning algorithms in simulations, although in their best runs (shown in figure 3.21) the errors were all within a millimeter or so of each other. Issues associated with implementing the adaptation algorithms in analog VLSI are discussed in appendix B.

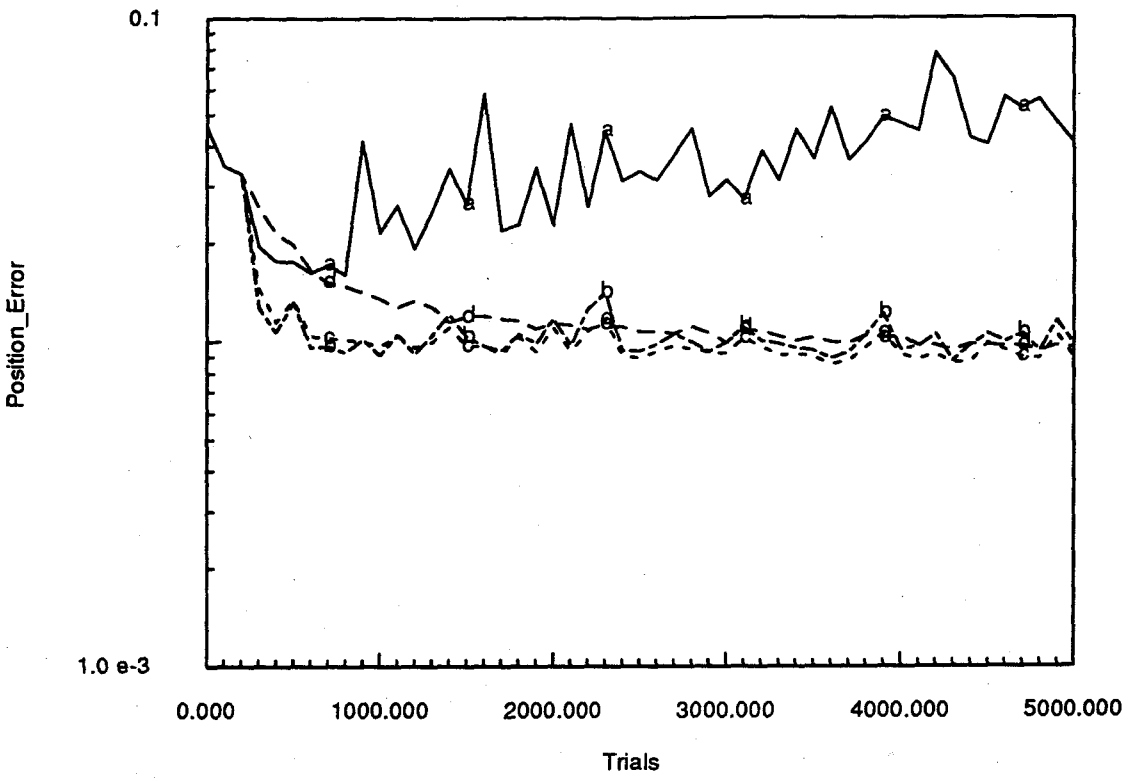

Figure 3.20: Calibration error correction performance of CMAC adaptation for different learning rates. The learning rates are 1.0, 0.1, 0.06, and 0.01 in plots (a) through (e) respectively. The network was limited to 200 randomly-positioned neurons. All output weights were initialized to 0.001.

3.5.4 The Preferred Network

Based on the results of the allocation and adaptation tests, the network selected for the remainder of this work uses virtual neuron adaptation and the allocation algorithm of eq. 3.54, in which the allocation threshold is just set to zero. The algorithm of 3.54 is the simplest and, in the noisy case, learns as fast and as accurately as the others.[11] It effectively eliminates the threshold from consideration, thereby reducing

[11]The algorithms of eqs. 3.47 and 3.48 have slightly better performance in the noise-free case.

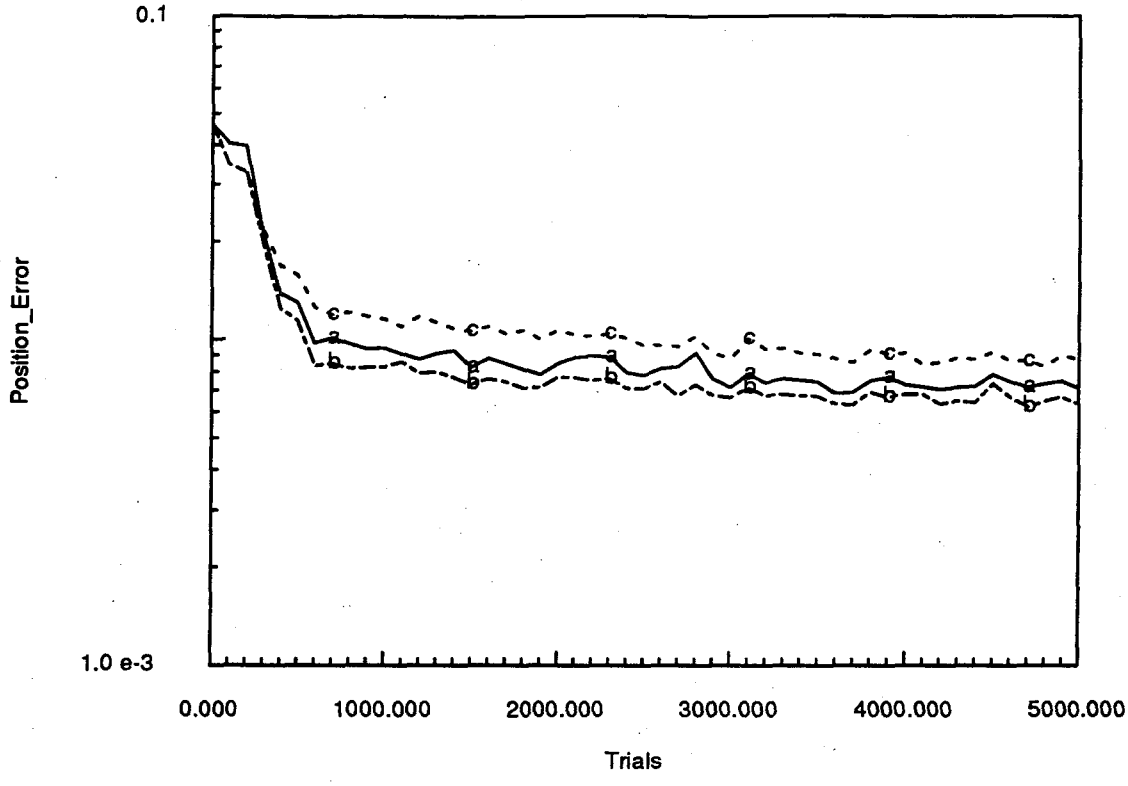

Figure 3.21: Best error-correction performance of the gradient descent (plot (a)), virtual neuron (plot (b)), and CMAC (plot (c)) adaptation algorithms. Learning rates were 0.02, 0.06, and 0.02 respectively. In each case networks were limited to 200 randomly-allocated neurons, and output weights were initialized to 0.001. The virtual neuron algorithm has the best performance.

the complexity of neuron allocation and simplifying its potential realization in hardware. Virtual neuron adaptation was selected because it has the best performance.

Preferred Network Parameters

Principal network parameters included the half maximum fraction, h, which determines tuning width, the neuron on-threshold θ_o, the learning rate λ, the learning error threshold θ_l, the first neuron tuning width R_1, and the number of neurons. A neuron is considered on, or active, by the virtual neuron and CMAC adaptation algorithms if its output is $> \theta_o$. Similarly, learning or adaptation is enabled whenever the network

output error norm is greater than θ_l, which was set to 0.001. The value of θ_l should probably be determined by the noise magnitude, but that was not pursued.

Network parameter values giving the best performance were $h = 1.2$, $\theta_o = 0.1$, and $\lambda = 0.06$. The first neuron tuning width, R_1, was set to $0.1hD$, where D is the diameter (the diagonal) of the input space. The initial width matters since it sets the scale for the range over which adaptation changes will have an effect. A large initial neuron will be above threshold for nearly any input and hence changes in its output weights will have an effect over a significant portion of the input space. Empirically, this makes it more difficult for the network to stabilize. Networks in noisy situations were usually limited to 200 neurons, which gave adequate performance. Adding neurons gave little improvement in performance. In noise-free situations, adding neurons was beneficial.

The Preferred Network's Performance

Figure 3.22 shows the positioning performance of the selected network structure with and without neuron limits and with and without noise. Figure 3.23 shows the positioning error sample standard deviations for the plots in figure 3.22. Performance is shown over 5000 trials. Arm and visual eccentricity values were selected randomly during both training and testing. After each 100 training trials, system performance was evaluated over 1000 randomly-selected positions.

3.5.5 Visual Noise

The presence of visual noise has a profound effect upon vision calibration performance. Compare figures 3.10 and 3.12, which show the error correction performance of the

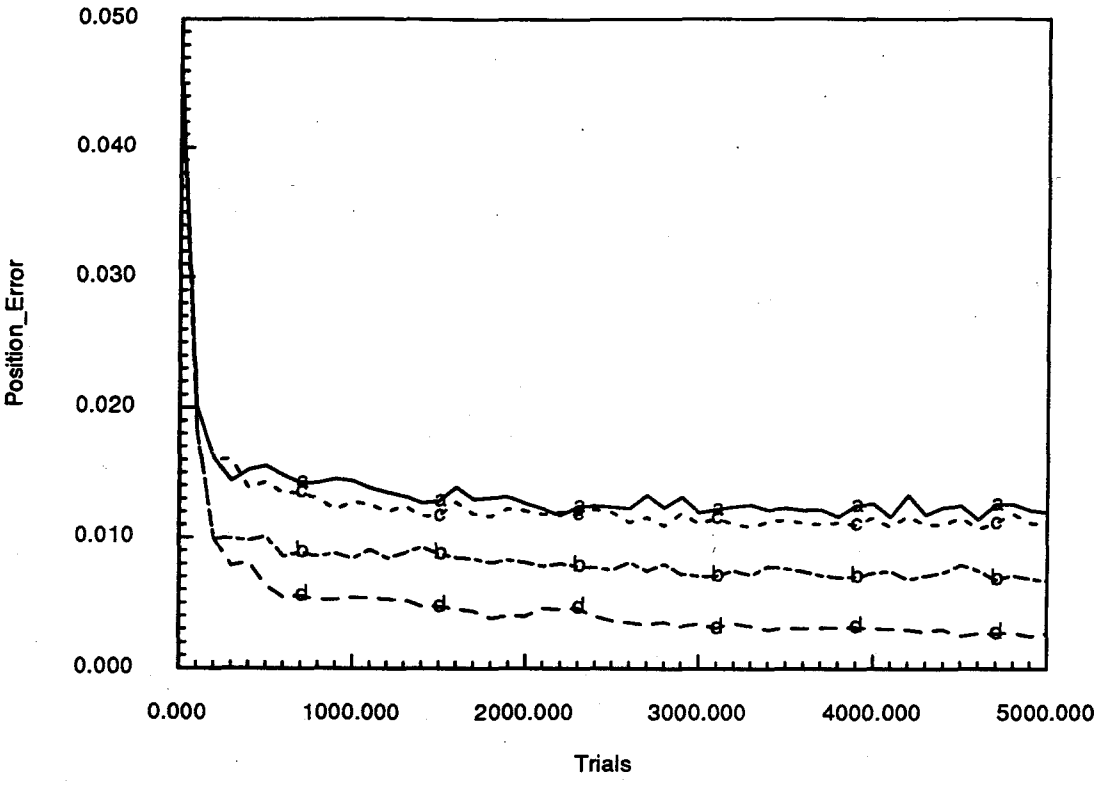

Figure 3.22: Hand-eye system positioning accuracy using virtual neuron networks of the preferred structure. Plot (a) is for a network restricted to 200 neurons with noise. Plot (b) is for a network restricted to 200 neurons but without noise. Plot(c) is for a network with no restriction on the number of neurons, with noise, and plot (d) is for an unrestricted network without noise. Network parameters were identical in all cases. Neurons were allocated at random locations subject to threshold constraints. The network of plot (c) allocated 4770 neurons while that of plot (d) allocated 3771 neurons.

different neuron allocation algorithms in the presence of visual noise, with figures 3.24 and 3.25, which show performance of the same algorithms in the absence of noise, and figure 3.26, which shows the adaptation performance of the 200-neuron network with and without visual noise.

In the noise-free case, any of the neuron allocation algorithms approximates the target angle correction vector function $\mathbf{T}(\mathbf{x})$ perfectly at the points where neurons are allocated, so there is no average position error over those points. In the noisy case, however, there will be an error at each allocation point due to errors in visually

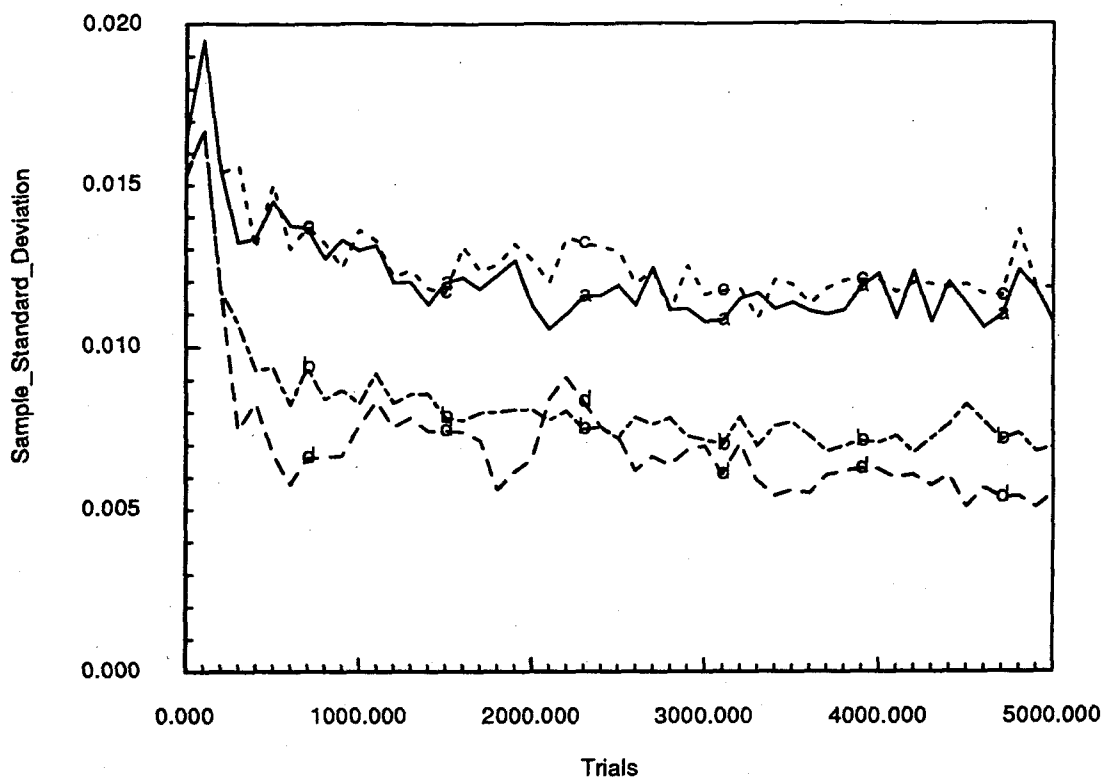

Figure 3.23: Hand-eye system positioning error sample standard deviations for the plots shown in figure 3.22. The order of the plots in the two figures is identical.

determining the position of the arm tip. These errors will manifest themselves as an input error to the network, since the actual target angle vector will not correspond to the target input vector because of the noise. The network will learn the correct target, but for a slightly incorrect position, meaning that the network output vector for the correct input vector will be in error (this depends upon the target function being learned) by some amount **e**. Over the set of n target, or training, inputs, then, there will be an average expected recall error of

$$\frac{1}{n}\sum_{j=1}^{n}\langle \mathbf{e}_j \rangle \;=\; \frac{1}{n}\sum_{j=1}^{n}(\langle e_{j,\alpha}\rangle, \langle e_{j,\beta}\rangle), \tag{3.74}$$

where $\langle \mathbf{e}_j \rangle$ is the expected angle vector error for target j, and $\langle e_{j,\alpha}\rangle, \langle e_{j,\beta}\rangle$ are its

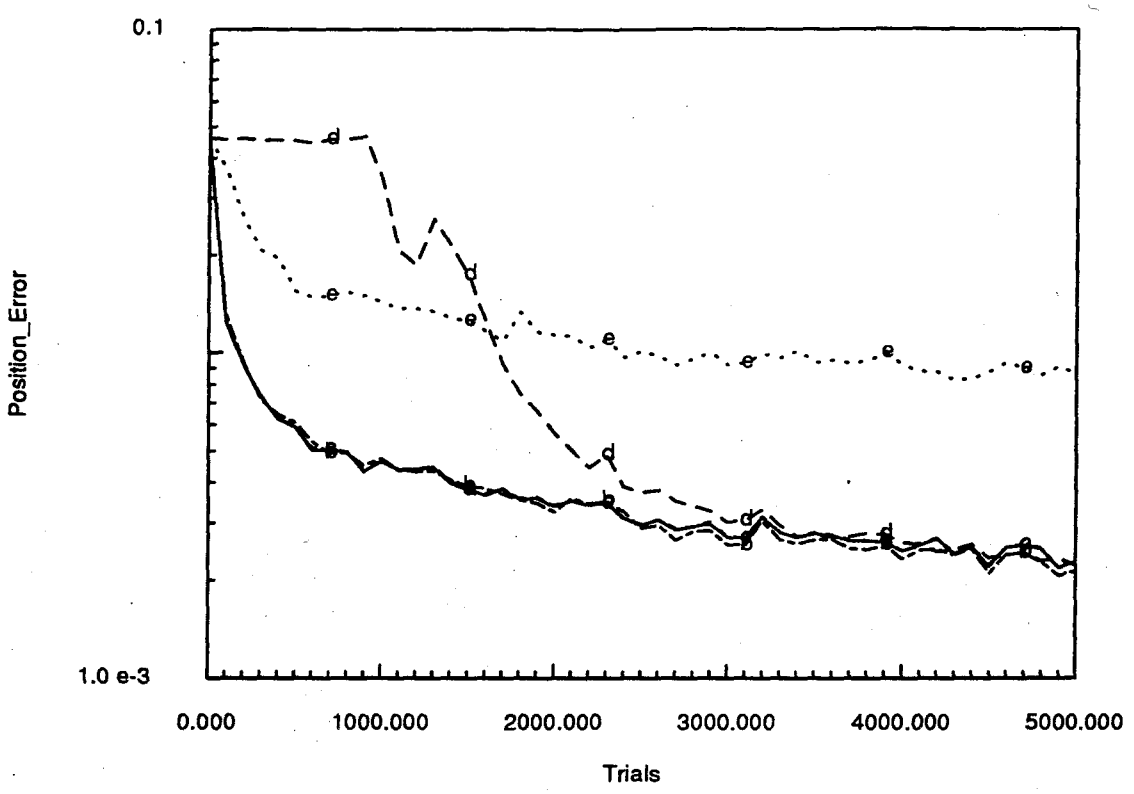

Figure 3.24: Positioning error for allocation algorithms of eqs. 3.47–3.51 without visual noise. Graphs are labeled (a)–(e), with (a) corresponding to 3.47, (b) to 3.48, (c) to 3.49, (d) to 3.50, and (e) to 3.51. Graphs (a), (b), and (c) largely overlap. The Gaussian (graph d) allocates no neurons for over 900 trials due to the slowly decreasing allocation threshold.

components. The average root mean square position error is

$$\frac{1}{n}\sum_{j=1}^{n}\sqrt{\langle \mathbf{e}_j^2\rangle} \;=\; \frac{1}{n}\sum_{j=1}^{n}\sqrt{\langle e_{j,\alpha}^2\rangle + \langle e_{j,\beta}^2\rangle}. \tag{3.75}$$

The recall error will tend to average to zero over the training inputs, but the root mean square position error, being always nonnegative, will not. General analysis for points not in the training set is more complicated and is not addressed here. In addition to limiting attainable accuracy, noise limits the efficiency of representation by uselessly increasing the number of neurons. Accuracy without noise plotted as a function of the number of neurons (figure 3.27) shows clearly that in the noise-free case, different allocation algorithms *do* have different representational efficiencies. This is not true

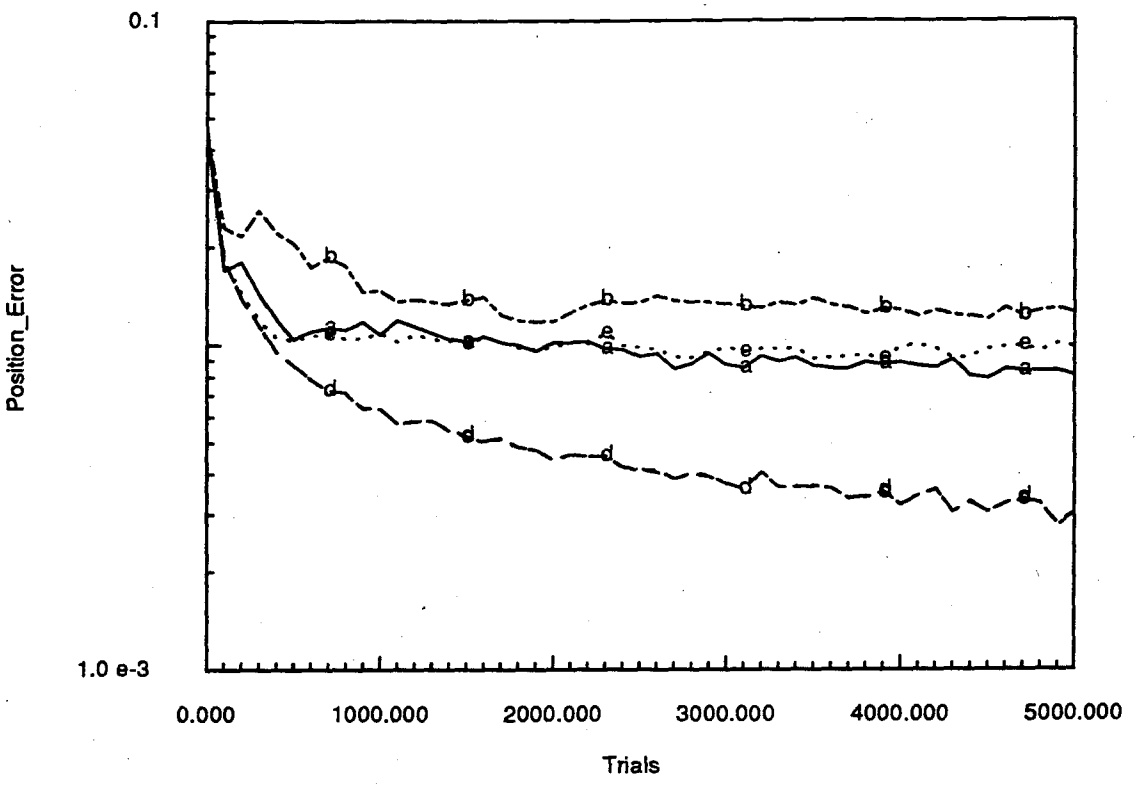

Figure 3.25: Positioning error for allocation algorithms of eqs. 3.52–3.56 without visual noise. Graphs are labeled (a)–(e), with (a) corresponding to 3.52, (b) to 3.53, (c) to 3.54, (d) to 3.55, and (e) to 3.56. Graphs (c) and (d) overlap.

in the noisy case because the error due to noise, being random and above the learning threshold, triggers neuron allocation even where it is unnecessary, ultimately flooding the network with neurons that can provide no accuracy improvement.

Noise and Error-Free Kinematics

Consider a hand-eye system with no kinematic errors or perturbations that employs a virtual neuron network with 200 neurons. In the noise-free case, the system will have no position errors, while in the noisy case position errors will reflect only the effects of noise. The average positioning error over 5000 trials for both cases is shown in figure 3.28. In the noisy case the network allocated its full complement of 200 neurons and

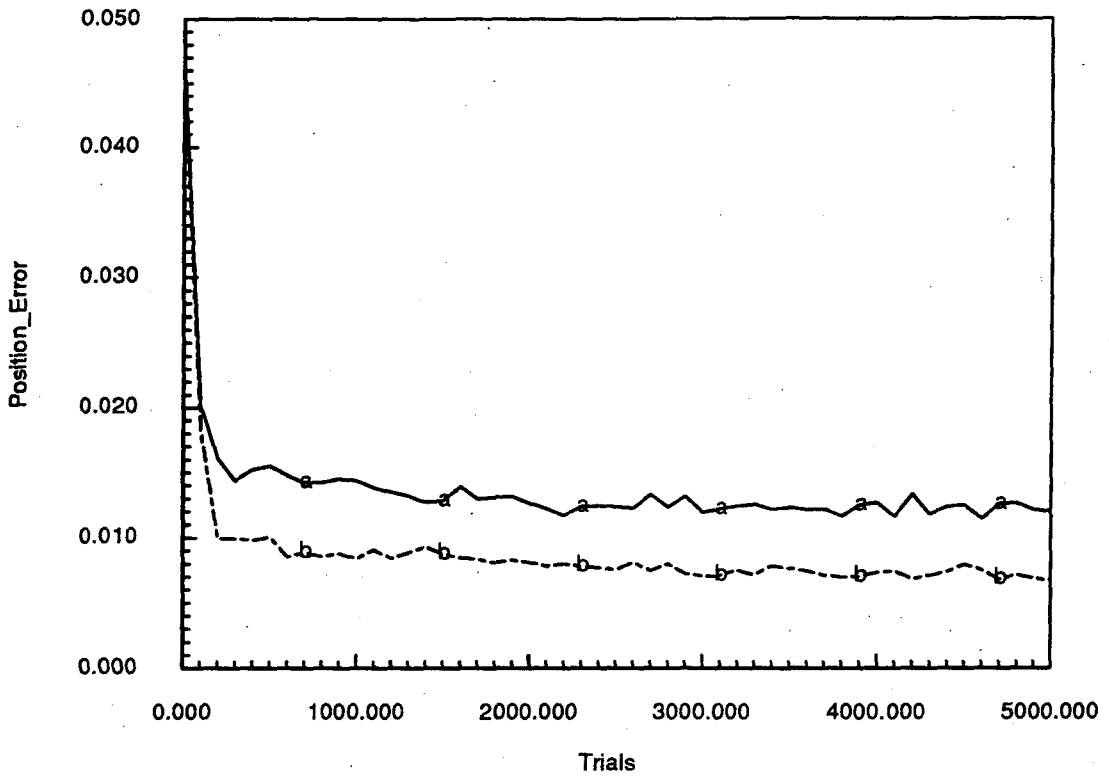

Figure 3.26: System allocation and adaptation behavior with and without noise using a virtual neuron network with 200 neurons. Plot (a) is with noise; plot (b) is without.

had an average position error of about 9 mm; in the noise-free case there was no error and no neurons were allocated.

A Simplified Noisy Adaptation Model

To help understand network adaptation behavior, consider the simplified problem of learning to generate a fixed output vector $\mathbf{o}$ equal to a fixed target vector $\mathbf{T}$ that corresponds to a fixed but noisy network input vector $\mathbf{x}$. That is, each network input is of the form

$$\mathbf{i} = \mathbf{x} + \nu, \tag{3.76}$$

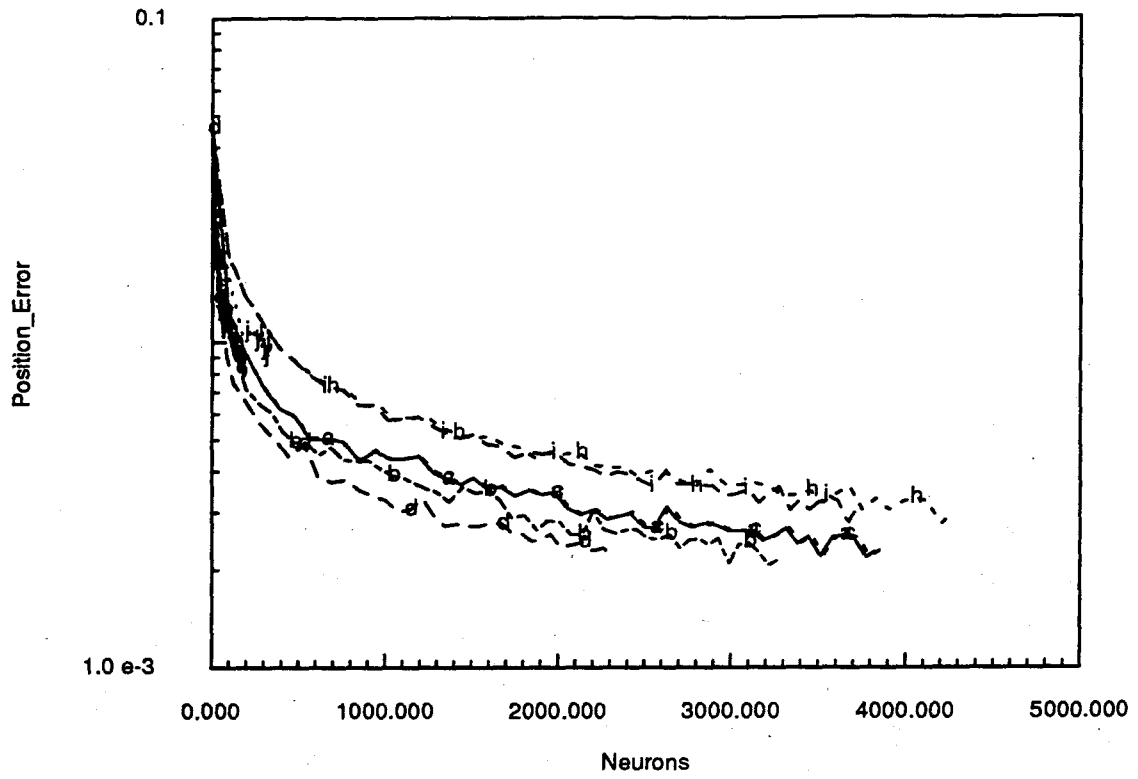

Figure 3.27: Positioning error as a function of neuron count for allocation algorithms of eqs. 3.47–3.56 without visual noise. Noisy behavior at low neuron counts is caused by error fluctuations arising from random position sampling while the neuron count for some allocation algorithms is changing slowly. Graphs are labeled (a)–(j), with (a) corresponding to 3.47, (b) to 3.48, (c) to 3.49, (d) to 3.50, (e) to 3.51, (f) to 3.52, (g) to 3.53, (h) to 3.54, (i) to 3.55, and (j) to 3.56. Allocation algorithm 3.50, corresponding to graph (d), has the most efficient representation.

where $\mathbf{i}$ is the network input and ν is additive noise. As described above, noise appears as a network access error. In the current situation this corresponds to repeatedly positioning the arm at a fixed location, sensing its position using a noisy vision sensor, and using the noisy vision output as network input to generate (and learn) the arm position angle corrections. These corrections will then be added to the angles calculated using the nominal kinematics to generate arm command angles. Noise will cause the command angles so calculated to deviate from the actual angles. Since we are interested in the network's long-term behavior, and allocation is a finite transient that is active only until all the neurons have been allocated, we assume that a network

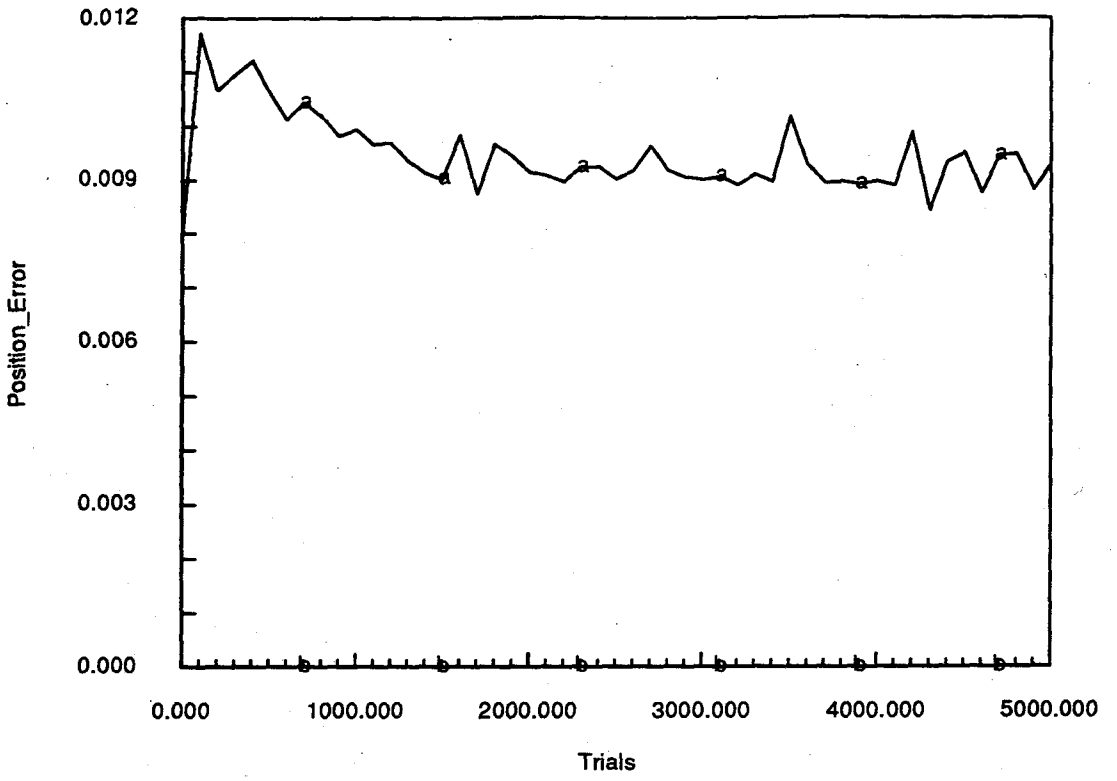

Figure 3.28: Average corrected position error in a hand-eye calibration system with no kinematic errors. Plot (a) shows positioning error in the presence of noise. Plot (b), which lies close to the bottom of the graph, shows positioning error without noise. In both cases the network employed 200 neurons and the virtual neuron learning algorithm.

already exists and consider the way it adapts to the noisy inputs.

Letting λ be the learning rate, and considering a single component of the output, from the virtual neuron learning algorithm we can write (exactly) the component output at $\mathbf{x} + \delta\mathbf{x}_j$ for iteration $j+1$ in terms of the output for iteration j as

$$o_{j+1}(\mathbf{x} + \delta x_j) = o_j(\mathbf{x} + \delta\mathbf{x}_j) + \lambda(T - o_j(\mathbf{x} + \delta\mathbf{x})). \tag{3.77}$$

Assuming that the errors $\delta\mathbf{x}_j$ are small relative to the scale of the arm kinematics,[12] we can expand each component of the position error as a power series around the

[12]This is reasonable in our case since the rms vision distance error is on the order of a centimeter.

input vector $\mathbf{x}$ and write, keeping first-order terms

$$o_{j+1}(\mathbf{x}+\delta\mathbf{x}) \simeq (1-\lambda)(o_j(\mathbf{x}) + \nabla o_j(\mathbf{x}) \cdot \delta\mathbf{x_j}) + \lambda T. \tag{3.78}$$

But $o_{j+1}(\mathbf{x}+\delta\mathbf{x_j})$ can also be expanded around $\mathbf{x}$ and combined with the expression above yielding

$$o_{j+1}(\mathbf{x}) \simeq (1-\lambda)(o_j(\mathbf{x}) + (\nabla o_j(\mathbf{x}) - \nabla o_{j+1}(\mathbf{x})) \cdot \delta\mathbf{x_j} - \lambda(T - \nabla o_j(\mathbf{x}) \cdot \delta\mathbf{x_j}). \tag{3.79}$$

The difference of gradient term is a sum of terms over all neurons. Each neuron's term is proportional to the weight increment between iterations j and $j+1$ coupling the neuron to the output. From the virtual neuron learning algorithm, this is proportional to λ and the error as well as the neurons's output and the output of the virtual neuron at the neuron's center. Since these terms are small, they will be neglected. Setting $\nabla o_j(\mathbf{x}) \cdot \delta\mathbf{x_j} = \nu_j$, a noise term, we obtain the recursion

$$o_{j+1}(\mathbf{x}) = (1-\lambda)o_j(\mathbf{x}) + \lambda(T - \nu_j). \tag{3.80}$$

Starting at iteration zero, and noting that the target T is a constant, we obtain after summing the geometric series involving T:

$$o_j(\mathbf{x}) = (1-\lambda)^j o_0(\mathbf{x}) + (1-(1-\lambda)^j)T - \sum_{k=0}^{j-1}(1-\lambda)^{j-1-k}\nu_k). \tag{3.81}$$

If we can assume that the noise terms are independent and essentially identically distributed, the expected output becomes:

$$\langle o_j(\mathbf{x})\rangle = (1-\lambda)^j\langle o_0(\mathbf{x})\rangle + (1-(1-\lambda)^j)\langle T\rangle - \langle\nu_c\rangle\sum_{k=0}^{j-1}(1-\lambda)^k). \tag{3.82}$$

Summing the geometric series and letting j become large, this becomes:

$$\langle o_j(\mathbf{x})\rangle = T - \langle \nu_c \rangle, \tag{3.83}$$

where ν_c is the common expectation of the noise terms. We have used $0 < \lambda < 1$. In a particular instance, the sequence of noise terms will induce a net error called e_{net} below.

In this simplified analysis, network output converges to the target value plus the expected value of the noise term. The contribution of the initial state vanishes. A more thorough analysis would include the effects of adaptation at other sites on the network behavior at $\mathbf{x}$ as well as the average error over the space accessible to the robot.

The error for an output component due to reading a trained net with a noisy input is, using an expansion again:

$$T - o(\mathbf{x} + \delta\mathbf{x}) = T - (o(\mathbf{x}) + \nabla o(\mathbf{x}) \cdot \delta\mathbf{x}) \tag{3.84}$$

$$= T - (o(\mathbf{x}) + e_r) = T - (T - e_{net}) + e_r = e_{net} - e_r, \tag{3.85}$$

where e_{net} is the net error due to training by noisy inputs and e_r is the error due to the noisy reading vector. This is a reasonable result because training a net with a constant input vector offset and reading it with an input vector having the same offset will result in zero error.

In the hand-eye system we are considering, arm position commands are generated by calculating the nominal angles from visual input and adding a correction $\Delta\theta$ generated by a neural network $\mathcal{N}$ that uses the nominal angles θ_{nom} as input. Writing θ for θ_{nom} and letting $\mathcal{K}$ and $\mathcal{J}$ be the forward kinematics and forward Jacobian respectively, and $\mathcal{K}^{-1}$ and $\mathcal{J}^{-1}$ be the inverse kinematics and inverse Jacobian respectively,

the Cartesian position error $\mathbf{e}_p$ between an actual arm tip position $\mathbf{t}$ and the tip position calculated from $\mathbf{t}+\delta\mathbf{t}$, a noisy visual measurement of $\mathbf{t}$, is given approximately by:

$$\begin{aligned}
\mathbf{e}_p &= \mathbf{t} - \mathcal{K}(\theta + \Delta\theta) && (3.86)\\
&= \mathbf{t} - \mathcal{K}(\mathcal{K}^{-1}(\mathbf{t}+\delta\mathbf{t}) + \mathcal{N}(\mathcal{K}^{-1}(\mathbf{t}+\delta\mathbf{t}))) && (3.87)\\
&= \mathbf{t} - \mathcal{K}(\mathcal{K}^{-1}(\mathbf{t}) + \mathcal{J}^{-1}(\theta)\delta\mathbf{t} + \mathcal{N}(\mathcal{K}^{-1}(\mathbf{t}) + \mathcal{J}^{-1}(\theta)\delta\mathbf{t})). && (3.88)
\end{aligned}$$

Noting that $\mathcal{K}(\mathbf{x}+\delta\mathbf{x}) \simeq \mathcal{K}(\mathbf{x}) + \mathcal{J}(\theta)\delta\mathbf{x}$, this becomes:

$$\begin{aligned}
\mathbf{e}_p &= \mathbf{t} - \mathcal{K}(\mathcal{K}^{-1}(\mathbf{t})) - \mathcal{J}(\theta)(\mathcal{J}^{-1}(\theta)\delta\mathbf{t} + \mathcal{N}(\mathcal{K}^{-1}(\mathbf{t}) + \mathcal{J}^{-1}(\theta)\delta\mathbf{t})) && (3.89)\\
&= -\delta\mathbf{t} - \mathcal{J}(\theta)(\mathcal{N}(\mathcal{K}^{-1}(\mathbf{t}) + \mathcal{J}^{-1}(\theta)\delta\mathbf{t})). && (3.90)
\end{aligned}$$

In the case of perfect kinematics, the neural network's target correction vector is zero, and so, going to the vector form, writing $\mathbf{e}_r$ out explicitly, including the net error, and letting $\mathcal{I}$ be the unit matrix, we obtain:

$$\begin{aligned}
\mathbf{e}_p &= -\delta\mathbf{t} - \mathcal{J}(\theta)(\nabla(o(\mathbf{t}))\mathcal{J}^{-1}(\theta)\delta\mathbf{t} - \mathbf{e}_{net}) && (3.91)\\
&= -(\mathcal{I} + \mathcal{J}(\theta)(\nabla(o(\mathbf{t}))\mathcal{J}^{-1}(\theta))\delta\mathbf{t} + \mathcal{J}(\theta)\mathbf{e}_{net}. && (3.92)
\end{aligned}$$

Since the operators are all linear, the expected value of $\mathbf{e}_p$ is

$$\langle\mathbf{e}_p\rangle = -\langle\delta\mathbf{t}\rangle - (\mathcal{J}(\theta)(\nabla(o(\mathbf{t}))\mathcal{J}^{-1}(\theta))\langle\delta\mathbf{t}\rangle + \mathcal{J}(\theta)\langle(\mathbf{e}_{net})\rangle. \qquad (3.93)$$

If the system is trained and evaluated on vectors $\delta\mathbf{t}$ drawn from the same population, then

$$\langle\mathbf{e}_{net}\rangle = (\nabla(o(\mathbf{t}))\mathcal{J}^{-1}(\theta))\langle\delta\mathbf{t}\rangle, \qquad (3.94)$$

and the expected error is just

$$\langle \mathbf{e}_p \rangle = -\langle \delta \mathbf{t} \rangle. \tag{3.95}$$

To illustrate these ideas, the vision pointing angle γ was fixed at zero and the arm position was fixed at $x = 1, y = 0$, which corresponds to arm angles of $\alpha = -60°$ and $\beta = 120°$, and to a visual radius of two meters. The system was trained and tested using the visual noise parameters $\sigma(n_r) = 8.957 \times 10^{-3}$ and $\sigma(n_\epsilon) = 1.112 \times 10^{-3}$ defined in eqs. 3.24 and 3.28 and evaluated for the arm position. The root mean position error $\delta \mathbf{t}$ for this configuration is

$$\sqrt{\sigma^2(n_r) + \sigma^2(n_\epsilon)} = 9.0267 \times 10^{-3}. \tag{3.96}$$

Figure 3.29 shows the magnitude of noisy visual measurements (the length of $\delta \mathbf{t}$) over 100 trials, while figure 3.30 shows the resulting average error magnitude over 5000 trials (about 8 mm). The predicted root mean square error compares well with the observed visual error and the observed network learning error of 9 mm shown in the upper graph of figure 3.28. Systems that do not have perfect kinematics will be more complex in that there will be errors due to fundamental representational accuracy limitations in addition to noise errors.

3.5.6 Tuning, or Receptive Field Width

System performance depends critically upon the tuning, or receptive field, widths of the neurons. Figure 3.31 shows the position error correction performance for identical 200-neuron virtual neuron networks with different values of h, the tuning width half-maximum fraction. Noise was omitted to avoid masking differences in performance.

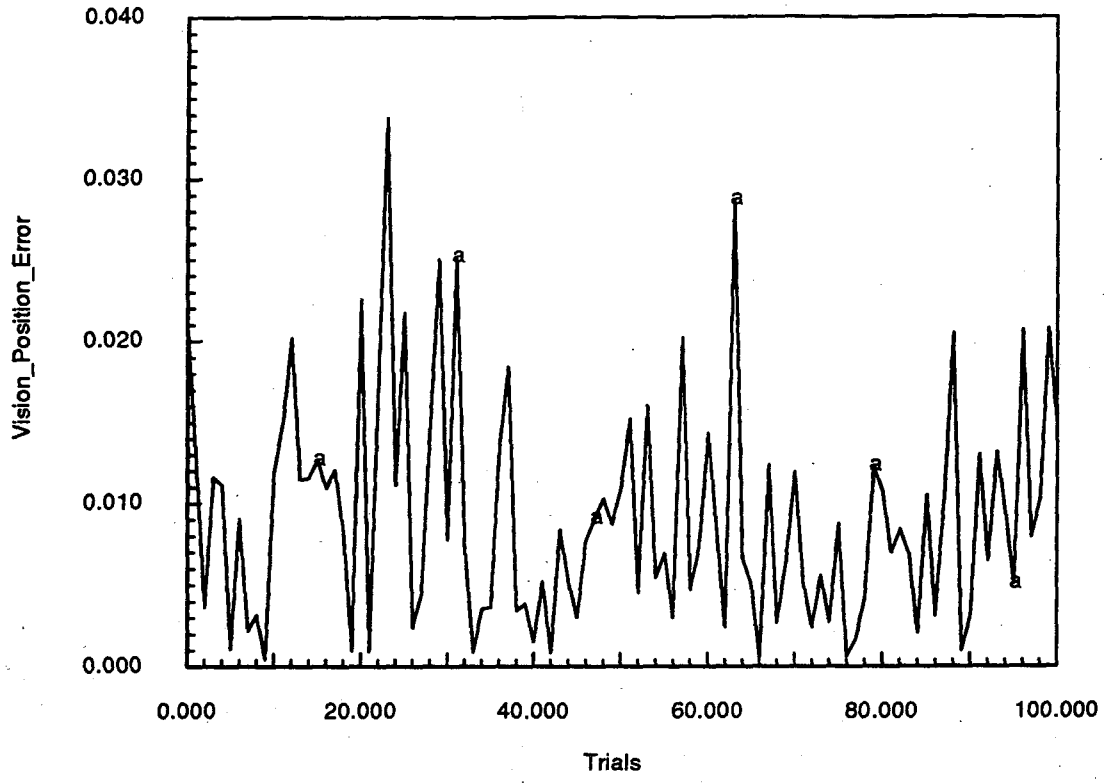

Figure 3.29: Visual measurement error magnitude over 100 trials.

Both narrow and broad receptive fields have poor performance. A sparse network with narrow receptive fields[13] can represent functions well at neuron locations, but generalizes (interpolates) poorly at locations between neurons [Mel (1989)]. In fact, very narrow receptive fields ($h \leq 0.2$) were unstable in simulations. Increasing the width up to a limit is intuitively appealing, and half-maximum fractions in the range of $0.4 \leq h \leq 2.0$ had good performance. Broad receptive fields ($h > 2.0$) performed poorly in simulations. This seems reasonable since using neurons with wide, overlapping, receptive fields would seem to reduce the specificity of the network. Interestingly, performance did not become increasingly bad with large values of h. In simulations, networks with $h = 100$ performed about as well as networks with $h = 10$, which may indicate that large receptive fields have useful properties. Baldi and Heili-

[13]Narrow with respect to the interneuron spacing and hence essentially non-overlapping.

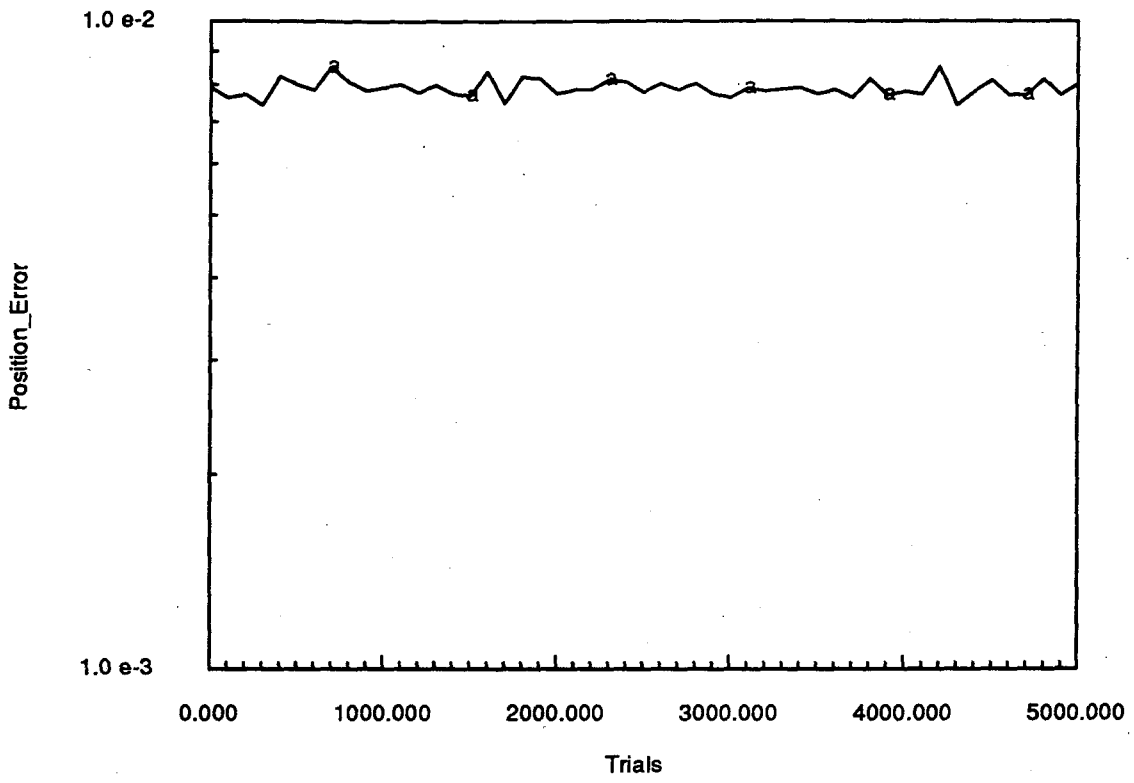

Figure 3.30: Average visual measurement error magnitude over 5000 trials. Averages were calculated every 100 trials, and were based on 1000 samples.

genberg [Baldi and Heiligenberg (1988)] have analyzed a network similar to the one employed here[14] in which increasing the receptive field width to large values with fixed neuron spacing increased resolution, accuracy, and gain for simple functional approximations.

3.5.7 Network Step and Frequency Responses (Tracking a Drifting Plant)

The ability of a network to maintain hand-eye calibration corrections in the presence of plant variations depends fundamentally upon its step and fre-

[14]Neurons are located on a lattice rather than being randomly allocated.

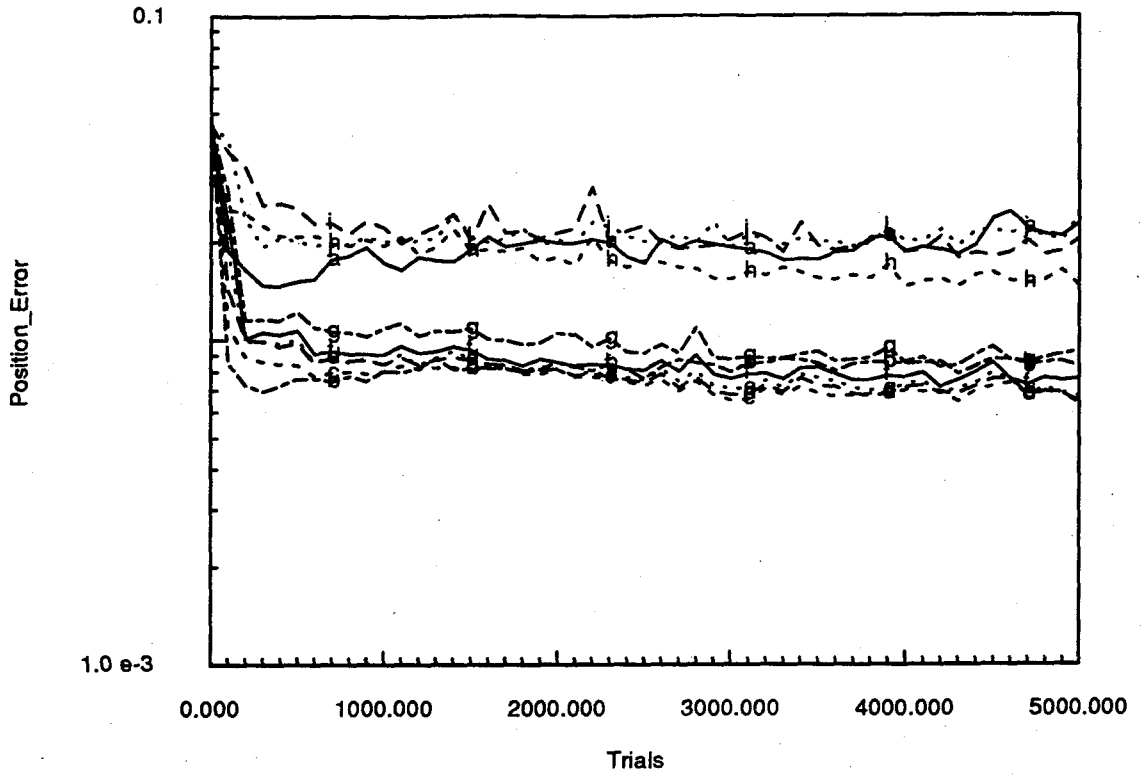

Figure 3.31: Hand-eye system positioning error correction performance for different values of the neuron tuning width parameter, h. Values are .4, .6, .8, 1.0, 1.2, 1.5, 2.0, 4.0, 10.0, and 100.0 in plots (a)–(j) respectively. Visual noise was not present.

quency responses. While a neural network is not a linear system, it is shown below that the ability of the networks constructed above to track plant variations exhibits surprisingly linear behavior in terms of phase lag and bode plot rolloff [Franklin, Powell and Emami-Naeini (1986), Dorf (1983), Distefano, Stubberud and Williams (1967)]. This means that once the step response and steady-state error are obtained experimentally, network tracking behavior can be predicted. Visual noise is present in all the step and frequency-response results presented here. Results for the noise-free case are nearly identical except for greater accuracy.

Referring to figure 3.1, drifts due to mechanical and thermal effects are simulated by systematically varying the lengths, a and b, of the arm links, the components of

the vector $\mathbf{v}$ that locates the vision pointing system, the arm and vision pointing system encoder offsets O_α, O_β, and O_γ, and the vision range scale factor. Modifying the encoder offsets simulates link bending due to both differential heating and to yield resulting from collisions or slippage in the manipulator power transmission mechanisms.

Step Response

A maximum offset of two degrees, a value typical of those observed in practice, was assumed for yield due to collisions and transmission slippage. Yield was simulated by training a network for 5000 trials using a particular manipulator kinematic configuration, making a step change in a single encoder offset (taken here to be O_α, and running another 5000 training trials. The network had 200 neurons and employed the virtual neuron learning algorithm. Figure 3.32 shows the system behavior during the initial training trials while figure 3.33 shows the system position error step response. During initial training the system began with an average position error of 4.8 cm and stabilized at an average position error of 1.3 cm, allocating 200 neurons in the process. During its step response the position error decayed from an initial value of 5.4 cm to 1.3 cm. From figure 3.33 the time constant is seen to be about 120 trials. In figure 3.34 the system step response is displayed along with an ideal step response having the same amplitude and offset and a time constant of 120 trials for comparison.

The network error-correction behavior is closely approximated by that of a first-order linear system with a decay constant of 120 trials. The trials involved randomly-selected positions uniformly distributed over the accessible input space. If the robot operates only in a small number of localized regions, as around parts acquisition and assembly sites, the time constant will be much smaller because less of the input space must be corrected.

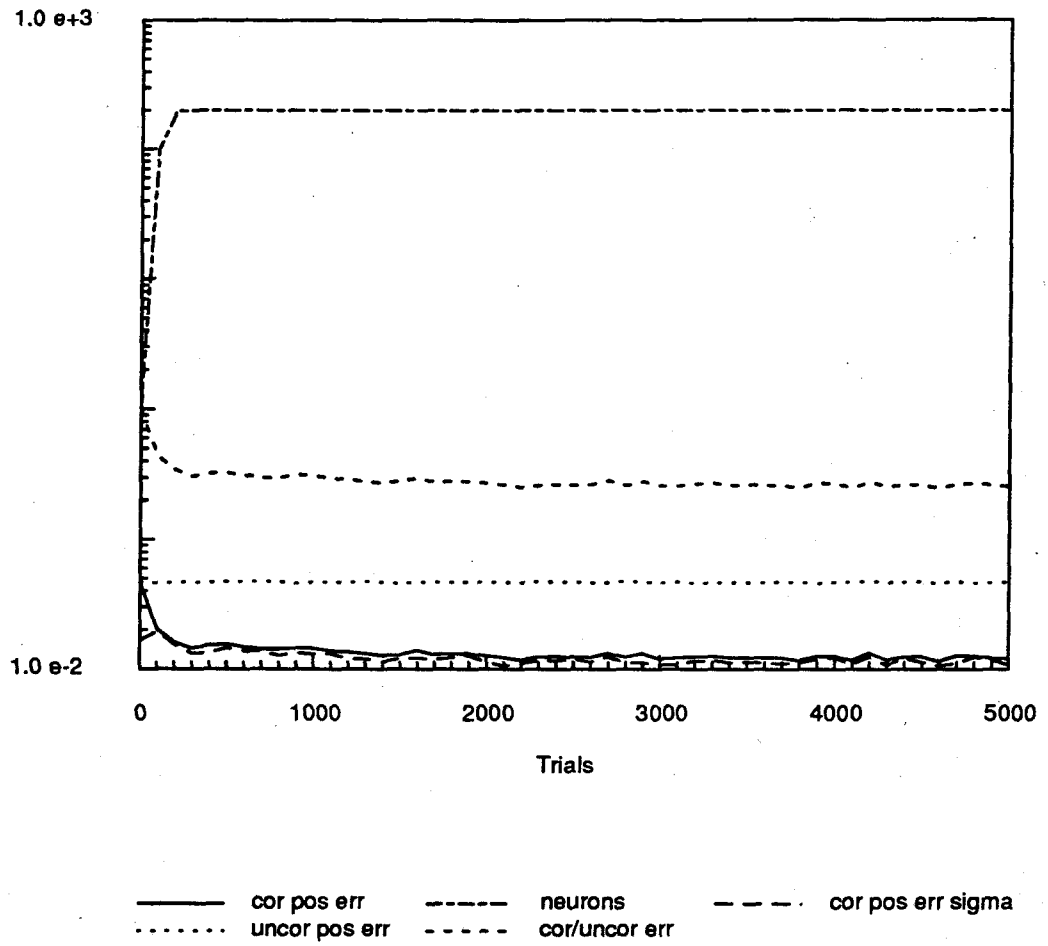

Figure 3.32: System network behavior during initial training trials. The network had 200 neurons and used the virtual neuron learning algorithm. Visual noise was present. Plots show corrected position error magnitude, uncorrected position error magnitude, neuron count, the ratio of the corrected to the uncorrected position error magnitudes, and the corrected position error magnitude standard deviation. The verical axis is the number of neurons in the case of the neuron count plot; it is meters for the other plots.

Response to Periodic Drift

System position error frequency response was investigated in the context of periodic differential heating, which simulates many thermal drift problems. [15]

[15]For example, daily temperature changes in factories and the differential heating experienced by orbiting spacecraft can cause thermal drifts that are roughly periodic.

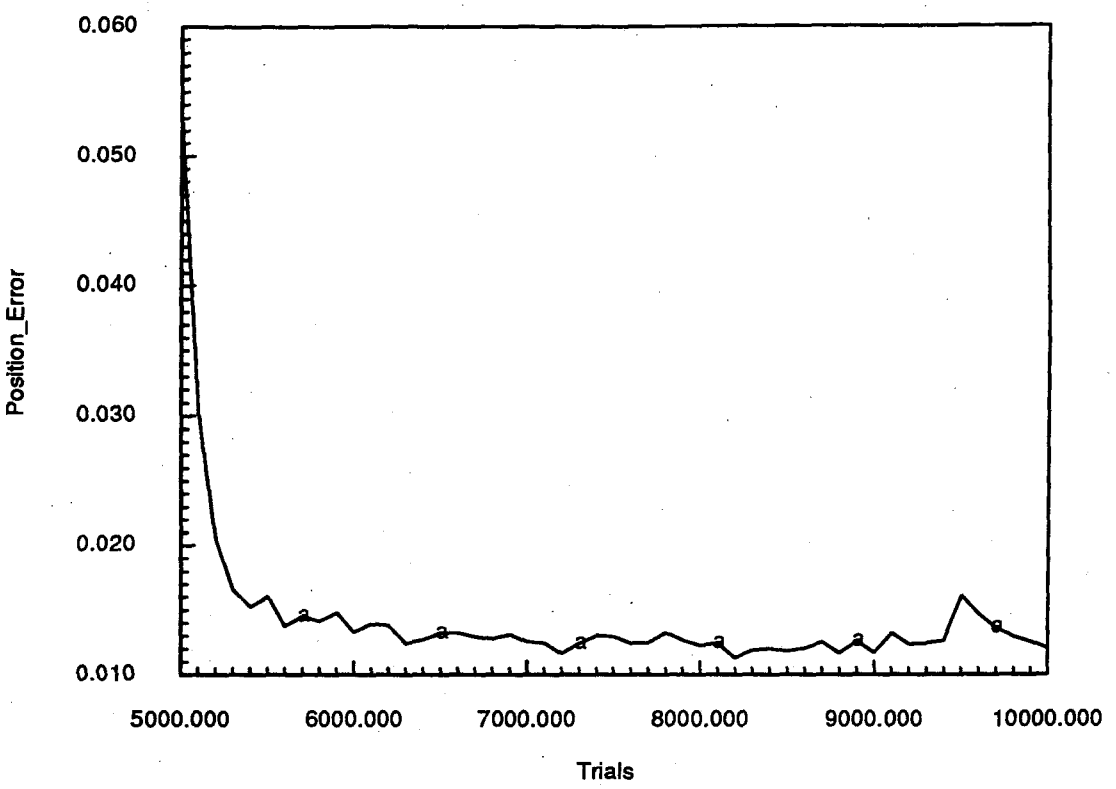

Figure 3.33: System positioning error step response for a 200-neuron virtual neuron network with visual noise. The system has a time constant of about 120 trials.

In differential heating, the arm links are assumed to bend so as to move the elbow and tip in the direction of increasing α and β respectively (counterclockwise) and the strut supporting the camera pointing system is assumed to bend so the camera pointing base rotates clockwise slightly around the arm base frame. As the camera pointing base rotates, it carries the vision encoder with it, inducing a change in the effective vision encoder offset. Bending due to differential heating increases the effective arm encoder offsets and decreases the effective vision encoder offset, and is intended to simulate radiant heating of the arm and vision system from the same direction. A maximum angular offset of one degree is assumed, including the orientation of the vector $\mathbf{v}$. The vision scale factor is assumed to change as well. The maximum vision range scale factor error is .02, which induces a two-centimeter error at a radius of one meter.

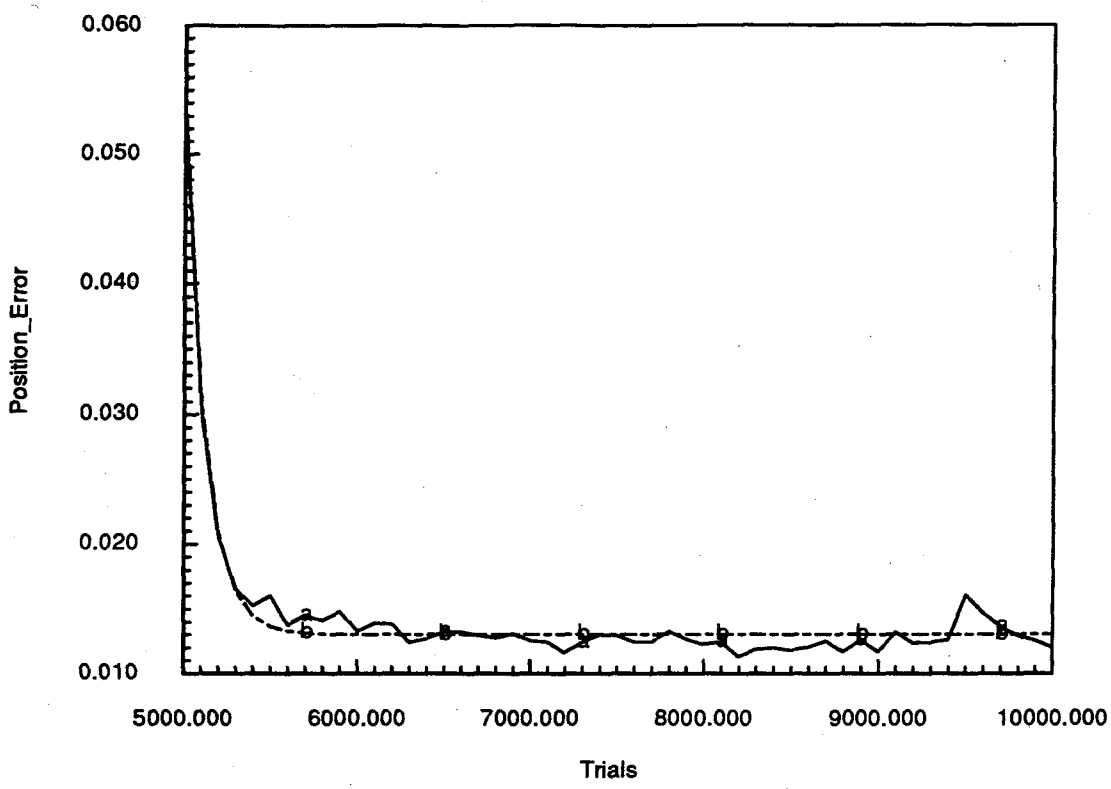

Figure 3.34: System position error step response and ideal step response. Plot a is the step response of a 200-neuron virtual neuron network with visual noise. Plot b is an ideal first-order system step response with a time constant of 120 trials.

In operation, the changeable parameters are varied as a function of the number of trials by adding periodically-varying increments to their actual values. For lengths, a multiplicative expansion factor f is calculated so that:

$$l_{exp} = f l_{ref}, \tag{3.97}$$

where

$$f \stackrel{\text{def}}{=} 1 + e \sin(2\pi \frac{n}{N}), \tag{3.98}$$

and l stands for a, b, v_x, or v_y. The subscripts exp and ref indicate the respective expanded and reference lengths, e is the expansion amplitude, n is the trial number,

and N is the number of trials in a period. The expansion amplitude, e, is the product of the thermal expansion coefficient of aluminum, 26×10^{-6}/ deg C, and the temperature amplitude, which is assumed to be 50 deg C. Similarly, the encoder offsets, vision range scale factor, and the location of the $y-$component of the camera pointing system base are given by

$$O_{exp} = O_{ref} + \tau \sin(2\pi \frac{n}{N}), \tag{3.99}$$

where O represents any of the encoder offsets, the vision range scale factor, or the $y-$ component of the camera pointing system base, and the amplitude, τ, is one degree for arm encoder offsets, minus one degree for the vision encoder offset, 0.02 for the vision range scale factor, and, for the camera pointing system base, is the linear displacement (1.745 cm) corresponding to the rotation of the base location by one degree clockwise around the arm base frame.

While the basic excitation function is a pure sinusoid, the periodic change in the uncorrected positioning error is not, because of the complex interplay between kinematic parameters and accuracy. It is approximately sinusoidal, however, as can be seen in the line labeled "uncor pos error" in figure 3.35.

A Simple Frequency Response Model

Since the adaptation algorithms used here make output corrections that are proportional to output error, it is natural to consider a first-order model for the system response. We will construct a model that predicts the (nonnegative) position error by taking the absolute values of the output of a model that has a signed response.

The neural network will have an inherent steady-state positioning error $\epsilon_0 i \geq 0$ due to its intrinsic structure. If we consider o to be the signed scalar output of the

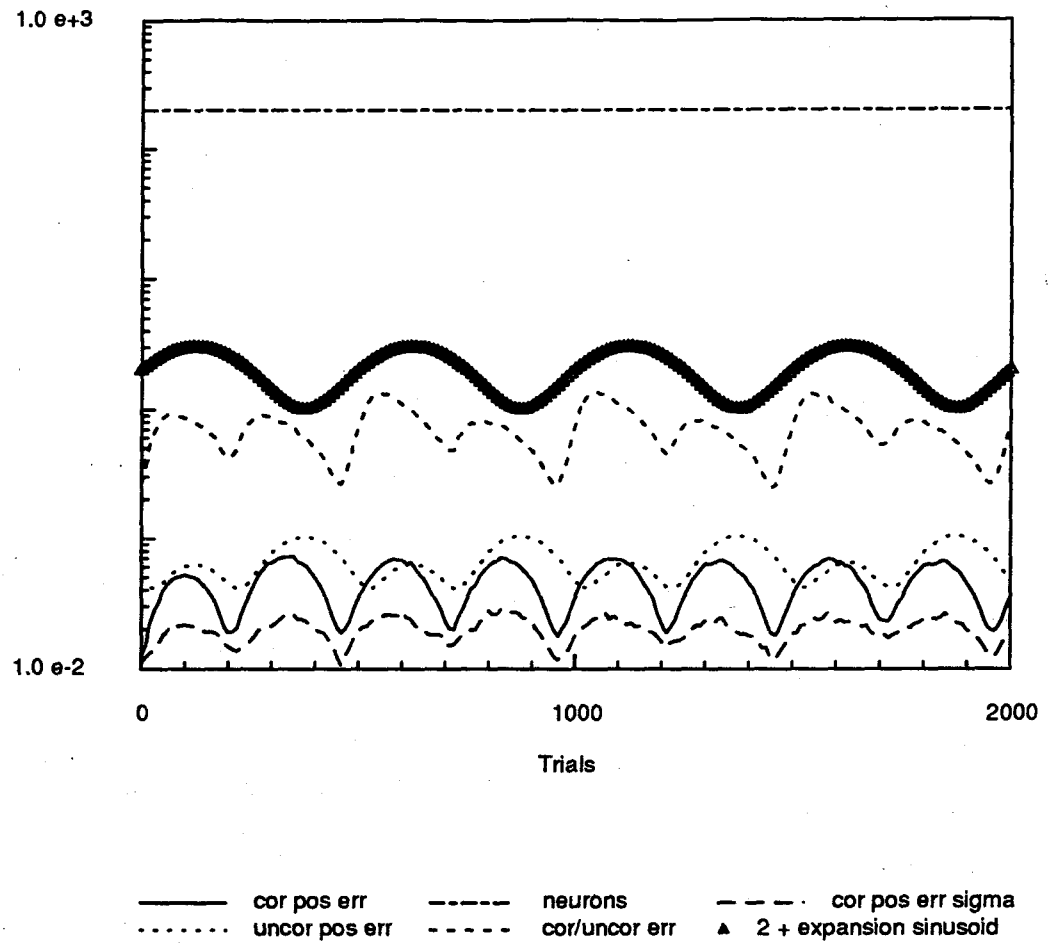

Figure 3.35: System response as a function of the number of trials to a periodic thermal distortion with a period of 500 trials. The error induced by the thermal excitation function is the line labeled "uncor pos error." The thermal excitation sinusoid is indicated with Δ's. A constant offset of 2 has been added to it to prevent logarithmic plotting problems. The verical axis is the number of neurons in the case of the neuron count plot and is just real numbers for the sinusoid; it is meters for the other plots.

network, and output corrections are taken to be proportional to the difference between the target scalar output T and the network output o, then we have, approximately,

$$\frac{do}{dt} = k(T - o - \epsilon_0), \tag{3.100}$$

where k is a constant and t represents time. If $T = A\sin(\omega t)$, then the steady-state

solution for the network output is

$$o(t) = A(\frac{k^2\sin(\omega t) - k\omega\cos(\omega t) + k\omega e^{-kt}}{k^2+\omega^2}) + \epsilon_0(e^{-kt}-1) + Ce^{-kt}, \qquad (3.101)$$

with C a constant. The network error $\epsilon = T - o$ is, then,

$$\begin{aligned}\epsilon &= A\sin(\omega t) - A(\frac{k^2\sin(\omega t) - k\omega\cos(\omega t) + k\omega e^{-kt}}{k^2+\omega^2}) - \\ &\quad \epsilon_0(e^{-kt}-1) - Ce^{-kt}. \qquad (3.102)\end{aligned}$$

If the network has been trained to a steady-state error with a constant plant so that $\epsilon = \epsilon_0$ at $t = 0$, then $C = -\epsilon_0$, and, after the transients die out,

$$\epsilon = \frac{A}{\omega^2+k^2}(\omega^2\sin(\omega t) + \omega k\cos(\omega t)) + \epsilon_0. \qquad (3.103)$$

Placing this in the form $\epsilon = \rho(\omega)\sin(\omega t + \phi)$, with $\rho(\omega)$ the amplitude function and ϕ the phase angle, yields:

$$\rho(\omega) = A\omega/\sqrt{\omega^2+k^2} \qquad (3.104)$$

$$\tan(\phi) = k/\omega, \qquad (3.105)$$

so that finally, taking the absolute value of the temporal term,

$$\epsilon = \rho(\omega)|\sin(\omega t + \phi)| + \epsilon_0. \qquad (3.106)$$

For high frequencies ($\omega \gg k$) these results predict $\phi \simeq 0$ and $\rho \simeq 1$. That is, the network cannot correct positioning errors for the constantly changing plant. For low frequencies ($\omega \ll k$) these results predict a phase *lead* of $\phi \simeq \pi/2$ along with $\rho \simeq 0$.

System Frequency Response

The 200-neuron system's response to excitation periods of 100, 200, 500, 1000, 2500, and 5000 trials is shown in figure 3.36. Periods of 10000 and 20000 trials were run as well, but the responses were too low for meaningful estimation. Bode plots of the amplitude response and phase shift, which were constructed from measurements of the graphical output, are plotted in figures 3.37 and 3.38 respectively. To constuct the plots, the amplitude of the network error in each period was taken as the difference between the average maximum and average minimum errors, and the error input amplitude was taken as 5.3 cm, the difference between the maximum and initial uncorrected position error. Phase shifts were estimated from the positions of the input and response peaks. The corner frequency [Distefano, Stubberud and Williams (1967)] is $\omega_{corner} = k = 1/\tau$, where tau is the step response time constant. This implies that $N_{corner} = 2\pi\tau$, where N_{corner} is the period corresponding to ω_{corner}.

For periods less than N_{corner} the bode amplitude approaches 0 db, and it rolls off at 20 db per decade asymptotically for periods above N_{corner}. The phase lead asymptotically approaches zero degrees for periods less than N_{corner}, and becomes larger for periods greater than N_{corner}. Corner periods estimated from the step response and bode plots are quite consistent, lying in the range from 600-800 trials.

The system's step responses to the (fixed) extreme values of position errors are shown in figure 3.39. These plots were generated by training the network for 5000 trials as before, fixing the thermal excitation driver phase at 180 degrees, 21.6 degrees, and 90 degrees, locations of the position error extrema, and letting the network evolve to reduce the error. These values are meaningful because they establish the minimum error the system can achieve for the given kinematic configurations. For very long periods these errors will dominate the system error response in the appropriate phase region because they represent intrinsic network accuracy limits for the configuration.

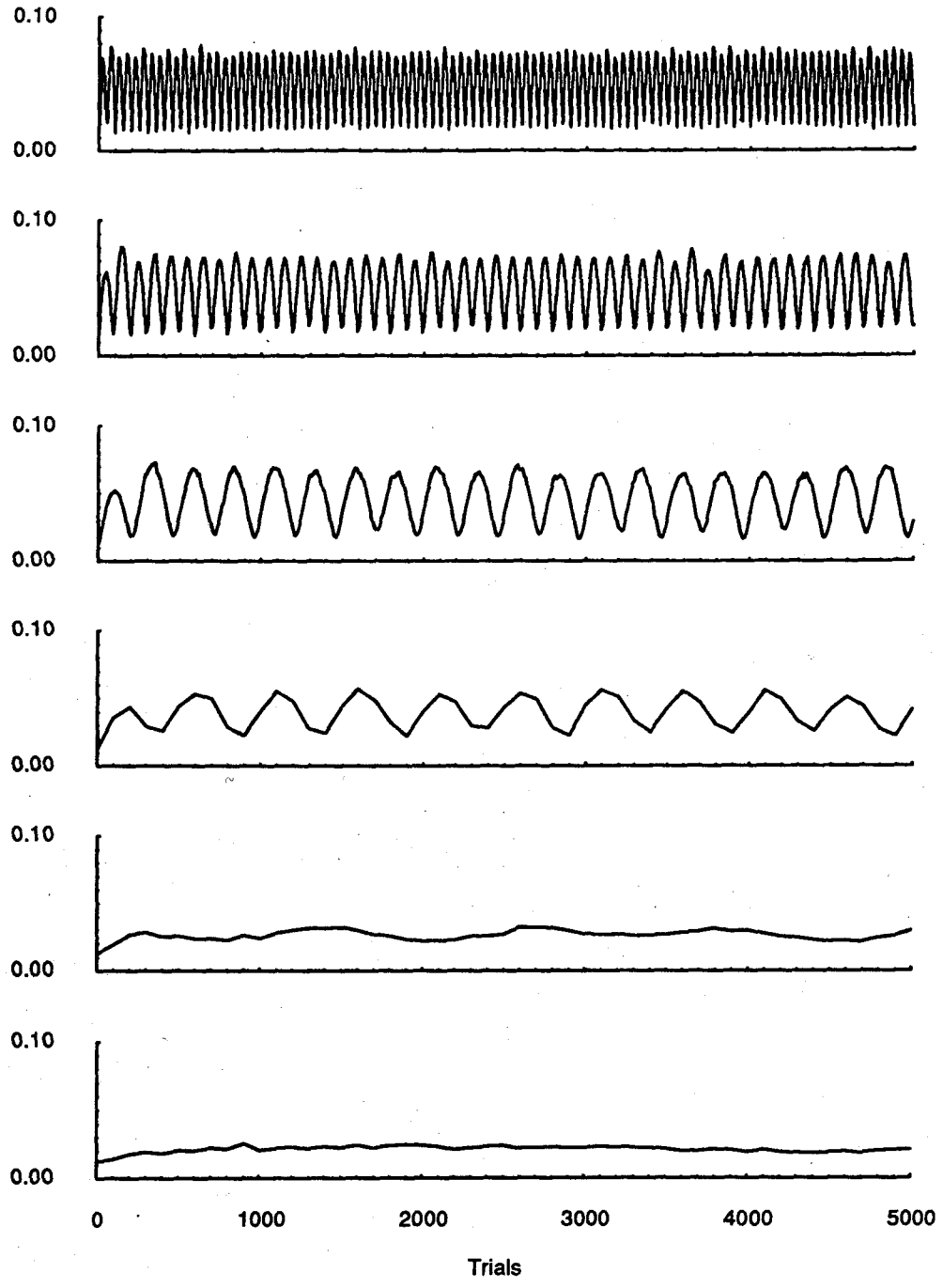

Figure 3.36: Hand-eye system positioning error response for sinusoidal expansion inputs with periods of 100, 200, 500, 1000, 2500, and 5000 trials, respectively, beginning at the top. Vertical axis units are meters.

These plots show again that the system error response time constant is about 120 trials.

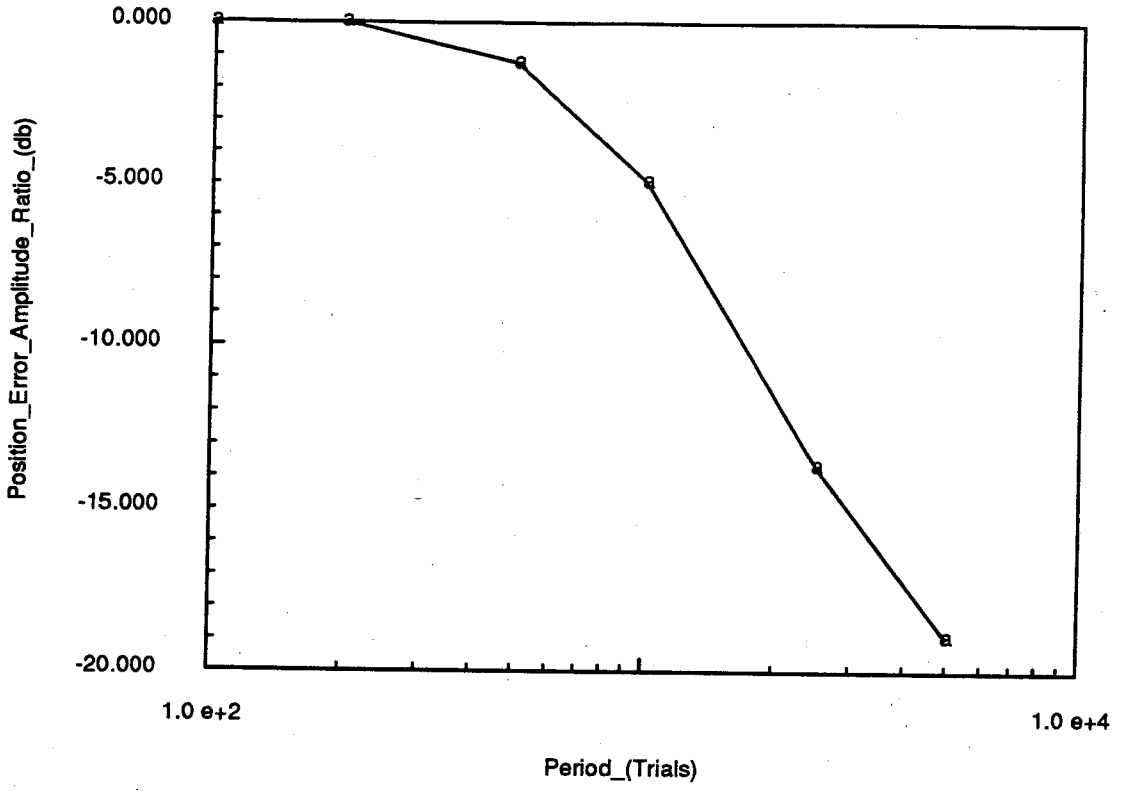

Figure 3.37: Bode amplitude plot of the system position error for periods from 100 to 5000 trials. The corner period estimated from the plot is about 600 trials.

Adaptation Algorithn Effects on the System Time Constant

The 200-neuron virtual neuron network used in the frequency and step response investigations was chosen because of the simplified nature of the learning algorithm and its good learning and accuracy performance. Increasing the number of neurons in such a network (with fixed input dimension) improves positioning accuracy performance, but degrades the network step response. This is due to the fact that the width of the virtual neuron used to allocate weight corrections among the active neurons is determined by its nearest neighbor. Increasing the number of neurons decreases the distance between them, decreasing the range of influence of the virtual neuron and hence of corrections based upon it. This is a complex issue, of course, because for complex mappings (ones with a great deal of structure), restricting the range of cor-

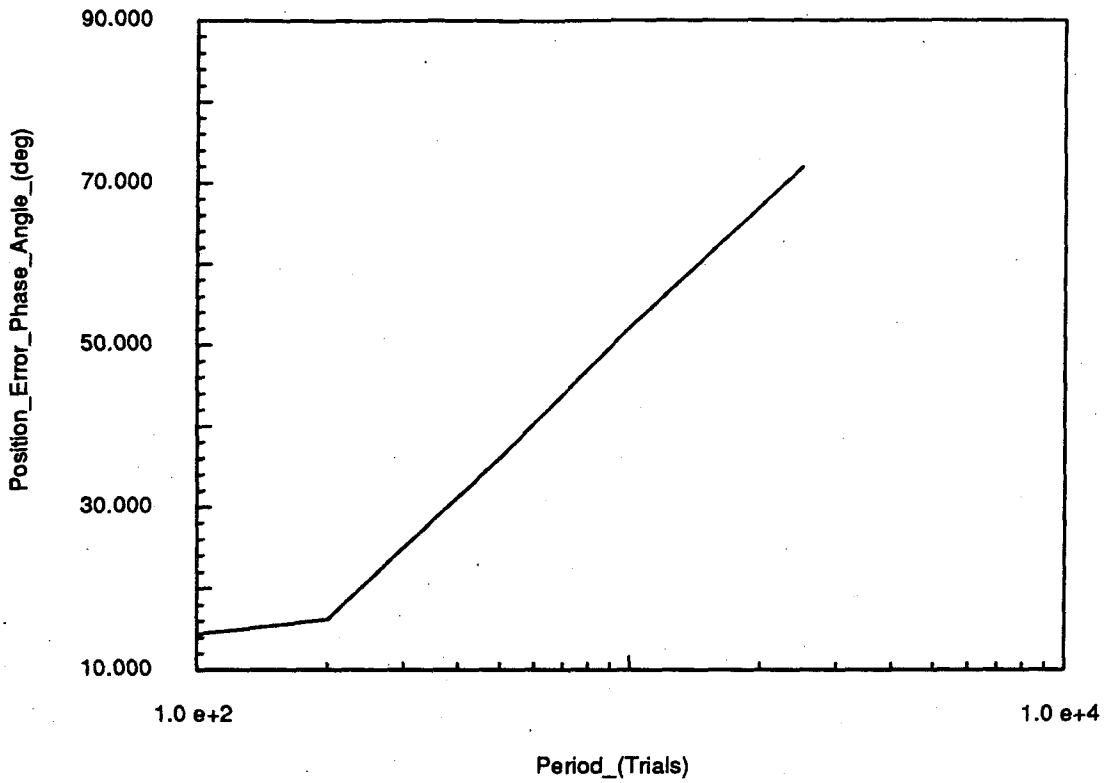

Figure 3.38: Bode phase angle plot of the system position error for periods from 100 to 5000 trials. The system error has a phase lead. The corner period estimated from the plot is about 700 trials.

rections is important so that the accuracy at distant points is not appreciably affected by local modifications.

In networks employing the gradient descent adaptation procedure, the correction range is not limited by nearest neighbors since virtual neurons are not employed. Figure 3.40 shows the step responses of the system with virtual neuron networks containing from 200 through 3000 neurons. The network time constant increases monotonically with the number of neurons. As pointed out in sections 3.5.2 and 3.5.5, the extra neurons do not improve accuracy because of noise. Figure 3.41, which plots the step responses of the system with gradient descent networks containing 200, 1000, and 3000 neurons, clearly shows that the time constant is minimally affected as the number of neurons increases. For both figures, networks were trained

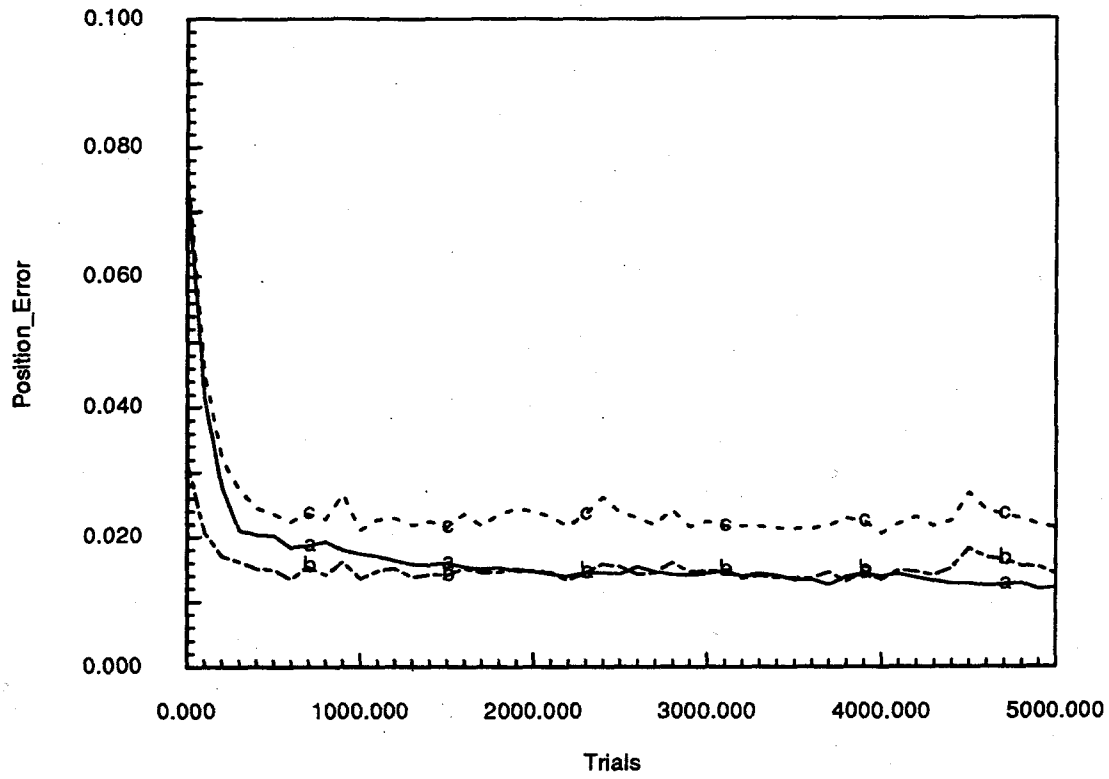

Figure 3.39: Hand-eye system error response to step inputs corresponding to distortions caused by fixing the thermal excitation driver phase at (a) 180 degrees, (b) 21.6 degrees, and (c) 90 degrees.

for 5000 trials using a constant plant, were subjected to the same step input used in section 3.5.7 for mechanical drift, and were trained for another 5000 trials to allow the networks to adapt. We can conclude that network error rejection characteristics for high-frequency (short-period) disturbances, at least for the smooth mappings considered in kinematic error compensation, can be predicted using linear first-order models if the system step response can be empirically determined. Error rejection characteristics for low- frequency (long-period) disturbances will be dominated by the intrinsic network accuracies associated with the kinematics. We also conclude that good error response may require employing gradient descent adaptation rather than virtual neuron adaptation.

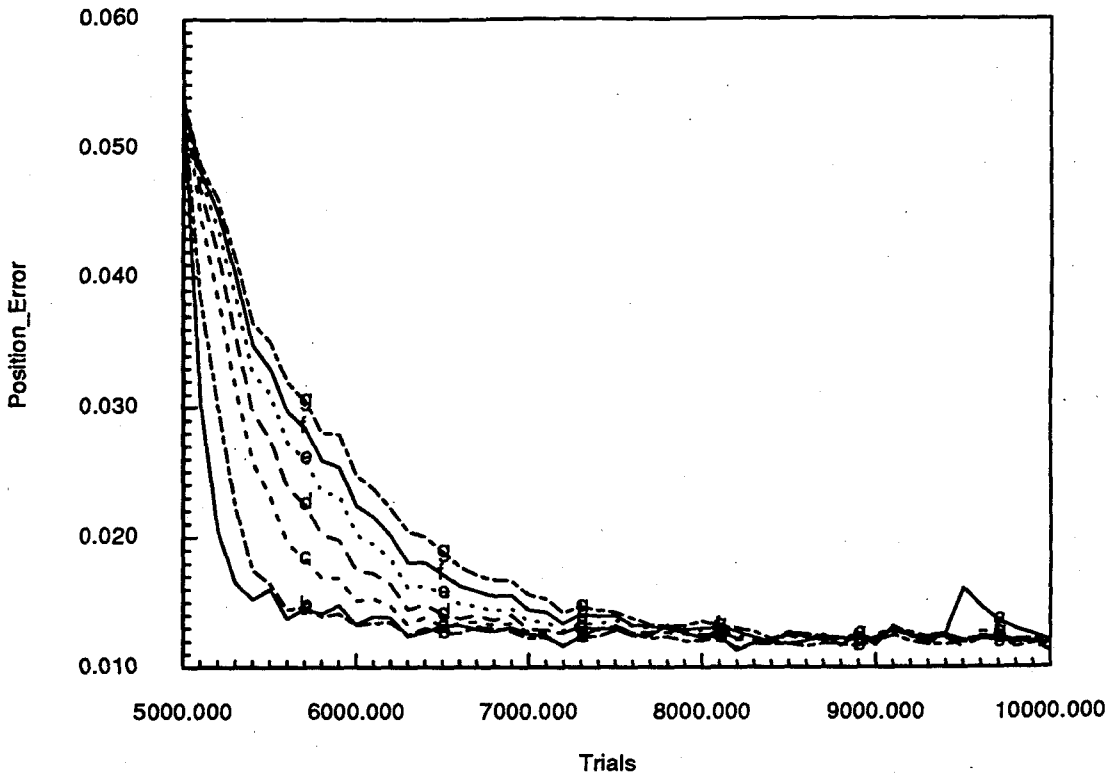

Figure 3.40: System position error step response using virtual neuron networks with (a) 200, (b) 500, (c) 1000, (d) 1500, (e) 2000, (f) 2500, (g) 3000 neurons in the presence of visual noise. The input space dimensionality for each network. is the same. The output dimensionality for each network is identical as well. The time constant increases monotonically with the number of neurons.

3.5.8 Effects of the Various Error Sources on System Performance

This section describes the effects of various inaccuracies, or modeling errors, on system performance.

Effects of Nonlinear Perturbations

Noise-free system error-correction performance with and without the nonlinear perturbations is shown in figure 3.42. The two upper plots are for a virtual neuron

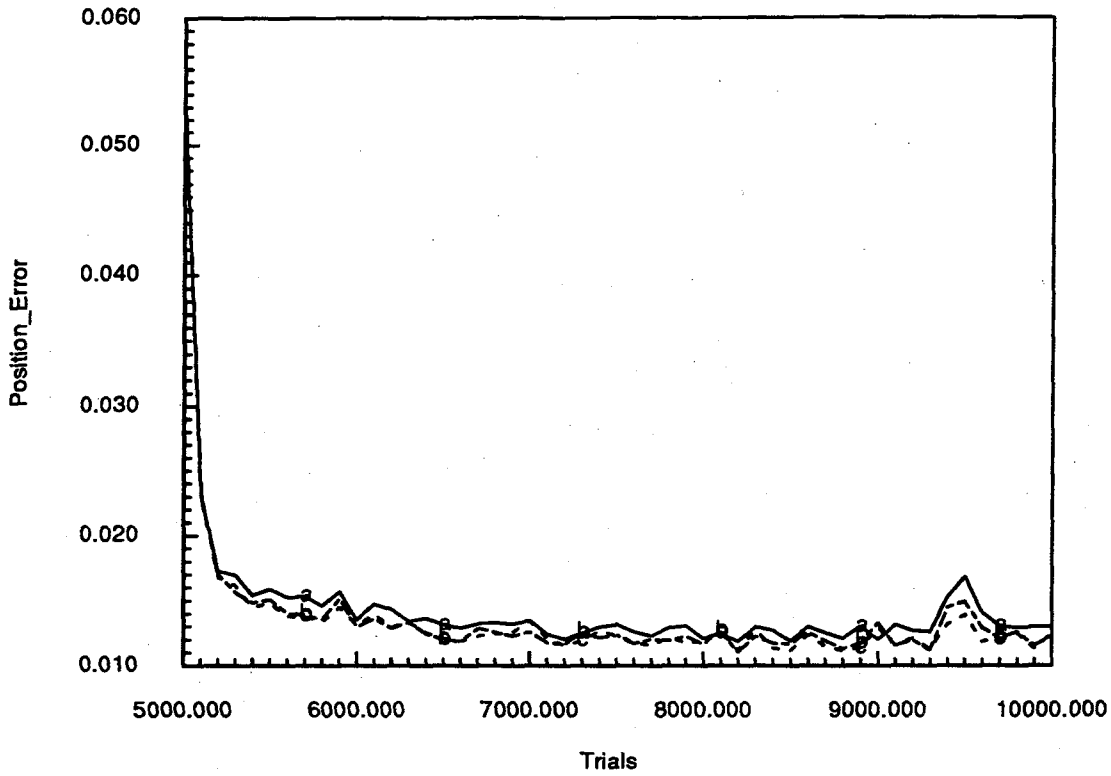

Figure 3.41: System position error step response using gradient descent networks with (a) 200, (b) 1000, (c) 3000 neurons in the presence of visual noise. The input space dimensionality for each network. is the same. The output dimensionality for each network is identical as well. The time constant is essentially unaffected by the number of neurons.

network with 200 neurons; the bottom plots are for the same virtual neuron network except that the number of neurons was not limited. Removing perturbations enhances performance by a few mm, and reduces the number of neurons allocated by the unconstrained networks from 3771 to 3229. The nonlinear perturbations do not appreciably affect the uncorrected error; they increase the complexity of the hand-eye map.

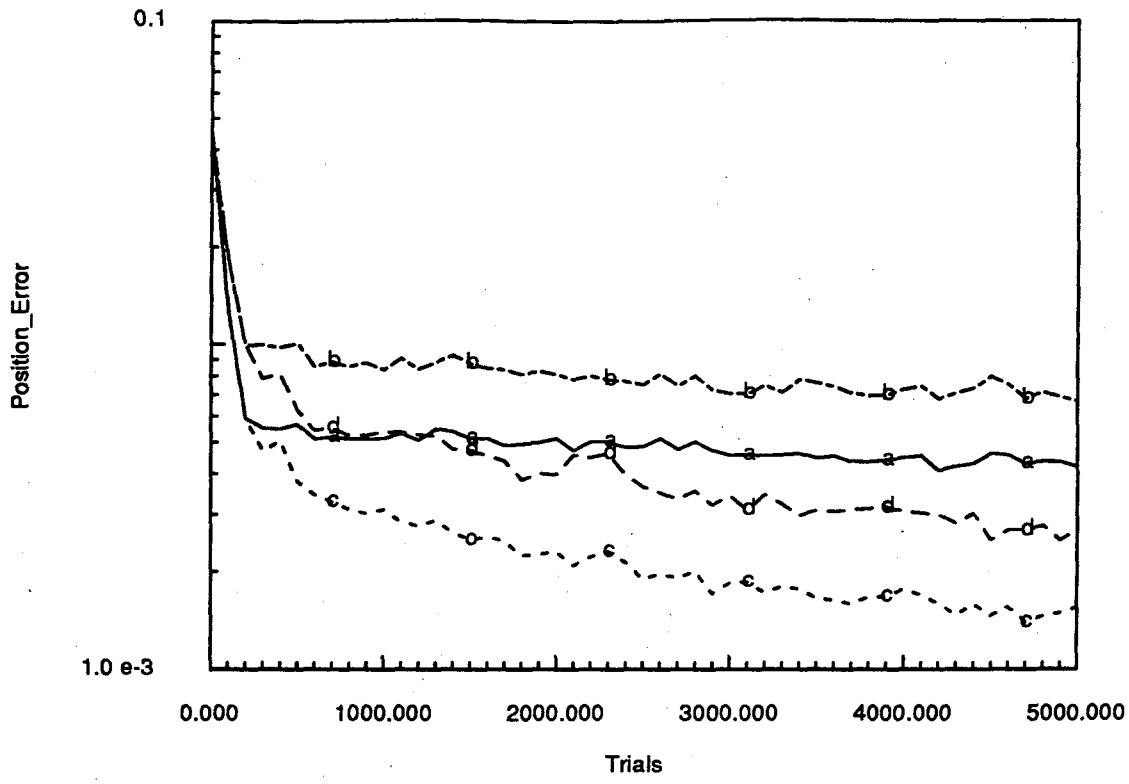

Figure 3.42: Noise-free error-correction performance with and without nonlinear perturbations. Plots (a) and (b) are for a 200-neuron network; Plots (c) and (d) are for the same network with no constraints on the number of neurons. Plots (a) and (c) are without perturbations.

Visual Range Scale Factor Errors

It is not necessary to employ a pre-calibrated vision system in order to obtain adequate positioning performance. The system can accommodate visual scale factor errors in the same way that it accommodates errors of other types. Figure 3.43 shows noise-free system performance with visual scale factors of 0.5, 0.9, and, for reference, 1.0. Plots (a) and (c) are for a scale factor of 0.5, which implies a position error of one meter at a radius of two meters due to the vision scale factor alone. The virtual neuron network of plot (a) was limited to 200 neurons, while that of plot (b) allocated a total of 4999 in 5000 trials. Plots (b) and (d) are for a scale factor of 0.9, which implies a visual position error of 20 cm at a radius of two meters due to the vision scale

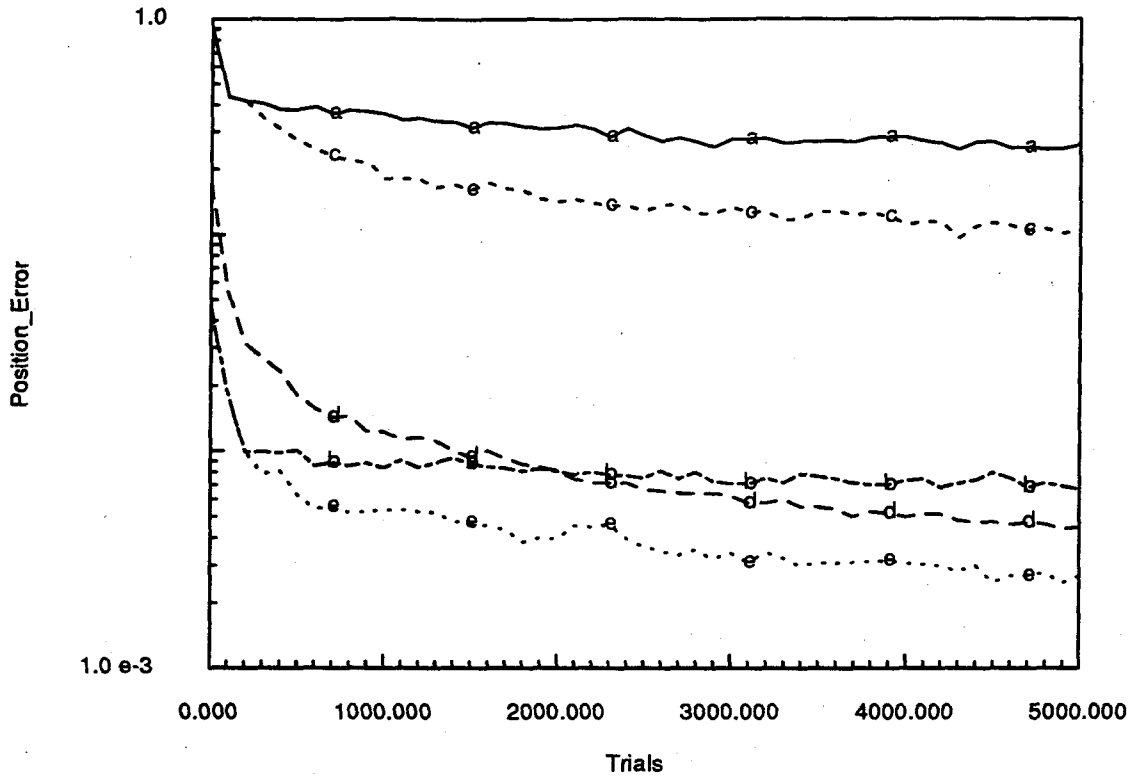

Figure 3.43: Noise-free positioning performance with visual scale factor errors. plots (a), (b) and (e) are for a 200-neuron network; the others are for a network with no constraints on the number of neurons. Plots (a) and (c) have a visual scale factor of 0.5, while plots (b) and (d) have a scale factor of 0.9. Plots (e) and (f) have a scale factor of 1.0

factor. Plot (b) is for a network limited to 200 neurons, while the network of plot (d) allocated a total of 4805 neurons in 5000 trials. It required 1878 neurons to equal the accuracy of the 200 neuron network with scale factor of 1.0 shown in plot (e). Plot (f) is for an unconstrained network with scale factor of 1.0. It allocated 3771 neurons over 5000 trials. The plant was the same for all plots, and included nonlinear perturbations. The scale factor errors employed in these simulations are significant, and yet the system is able to achieve reasonable performance. In the presence of more moderate errors, neural network performance should be entirely adequate.

3.5.9 Additional System Tests

This section describes additional hand-eye calibration system tests. The tests include the effects of foveation, or aligning the optical axis with the visually-sensed target, hand-eye calibration without using kinematic information, hand-eye calibration with large kinematic errors, and system performance with non-gaussian neurons.

Foveation

As mentioned in section 3.4.3, the vision pointing system used in the present research does not align the vision system's optical axis with visual stimuli (does not foveate on the stimuli) in order to calculate arm angle corrections; the vision system is merely pointed in their general direction. Because of visual eccentricity perturbations (astigmatism), this approach increases the dimensionality and complexity of the hand-eye correction map since it is necessary to map eccentricity values into arm angle corrections and to correct for errors induced by eccentricity perturbations.

If a robot system were to point directly at visual stimuli rather than in their general direction, it should be possible to reduce the correction map's dimensionality and complexity since eccentricity would no longer be a required input and eccentricity perturbations would no longer be sources of error. Pointing directly at stimuli, then, should manifest itself in improved system accuracy and reduced neuron count.

Figure 3.44 shows system performance with and without foveation. The upper plots include visual noise; the lower plots do not. In all cases the plants were identical except for foveation, and employed a virtual neuron network with an unconstrained number of neurons. Plots (a) and (c) employ foveation; the networks they employ do not provide an eccentricity input. Plots (b) and (d) do not employ foveation and do

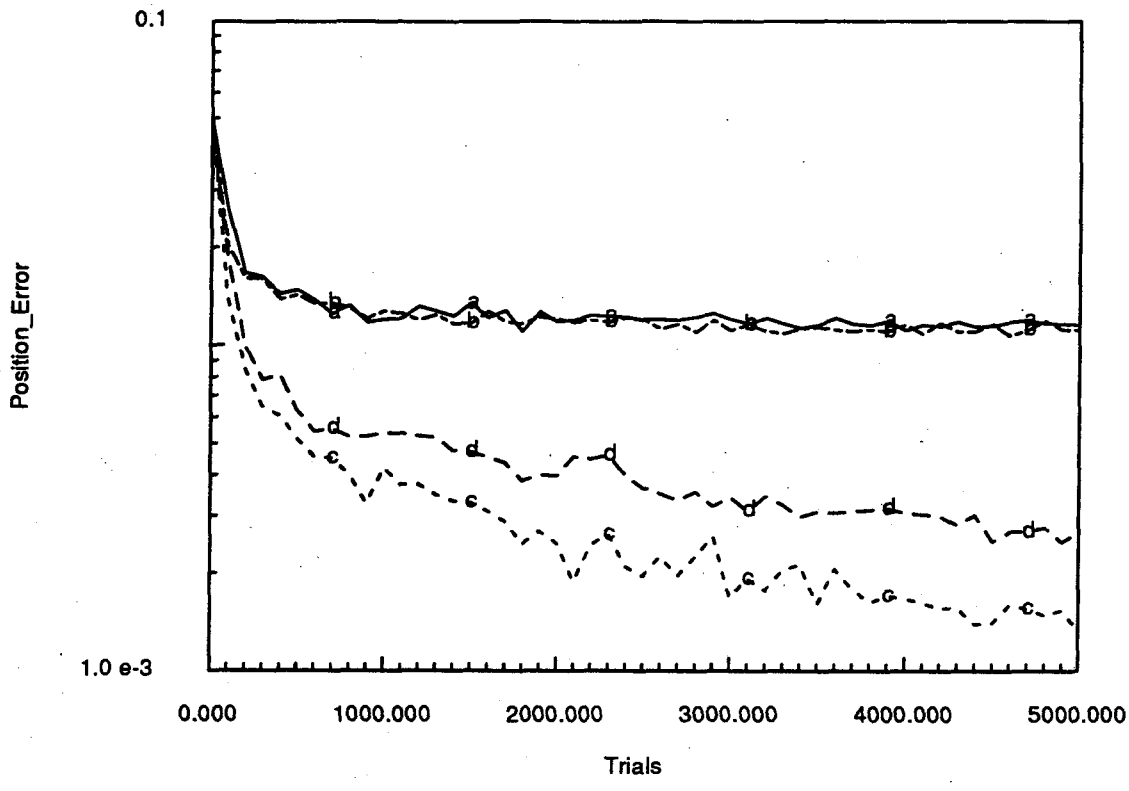

Figure 3.44: System positioning error performance with and without foveation. The upper plots include visual noise; the lower plots do not. In all cases the plants were identical except for foveation, and employed a virtual neuron network with an unconstrained number of neurons. Plots (a) and (c) employ foveation; the networks they employ do not provide an eccentricity input. Plots (b) and (d) do not employ foveation and do provide eccentricity inputs.

provide eccentricity inputs. In the noisy case performance was essentially identical; both plants had the same positioning accuracy and both networks allocated over 4500 neurons. In the noise-free case foveation provided somewhat improved accuracy (slightly over 1 mm), and reduced the total neuron count from 3771 to 2886 over 5000 trials.

Foveation improves noise-free system performance at the expense of a fast vision pointing system. In the noisy case, it provides no accuracy performance improvement at all, but does reduce the number of neurons.

Learning Hand-Eye Calibration Without Kinematics

To compare the performance of learning a hand-eye correction map with learning the entire hand-eye map, the system was run with and without nominal kinematics. Without nominal kinematics, the system has no *a priori* algorithm for estimating the joint angles that correspond to visual stimuli, and must learn to associate visual inputs with the appropriate arm angles. This is a much more difficult problem than learning corrections, as figure 3.45 illustrates.

The figure compares a virtual neuron network learning the entire map with a virtual neuron network learning hand-eye map corrections for the same plant. Comparisons are made with and without visual noise. Networks, which did not constrain the number of neurons, were identical except for their inputs. Network inputs for learning the entire map are the visual radius R, the visual pointing angle γ, the visual eccentricity ϵ, and the arm solution flag (elbow up or down), which is necessary because there is no kinematic package to determine angle estimates. Network inputs for correction learning are described in section 3.5.1. Entire map learning reduces the average error from over two meters to about six cm after 5000 trials, with essentially no difference between the noisy and noise-free cases, which allocated 5000 and 4997 neurons, respectively. Correction map learning reduced the error from about 4.8 cm to 1.2 cm in the noisy case and from 4.8 cm to 0.3 cm in the noise-free case.

The figure shows that learning the entire map takes far longer and requires many more neurons than learning a correction map. Since A 200-neuron network gives adequate speed and accuracy for learning corrections, especially when accuracy is limited by noise, we can conclude that exploiting engineering knowledge significantly improves the performance of neural networks in adaptive hand-eye calibration.

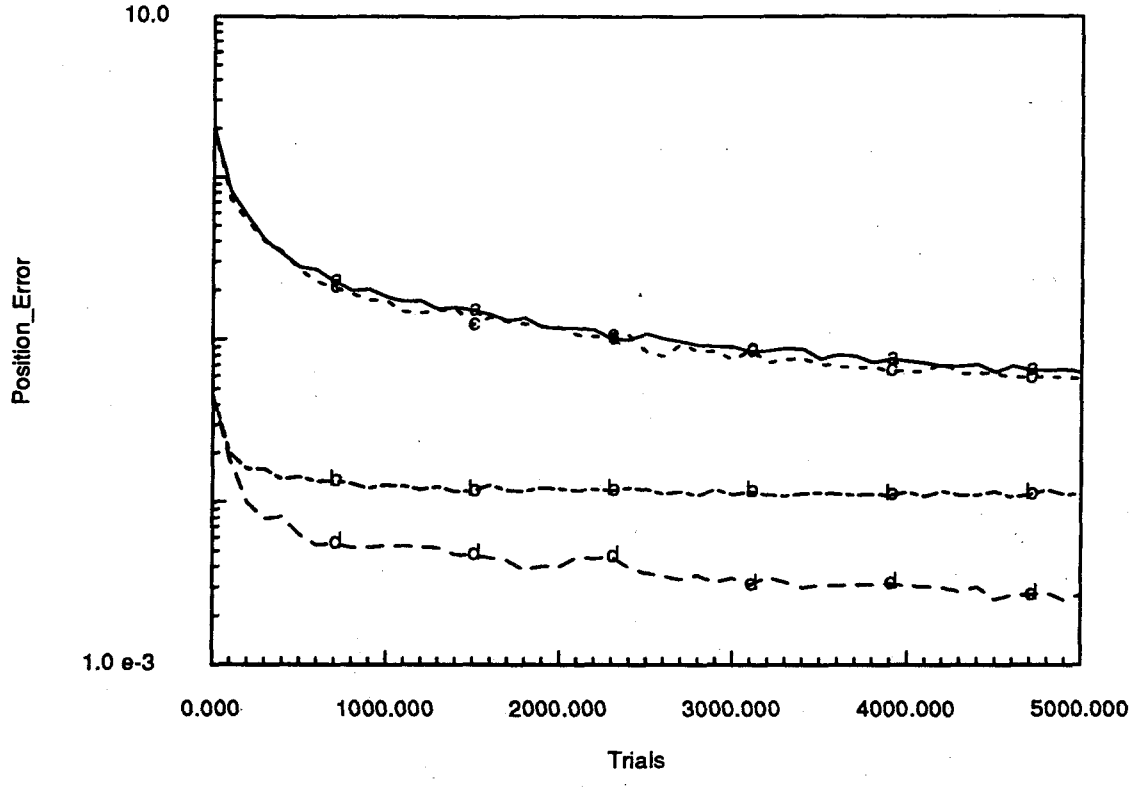

Figure 3.45: Entire map learning compared with correction learning with and without visual noise. Plot (a) shows the position error performance of entire map learning without noise; plot (c) shows the position error performance with visual noise. Plots (b) and (d) are noisy and noise-free correction learning respectively.

Network Performance With Large Kinematic Errors

Kinematic and nonlinear perturbation parameters were modified to test calibration system performance in the presence of large parameter errors. This is similar to learning the entire map in that the initial kinematic errors are significant. The difference is that the system still employs nominal kinematics to obtain an initial arm angle estimate. Thus the system learns a correction map rather than an entire map.

Kinematic parameters were increased from those in table 3.1 to those in table 3.3. Nonlinear perturbation parameters were changed from those in table 3.2 to those in 3.4, which increased maximum tangential perturbations from one to five cm at two

Parameter	Nominal Value	Actual Value
$\|\mathbf{a}\|$	1.00 Meter	1.05 Meter
$\|\mathbf{b}\|$	1.00 Meter	0.96 Meter
$\mathbf{v}_x$	-1.00 Meter	-1.08 Meter
$\mathbf{v}_y$	0.00 Meter	0.10 Meter
O_α	0.00 Radian	0.00 Radian
O_β	0.00 Radian	0.10 Radian
O_γ	0.00 Radian	0.20 Radian

Table 3.3: Nominal and actual kinematic parameters for the case of large kinematic parameter errors.

Parameter	Value
Vision Perturbation Parameters	
η	7.500×10^{-2}
ζ	1.250×10^{-2}
Vision Pointing Perturbation Parameter	
ν	1.000×10^{-2}
Arm Perturbation Parameters	
ξ	-8.686×10^{-3}
κ	9.143×10^{-3}

Table 3.4: Nonlinear perturbation parameters for the case of large kinematic parameter errors.

meters for each angular degree of freedom and, the maximum radial perturbation from two to five cm at three meters. Five centimeters was selected as being representative of fairly large errors. Other than scaling, the basic form of the perturbations was unchanged. Plots of the visual distortion and perturbations are shown in figures 3.46 and 3.47 respectively. Noise parameters were not modified.

Plots of system position error performance are shown in figure 3.48 for the large-error case. Plots of the unchanged system's error performance are also shown for comparison. The virtual neuron network with no constraints on the number of neurons was used in all cases. The network was able to reduce the average error from 40 cm

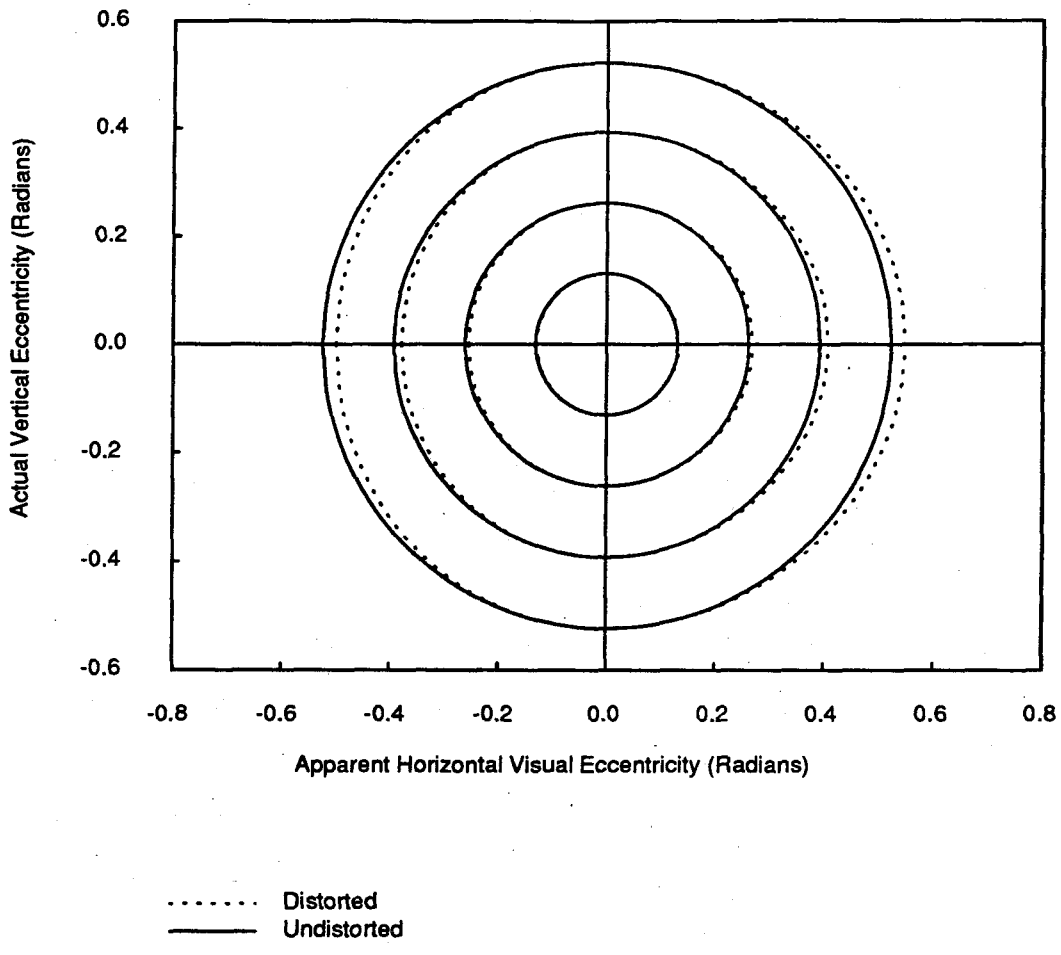

Figure 3.46: Large nonlinear visual angular eccentricity perturbation. One circle in each of the plotted circle pairs is distorted by the eccentricity perturbation. The other is not. In the left half plane the distorted circle is closer to the origin, while in the right half plane it is farther away. The distortion is zero on the y-axis. Units are in radians.

to 4.5 cm in 1000 trials in the noisy case, allocating 999 neurons in the process. After 5000 trials, error had decreased to about 3 cm and the system had allocated 4982 neurons. Noise-free performance was just a few mm better with nearly as many neurons allocated.

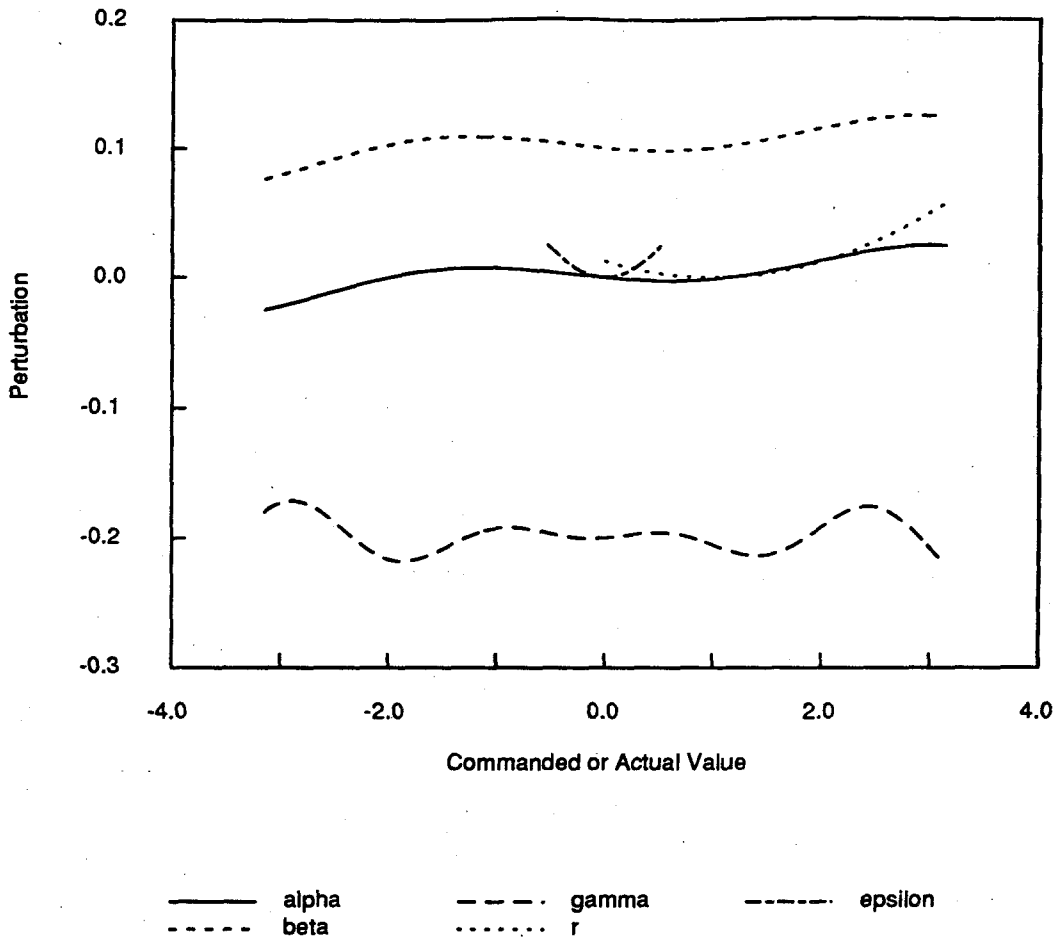

Figure 3.47: Plots of the $\delta\alpha$, $\delta\beta$, $\delta\gamma$, $\delta\epsilon$, and δr perturbations over their accessible domains for the case of large errors. Offsets O_α, O_β, and O_γ are added to the $\delta\alpha$, $\delta\beta$, and $\delta\gamma$ plots respectively. Angular units are radians; linear units are meters. Note that in all cases $r \geq 0$.

Non-Gaussian Neurons

Networks using non-Gaussian neurons were explored by constructing neurons with response functions composed of parabolic and cosine segments and evaluating them with the plant and network structures already employed for Gaussian neurons.

The cosine response function is just one-half period of a scaled cosine plus a

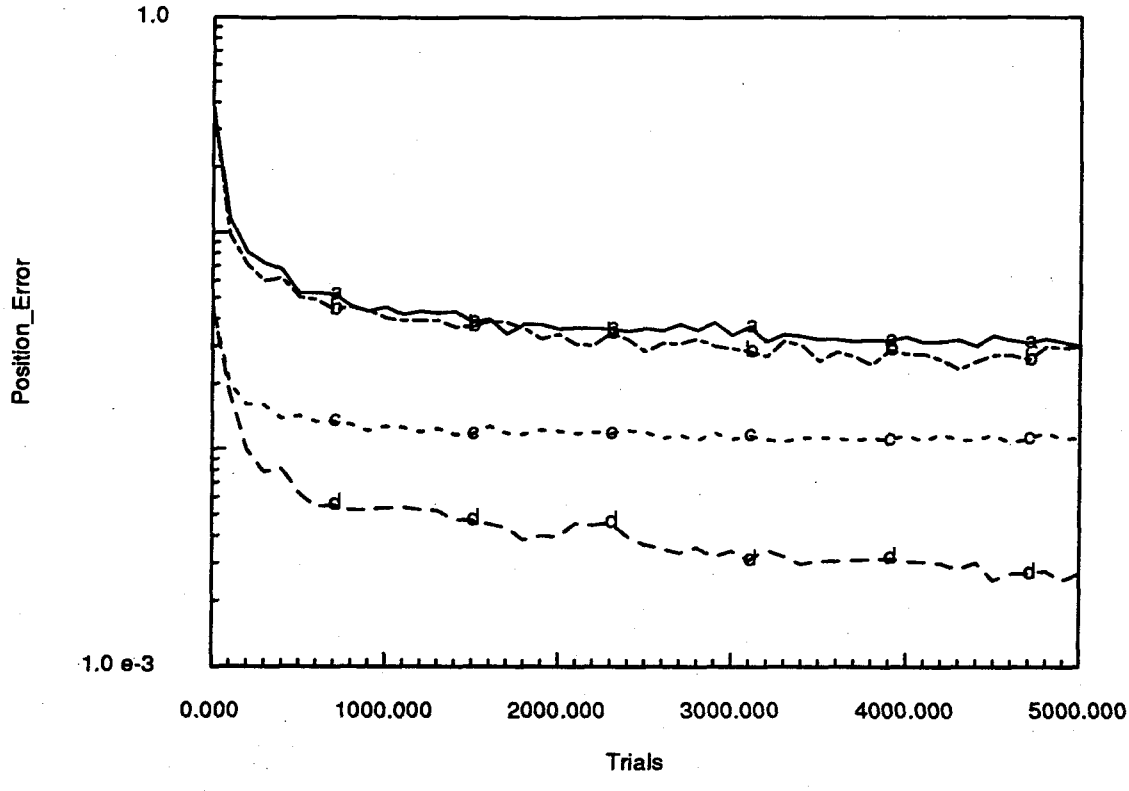

Figure 3.48: Positioning error performance for large kinematic parameter errors. Plots (a) and (b) show position error performance in the noisy and noise-free cases, respectively. For reference, plots (c) and (d) respectively show noisy and noise-free position error performance for the unchanged system.

constant offset:

$$o = \begin{cases} \frac{1}{2}(1 + \cos(\pi \frac{d}{R})) & \text{if } d < R \\ 0 & \text{otherwise.} \end{cases} \tag{3.107}$$

Here the radius of the receptive field is R.

The parabolic response function is composed of parabolic segments with upward

and downward concavities that are joined so the first derivative is continuous:

$$o = \begin{cases} 1 - 2(d/R)^2 & \text{if } 0 \leq d \leq R/2 \\ 2(1 - (d/R))^2 & \text{if } R/2 < d \leq R \\ 0 & \text{otherwise.} \end{cases} \tag{3.108}$$

Again, the radius of the receptive field is R.

The cosine and parabolic response functions are identically zero outside their receptive fields.

For the cosine response function, gradient descent parameters of section 3.5.3 are:

$$o_j'(z_j) = \begin{cases} -\pi \sin(\pi d_j/R_j)/4R_j d_j & \text{if } d_j < R_j \\ 0 & \text{otherwise} \end{cases} \tag{3.109}$$

$$\frac{\partial o_j(z_j)}{\partial R_j} = \begin{cases} \pi d_j \sin(\pi d_j/R_j)/2R_j^2 & \text{if } d_j < R_j \\ 0 & \text{otherwise,} \end{cases} \tag{3.110}$$

and for the parabolic response function we have:

$$o_j'(z_j) = \begin{cases} -2/R_j^2 & \text{if } d_j < R_j/2 \\ -2(R_j - d_j)/(d_j R_j^2) & \text{if } R_j/2 \leq d_j \leq R_j \\ 0 & \text{otherwise} \end{cases} \tag{3.111}$$

$$\frac{\partial o_j(z_j)}{\partial R_j} = \begin{cases} 4d_j^2/R_j^3 & \text{if } d_j < R_j/2 \\ 4(R_j - d_j)d_j/R_j^3 & \text{if } R_j/2 \leq d_j \leq R_j \\ 0 & \text{otherwise.} \end{cases} \tag{3.112}$$

Since Gaussian neurons decay from a value of 0.0625 for distances $\geq R$, while the

outputs of the cosine and parabolic neurons vanish at distances $\geq R$, learning adjustments in networks based on the non-Gaussian neurons are somewhat more highly localized. This is manifest in a greater sensitivity to the half-maximum fraction, h. In simulations using 200-neuron virtual neuron networks, both non-Gaussian networks were unstable for $h \leq 0.8$. In contrast, Gaussian networks, were unstable for $h \leq 0.2$. Figure 3.49 shows the results of these simulations for the parabolic neurons in the noise-free case. The best accuracy performance was inferior, but by only about one mm, to that of the Gaussian networks shown in figure 3.31, and followed the same trends. Large values of h were stable, but were less accurate than smaller values. Performance of the cosine neurons was virtually identical to that of the parabolic neurons and is not shown.

The non-Gaussian neurons investigated here are quite similar to Gaussian neurons, and have similar performance. The choice of neuron used in applications should be dictated by the ease of implementation. The use of neurons with other activation functions is also possible. Baldi and Heiligenberg [Baldi and Heiligenberg (1988)] have used neurons with triangular activation functions with some success in studies of hyperresolution.

3.6 Hand-Eye Calibration Using Nonlinear Estimation

A calibration correction system using nonlinear estimation was implemented to provide a comparison with the neural network approach. The adaptation process based on nonlinear estimation is shown in figure 3.50. It is similar to the process based on neural networks shown in figure 3.5, except that instead of learning a correction to

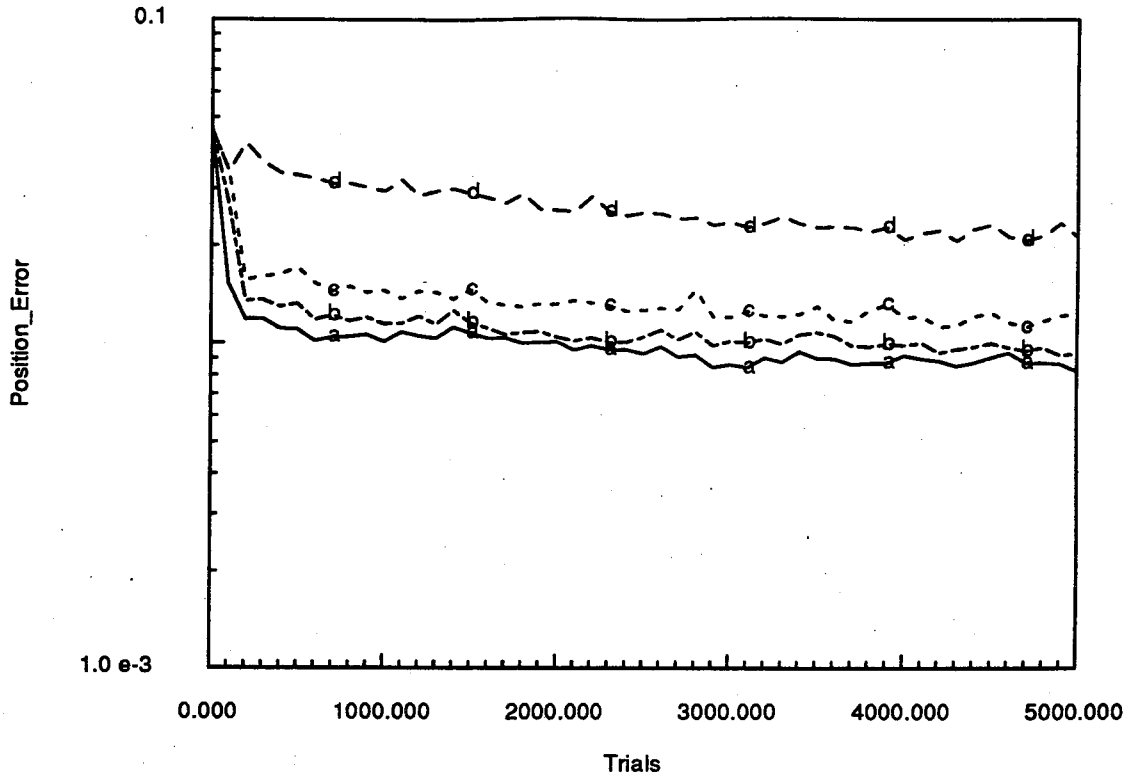

Figure 3.49: System positioning error performance using a 200-neuron network with parabolic neurons. Plots (a)-(d) have h = 1.0, 1.5, 2.0, and 4.0 respectively.

the nominally-calculated arm angles, the system uses the difference between calculated and observed kinematic behavior to identify the arm's actual kinematic model parameters, which are initially set to their nominal values. System performance is thus intimately tied to model sophistication. Once identified, the parameters are available for all kinematic computations, both forward and reverse. Both processes are described in section 3.4.3.

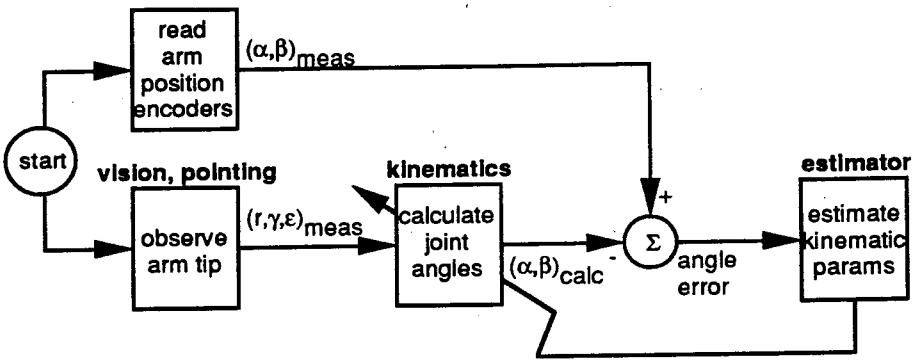

Figure 3.50: Adaptation process based on nonlinear estimation. Reverse Marquardt is shown.

3.6.1 Nonlinear Estimation Approach

To identify parameters in the Levenberg-Marquardt method, a merit function χ^2 is defined [Press et al (1988)]:

$$\chi^2 = \sum_{i=1}^{N} [\frac{y(x_i) - y(x_i; \mathbf{a})}{\sigma_i^2}]^2, \tag{3.113}$$

where $y(x_i)$ are observed, or measured, values at the points x_i, $y(x_i; \mathbf{p})$ are values calculated at the points x_i using the parameter estimates in the parameter vector $\mathbf{p}$, and σ_i is the standard deviation of the measurement at point i, which is set to one in all cases.

χ^2 as a function of $\mathbf{p}$ is minimized to determine the best parameter vector p. Minimization is by gradient descent far from the minimum, and by the inverse Hessian method as the minimum is approached. The inverse Hessian method is given by:

$$\delta \mathbf{p} = \mathbf{H}^{-1} \cdot [-\nabla \chi^2(\mathbf{p}_{\text{curr}})], \tag{3.114}$$

where the subscript *curr* means the current value, and $\mathbf{H}$ is approximately the Hessian, the matrix of second partial derivatives of χ^2 with respect to the components of

$\mathbf{p}$. $\mathbf{H}$ is given by

$$\mathbf{H}_{kl} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} [\frac{\partial y(x_i; \mathbf{p})}{\partial p_k} \frac{\partial y(x_i; \mathbf{p})}{\partial p_l}]. \tag{3.115}$$

Continuous switching between the two methods is accomplished by defining a matrix $\mathbf{M}$ [Press et al (1988), Ralston and Rabinowitz (1978)]:

$$\mathbf{M}_{kl} = \mathbf{H}_{kl} + \lambda \delta_{kl} \mathbf{H}_{kl}, \tag{3.116}$$

which captures both the gradient descent (diagonal matrix proportional to λ) and the inverse Hessian algorithms. The factor δ_{kl} is the Kronecker delta. The estimation algorithm controls λ, which is large far from the minimum and small near the minimum. This switches between the two matrices and hence between the two algorithms.

Using the algorithm requires providing the functions that calculate $y(x_i; \mathbf{p})$ and their partial derivatives with respect to the components of $\mathbf{p}$ at the points x_i, and providing corresponding data measurements $y(x_i)$ at the same points for calculating χ^2.

The kinematic model was identical to the one used in the neural network system. Kinematic variables were the arm link lengths $a = \|\mathbf{a}\|$ and $b = \|\mathbf{b}\|$, the two components of the vector locating the vision system $\mathbf{v}_x$ and $\mathbf{v}_y$, and the three encoder offsets O_α, O_β, and O_γ. No attempt was made to model the nonlinear perturbations.

3.6.2 Calibration Using Forward Marquardt Estimation

To avoid problems with arm singularities when $\beta = 0$, the function y was defined in terms of the arm tip radius r and visual angle θ relative to the vision system

base frame. These can be calculated from joint angles without worrying about elbow singularities, and thus correspond to forward kinematics; hence the name. Measured values of these quantities are available at each training trial. The $y(x_i; \mathbf{p})$ are given by:

$$y = r^2 + \theta^2 = \mathbf{r}_x^2 + \mathbf{r}_y^2 + (\arctan(\mathbf{r}_y, \mathbf{r}_x) - O_\theta)^2. \tag{3.117}$$

Defining $\alpha = \alpha_{meas} + O_{alpha}$, $\beta = \beta_{meas} + O_\beta$, the components of $\mathbf{r}$ are;

$$\mathbf{r}_x = a\cos(\alpha) + b\cos(\alpha + \beta+) - \mathbf{v}_x, \tag{3.118}$$

$$\mathbf{r}_y = a\sin(\alpha) + b\sin(\alpha + \beta+) - \mathbf{v}_y. \tag{3.119}$$

The subscript *meas* indicates the measured value. Using the normal type face for vector magnitudes and components and defining $\theta = \epsilon_{meas} + \gamma_{meas} - O_\gamma$ in accordance with our earlier kinematic definitions, the expressions for the partial derivatives are given by:

$$\frac{\partial y}{\partial a} = 2(r_x\cos(\alpha) + r_y\sin(\alpha)) + \frac{2\theta}{r^2}(r_x\sin(\alpha) - r_y\cos(\alpha)) \tag{3.120}$$

$$\frac{\partial y}{\partial b} = 2(r_x\cos(\alpha+\beta) + r_y\sin(\alpha+\beta)) + \frac{2\theta}{r^2}(r_x\sin(\alpha+\beta) - r_y\cos(\alpha+\beta)) \tag{3.121}$$

$$\frac{\partial y}{\partial v_x} = -2(r_x - \frac{\theta r_y}{r^2}) \tag{3.122}$$

$$\frac{\partial y}{\partial v_y} = -2(r_y + \frac{\theta r_x}{r^2}) \tag{3.123}$$

$$\frac{\partial y}{\partial O_\alpha} = -2(r_x - \frac{\theta r_y}{r^2})(a\sin(\alpha) + b\sin(\alpha+\beta)) + 2(r_y + \frac{\theta r_x}{r^2})(a\cos(\alpha) + b\cos(\alpha+\beta)) \tag{3.124}$$

$$\frac{\partial y}{\partial O_\beta} = -2(r_x - \frac{\theta r_y}{r^2})(b\sin(\alpha+\beta)) + 2(r_y + \frac{\theta r_x}{r^2})(b\cos(\alpha+\beta)) \tag{3.125}$$

$$\frac{\partial y}{\partial O_\gamma} = -2\theta. \tag{3.126}$$

A reverse Marquardt approach, based on defining $y = \alpha^2 + \beta^2$ was also implemented. In most cases it gave similar results except for the problems with singularities mentioned above. The one exception was that it achieved significantly greater accuracy than forward Marquardt in handling vision scale factor errors.

Estimation Performance on Normal Plant

The so-called normal plant uses the kinematic parameters defined in table 3.1 and the joint angle limits defined in eqs. 3.29 and 3.30. In trials without visual noise or nonlinear perturbations, the estimator-based calibration system was able to identify these parameters to within $\approx 0.001\%$, achieving an average positioning accuracy of two microns after ten trials. This performance is far better than that of a system employing a 500-neuron virtual neuron network, which was run on the same plant for comparison. The neural network-based system was able to achieve an accuracy of about four mm after 500 trials. The uncorrected positioning error in both cases was about 4.6 cm. Performance of the two systems is plotted in figure 3.51.

Estimation Performance with Large Kinematic Errors

The actual kinematic parameters were changed from those in table 3.1 to those in table 3.3, increasing the kinematic parameter errors and increasing the average uncorrected positioning error to over 0.4 meter. The estimator-based calibration system was able to identify the parameters to within $\approx 0.001\%$, and to obtain an average positioning error of less than .02 mm. The network-based calibration system, in contrast, was able to obtain an average positioning error of about five cm, allocating all 500 neurons in the process. Again, no nonlinear perturbations or noise were present. Performance of the two systems is plotted in figure 3.52.

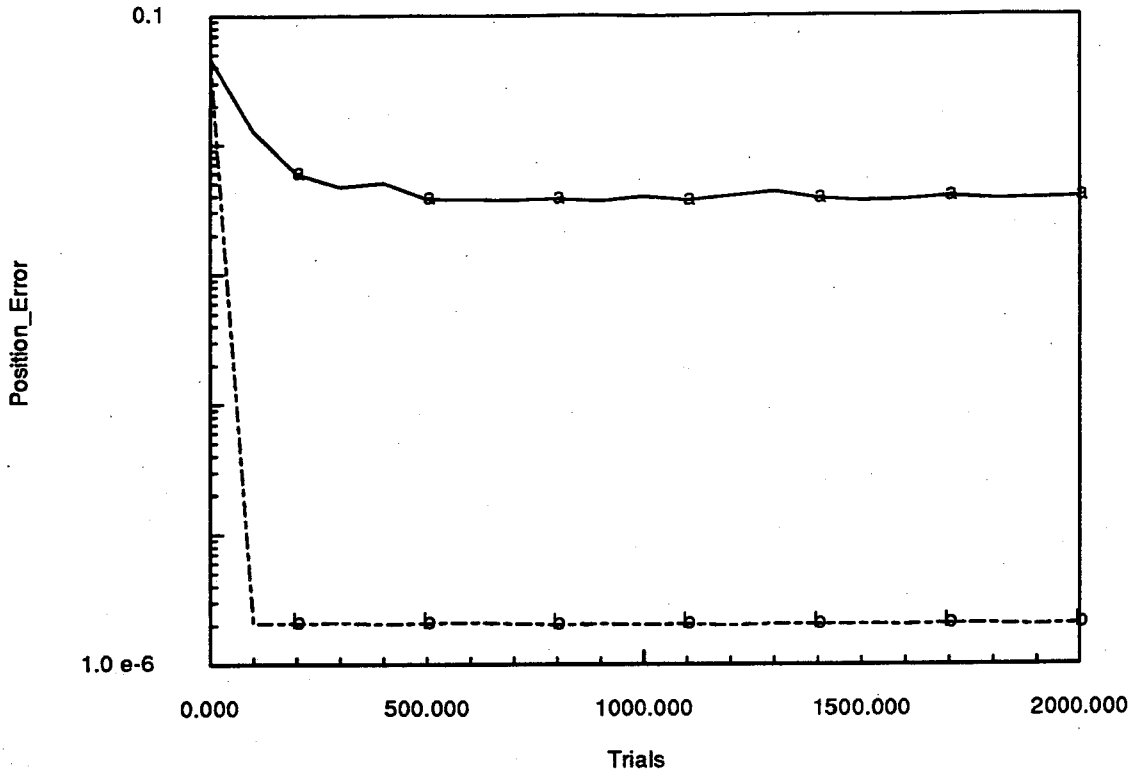

Figure 3.51: Error-correction performance of 500-neuron virtual neuron network and nonlinear estimator in hand-eye calibration. Plot (a) is the neural network. Plot (b) is the estimator. Noise and nonlinear perturbations were not present.

Estimation Performance in Localized Regions

In the absence of noise and unmodeled perturbations estimator performance is excellent when the region accessible to the manipulator is sampled at points with sufficient separation. In applications, however, a manipulator may work in localized regions for protracted periods, creating numerical problems that must be addressed in deciding when the estimator should be invoked. To assess the effects of localization on accuracy, the region accessible to the normal plant was restricted to the region $\alpha = -\frac{\pi}{3} \pm 0.1$ radians and $\beta = \frac{2\pi}{3} \pm 0.1$ radians. This defines a small, roughly parallelogram-shaped, region with sides about 20 cm in length that is nominally located two meters from the vision base and one meter from the arm base along the positive $x - axis$. The

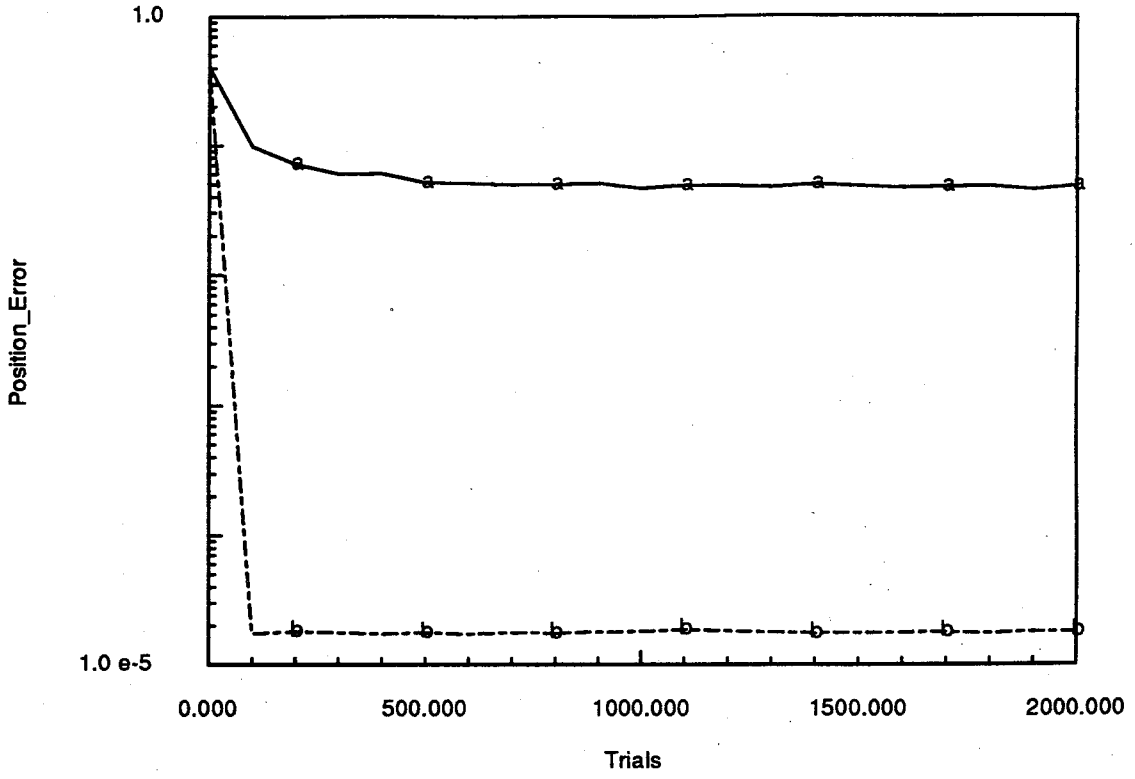

Figure 3.52: Error-correction performance of a 500-neuron virtual neuron network and the nonlinear estimator in hand-eye calibration with large kinematic errors. Plot (a) is the neural network. Plot (b) is the estimator. Noise and nonlinear perturbations were not present.

elbow is below the axis. In addition, the vision eccentricity limits were reduced to ± 0.05 radian around the optical axis. In this situation, which had an average uncorrected positioning error of 4.2 cm, the neural network outperformed the estimator, using 205 neurons to achieve an average positioning accuracy of about 0.4 mm in the region, compared to the estimator's average positioning accuracy of 0.6 cm. The estimator was able to identify the kinematic parameters only to within about one percent. Performance of both systems is shown in figure 3.53.

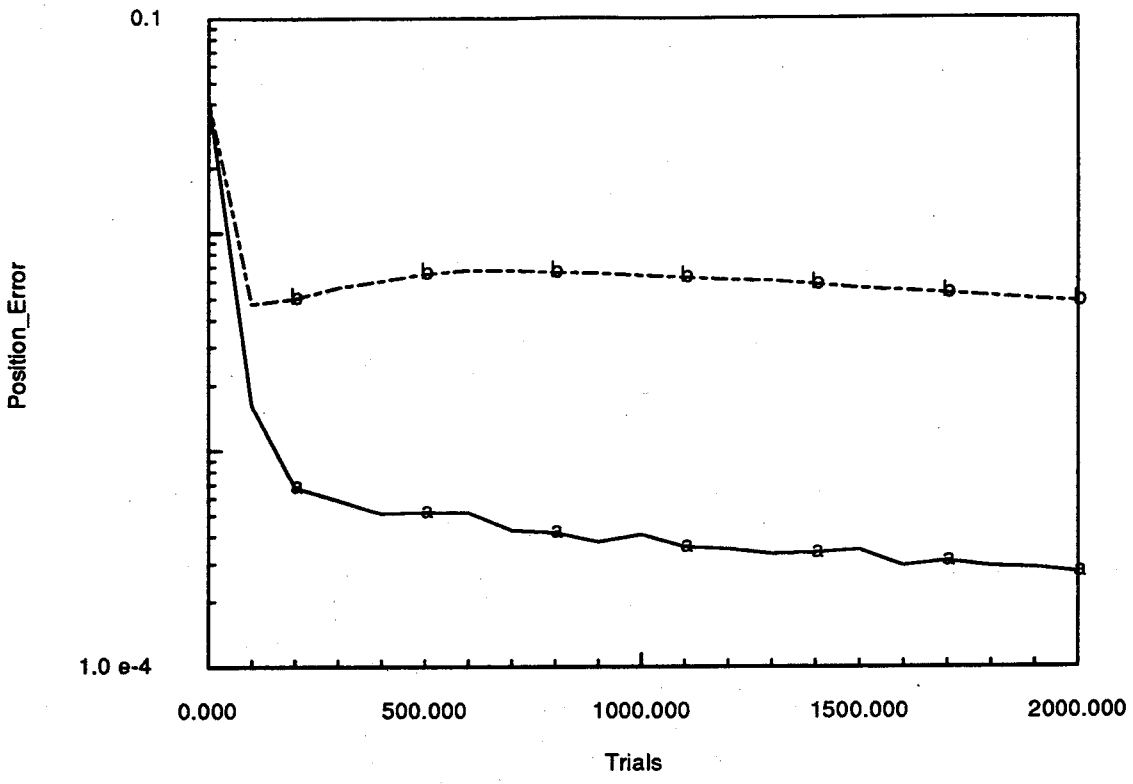

Figure 3.53: Calibration error-correction performance of a 205-neuron virtual neuron network and nonlinear estimator in a localized region. Plot (a) is the neural network. Plot (b) is the estimator. Noise and nonlinear perturbations were not present.

The Effects of Unmodeled Structure on Estimator Performance

Nonlinear estimation performed well in the above tests, in which nonlinear perturbations and noise were not present. Except for visual range scale factor errors (not shown), however, where average positioning error with the normal plant and a vision range scale factor of 0.9 was six microns,[16] estimation was unable to cope with noise, nonlinear perturbations, and plant drift until the magnitude of the disturbances was reduced nearly to zero. The reason for this, which will be the subject of further work, is the so-called large residual problem which arises when unmodeled effects cause χ^2 to have large values. Possible remedies include increasing the sophistication, and hence

[16]Reverse Marquardt result.

the complexity, of the model, which will involve identifying more parameters, and modifying the estimator to use a better algorithm for calculating λ, which switches between the gradient descent and inverse Hessian matrices in equation 3.116.

3.7 Discussion

This section compares the neural network- and estimation-based approaches to hand-eye calibration, and discusses issues relating to hand-eye mappings.

3.7.1 Comparison of The Network and Estimation Approaches to Calibration

These main points emerged from this work:

1. Platt networks coupled with virtual neuron adaptation give good noise-limited performance and handle nonlinear perturbations. They do not need parametric plant models, but have many parameters that must be adjusted to obtain the best performance.

2. Plant description information is held implicitly in a neural network, and so is inaccessible unless the network is designed explicitly to identify parameters. A separate network is required for reverse kinematics.

3. The neural networks employed in this work adapt relatively slowly to plant changes, because the network must sample the entire space. Architectural changes, a subject for future work, may be able to ameliorate this.

4. Nonlinear estimation has outstanding performance when it works, but it requires a parametric model, and was not able to accommodate noise and perturbations in these tests, probably due to the algorithm employed. Perturbations could possibly be accommodated by developing an approximate model and estimating the parameters. Performance could also be improved by estimating vision parameters.

5. Nonlinear estimation provides information that is globally useful in a convenient form (formulas in this context are compact).. In particular, the information is useful for both forward and reverse kinematics and thus is useful for planning, measurement, and other robot activities.

6. Nonlinear estimation adapts quickly, in principle, to plant variations because it is not necessary to sample the entire space.

3.7.2 Efficient Neural Representations and Hand-Eye Mappings

In the hand-eye system described in this thesis five inputs, the range, angular eccentricity, pointing angle and the two nominally-calculated joint angles, are required to calculate arm joint angle corrections. This is inefficient because the corrections for visual errors must be learned for all joint angle pairs, and in multi-arm systems they would have to be separately learned for each arm.

Assuming each of the five input degrees of freedom requires n neurons, a hand-eye map would require n^5 neurons to generate the two joint angle corrections. All the inputs are required because of the assumed visual and kinematic perturbations. If an intermediate representation of spatial coordinates were used instead of going directly

from visual stimuli to joint angles, the mappings would require fewer neurons and could be used by all arms (and other kinematic degrees of freedom). In particular, if the range, angular eccentricity, and pointing angle values were to map into a common two-dimensional representation of the spatial location of the visual stimulus and this was mapped, in turn, into the joint angles, $n^3 + n^2$ neurons would be required, assuming again that n are required per degree of freedom. For $n > 1$ it is always true that $n^3 + n^2 < n^5$.

A common internal spatial representation is also important for the integration of mental and observational processes.[17] Some positioning targets, for example, may not be visible because of occlusions and some, like an intermediate position in front of an object, either are not visible or can be confounded with the background because they have no material reality. The positions of such targets must be derived indirectly, either from memory, or by applying known coordinate transforms to nearby objects that *can* be seen. Reaching for targets that are not visible or are not being observed thus involves using an indirectly-derived representation of the target's position rather than a representation resulting from direct visual observation.

In addition to using neurons suboptimally, the hand-eye representations of the type employed in this work are slow to adapt to plant changes. This is because adaptation can occur only where there has been hand-eye experience. Correcting drifts over the entire space means that the entire space must be scanned. Animals probably implement hand-eye coordination differently because they appear to be able to shift entire maps quickly.

Assuming that animals possess structures that facilitate learning things important to their survival, and that such an approach would be useful in artificial systems, modifying the network architecture to calculate specific parameters with broad (as

[17]This is true for both neural network- and estimation-based systems.

opposed to local) relevance, such as offsets and scale factors, could significantly improve system response to plant drifts. In that case large-scale behavior modification could be based on local experience and adaptation to certain classes of drifts could occur quickly. Calculating specific parameters requires developing specialized circuitry adapted to identifying the parameters in question, and combining that circuitry with normal learning circuitry in a consistent way. Quickly identifying image scale factors and pointing offsets, for example, would be beneficial for maintaining an accurate internal spatial representation, because once identified from a few observations, they could be globally applied without requiring exhaustive sampling.

Chapter 4

Learning and Executing Hand-Eye Motor Skills

4.1 Introduction

This chapter details the learning robot motor control system introduced in chapters 1 and 2. It describes objectives, previous work, the overall approach and implementation, and system performance. It closes by comparing the performance of the learning control system with that of a conventional linear controller at ball catching.

4.1.1 Objectives

The overall objective is to investigate learning hand-eye robot motor control in simple motor tasks. More specifically, the objective is to design and simulate a system that can, without being explicitly programmed:

1. Learn to recognize arm responses and control simple arm behaviors, compensating for changes in mass.
2. Learn to recognize and predict simple behavior of external objects.
3. Learn to interact with external objects to satisfy simple goals.

In the context of ball catching, which is the task considered here, the system should be able to learn to catch different balls that follow a variety of trajectories using an arm with variable physical parameters. The system should not require being specifically programmed and should not require the design of a specific controller. It must learn to recognize and predict arm and ball behavior, and learn how to control the arm so the ball is caught. Predicting the behavior of an object means predicting the way its state will evolve in time.

4.2 Previous Work

As mentioned in chapter 1, a great deal of the previous work on robot motor learning has addressed the problem of compensating for manipulator dynamics [Albus (1972), Albus (1975b), Albus (1975a), Atkeson (1986), Goldberg and Pearlmutter (1988), Miller (1987), Raibert (1977), Raibert and Horn (1978)]. This problem has

also been addressed by control theorists [Craig (1986a), Slotine and Li (1986), Slotine and Li (1991)], and on-line adaptive compensation algorithms employing parameter estimation are now being commercially applied to improve the tracking accuracy of high-performance industrial robots [Larkin (1993)].

More recent motor learning work addresses learning in the task domain. Noting that a system must experience its own control situations to learn to issue appropriate control commands, Handelman and co-workers [Handelman, Lane and Gelfand (1989)] use declarative knowledge in the form of an internal critic to guide the motor learning process. They have focused on particular motions and on well-defined repetitive events such as learning to swing a tennis racket. Using a human supervisor, Pomerleau has trained a neural network to steer a robotic vehicle [Pomerleau (1990)].

The research reported here addresses a more global framework for sensory perception and motor learning that can be extended to a wide variety of sensor-motor tasks including those in which the system must interact with external objects. The framework is based upon both learning by trial and error and the use of declarative knowledg (rules), which are applied by an internal critic, called the mentor. The mentor makes specific recommendations for improving motor performance which are learned by the system.

4.3 Approach to Robot Motor Learning and Control

The system draws deeply upon biological ideas. Basic design principles are that accurate task execution is achieved by comparing predicted with observed behavior

and that, at least from a motor control standpoint, a system comprehends a situation when it can reliably predict how it will evolve. Surprise, or deviation from prediction, triggers learning if the unexpected situation is not known, and/or invoking a different behavior if the situation is known.

The system includes a temporally-ordered memory of the motor command, sensor observation, and sensor prediction[1] streams in the current training or execution episode, along with a comparator that determines whether experience matches predictions. Comparison of predicted and observed behavior is thought to be important in biological motor control [Brooks (1986)]. Neural networks learn to recognize and predict internal and external (arm and ball, respectively) behaviors and to issue motor commands (voltages) appropriate to the control situation.

The system mimics the observed hierarchical behavior of animals, in which low-level processes in understood, low-complexity situations are nearly automatic, while confusion or misunderstanding requires conscious attention,[2] and may cause a change in behavior as well as learning.

Learning, which is taken here to mean improving performance according to some metric, requires the ability to recognize and reinforce better performance [Brooks (1986)]. Learning capabilities may be nearly automatic, as in learning to walk or control arm movements, or they may require conscious attention, as in learning to play a musical instrument.

Learning effectiveness can be improved if specific knowledge rather than random

[1]This is inspired by the efference copy, or corollary discharge, of biological motor control in which copies of efferent, or outgoing, motor commands are relayed to higher motor control centers to allow comparison of commanded and actual behavior [Brooks (1986)].

[2]Driving on a smooth highway in light traffic is an example of an understood, low complexity situation which is nearly automatic. The sudden appearance of an oncoming car in your traffic lane requires conscious attention.

trials is used to suggest performance improvements. The role of the mentor is to suggest specific ways to improve performance at a functional level (e.g., move farther next time). These suggestions are transformed into behavior modifications by a low-level process that trains networks to invoke pursuit motor commands that satisfy the mentor's suggestions in particular control situations. A mentor, of course, may be internal or external to a particular system.

The learning system considered here has no closed-loop arm controller, no initial arm or external object models and no explicit program describing how to catch balls. It does have a great deal of internal structure, along with innate motivation, that allows it to observe the behavior of itself and external agents, to correlate its actions with their observed effects, and makes it want to move its arm, catch balls, or perform other tasks. It also has innate structures that cause it to issue randomly directed, but structured, pursuit commands [Brooks (1986)], that lead ultimately to its capability for purposive movement. As we shall see it learns very accurate and stable behavior.

To make the learning problem tractable, capabilities are learned in isolation and order, again as in learning to play an instrument. The system first learns, much like an infant, to perform controlled pursuit movements by exercising its arm a great deal and observing its response to the movement commands. Arm movements are random, but they have a stereotypical form mimicking pursuit movements observed in animals[3] [Brooks (1986)]. Since arms have different behaviors if they are loaded or unloaded, that is, whether they are or are not carrying objects, different arm masses are used. After the system has learned to control its arm movements, it observes the behavior of simple external objects,[4] learning to identify different objects and to predict their behavior. Once the system knows how to perform pursuit movements

[3]Relying on a stereotypical form, which could arise innately, is important to simplify the learning problem for simple movements. There are an infinite number of ways to move to a location.

[4]Light and heavy balls in this case, which have very different behaviors due to aerodynamic drag.

and can recognize and predict ball motion, it learns to catch balls with the advice of the mentor.

Since a sense of time ordering is essential, the existence of a system clock has been assumed, along with the ability to perform implicit time stamping.[5] In the present system, the clock, which drives the shift memory (see below), operates at a fixed frequency of 40 Hz.

While the overall approach, with augmentations, is believed to be extensible to more complex situations, such as recognizing and learning the behavior of complex plants, the system here deals with smooth motions, simple tasks, and uncluttered environments. The system, however, is able to detect anomalies, such as the ball's colliding with the ground or the arm because the comparator notes the inconsistency between the predicted and observed sensory streams.

4.4 The Motor Learning System

The overall system considered in this work consists of the robot, which has one linear degree of freedom, with its sensor and control systems, and the external agents which are light and heavy balls. The main robot elements are the light and heavy arm and its sensor system, the visual system, and the controller. During both learning and execution the controller relies heavily upon a temporally-ordered memory, which is modeled as a multistream shift register. Values stored in the shift register during a trial are used to construct the recognition, prediction, and control networks during

[5]There is debate on the neural representations of time. The olivary structures in the brain are known to oscillate stably, which may have some significance to motor learning and control. The sense of time associated with an active system clock, however, which is used in regulation and coordination, is distinct from the sense of time associated with high-resolution temporal discrimination processes like echlocation, which uses neural delay lines and coincidence detectors.

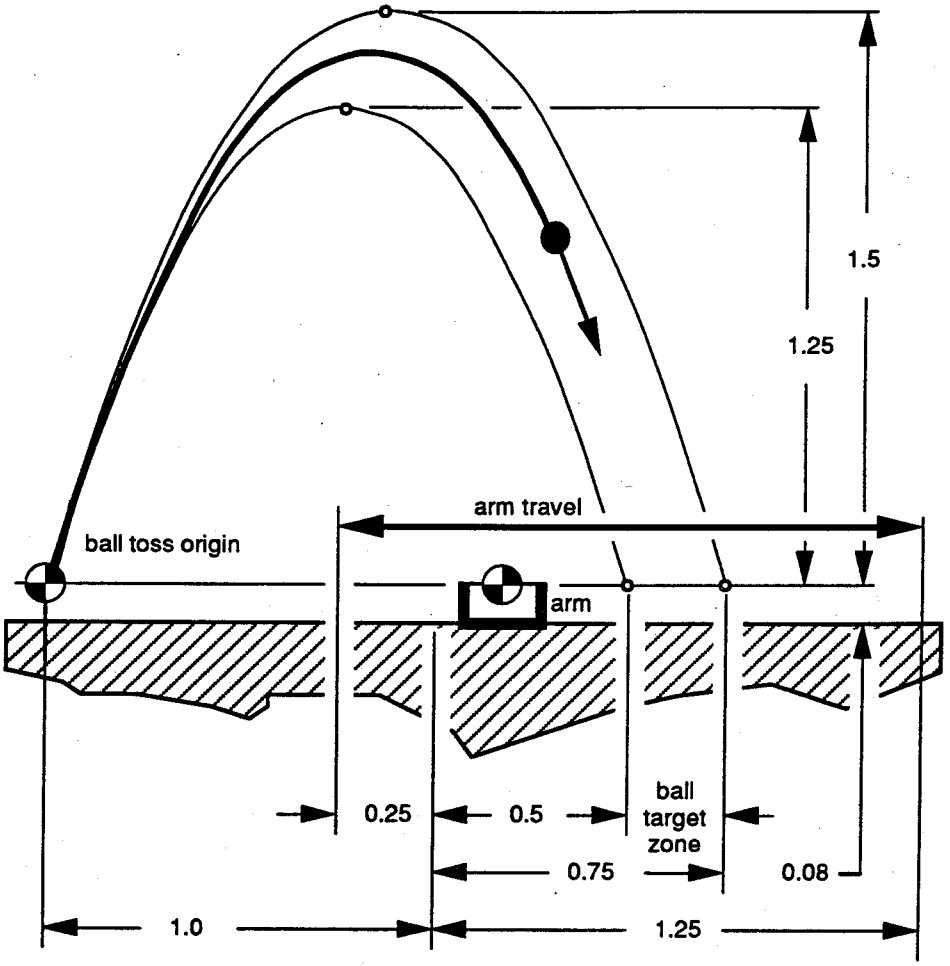

Figure 4.1: Catching layout. Dimensions are in meters.

learning and as network inputs during execution to generate behavior predictions and arm commands. The shift register memory stores both the observed and predicted sensor values as well as the instantaneous identity of the arm and the external object, if they are active, along with the output arm commands, which are voltage values.

The ball and the robot are simulated in some physical detail, including the effects of viscous and frictional damping, rebound, back motor emf, control nonlinearities, and so on. See figure 4.1 for a geometrical description of the robot, its environment, and the arm and ball travel limits. The overall robot control system structure is shown in figure 4.2.

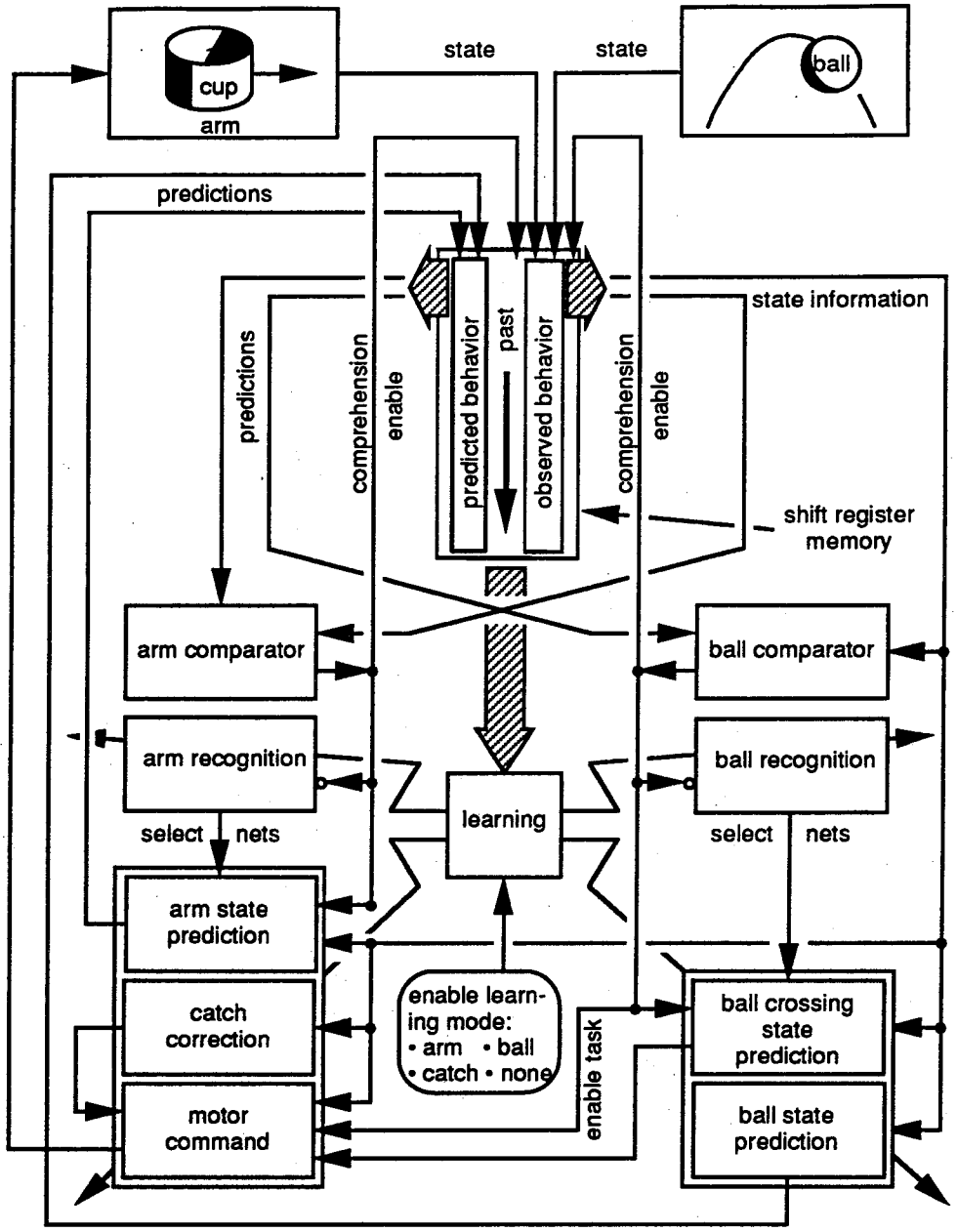

Figure 4.2: Learning control system structure showing principal information flow. The mentor and other learning functions, which are discussed in section 4.4.2, reside in the block labeled learning. For clarity, logic is not shown in detail.

4.4.1 System Elements

The system is made up of several elements. The arms, balls, comparators, mentor, neural networks, shift register memory, and vision system are briefly described here.

Arms. The two arms, which differ only in mass, each have one horizontal degree of freedom, which positions the cup that is used for catching. The different arm masses simulate changes in dynamics, as when grasping or releasing objects. The cup, which is 20 cm in diameter and 8 cm high, can be positioned anywhere within limits shown in figure 4.1. The inner cup surface is bounce-absorbent,[6] meaning that objects rebound from an inner surface with their velocity components normal to the surface essentially zero. The outer surface is hard, so objects rebound according to their elastic properties and the cup velocity. All the cup surfaces are taken as frictionless, so there is no change in a colliding object's tangential velocity component during a collision. Thus balls, because of their symmetry, undergo no changes in rotational energy when they collide with the cup. Arm motion is assumed to be unaffected by collisions.

The light arm mass is 0.1 kg, while that of the heavy arm is 0.5 kg. The arm is driven by a motor coupled to a traction transmission, which is a wheel of radius $\alpha = 8$ cm on the motor shaft. The linear arm velocity, $\dot{a}_x$ is thus given by $\dot{a}_x = \alpha\omega$, where ω is the angular velocity of the armature shaft. Damping is assumed to be proportional to arm velocity, with the damping force f given by $f = b\dot{a}_x$. It is assumed that $b = \mu gM + \nu dh$, where $\mu = \nu = 0.1.$, g is the acceleration of gravity, M is the arm mass, d is the cup diameter, and h is the cup height. The νdh term makes the viscous damping force dependent upon the cup size. The motor terminal resistance, R, is 3.3 ohms, and the motor torque constant κ_m is 0.125 newton-meters/amp; the mass of the motor armature is absorbed into the arm mass. With 20 volts across the motor, the arm's terminal velocity is 9.6 m/sec.

[6]The ball-surface interaction is modeled as a highly overdamped second-order system.

Making the conventional assumption that armature inductance is negligible, the arm's voltage-position transfer function is given by:

$$\frac{\hat{a}_x(s)}{\hat{V}(s)} = \frac{(\kappa_m \tau)/(\alpha RM)}{s(\tau s + 1)}. \tag{4.1}$$

The motor time constant, τ, is given by:

$$\tau = \frac{1}{b/M + \kappa_m^2/\alpha^2 RM}. \tag{4.2}$$

The motor amplifier is nonlinear, and can be saturated. The output is given by:

$$V_m = A \tanh(G_i V_c). \tag{4.3}$$

V_m is the voltage applied to the motor windings, $A = 20$ is the amplifier gain, $G_i = 4.0$ is the input voltage gain, and V_c is the command voltage. Arm sensors include position (of the cup centerline), velocity, and drive force. The drive force sensor sees the inertia of the entire system.

Balls. The light ball has a density of .03 kg/m^3, giving it the behavior of a beach ball, while the heavy ball has a density of 500 kg/m^3, giving it the behavior of a child's small rubber ball. Both balls are six cm in diameter. Ball behavior in simulation is calculated using exact formulas for damped ballistic motion and either underdamped or overdamped rebound, depending upon the surface with which the ball collides. The ground and exterior cup surfaces are hard, and the ball rebounds from them in an underdamped manner. The ball rebounds from the interior cup surfaces in a highly overdamped manner as discussed above under Arms. The ball is assumed to be subject to a viscous drag force, f, given by

$$\mathbf{f} = -\eta 6\pi r \mathbf{v}, \tag{4.4}$$

where $\eta = 1.81 \times 10^{-5}$ kg/meter-sec., the coefficient of viscosity, $r = 0.03$ meter is the ball radius, and $\mathbf{v}$ is the ball velocity.

A ball's position is taken as that of its center of mass. Balls are constrained to move in the $x - z$ plane. Light and heavy ball trajectories are shown in figures 4.3 and 4.4. respectively.

Comparators. The comparators compare predicted and observed sensory signatures for both the active ball and the active arm. They operate on information stored in the shift register memory, subtracting sampled observed values from the corresponding sampled predicted values over the ten most recent time steps. If the difference is below a predetermined threshold, the predicted and observed streams are taken to agree.

Mentor. As mentioned above, the mentor is an internal critic or advisor that helps the system improve its ability to catch. The mentor observes the system's performance and uses rules to determine how to correct inputs to the motor command network to improve catching success. Mentor output is used to train a network to emit the corrections, which are added to the motor command network inputs. The mentor is described further in sections 4.4.2 and 4.7.

Neural Networks. The system uses feedforward resource-allocating neural networks, with the virtual neuron adaptation algorithm and the distance allocation threshold set to zero, as described in section 3.5.4. The tuning width is set so the (Gaussian) response function of a newly-allocated neuron drops to one-half at a distance fraction, h, of 0.8. Resource-allocating networks are used because they offer rapid learning. There is no restriction on the number of neurons that can be allocated in each network.

For each ball there is a state prediction network and a crossing state prediction network. For each arm there is a state prediction network, a motor command

network, and a catch correction network. The catch correction network is created under the tutelage of the mentor as described in section 4.4.2. In addition, there is a ball recognition network that recognizes which ball is being observed, and an arm recognition network that recognizes which arm is being used. The recognition values are used to select the appropriate prediction and motor control networks. The individual networks are described more completely in section 4.4.2. Network inputs and outputs are summarized in tables 4.1 and 4.2.

Shift Register Memory. The shift register memory, which is cycled by the clock, holds the predicted and observed sensory traces, the motor commands, and the ball and arm identities. Information from the memory is used by the comparators and by the learning algorithms. It is also used as input to the networks that generate motor commands and behavior (state) predictions and recognize the arm and ball identities. Sensory memory is old because of perceptual time delay and shifting, but relative phasing is still valid because delay times are consistent. Delayed perception can thus be used for motor learning.

Vision. The vision system is monocular with a square retinal visual field that lies in the (vertical) motion plane. The optical axis is pointed in a fixed direction perpendicular to the motion plane, and passes through the center of the visual field. The vision projection is orthogonal and the visual scale is constant. The vision system is assumed to provide position and velocity information for objects of interest. The system has position and velocity (x,y) sensors that return analog values corresponding to the x and z positions and velocities of objects (centroids) in the visual field. The velocity sensors are linear and have signed output. It is assumed here that the vision and arm subsystems are calibrated, meaning that accurate feedforward position commands to visually-observed points can be generated.

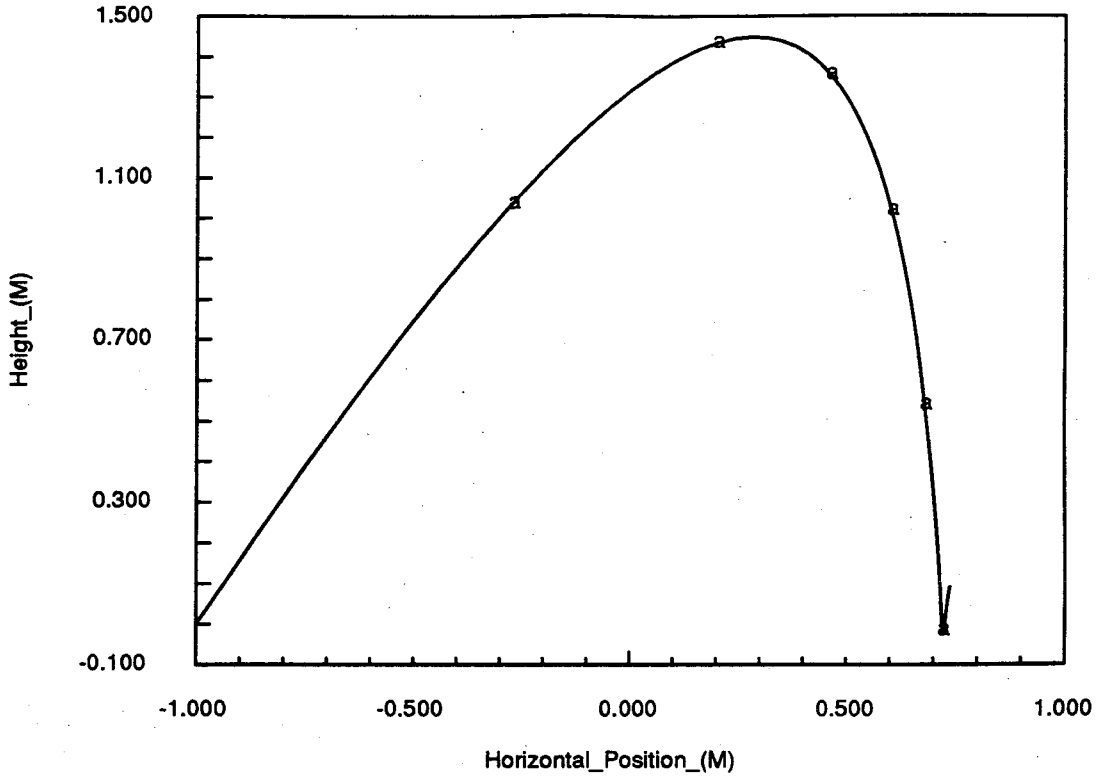

Figure 4.3: Typical light ball trajectory with a bounce.

4.4.2 System Operation

The system initially has no motor knowledge of itself or external objects. It has an arm learning mode, a ball learning mode, a catching mode, and a catch training mode, mediated by the mentor, in which catching performance is improved. The arm and ball learning modes are independent. Catching cannot be successfully performed or improved until the system is capable of controlling its arm and recognizing and predicting ball behavior. The modes are described in the following subsections.

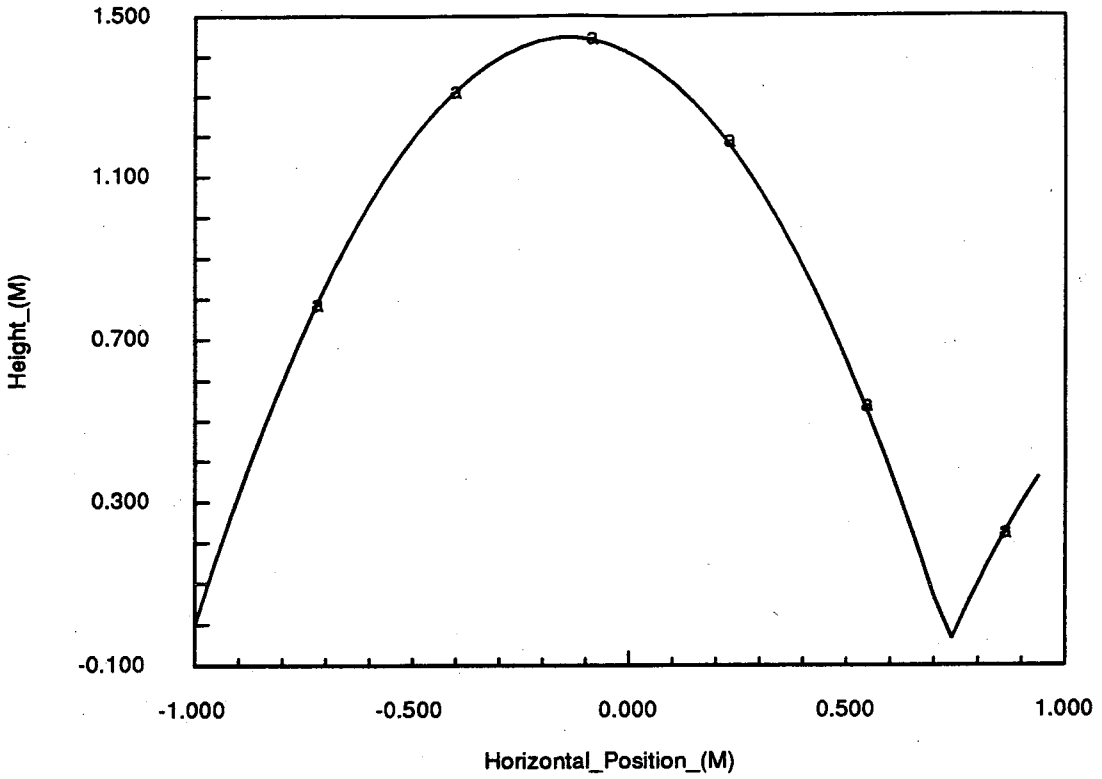

Figure 4.4: Typical heavy ball trajectory with a bounce.

Arm Learning

In the arm learning mode, either the light or heavy arm is selected and stereotypical pursuit movements, which are generated with the help of a random number generator, are executed by the arm. Each movement accelerates, decelerates, and stops the arm by issuing bang-bang voltage commands of fixed magnitude and a polarity that depends upon the direction of travel. The random number generator selects initial and target arm positions from within the arm travel limits along with a maximum voltage amplitude in the range $[\epsilon, 1]$, where $\epsilon > 0$ is set high enough to prevent glacially slow movements and provide good sensor excitation. The arm moves under the initial voltage until it crosses the midpoint between the initial and final positions, where the polarity reverses. When the arm stops, the voltage is set to zero. Because of damping, sampling, and nonlinear amplifier effects, the arm does not necessarily

stop at the position selected by the random number generator. This does not matter, since the system only needs to associate commands with observed behavior. During the movement, the behavior (succession of states) of the arm as sensed by the arm sensors, including the arm drive force sensor, is recorded in the shift register memory, along with the command inputs (voltages).

At the conclusion of the movement, the motor command network for the current arm is trained, for every time step in the movement, to emit the movement's command voltage, with the appropriate sign, in response to the arm state (velocity) at the time step, the displacement of the final arm position from the current arm state, and the time remaining from the current time until the arm comes to rest (its arrival time). In this way the system learns to command the correct voltages necessary to reach desired pursuit objectives (stop at a specified position at a specified time) from a given arm state. Arm state and target position information are stored in relative form[7] to eliminate the need to learn arm behavior separately for all positions in space. This is reasonable given the prismatic nature of the arm being used. It would not be acceptable for revolute arms. Arm network inputs and outputs are summarized in table 4.1.

The arm recognition and state prediction nets are also trained using information in the shift register memory. The current arm's state prediction net learns to predict the arm state and drive force at the next time step given the current arm state (velocity) and command voltage. The arm recognition net learns to associate the current arm identity with command voltages, observed arm behavior, and arm drive force sensor readings over recent time steps. It does this by setting an output line high. Thus observed arm behavior can be used to invoke the appropriate control and recognition nets when necessary. In operation, the output line for each arm will have

[7]Relative to the current arm position.

some activity, so the line with the highest activity is chosen, providing the activity of at least one is above a threshold.

Recognition in this context is not the same as system identification, although it has the same goal of accurate control. In the current work, we determine which arm is active so the appropriate networks can be used to generate control commands. In system identification [Ljung and Soderstrom (1987)], system response is observed and the parameters, such as masses, damping factors, etc., that describe the system (plant) according to a system model are determined. These parameters are then used by the control system to generate accurate control commands.

Ball Learning

In the ball learning mode, the light or heavy ball is selected, the control system is informed of the ball choice, ball trajectory parameters are selected, and the ball is tossed. Ball trajectory parameters, which are unknown to the control system, are randomly chosen so the ball will cross the plane $z = 0$ in the region specified in figure 4.1 and will have a maximum height between 1.25 and 1.5 meters as shown in the figure. Trajectory limits are imposed to limit problem complexity (hence neuron count) and to assure that all balls can be caught[8] by the arm using simple pursuit movements.

A trial begins when the ball is launched and ends when it crosses the plane $z = 0$.[9] During ball learning the arm is moved out of the way so no arm-ball collisions can

[8]In all cases it is physically possible for the arm (the center of the cup) to reach the position where the ball crosses the plane $z = r$, with r the ball radius, by the time the ball arrives. Because of the finite cup size and complex ball-cup collision dynamics, this is a sufficient, not a necessary, condition.

[9]The event of the ball's descending through the plane $z = 0$ is called *zero crossing* or just *crossing*. The ball's state at that time is called the crossing state.

Inputs	Outputs
Arm Recognition Net	
$V(t-\Delta t)$	enables arm id gate
$V(t-6\Delta t)$	
$\dot{a}_x(t)$	
$\dot{a}_x(t-5\Delta t)$	
$a_x(t)-a_x(t-\Delta t)$	
$a_x(t)-a_x(t-6\Delta t)$	
$f(t)$	
$f(t-6\Delta t)$	
Arm State Prediction Net	
$V(t)$	$a_x(t+\Delta t)-a_x(t)$
$\dot{a}_x(t)$	$\dot{a}_x(t+\Delta t)$
	$f(t+\Delta t)$
Motor Command Net	
$\dot{a}_x(t)$	$V(t+\Delta t)$
$a_{xs}-a_x(t)$	
t_s-t	
Catch Correction Net	
$\dot{a}_x(t)$	δx
$b_{xc}-a_x(t)$	δt
t_c-t	

Table 4.1: Arm Neural Networks. As discussed in the text, t is the current time, Δt is the clock time increment, a_x is the arm position, f is the drive force, V is the control voltage, a_{xs} and t_s are respectively the position and time the arm comes to rest, b_{xc} is the ball crossing location, t_c is the ball crossing time, δx is the catching position correction, and δt is the catching time correction. Inputs shown are used for both training and execution except as follows: During execution a_{xs} is replaced by $b_{xcp}(t)+\delta x(t)$, and t_s is replaced by $t_{cp}(t)+\delta t(t)$, where $b_{xcp}(t)$ is the current predicted ball crossing location and $t_{cp}(t)$ is the current predicted ball crossing time. For catching, the ball crossing location and time correspond to the arm stopping location and time, since simple catching involves the arm's being at rest at the crossing location when crossing occurs. The δx and δt values, which are learned with the help of the mentor, correct the position and time, and make the arm arrive 0.1 second before the ball.

occur.

During the trial, ball state variables, its horizontal and vertical velocities and positions, are stored in separate streams in the shift register memory. At the conclusion of the trial, the ball state variable memory trace is used to train the state prediction net and the crossing state prediction net for the current ball, as well as the ball recognition net. The state prediction net associates the ball's velocity and position displacement (from its current position) at the next time step with the (vector) ball velocity at the current time step and the ball's present distance from its position two time steps earlier. The state predictor, which, like the arm, uses a relative position representation, thus predicts future behavior based on past behavior. Ball network inputs and outputs are summarized in table 4.2.

The crossing state predictor associates the time remaining from the current step until ball crossing, the ball's horizontal velocity at crossing, and the horizontal distance from the current ball location to the crossing state with the (vector) ball velocity at the current time step; the ball's present height; and the ball's present distance from its position two time steps earlier. That is, the crossing state predictor net predicts, relative to the ball's current position and the current time, when and where the ball will cross the x-axis, and how fast it will be going.

The ball recognition net learns to identify the ball based upon its current velocity and its position displacement from its states one, three, and five time steps earlier. In the current implementation, it indicates recognition by setting an output line high. In operation, the output line for each ball will have some activity, so selection is made by choosing the line with the highest value (above a threshold), as is done for the arm.

Inputs	Outputs
Ball Recognition Net	
$\dot{b}_x(t)$	enables ball id gate
$\dot{b}_z(t)$	
$b_x(t) - b_x(t - \Delta t)$	
$b_z(t) - b_z(t - \Delta t)$	
$b_x(t) - b_x(t - 3\Delta t)$	
$b_z(t) - b_z(t - 3\Delta t)$	
$b_x(t) - b_x(t - 5\Delta t)$	
$b_z(t) - b_z(t - 5\Delta t)$	
Ball State Prediction Net	
$\dot{b}_x(t)$	$b_x(t + \Delta t) - b_x(t)$
$\dot{b}_z(t)$	$b_z(t + \Delta t) - b_z(t)$
$b_x(t) - b_x(t - 2\Delta t)$	$\dot{x}(t + \Delta t)$
$b_z(t) - b_z(t - 2\Delta t)$	$\dot{z}(t + \Delta t)$
Ball Crossing State Predictor Net	
$\dot{b}_x(t)$	$\dot{b}(t_c)$
$\dot{b}_z(t)$	$t_c - t$
$b_x(t) - b_x(t - 2\Delta t)$	$b_{xc} - b_x(t)$
$b_z(t) - b_z(t - 2\Delta t)$	
$b_z(t)$	

Table 4.2: Ball Neural Networks. In the table, t is the current time, Δt is the clock time increment, b_x and b_z are the x- and z-components of the ball position, b_{xc} is the x-component of the ball's crossing position, and t_c is the ball crossing time.

Catching

When the system is in catching mode, a trial begins when a light or heavy ball is randomly tossed and the light or heavy arm is enabled. The trial ends at ball zero crossing, or when the ball is caught or knocked out of bounds. Since the system is memory-based, it must know which arm and ball are in use so it can access the correct prediction and command networks.

Considering the balls for the moment, if the system knows, or believes it knows, the ball identity, it enables the associated state prediction nets. Using the net corresponding to the ball identity, the system begins predicting ball behavior. When enough data have accumulated in the shift register, the ball comparator begins comparing the predicted behavior with the observed behavior. If they are not sufficiently close, the system disables the prediction net, and enables the ball recognition net, which tries to identify the ball based upon its observed behavior. This can happen even in mid-trial. If ball recognition is successful, the appropriate state prediction net is enabled, the recognition net is disabled, and the ball comparator again begins comparing predicted and observed behavior. The ball recognition net continues to attempt recognition until it is successful or until the trial is over.

If the predicted and observed ball behavior are sufficiently close, the system sets a ball comprehension flag high and the motor command net begins to generate voltage commands based upon the arm state, the predicted ball zero-crossing state, and the output of the catch correction net, which will be described below. If the ball comprehension flag is not high, the system puts the arm into a damped safe state.

Now consider the arms. Whenever an arm is active, the system continuously predicts its behavior using the identity of the arm it believes is active. As the arm moves under the influence of the voltage commands generated by the motor command

net, the comparator will, as in the ball case, compare observed and predicted behavior, keeping an arm comprehension flag high and storing the comprehended arm identity in the shift register memory as long as the behaviors are sufficiently close.

If they are not close, the arm state prediction net is disabled and the system attempts to identify the arm using the recognition net. The arm, however is not disabled, since to do so would make it impossible to gather the data necessary for its recognition.[10] Instead, the arm is excited using voltage commands generated by the (apparently) incorrect motor command net. As in the ball case, the arm recognition net attempts to recognize the arm until it is successful or until the trial is over.

If the arm and ball are both recognized, the situation is considered to be comprehended and the system continues to generate and issue arm voltage commands until the trial is over, or until the situation is no longer comprehended. The voltage commands are designed to bring the arm to rest at the ball crossing point slightly before ball crossing. If this is done with sufficient accuracy, the system will catch the ball.

In addition to catching using single pursuits, which terminate with the arm at rest (and possibly missing the ball), there is a somewhat more adaptive mode in which, at the completion of one pursuit motion, the system is allowed to initiate another. Since the ball is closer to crossing and therefore its crossing state estimates should be better, this improves performance, often markedly so, in cases where the arm is not fully trained. This is not surprising, as the mode is a form of loop closure.

[10]Arms are recognized kinesthetically by exciting them and observing the response.

Catch Training

Once the system has learned to control its arm and predict ball behavior, it can catch balls with a high degree of accuracy and success, as will be seen below. The mentor improves catching performance by observing catching behavior, suggesting specific corrections, and training a network (the catch correction network) to emit the corrections appropriate for control situations. The corrections, which compensate for arm control and ball crossing state estimation errors, are added to the motor command network inputs.

In catch training trials, randomly-selected balls are thrown one at a time. The objective is to to catch or intercept each ball before it hits the ground. Allowable arm and ball initial conditions are such that it is possible for the arm to catch all the balls. At the completion of a trial, the mentor observes the catching performance, noting the improvement needed in both temporal and positioning performance by the arm in question.

Temporal performance improvement is needed if the arm is not yet at rest, or has been at rest too long at ball crossing. If the arm is not at rest at ball crossing, it is not exerting enough effort; if it has been at rest too long, it is exerting too much effort. Position performance improvement is needed if the arm rest position does not coincide with the ball crossing position, where the correct position is defined as having the ball centered in the cup at crossing.

The qualitative learning rule, or heuristic, is that the arm should arrive at the crossing location one-tenth of a second before the ball. In this implementation this heuristic is transformed into definite position and time adjustments δx and δt accord-

ing to the following rules:

$$\delta x = b_{xc} - a_{xs} \tag{4.5}$$

$$\delta t = t_c - t_s - 0.1 \tag{4.6}$$

where b_{xc} is the ball's horizontal position at crossing, a_{xs} is the arm's final position, t_c is the time of crossing, and t_s is the time the arm arrives at its rest position.

Since it is the *effective* arm pursuit behavior we wish to modify, for each time step the position and time adjustments are associated, in the catch correction net, with the same input information used by the arm control nets, namely the arm state (velocity) at the time step, the displacement of the final arm position from the current arm state, and the time remaining from the current time until the arm comes to rest. This is shown in figure 4.5.

Given this information, then, the catch correction net generates a target position correction and an arrival time correction. When the system is attempting to catch a ball, these increments are added to the arm target increment and arrival time increment inputs to the motor command net as shown in figure 4.6.

In this way the primitive behavior, the pursuit, becomes the basis for a skill, catching, without being itself modified. The catch correction net inputs and outputs are summarized in table 4.1. Performance is described is section 4.7.

In cases where the ball hits the arm and is therefore deflected from its normal crossing state, the mentor predicts the crossing state based upon the ball state immediately prior to the collision.

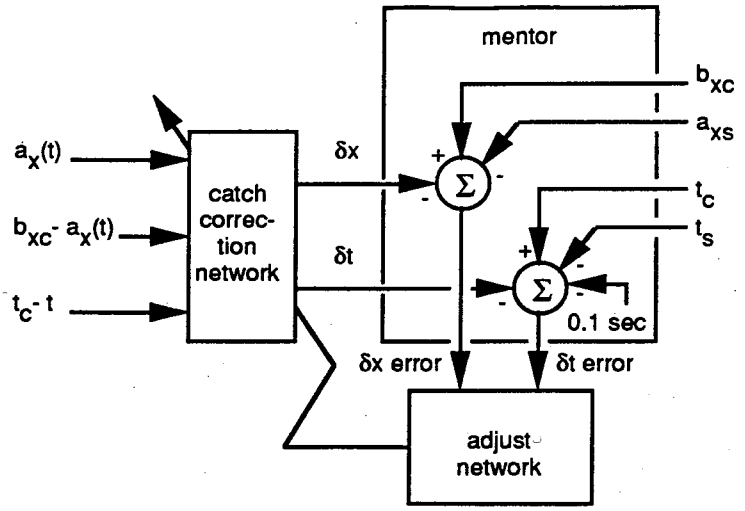

Figure 4.5: Schematic diagram of mentor operation. The mentor observes catching performance and trains the catch correction network to emit corrections δx and δt to the target catching location and arrival time. The catch correction network is trained for each time step in the trial using observed crossing information and arm state information as well as arm state information recorded in the shift register memory. In the diagram, b_{xc} is the ball crossing location, a_x is the arm location, a_{xs} is the arm stop position, t is the current time step, t_c is the crossing time, and t_s is the time the arm stops (the arrival time).

4.4.3 System Implementation

The system has been implemented as a "C" program. Simulations were run on Sun Sparcstations. For physical plausibility the implementation includes detailed models of the arm and ball behavior and their interactions. Recognition, control, and prediction neural networks were implemented in software. No attempt was made to simulate other systems, such as the shift register memory and the vision system, using neural elements.

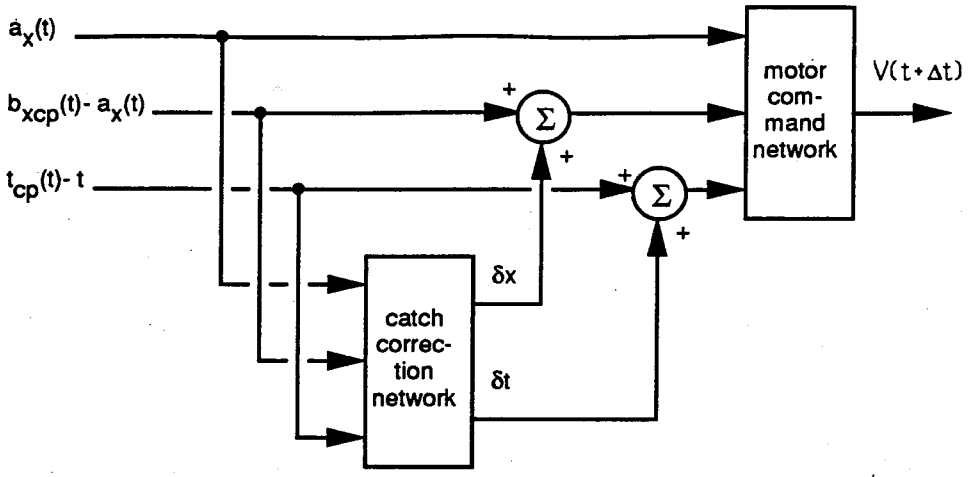

Figure 4.6: Catching control diagram. When the system attampts to catch a ball, the target position and arrival time corrections, δx and δt, generated by the catch correction network, are added to the motor command network inputs to improve catching performance. In the figure, a_x is the arm position, b_{xcp} and t_{cp} are respectively the predicted ball crossing location and crossing time as generated by the ball crossing state network, and $V(t + \Delta t)$ is the command voltage for the next time step.

4.5 Ball Behavior Recognition and Prediction Performance

This section describes the system's performance at learning to recognize and predict ball behavior.

Each ball was tossed two hundred times subject to the trajectory constraints illustrated in figure 4.1. After the two hundred trials the recognition network had allocated 2047 neurons; the light and heavy ball state prediction nets had allocated 3600 and 1736 neurons respectively; and the light and heavy ball crossing state prediction nets had respectively allocated 1242 and 3007 neurons. The system was able to learn to estimate ball crossing position and crossing times very accurately in both cases. Figure 4.7 shows the heavy ball crossing position prediction error for a typical ball flight as a function of time. The system cannot make accurate predictions until it

has gathered enough observations. When it has done so, accuracy increases markedly, decreasing after the ball bounces. Results for the light ball are similar. Figure 4.8 shows the crossing time prediction error for the light ball, again as a function of flight time. These are also very good. Crossing time prediction results for the heavy ball are similar.

Recognition performance for each ball was tested over twenty trials with the ball identity initially unknown to the system. In all cases, the trained nets were able to determine the correct ball identity. Furthermore, when recognition performance was tested over twenty trials in the context of well-trained catching as discussed in section 4.7, the system was able to determine the ball identity quickly enough to achieve perfect catching success with an rms position error of slightly more that 2 cm. This was also the case when the ball identity, instead of being unspecified, was initialized to be that of the other ball. From this we can infer that the comparators were working well, though their performance was not investigated in detail.

4.6 Arm Control, Behavior Prediction, and Recognition Performance

As described above, the system learns about the arm by exercising it, which is the same basic idea used in the hand-eye calibration system of chapter 3. During training, the arm is excited with antisymmetric square pulses as illustrated in figure 4.9. The response of the heavy arm to this voltage profile is shown in figure 4.10.

The heavy arm was trained over 1000 pursuit learning trials, while the light arm was trained over 500. In these trials the heavy and light arm motor command nets allocated 13245 and 3256 neurons respectively; the heavy and light arm state pre-

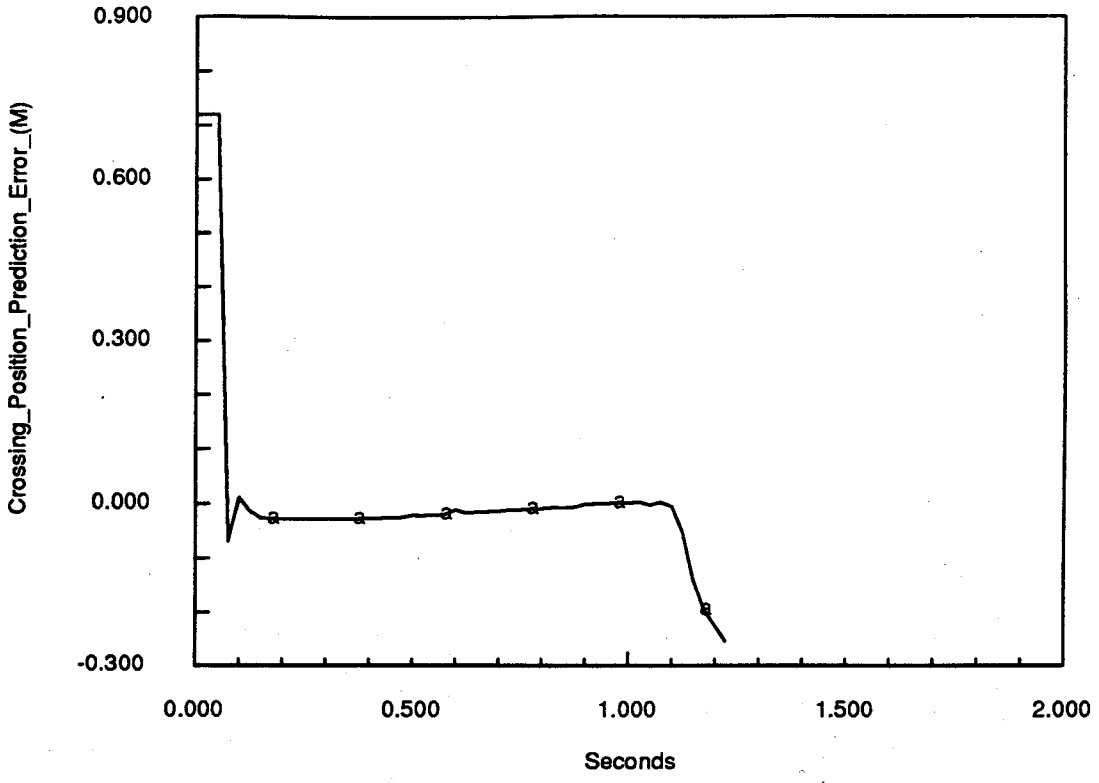

Figure 4.7: Crossing position prediction error as a function of flight time for the heavy ball. The decrease in accuracy beginning at about 1.1 seconds is due to software artifacts associated with the end of the trial and the system's latency in predicting ball behavior after the ball bounces.

diction nets allocated 8862 and 2186 neurons respectively; the arm recognition net allocated 1681 neurons. After 10, 100, 500, and 1000 training trials the heavy arm's performance was tested over a fixed set of twenty non-learning catching trials, in which the arm position error (the difference between the arm and target positions) at the instant of ball crossing was noted. The rms position error over these testing trials is shown in figure 4.11 as a function of the number of training trials. The arm clearly gets better at performing pursuit movements as the number of trials increases. The ball catching probability over the twenty testing trials as a function of the number of training trials is shown in figure 4.14. The catching success is excellent.

The voltage command profiles and the ball trajectory relative to the arm for

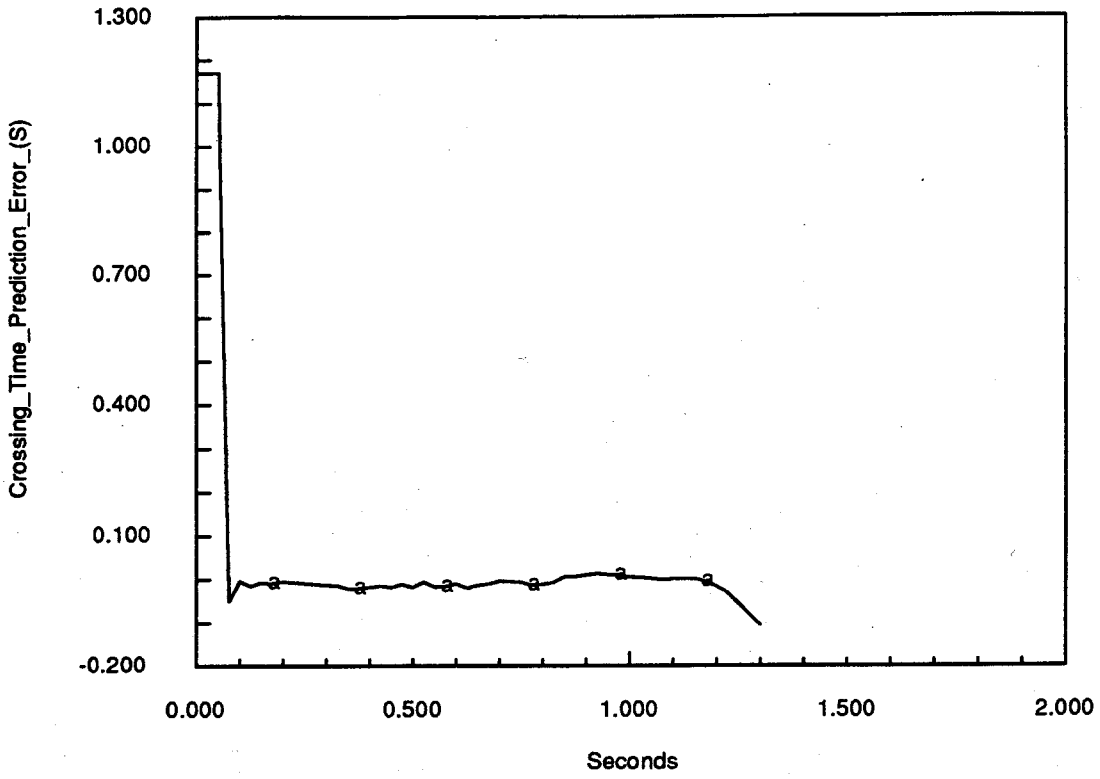

Figure 4.8: Crossing time prediction error as a function of flight time for the light ball. The decrease in accuracy beginning at about 1.2 seconds is due to software artifacts associated with the end of the trial and the system's latency in predicting ball behavior after the ball bounces.

identical initial conditions are shown in figures 4.12 and 4.13 after 10, 100, 500, and 1000 training trials. The voltage commands are spiky, but they are clearly becoming more controlled with increasing training.

In addition to the pursuit and catching trials, the system's arm recognition capabilities were tested by initializing the system with the incorrect arm identity. It was not set to *unknown* identity because, as described above, the system must excite the arm in order to identify it. For both the light and the heavy arm the system identified the correct arm in each of the twenty testing trials. In the case of initializing the heavy-arm system with the the identity of the light arm, the only case tested, the system was able to identify the correct arm and recover quickly enough to achieve

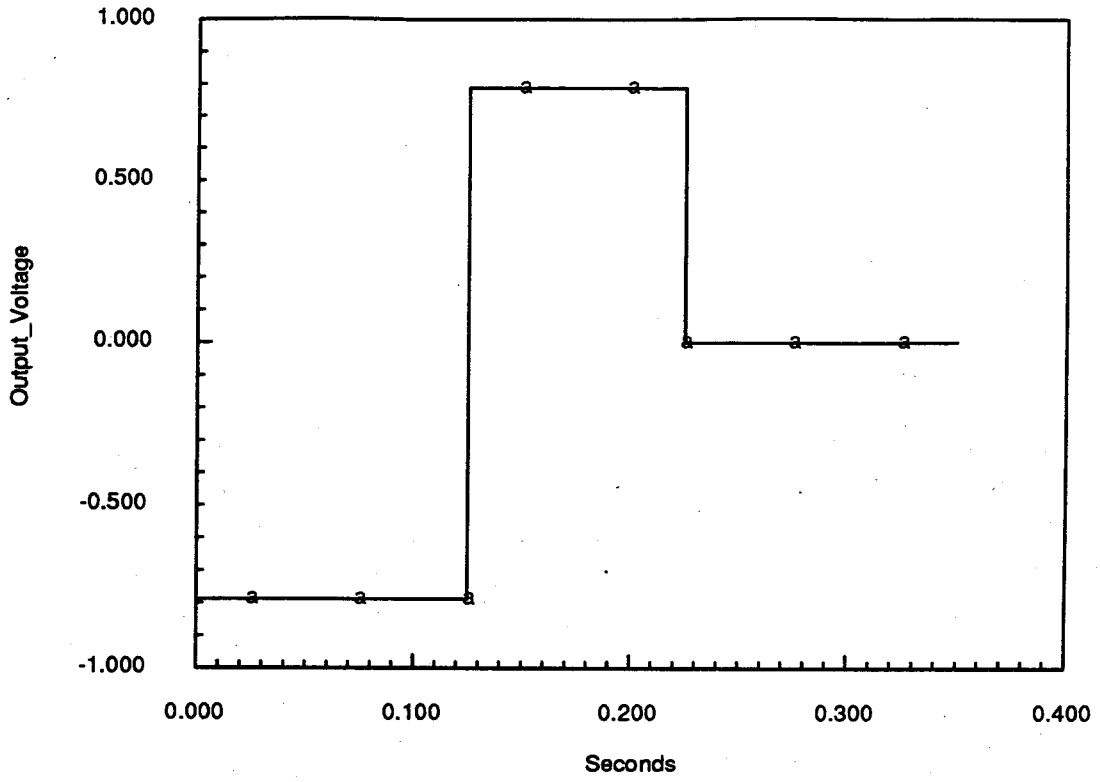

Figure 4.9: Pursuit training command voltage profile.

perfect catching success[11] over the twenty testing trials with an rms position error of slightly less than 4 cm.

4.7 Improving Catching Success with Help From the Mentor

At this point the system knows how to identify and predict ball and arm behavior, and how to command pursuit movements that will cause the arm (the cup) to reach a desired position at a specified time. If a pursuit motion is invoked using the predicted

[11]The system had been trained by the mentor as described in section 4.7.

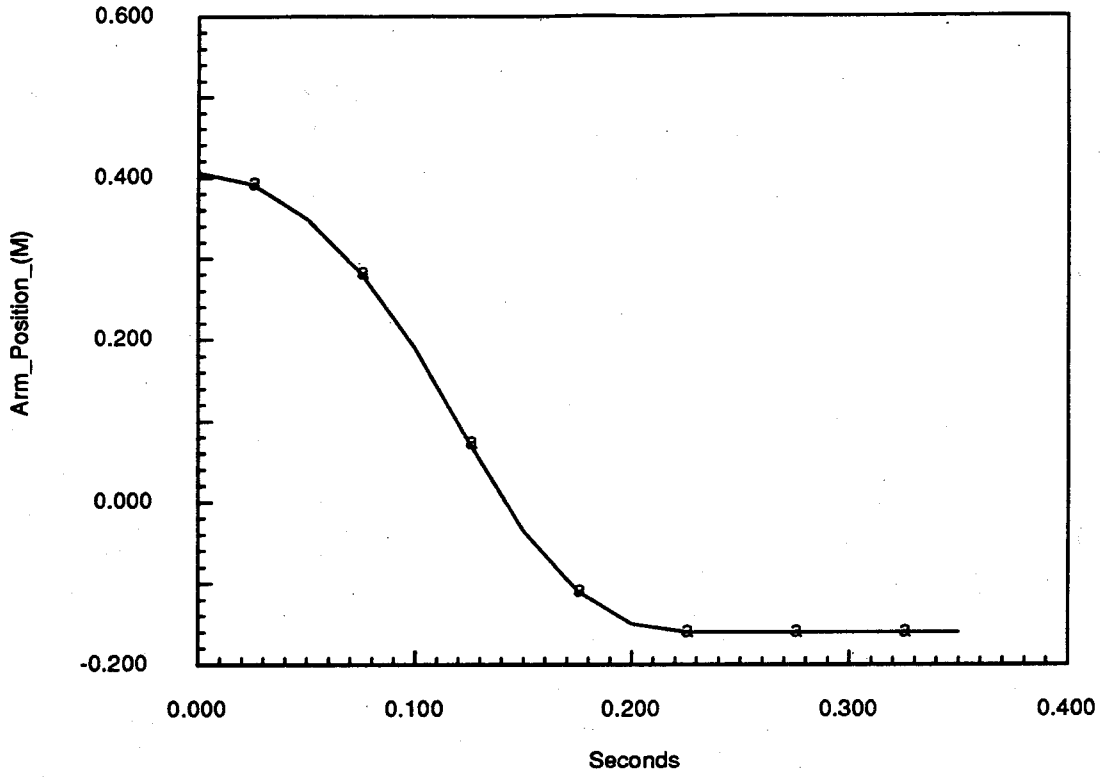

Figure 4.10: Response of heavy arm to training voltage profile.

ball crossing time and state, the system will attempt to get the hand to the ball crossing location at the time of ball crossing. The system is capable of excellent catching performance using pursuits by themselves. As discussed in section 4.4.2, however, its performance can be improved if it invokes pursuit motions that have slightly modified arrival times and, possibly, target positions. The mentor helps the system learn to make the necessary modifications.

For the high balls studied here, which do not require synchronizing the arm with the ball motion at crossing, it is sensible to arrive early, which gives time for some position adjustment if that is necessary. The mentor has a simple learning rule: Select a pursuit motion that will arrive 0.1 second early at the correct crossing location. If the arm ends up at the wrong location, adjust the target position increment generated by the catch correction net by the appropriate amount. Do the same for the time

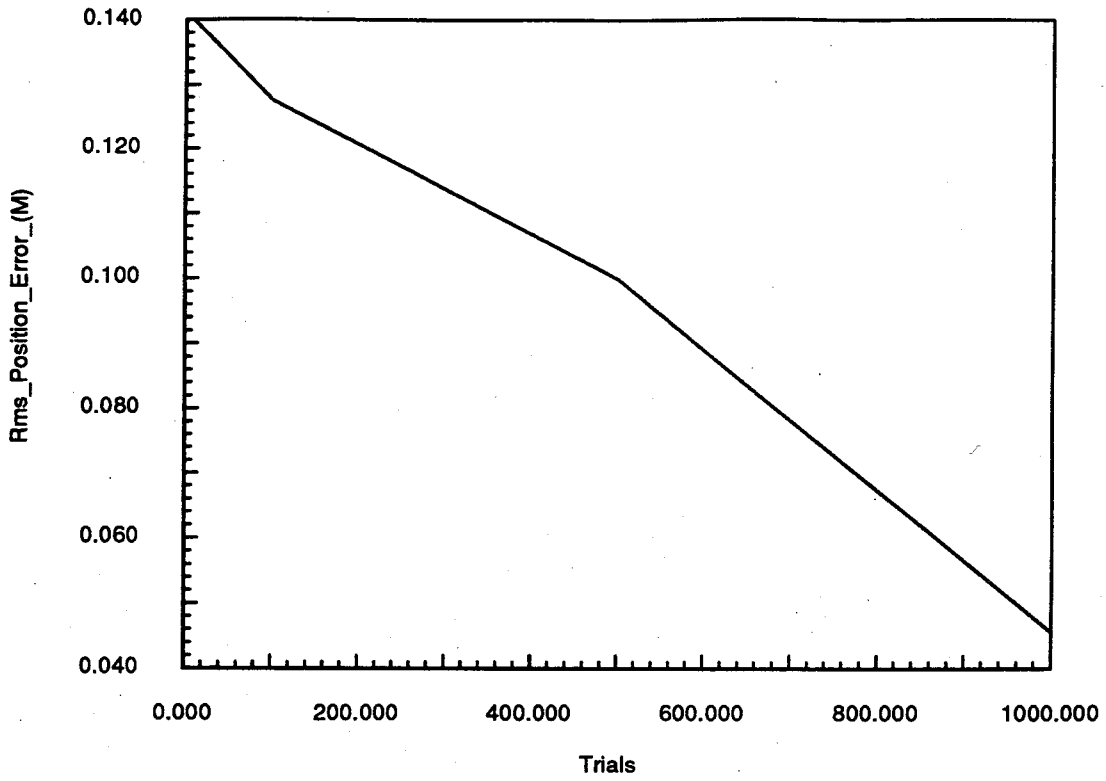

Figure 4.11: Rms pursuit position error over twenty trials as a function of the number of training trials.

increment generated by the catch correction net if arm arrival time is too late or too early.

The system, using the heavy arm and heavy ball, was trained to modify pursuit arrival times and targets with the help of the mentor for 100 trials. At the conclusion of the training trials, catching success, crossing error, and arrival time were evaluated using the twenty-trial testing sequence described in section 4.6. In these trials the rms position error was reduced from slightly more than 4.5 cm with a catching probability of 0.95 to an rms positioning error of slightly less than 2.9 cm with a catching probability of 1.0.[12] The catch correction net allocated 2952 neurons. Figures 4.15 and 4.16 show time plots of system performance with pursuit-only and mentor-trained

[12]These values are those reported above for pursuit-only catching after 1000 arm training trials.

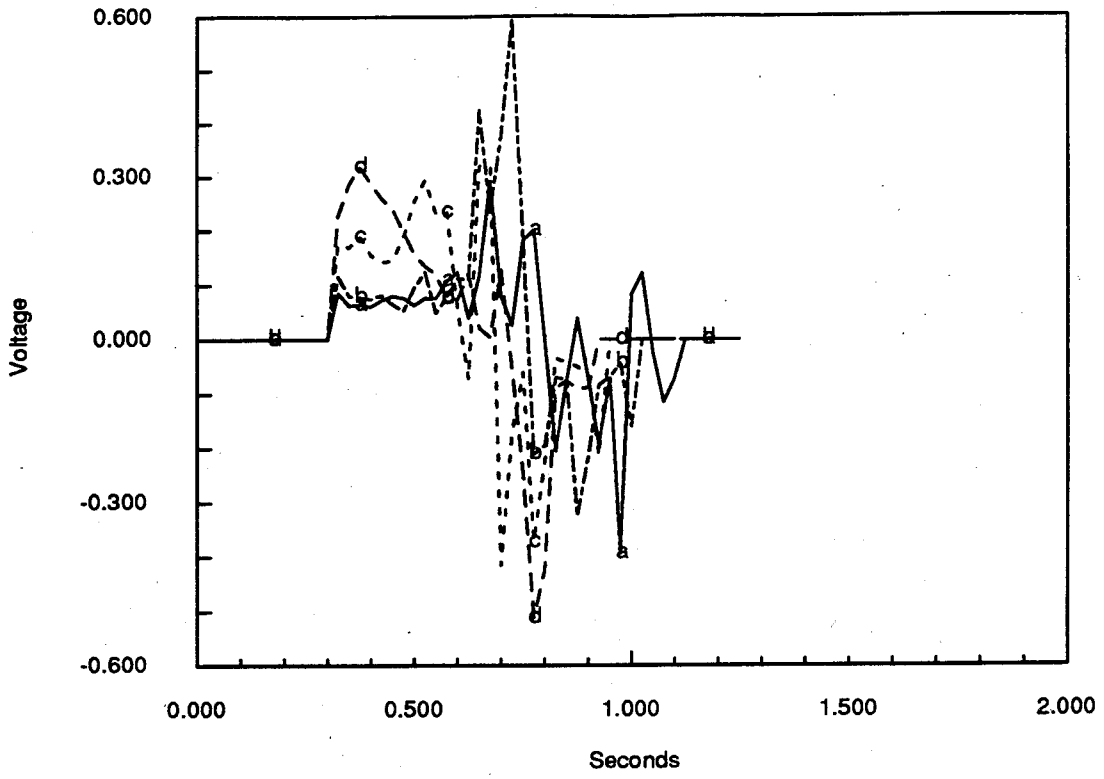

Figure 4.12: Arm output voltage profiles for identical ball initial conditions. (a), (b), (c), (d) are after 1000, 500, 100, and 10 training trials, respectively.

catching respectively. These figures clearly show that with mentor training, the arm arrives earlier, by about a tenth of a second, as it was coached to do. Terminal control and precision are also better. Similar results were obtained with the light arm.

4.8 Ball Catching Using a Conventional Linear Controller

As we have seen, the learning control system can acquire excellent catching skill. The learning control system is complex, however, and requires training, so from an engineering standpoint it is useful to compare learning controller performance with

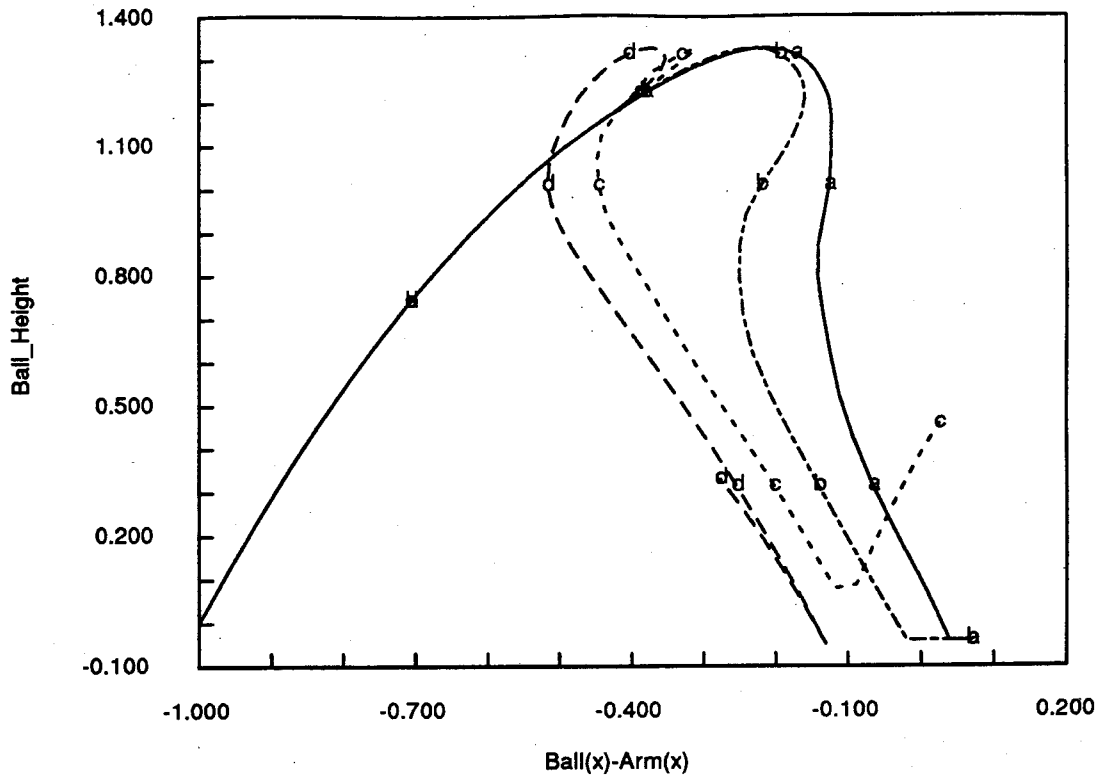

Figure 4.13: Ball trajectories relative to the heavy arm. (a), (b), (c), (d) are after 1000, 500, 100, and 10 training trials, respectively. (c) and (d) show the ball bouncing off the cup. (a) and (b) show it sliding along the bottom.

that of conventional closed-loop control, which can be implemented relatively simply.

4.8.1 Closed-Loop Control System Design

As with learning control, the control objective is to get the cup to the crossing point in time to catch the ball. In the limiting case of zero drag, the ball's horizontal motion is a ramp. For macroscopic drag the ball exhibits damped motion[13] that approaches a finite horizontal position limit C/a. The ball crossing point by itself is a horizontal step for any value of drag.

[13]Damped ball motion is of the form $b(t) = (C/a)(1 - e^{-at})$, with C and a constants, $a > 0$, and t time. The Laplace transform is $C(1/s - 1/(s + a))$.

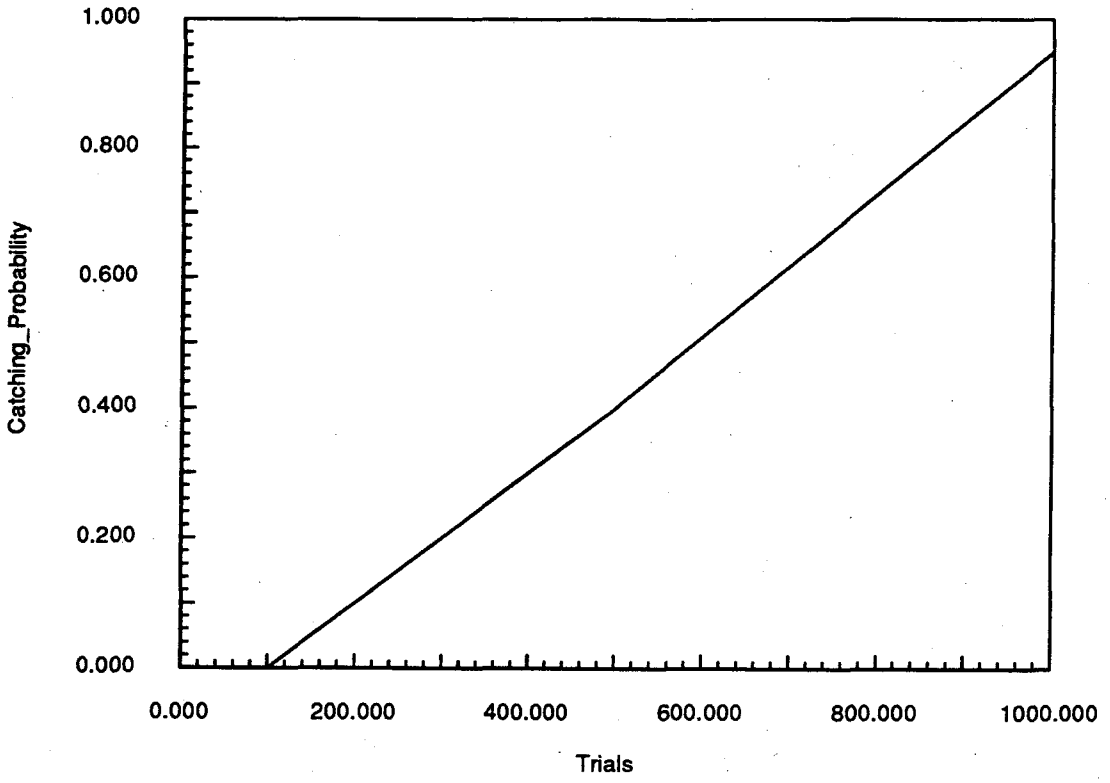

Figure 4.14: Heavy arm ball catching probability using pursuit movements as a function of the number of training trials. Probability is calculated over twenty trials.

If the system can asymptotically track ramps, it can necessarily asymptotically track damped ball motion and steps, so if transient tracking errors are small, it should be able to catch lightly or heavily damped balls using only the horizontal ball position as a command reference input. If ball crossing position estimates are available, the system only needs adequate step response.

A PID controller with unit feedback [Franklin, Powell and Emami-Naeini (1986)] was chosen for the closed-loop system. Such a controller has the form:

$$c(s) = \kappa_p(1 + T_d s + 1/T_i s). \tag{4.7}$$

Here κ_p, T_d, and T_i are the proportional gain, the derivative time, and integral time, respectively. The system block diagram is shown in figure 4.17. The command

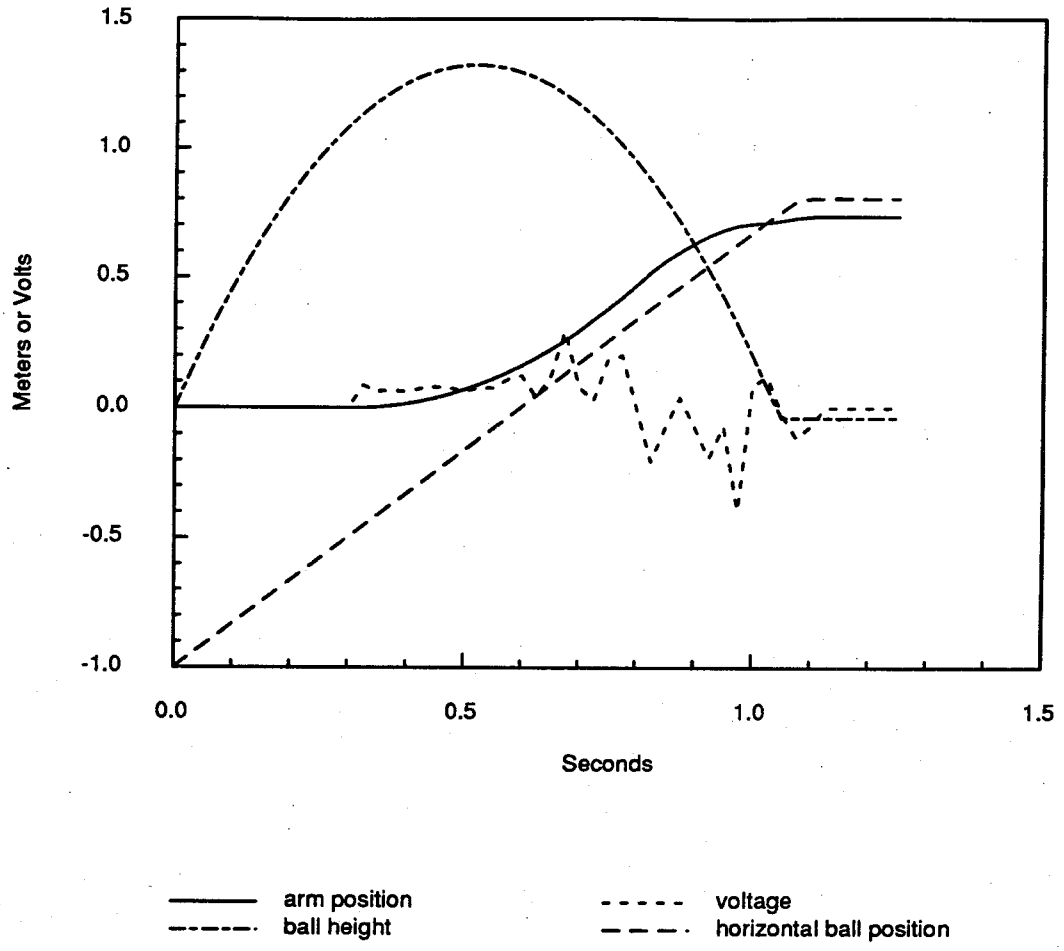

Figure 4.15: Heavy arm temporal response without coaching after 1000 pursuit learning trials.

reference input, b_x, is either the horizontal ball position or an estimate of its crossing location; the controlled output is the arm position, a_x.

The controller was parameterized using the Internal Model Control (IMC) design approach developed by Morari and co-workers [Morari (1989)]. In our case IMC specifies control parameters in terms of a single adjustable parameter, λ, which is essentially the system time constant. A single-parameter controller was selected because it is simpler to tune and is itself a good candidate for a learning control system. In that case, the learning control system would learn to generate values of λ appropriate for particular control situations. Different values of λ are necessary because a

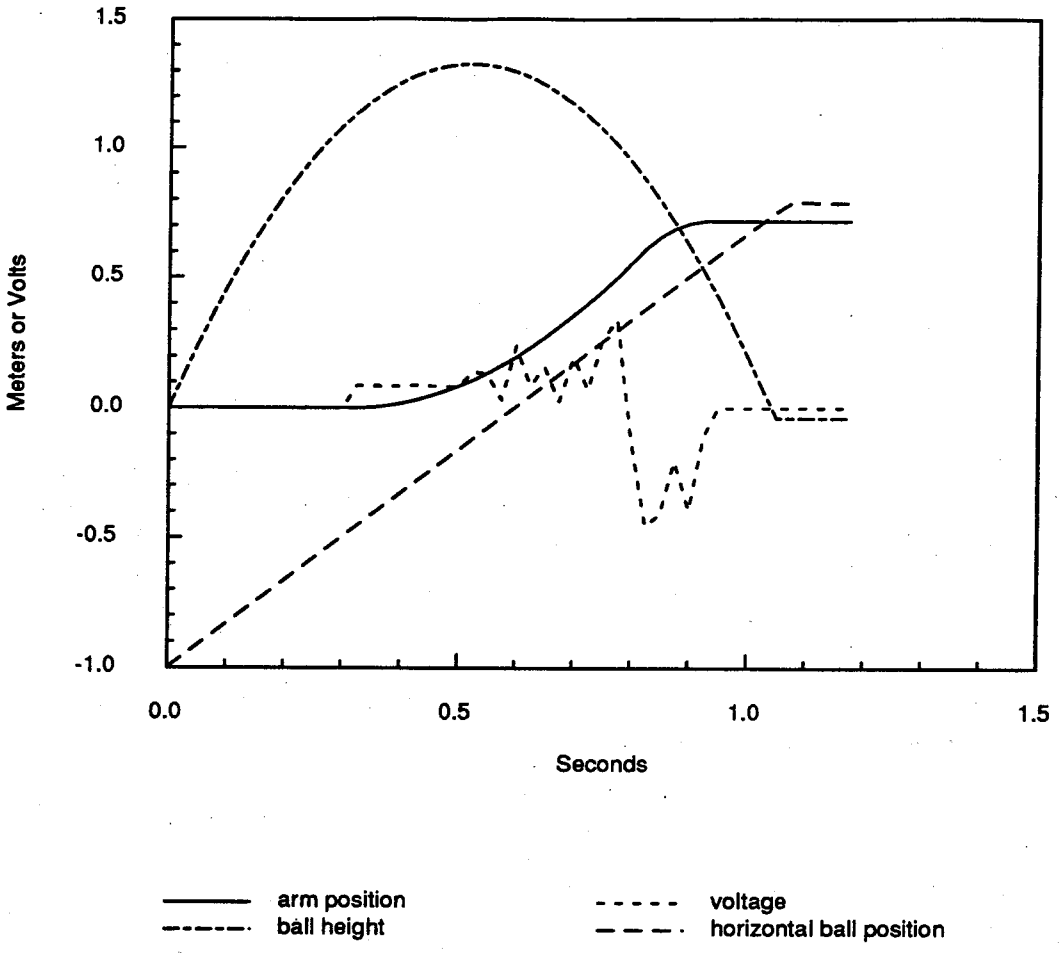

Figure 4.16: Heavy arm temporal response after 1000 pursuit learning trials and 100 coaching trials.

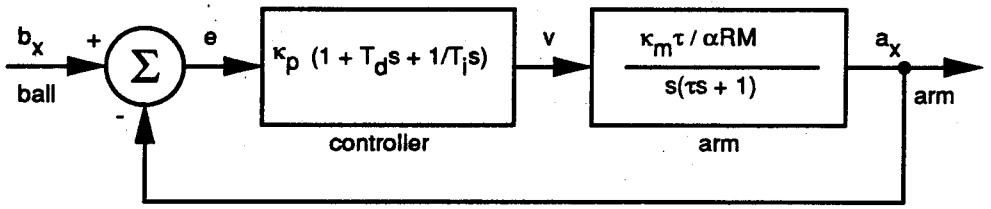

Figure 4.17: Block diagram of closed-loop ball catching system using conventional PID controller.

fixed value is not optimum for all cases as will be seen below.

For ramp inputs and the given arm,[14] the IMC controller parameters are given by:

$$\kappa_p = \frac{2\lambda+\tau}{\lambda^2}\frac{\alpha RM}{\kappa_m \tau} \tag{4.8}$$

$$T_i = 2\lambda+\tau \tag{4.9}$$

$$T_d = \frac{2\lambda\tau}{2\lambda+\tau}. \tag{4.10}$$

The arm time constant τ is specified in eq. 4.2. The controller is then given by:

$$c(s) = \frac{\alpha RM}{\lambda^2 \kappa_m \tau}(2\lambda+\tau+2\lambda\tau s+\frac{1}{s}), \tag{4.11}$$

and the corresponding closed-loop arm response is:

$$\hat{a}_x(s) = \hat{b}_x(s)\frac{2\lambda+\tau+2\lambda\tau s+1/s}{\lambda^2(s(\tau s+1))+2\lambda+\tau+2\lambda\tau s+1/s}, \tag{4.12}$$

where, again, $\hat{b}_x(s)$ is the horizontal ball position, $\hat{a}_x(s)$ is the arm position, and λ is the adjustable time constant. The error is given by:

$$\hat{e}(s) = \hat{b}_x - \hat{a}_x = \hat{b}_x\frac{\lambda^2 s(\tau s+1)}{\lambda^2 s(\tau s+1)+2\lambda+\tau+2\lambda\tau s+1/s}. \tag{4.13}$$

Applying the final-value theorem [Franklin, Powell and Emami-Naeini (1986)]:

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} s\hat{f}(s), \tag{4.14}$$

it is easy to verify that the system asymptotically tracks ramps, steps, and damped motion with zero error.

The system with the parameterized controller is shown in figure 4.18.

[14]The amplifier was assumed linear for the conventional controller tests.

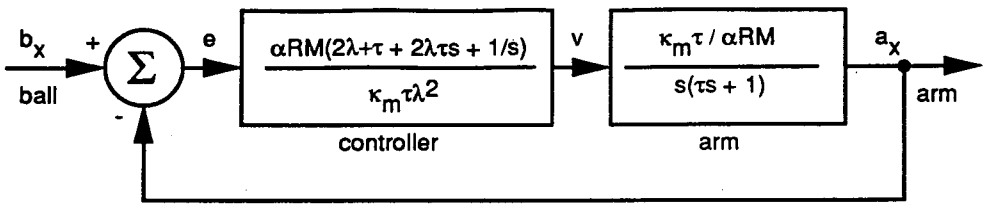

Figure 4.18: Block diagram of closed-loop ball catching system showing controller parameterized using the Internal Model Control design formalism.

4.8.2 System Implementation

The closed-loop system with the heavy arm was simulated in the time domain using Euler's method [Ralston and Rabinowitz (1978)]. It was embedded in the testing and control structure used for the learning control system, and used the same 40 Hz clock rate.

4.8.3 Closed-Loop Catching Performance

Catching performance was tested in both ball-tracking and crossing position modes using the same sequence of twenty testing trials used in section 4.6 for the learning controller. In ball tracking mode the command reference input is the ball position; in crossing position mode it is the predicted horizontal ball crossing position. For simplicity in the controller tests the ball crossing position was calculated by the ball-tossing subroutine rather than using an estimator.

In the ball-tracking mode, with $\lambda = 0.6$ second, the closed-loop system had perfect ball catching success with the heavy ball, along with an rms catching position error of slightly more than 4 cm and an average position error of -0.2 cm. These errors indicate that the arm sometimes led and sometimes lagged the ball at crossing. With $\lambda = 0.6$ second, however, the system was too sluggish for the light ball and had no

catching success. The average and rms catching position errors were nearly equal at 0.51 meters, indicating that the arm severely lagged the ball at crossing. Decreasing λ increased catching probability for the light ball, but decreased catching probability for the heavy ball. Decreasing λ increases ramp overshoot. This causes the arm to be ahead of the heavy ball position at crossing.

With $\lambda = 0.1$ second, the system was unable to catch the heavy ball, but was able to catch the light ball with perfect success and an rms position error of only 1.5 cm. The average position error in this case was -1.4 cm, indicating that the arm consistently, but only slightly, lagged the ball at crossing.

With $\lambda = 0.018$ second, the best case for both the light and heavy balls together, the system had perfect catching success with the light ball and a catching probability of 0.7 for the heavy ball. The corresponding rms catching position errors for the light and heavy balls were 5.8 cm and 7.7 cm, respectively.

In the crossing position mode, with $\lambda = .018$ second, the system was able to catch both the light and heavy balls with perfect success. This is a simpler control situation that just requires good step response. The system does, however, require a means for estimating the crossing position.

4.9 Catching Performance Discussion

The learning control system outperforms the closed-loop system, being able to achieve perfect catching success for both the light and heavy balls simultaneously. This performance, however, comes at the cost of significantly greater complexity and the need for training. The closed-loop system is simpler and performs reasonably well,

but requires tuning. In addition, the crossing position mode requires implementing a capability for predicting the crossing position.

Since the closed-loop system is capable of perfect catching success for both balls if the value of λ appropriate for the particular ball is used, its performance can obviously be improved by selecting λ according to the observed arm and ball behavior. This is just gain scheduling. Gain scheduling, of course, requires tuning to determine the gains and requires a scheduling algorithm. Closed-loop controller performance could also be improved by independently selecting κ_p, T_i, and T_d. Again, tuning would be required, and tuning PID controllers is difficult [Morari (1989)]. Finally, closed-loop system performance could be improved by increasing the clock rate, but so could that of the learning controller.

A self-tuning controller [Goodwin and Sin (1984)] could be employed. This would be similar in spirit to the learning controller, except that the learning controller requires no initial model of the arm or the ball. Furthermore, self-tuning controllers are significantly more complex than simple linear controllers.

We can conclude that achieving good ball-catching performance with variable or unknown plants requires a relatively complex control system, whether the control system employs gain scheduling, is adaptive in the traditional sense, or is a learning control system based on neural networks. The learning control system described in this work needs no initial model, but needs specialized computing resources (hardware neural networks). Traditional adaptive control systems require an initial analytical model, but can use conventional computational resources and may adapt more quickly. The choice will depend upon the application.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

5.1.1 Hand-Eye Calibration Conclusions

1. Resource-allocating neural networks coupled with virtual neuron adaptation have credible performance in hand-eye calibration that is far better than that of conventional multilayer back-error propagation networks. Experiments with multilayer back-error propagation networks gave unsatisfactory results and were not pursued. They required many tens of thousands of training trials to stabilize and achieved only poor performance.

2. Exploiting engineering knowledge and learning a hand-eye map correction is a practicable approach that has much better performance than learning the entire hand-eye map as has been done in previous work. The network learns

to approximate the map rather than identifying parameters, hence does not require a detailed parametric model. Many neurons, however, are required for complex maps and networks have many parameters that must be adjusted to get the best performance.

3. The choice of network basis functions is not too critical. Gaussian and non-Gaussian neurons perform about equally well.

4. The neural system reported here is able to learn the global hand-eye mapping only by sampling the entire space. Modifying the approach so key parameters, such as scale factors and offsets, are identified explicitly and only unmodeled perturbations involve sampling the entire space would make it possible to track systematic plant parameter drifts with just a few observations. Assuming that the unmodeled perturbations change slowly, this would significantly improve system performance.

5. Noise decisively limits network and estimator performance. This has not received much attention in previous neural network research in robotics. Noise sets a limit to attainable accuracy, and can cause useless neuron allocation.

6. Both the estimation- and neural network-based approaches can develop internally consistent hand-eye maps even when kinematic parameters are in error because they minimize an energy function. Obtaining accurate metrical information requires special procedures like explicitly forcing the system to learn distance scales.

7. Real arms have much greater input complexity than the model explored here, and consequently will need far more neurons or much more complex estimator models.

8. Within the limits of its model, the estimator-based system significantly outperforms the network-based system. If nonlinearities and noise are small, esti-

mation may be the best approach unless fast neural hardware is available. If nonlinearities and noise are significant, the need for a complex model may limit its attractiveness. In principle, the estimator can respond to plant changes well and quickly, but this needs to be demonstrated.

5.1.2 Motor Learning Conclusions

1. Learning controllers based on neural networks can, with a reasonable amount of training, learn extremely effective feedforward motor control. In these simulations, learning controllers, which required no initial plan models, outperformed the conventional controllers in terms of positioning accuracy, temporal response, and ball catching.

2. Learning feedforward controllers outperform fixed-gain closed-loop controllers when there is significant plant variation because they can switch control generation nets. A direct comparison, however, is complicated by the fact that the neural system is inherently adaptive, while the closed-loop system used in this work is not. Closed-loop controllers, of course, can be made to be adaptive as well, but at the cost of significant effort.

3. In this work, the antisymmetric motor drive profiles were used to train the motor command networks to emit the appropriate voltage at each time step. This resulted in spiky voltage profiles. A better approach would be to train the motor command networks to enable the antisymmetric motor drive profile generator, which would, in turn, generate the command voltage at each time step. This would smooth voltage profiles. The profile generator exists since it is used in arm training.

4. Resource-allocating neural networks are very effective in this situation since

they essentially provide real time learning with generalization. They do this at the expense of an efficient representation, however. Once allocation networks are trained, they might be used by a background process to train networks using more efficient representations. It is interesting to speculate on potential parallels with short- and long- term memory in biological systems. The robot, which was initially unable even to move its arm in response to tossed balls, learned nearly perfect catching success in only 1000 training trials. After another 100 training trials with the Mentor, its catching success was perfect and it exhibited direct, precise arm motion.

5. Monitoring progress by comparing predicted and observed behavior is a good approach to assuring robust task execution and recognizing when the control regime has changed.

6. Training a neural network to emit the plant identity in response to samples of the time course of plant inputs and responses is an effective way to perform implicit plant identification. If the learning loops are always active, inputs and responses will stay correlated with each other and with the plant identity even if there are gradual sensor and plant drifts.

7. The use of an internal or external critic (the mentor in this work) can significantly improve system performance. It allows skills to be constructed by invoking the appropriate primitive motions. The primitive motions remain unchanged and are therefore available for other tasks.

8. The shift register memory is an effective way to deal with real- time information that must be processed in a pipeline fashion. It handily solves the time stamping problem, and can be extended to deal with significant processing delays.

Based on this work, it appears that learning control systems are extremely promising. Once the basic architectural structures and hardware are in place, they may be

able to provide excellent, adaptive, control capabilities in a wide variety of situations without the need for laborious control system tuning or the need for developing detailed plant models.

5.2 Future Work

As usually happens in research, this work raised a host of questions. I would like to extend the work in the following directions with the dual aims of understanding how to engineer more effective robotic hand-eye systems and understanding the way hand-eye systems are organized in animals.

5.2.1 Future Work in Hand-Eye Calibration

1. Extend nonlinear estimation to handle noise and modeling errors that result in large residuals. This can be dealt with by acquiring and implementing more sophisticated nonlinear estimation software.

2. Extend nonlinear estimator model complexity to include visual parameters as well as basic distortions such as actuator scale factors.

3. Extend both neural and estimation approaches to tool frames and arbitrary points on arm links as well as redundant arms.

4. Implement learning algorithms in analog VLSI hardware.

5. Investigate dedicated adaptation architectures and internal spatial representations for improving system adaptation accuracy and response.

6. Investigate the role of contact and other non-visual information in developing hand-eye maps and internal spatial representations and a sense of spatial scale.

7. Apply both the neural and estimation approaches to real robot arms including multiple arm systems.

8. Investigate the use of contact information to estimate the location of the tool frame.

9. Investigate the formation and maintenance of hand-eye maps in human subjects using virtual reality and telerobot technology. By allowing the subject to view only computer-generated representations of his own limbs and to feel only computer-generated tactile stimuli, visual input and tactile cues could be distorted and delayed in time. This should make it possible to introduce artificial hand-eye errors and to distort causality, thereby helping to identify mechanisms for the acquisition and maintenance of hand-eye maps.

5.2.2 Future Work in Motor Learning

The robot system and behaviors investigated here are actually extremely idealized, and are not yet suitable for real robots. The following work is proposed with the aim of creating robot systems that are more like animals in their grace and adaptability.

1. Extend the motor learning results to realistic arms, especially to arms that have variable physical stiffnesses, and to robots that have multiple arms and a richer sensor suite. Learn to set stiffnesses for grasps and contact events.

2. Investigate neural representations that are biologically more realistic, such as using a series of active lines to represent values (place coding) and local populations to represent drive intensity. The learning control system considered

here used analog values, which is probably not realistic, especially for visual information.

3. Learn to deal with continuous rather than discrete variations in ball and arm masses.

4. Learn to anticipate and recognize events such as catching and collisions.

5. Learn to recognize and follow external agents that are more complex than balls, and investigate more complex control tasks, such as learning to catch a ball in a cup that is at the top of an inverted pendulum. This means the system must learn to schedule reaction forces to avoid being knocked over.

6. Extend the repertoire of available control modes and apply learning control to realistic sensor-motor tasks, such as assembly, that are composed of many primitive elements and involve many smooth transitions. Learn to recognize failure precursors.

7. Investigate structures that will allow the system to learn whether a particular control task is beyond its competence, e.g., that the ball is moving too fast or will impact too far away to be caught.

8. Investigate hierarchical task representations so large tasks can be composed of smaller ones and so low-level skill improvements are manifested through overall motor behavior, even for previously-learned tasks.

9. Consider noise and longer processing delays. The current delay is one clock tick, but complex processing may require longer delays.

10. Use the behavior prediction nets to estimate future states even though the elements in question may not be under observation. A possible approach is to gate net output back into the input. Errors would accumulate, but estimates might be reasonable.

11. Learn to regulate playback speed in response to temporally-distorted input.

Appendix A

Neural Networks

A.1 Introduction

This appendix introduces biological and artificial neural networks. It gives some basic biological details, briefly describes recurrent networks, and outlines the aspects of feedforward networks and learning algorithms that are used in this thesis. It closes with a section that relates feedforward neural networks to familiar mathematical ideas.

A.2 Biological Neural Networks

Biological nervous systems, which are composed of networks of interconnected cells called neurons, are remarkably complex computational mechanisms.[1] They are the primary reason that animals, at least multicellular animals, can sense and respond to their environments, learn, find food, maintain postural stability, find mates, and so on. Far from being unstructured, as in the old *tabula rasa* models, nervous systems are highly organized, with specialized anatomical and organizational features and spatial regions devoted to particular types of computations [Shepherd (1979), Brooks (1986), Kent (1981)].

Signals in biological neural networks are encoded using both place and value codes [Kent (1981)]. In place codes, the particular line that carries the signal is meaningful. In value codes, it is the intensity of the signal on a line that is meaningful. An example of a place code is activating the output line from a particular photoreceptor if the photoreceptor is being illuminated. The intensity of the signal on the output line is a value code. Clearly both can be combined. If there are many photoreceptors, examining the lines to determine which is the most active will determine which photoreceptor is being illuminated with the greatest light intensity. Place and value codes are very important in biology, and there are many such mappings. The retina, for example, is mapped in a topology-preserving manner onto the visual cortex using place coding. Similarly, spatial derivatives can be estimated by subtracting the value codes on place-coded lines that originate in a sensory area. See figure A.3. The extensive use of place codes provides for the massive computational parallelism of biological systems.

[1] Computational in the sense that inputs are transformed into outputs through some process.

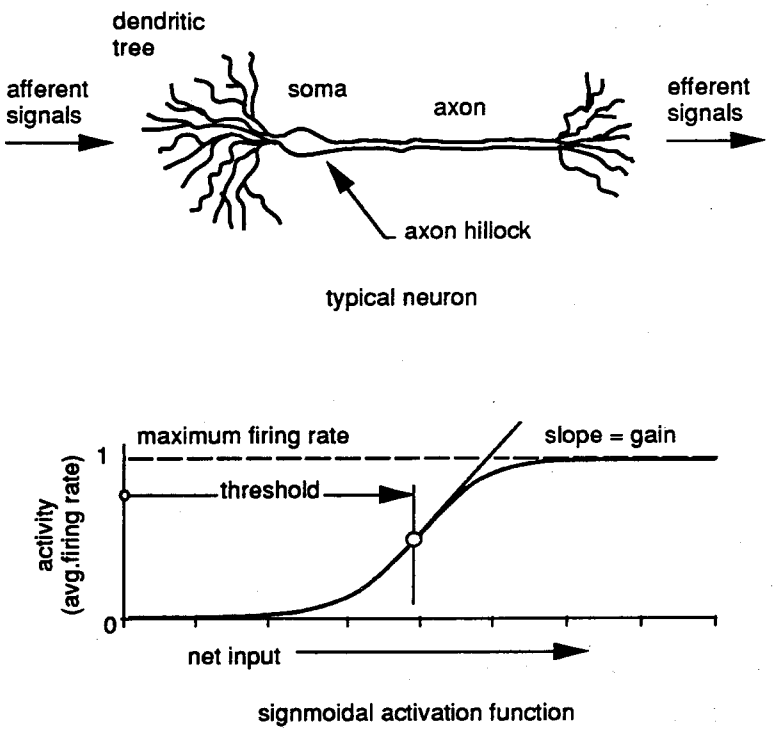

Figure A.1: Schematic biological neuron and activation function.

A.3 Biological Neurons

Neurons are a specialized cell type, schematically represented in figure A.1. They are active computational elements that are able to receive input signals and generate output signals in response. Input, or afferent, signals may come from other neurons or from sensory cells, such as photoreceptors.[2] Output, or efferent, signals are transmitted, in turn, to other neurons or, perhaps, to specialized tissues such as muscles or glands, where they can elicit physical responses [Katz (1966)]. Neurons may receive inputs from (fan-in) and send outputs to (fan-out) many tens of thousands, and, in some cases, hundreds of thousands, of other neurons. There are many types of neurons adapted for particular computational tasks.

[2]Other inputs, of a longer- term nature, are variations in circulating chemicals within the body that affect neurons' physical properties.

From a computational standpoint, a neuron's main features are the dendritic tree, the cell body, or soma, the axon hillock, and the axon. The dendritic tree receives inputs at sites called synapses,[3] of which there are many types [Mead (1989), Shepherd (1978), Shepherd (1979)]. Inputs may be excitatory, which tend to increase the neuron's level of activity, thereby increasing the strength of its output, or inhibitory, which tend to decrease the neuron's level of activity, decreasing the strength of its output. These inputs diffuse toward the soma, or cell body. The net input is essentially the sum of the excitatory and inhibitory inputs integrated over a small time interval.[4] When net input at the axon hillock, the conical region joining the axon to the cell body, exceeds a threshold, a so-called action potential, a voltage spike, is generated with a definite probability [Amit (1989), Mead (1989)], and the cell is said to be active. Action potentials, which are traveling electrochemical waves, have a stereotypical form. They propagate stably down the axon and its branches, connecting with neurons and other specialized cells. It is believed that a particular neuron makes only excitatory or inhibitory connections with other neurons (Dale's law) [Amit (1989), Shepherd (1979)].

If the net input is maintained above threshold, the neuron will remain active, firing a train of action potentials at some rate, known as the firing rate. A neuron's firing rate (its response, or activation) is a sigmoidal function of its net input, varying from a low (resting) value up to a maximum as shown in figure A.1. The gain is the slope of the response curve near the threshold. In the high-gain limit, the response curve becomes a step function and the neuron is either on, firing at its maximum rate, or off, firing at its resting rate.

A signal due to an action potential is propagated across the synapse to the tar-

[3]Sometimes synapses are on the cell body itself.

[4]The details are complex due to the capacitance of the cell membrane and metabolic processes that tend to maintain the resting cell potential. See [Mead (1989)] and [Koch and Segev (1989)].

get neuron by chemicals called neurotransmitters that are released into the synaptic cleft, the small gap between the neurons at the site of the synapse. Neurotransmitters are released in definite quanta [Katz (1966)]. After release, the molecules of neurotransmitter diffuse toward the target neuron. Upon arrival, they initiate a series of chemical events that are either excitatory or inhibitory, depending upon the nature of the synapse.

Neurons affect other neurons primarily through action potentials, but a great deal of computation is done at the analog level within the dendritic tree that does not necessarily result in action potentials [Mead (1989)].

Many neurons are responsive to (they detect) specific stimuli, such as a light spot at a particular location in the visual field, the strength of their responses to a particular input[5] being related to how exactly the input matches the specific stimulus in question. The neuron's receptive field is the region in its input space over which the neuron has appreciable activity to the stimulus it detects, the activity being effectively zero (resting) outside the region and maximum where inputs match the specific stimulus precisely. The neuron's response function over its input space[6] is called its tuning curve (for that space). Neurons with wide receptive fields are termed broadly-tuned; those with narrow receptive fields are termed narrowly-tuned. Tuning has a profound effect on network function, since it affects the ability of a network to interpolate and/or locate stimuli. See [Hinton, McClelland and Rumelhart (1986), Mel (1989), Kent (1981)] and section 3.5.6 in this thesis.

Since a neuron's response is a sigmoidal function of its net excitation, tuning curves for particular input spaces must arise as a computational result of the neuron's

[5]An input may be composed of very many individual elements.

[6]The input space is not necessarily the same as the net input at the axon hillock. Since a neuron receives inputs from a network with a large number of neurons and potentially a great deal of specialized processing, it may have many input spaces.

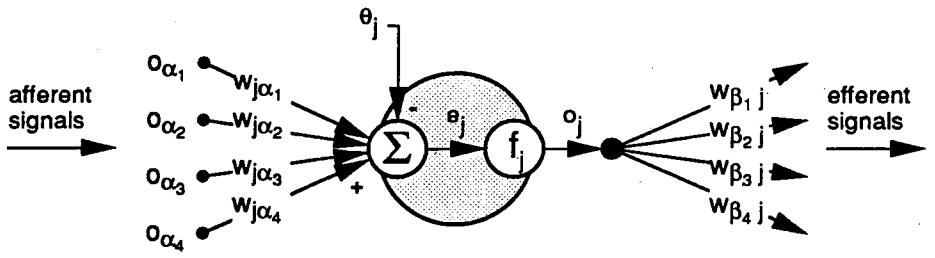

Figure A.2: Idealized neuron. The neuron shown is neuron "j". The o_α's are the afferent inputs, which are weighted by the corresponding input weights, the $w_{j\alpha}$'s, and summed, along with the threshold, θ_j, to yield the net excitation, e_j. Letting f_j be the activation function, the neuron output is $o_j = f_j(e_j)$. The output is coupled to the target neurons through the $w_{\beta j}$'s, which are the output weights.

response function combined with prior neural processing, dendritic computation, and the effects of propagation on input signals.

A.4 Artificial Neurons

As we have seen, an active physical neuron emits voltage spikes at a rate intermediate between its resting and maximum firing rates. While some neural network research, especially that which addresses mechanisms of biological information processing and control, is concerned with detailed neural models that include spiking behavior and dendritic computation,[7] a simplified neural model is often used. In this simplified model, which is shown in figure A.2, neural inputs are represented as firing rates averaged over a small time interval rather than as individual spikes, the firing rate of each input being that of the afferent neuron from which it originates. The neuron's net input is a weighted sum of its individual inputs; its net excitation is the net input minus the threshold, θ.[8] Excitatory and inhibitory connections are handled

[7]See, for example, [Mead (1989)] and [Koch and Segev (1989)].

[8]Thresholds can be considered weights associated with input neurons that are always on (always have unit output).

by respectively assigning positive and negative signs to the weights, with a weight's magnitude representing the connection strength. The threshold, which is equivalent to an inhibitory input, has the effect of shifting the location of the response curve along the net input axis. The neural response, which is scaled to represent the fraction of the neuron's maximum firing rate, is modeled as a sigmoidal function of the net excitation. Thus a resting neuron has zero output while a fully active neuron's output is one.

The sigmoid is commonly taken to be a hyperbolic tangent. In feedforward nets (described below) the hyperbolic tangent is shifted and scaled so the saturation limits (the asymptotes) are zero and one. The neuron's output response, o, to its net excitation, e, is then given by:

$$o = \frac{1}{2}(1 + \tanh(\alpha e)) \tag{A.1}$$

$$= \frac{1}{2}(1 + \tanh(\sum_{j=1}^{n} w_j a_j) - \theta), \tag{A.2}$$

where the a's are the separate inputs, the w's are the weights, n is the number of inputs, α is the gain (the slope of the tanh in the linear region), and θ is the threshold.

In recurrent nets (described below) the saturation limits are often taken as -1 and +1, which represent minimum and maximum firing rates, respectively. A neuron's output response in that case is given by:

$$o = \tanh(\alpha((\sum_{j=1}^{n} a_j) - \theta)), \tag{A.3}$$

where α, a, w, n, and θ are as before. As noted previously, the sigmoid becomes a step function in the high-gain limit. Step functions are known in neural network research as linear threshold functions. By suitable choice of gains, connection strengths, and gating inputs, neurons can be made to exhibit linear behavior over specified regions

as well.

The influence of a neuron's output (which is always a nonnegative firing rate) on a neuron to which it is connected can be controlled by varying the connection strength. The sign of the influence can be controlled by using excitatory (positive) or inhibitory (negative) synapses. In biological systems the connection strength corresponds to the number and strength of the synapses the afferent neuron makes on the target neuron.

Other simplifications are that a given neuron may make both excitatory and inhibitory connections with other neurons,[9] and that dendritic computation is modeled through connection types and connection strengths. Connection types commonly used are the direct excitatory and inhibitory connections already mentioned, where the total input is a weighted sum of the individual inputs, and so-called sigma-pi ($\Sigma - \Pi$) connections [Rumelhart and McClelland (1986), Mel (1989)] in which each individual input is a product of outputs from different neurons and the total input, I, is a weighted sum of the individual inputs:

$$I = \sum_{i=1}^{n} w_i a_i = \sum_{i=1}^{n} w_i \prod_{k=1}^{m} o_{ik}. \tag{A.4}$$

Here the a's are the individual inputs, n is the number of inputs, w are the weights, and the o's are the m outputs from the different afferent neurons that contribute to the ith individual input a_i. Sigma-pi connections are useful when it is necessary to use the output of one or more neurons to gate that of another.

[9] Dale's law is ignored.

A.4.1 Gaussian Neurons

Many neurons have tuning curves that are approximately Gaussian, and some biological visual processing employs tuning curves that are closely approximated by differences of Gaussians [Kent (1981), Marr (1982)]. Furthermore, since Gaussians have convenient mathematical properties,[10] they are attractive for neural modeling. It was pointed out above that, since a neuron's response is actually a sigmoid function of its net input, tuning curves must arise from dendritic and network computations. We will see below in section A.6.6 how sigmoids can be paired in such a way that good approximations to sinusoid and Gaussian tuning curves can be obtained. Such pairs are really composite neurons, but they are considered to be single neurons with sinusoid or Gaussian, rather than sigmoidal, response curves in this work.

A.5 Learning

Learning in biological systems is thought to involve both physical changes that modify synaptic strengths as well as connectivity changes in which new neural connections are formed [Shepherd (1979)].

Learning in artificial neural systems is principally effected by modifying neuron gains and thresholds, changing neuron connectivity, adding and deleting neurons, and changing connection strengths and signs. Modifications are made using learning algorithms as will be discussed below.

[10] Products of Gaussians are Gaussian, for example, making it possible to compose multidimensional Gaussians from one-dimensional Gaussians, and making sums of Gaussians an algebra.

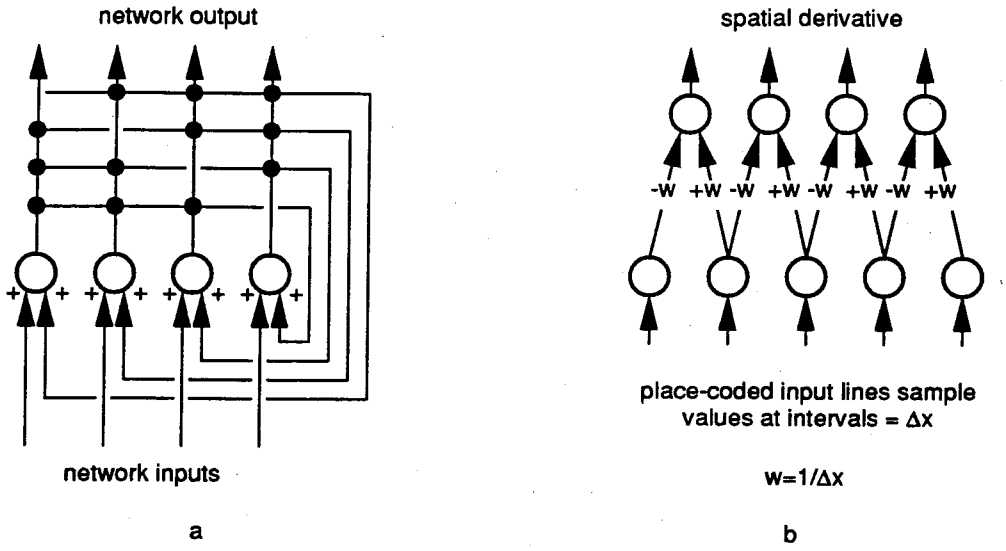

Figure A.3: Neural networks. The direction of signal flow is indicated by the arrows. The network labeled a) is a Hopfield network. It is recurrent since each neuron is connected to every other neuron except itself. The large dots indicate connections. The network labeled b) is a feedforward network that calculates, at each output, a value proportional to the approximate spatial derivative at the input point.

A.6 Artificial Neural Networks

Artificial neural networks are sets of interconnected artificial neurons, and so are often termed connectionist networks. Neural networks can be used for performing computations and are capable of learning. A neural network has a set of input neurons and a set of output neurons, and may, depending upon its architecture, have layers of internal (hidden) neurons. If the input to any neuron in the network is a function of its output, the network is termed recurrent. Otherwise it is a feedforward network. Feedforward networks are closely related to combinational circuits studied in electronics. Feedforward and recurrent networks are illustrated schematically in figure A.3. Networks may naturally have subnetworks of both types. The temporal behavior of a computation is determined by the network inputs and the network state, which is the aggregate of the individual neuron states. Important issues for neural networks are architecture, which is determined by network connectivity and

neuron types, network state evolution and stability, storage capacity, representational efficiency, the effects of noise, spurious states, and learning. The number of neurons their type, and connectivity is usually, but not always, selected in advance (see section A.6.2 below).

Time delays are also important [Amit (1989)]. In recurrent networks they can significantly affect network behavior. They are necessary in feedforward networks for operations such as computing temporal derivatives, which can be performed in the analog domain by subtracting a delayed value of a signal from its current value [Mead (1989)].

Some networks are used to calculate Boolean functions of binary strings, in which case neuron inputs and outputs are considered to be either zero or one. In this context, neural networks can be shown to be capable of computing any computable function, even though the complexity of the network may be high [Abu-Mostafa (1986), Abu-Mostafa (1989), Judd (1990)]. Other networks are used in analog (real valued) calculations, so their inputs are ϵR^k. Networks consisting of a single layer of Gaussian neurons are capable of uniformly approximating any continuous real-valued function [Hartman, Keeler and Kowalski (1990)], hence are able to approximate any real-valued function that has a finite number of finite discontinuities.[11] Since each component of a real vector-valued function is itself a real- valued function, Gaussian networks can calculate real vector-valued functions as well.

Because of noise and neurons' limited dynamic range, input representation is a critical issue for analog networks, often demanding clever use of place and value coding. Boolean (digital) representations of analog values are possible, but can lead to excessive network complexity. In the Boolean case it is often necessary to threshold

[11]This is easy to see. Determine the Fourier series for the function in question and then approximate a suitable number of its harmonics using Gaussian networks.

the outputs to be in the set $\{0,1\}$ unless the output neurons are linear discriminant (step) functions. This is because the sigmoids may not be saturated. Recurrent networks are useful in Boolean computations as the feedback can be used to saturate the (nonlinear) neurons, driving them into either the -1 or $+1$ state.

The bounded nonlinearity of neurons is essential both for providing saturation and for calculating arbitrary functions. Saturation corresponds logically to making a decision. A feedforward network composed of linear neurons with additive connections will give a linear response in all cases.[12] In the linear case, the activation function of the jth neuron will just be a constant α_j. The neuron's output o_j will be α_j times the total excitation, which is a weighted sum of its afferent inputs[13] as we have seen:

$$o_j = \alpha_j \sum_k w_{jk} o_k, \tag{A.5}$$

where the w_{jk} are the weights, and the o_k are the afferent inputs. But each afferent input o_k is, itself, either an output of the same form from another neuron or a network input. This is true of all neurons in the network. Thus, writing $W_{jk} = \alpha_j w_{jk}$ for all neurons j and inputs k, we have, for the ith output neuron and the network inputs a_{k_n}:

$$o_i = \sum_{k_1} W_{ik_1} \sum_{k_2} W_{k_1 k_2} \cdots \sum_{k_n} W_{k_{n-1} k_n} a_{k_n}. \tag{A.6}$$

This is just a linear matrix product, which can be written in the form:

$$\mathbf{o} = \mathcal{W}\mathbf{a} \tag{A.7}$$

$\mathcal{W}$ is the matrix product of the W's. This shows that an arbitrary network of linear neurons with additive connections is equivalent to a single layer of linear neurons.

[12]This is not necessarily true if sigma-pi connections are used. Multiplying an input by itself, for example, will give a parabolic response.

[13]Thresholds are considered as weights here.

Thus nonlinearities are essential for calculating more general functions.

A.6.1 Recurrent Neural Networks

Highly interconnected recurrent networks with many neurons are high-order coupled nonlinear systems in which system dynamics due to factors such as signal propagation delays, resistance, capacitance, and gain must be considered. As a consequence, they have rich behavior, and are the subject of much current study. Techniques from theoretical physics including statistical mechanics and spin glasses are being fruitfully applied to investigate their properties [Amit (1989)]. The basic idea is that the network will evolve (flow) downwards in energy space towards an attracting state or limit cycle.

Hopfield and others have investigated fully interconnected networks (each neuron is connected to each other neuron except itself) with symmetric and nonsymmetric interconnection matrices [Hopfield (1982), Hopfield (1984), Amit (1989)]. In Hopfield networks, the interconnection matrices are constructed from the sum of the outer products of the memories stored in the network. Hopfield networks with symmetric interconnection matrices can be shown to be stable. Introducing asymmetry in the connection matrix can, under the right conditions, force the network to generate prescribed sequences and enter limit cycles.

Other interesting behavior, such as finding global and local extrema of functions, is possible as well. Creating networks that mimic computational functions observed in biological systems is an active area of research [Koch and Segev (1989), Mead (1989)]. Recurrent networks are being used to generate and recognize sequences, remove noise, recall complete items from only partial inputs, generate commands for walking machines, etc. [Amit (1989), Hopfield (1982),

Hopfield (1984), Rumelhart and McClelland (1986), Beer, Chiel and Sterling (1991), Chiel, Quinn and Espenscheid (1992)].

In analytical models of Hopfield networks, neurons are modeled as nonlinear amplifiers that generate currents proportional to their activities, and the cell body is modeled as a capacitor that sums incoming currents, creating a voltage that drives the amplifier. Connection strengths are conductances.

A.6.2 Learning With Feedforward Neural Networks

The interest in feedforward neural networks arises primarily from their ability to learn functional approximations. In feedforward networks that have stable, simple neurons of the type described above, dynamics is not an issue except as it relates to network settling time. For each steady input, there is asymptotically a unique, steady output. In the high-gain limit, sigmoids become linear discriminant functions and feedforward networks become multilayer networks called perceptrons [Minsky and Papert (1969), Duda and Hart (1973)].

The best-known learning algorithm for feedforward networks is Back Error Propagation (BEP),[14] which is an application of the well-known gradient descent procedure [Rumelhart and McClelland (1986)]. After the network architecture has been fixed, learning proceeds by presenting the network with an input, noting the error in its output vector, **O**, and recursively adjusting the weights to reduce the error. It requires a teacher, or supervisor, that knows what **T**, the desired, or target, output vector should be. Hence it is an example of supervised learning. Thresholds can also be learned if they are considered to be weights for an afferent neuron with output

[14]The name comes from the fact corrections are made by recursively propagating errors back from the output layer.

that is fixed at 1. Training sessions are usually started by initializing the weights to random small numbers.

If an energy E is defined by summing the squares of the components of the output error over the m output components (neurons):

$$E = \frac{1}{2}\sum_{l=1}^{m}(T_l - O_l)^2, \tag{A.8}$$

then, with η the learning rate, f_j the activation function of neuron j, e_j the total excitation of neuron j, o_k the output of neuron k, and δw_{jk} the change in w_{jk}, the weight coupling the output of neuron k to neuron j, the back propagation algorithm is given by:

$$\begin{aligned}
\delta w_{jk} &= -\eta\frac{\partial E}{\partial w_{jk}} && \text{(A.9)}\\
&= -\eta\frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial w_{jk}} && \text{(A.10)}\\
&= -\eta\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial e_j}o_k && \text{(A.11)}\\
&= -\eta o_k f_j'(e_j)\frac{\partial E}{\partial o_j}. && \text{(A.12)}
\end{aligned}$$

If neuron j is an output neuron, this becomes:

$$\delta w_{jk} = \eta o_k f_j'(e_j)(T_j - O_j). \tag{A.13}$$

If neuron j is not an output neuron, then we have:

$$\delta w_{jk} = -\eta o_k f_j'(e_j)\sum_p w_{pj}\frac{\partial E}{\partial e_p}. \tag{A.14}$$

The sum over p is a sum over the neurons which have their inputs connected to the output of neuron j. It is recursive, starting with the output layer.

In some cases, as in learning to control an unknown plant with a neural network, for example, it is necessary to propagate errors backward across the plant. Assuming that the plant has summing inputs $a_j = \sum_k w_{jk} o_k$ and outputs O_l, equation A.10 becomes:

$$\delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \tag{A.15}$$

$$= -\eta \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{jk}} \tag{A.16}$$

$$= -\eta o_k \frac{\partial E}{\partial a_j} \tag{A.17}$$

$$= \eta o_k \sum_l (T_l - O_l) \frac{\partial O_l}{\partial a_j}. \tag{A.18}$$

The partial derivatives $\partial O_l / \partial a_j$ can be empirically determined [Psaltis, Sideris and Yamamura (1987)].

Multilayer BEP networks work well in some cases, but often require an excessive number of training trials, and are subject to becoming stuck in false minima. Errors do not propagate past high-gain or saturated/resting neurons because in those cases the derivative f' is too close to zero. There is also the design problem of determining how many layers to use in a network and how to connect them. Single-layer networks with more initial structure successfully address many of these problems, as described below in section A.6.6.

Another learning approach is a version of Hebbian learning [Hebb (1948)] in which a sigma-pi connection is established between a target neuron and a set of afferent neurons if the afferent neurons are active and it is desired to have the target neuron active. Mel [Mel (1989)] has used this approach in learning hand-eye coordination. It corresponds to table lookup, and is an example of learning in which connectivity is not set in advance. Learning is very fast, as connections are created in one pass as needed. As in all table lookup schemes, however, the number of units required to

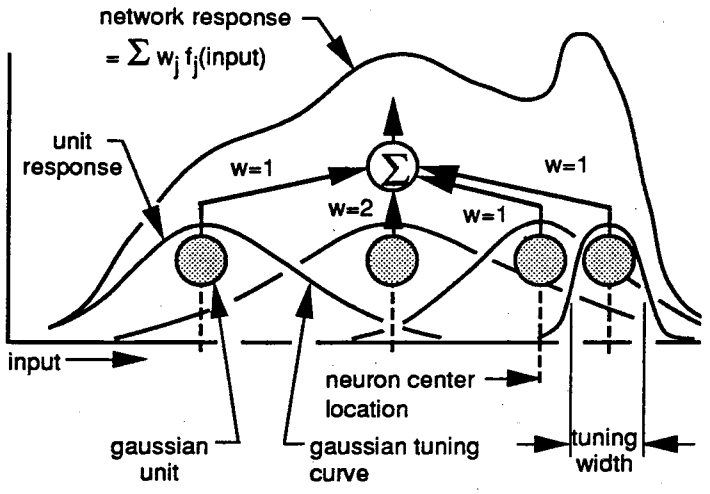

Figure A.4: Resource-allocating neural network. The network, which is a feedforward network, generates its response by superposing the responses of its individual neurons. The network is shown with Gaussian neurons, but other types may be used.

learn a mapping can become excessive.

A.6.3 Resource-Allocating Neural Networks

Resource-allocating neural networks [Platt (1990)], which map n-dimensional real spaces into m-dimensional real spaces, are also a version of table lookup. They employ a single layer of radially-symmetric Gaussian, or other unimodal neurons rather than sigmoidal neurons. The neurons are equivalent to the radial basis functions of approximation theory [Poggio and Girosi (1989)], and, as allocated during training, are related to Parzen windows that are used in estimating statistical distributions [Duda and Hart (1973)]. There is one input line for each degree of freedom in the input space. Every neuron is connected to all the input lines.

Each component of the output vector is generated by superposing the (weighted) responses of the network's neurons to the input vector as shown in figure A.4. Each

neuron calculates its distance from the input vector[15] and, based on the width of its response function, calculates its response as a function of the distance. The weighted responses are summed to get the output. Since the network response is generated from a finite set of neurons it is usually an approximation to a desired mapping.

In the normal case, a resource-allocating network initially has no neurons. The network is trained by selecting a set of input vectors that covers the domain of the desired function. Each vector is presented to the network in a separate trial and the error between the actual, or observed, network response and the desired, or target, response is noted. Then, if necessary, either a new neuron is allocated to eliminate the output error for that input completely, or the parameters of the existing neurons are adjusted to reduce the error. The parameters are the width of the response function (the tuning width in this case), the output weights that couple the neuron's output to the output component's summing junction, and the location of neuron's receptive field. Learning vector-valued mappings is simplified by the fact that the output components are independent of one another and depend upon separate output weights. More specifically:

1. If the error is acceptable, no action is taken.

2. A new neuron is allocated if the absolute error is above an error threshold, θ_e, additional neurons are available, and the current input vector is greater than a distance threshold, θ_d, from the center of the nearest existing neuron. Its parameters are chosen to make the network output exact for the current input vector, and the width of its response function is chosen so network output is disturbed only locally.

3. An adaptive algorithm adjusts network parameters to reduce error if: the error is too large to ignore but is below the allocation error threshold θ_e, or the input

[15]The distance from input vector to the center of the neuron's response function.

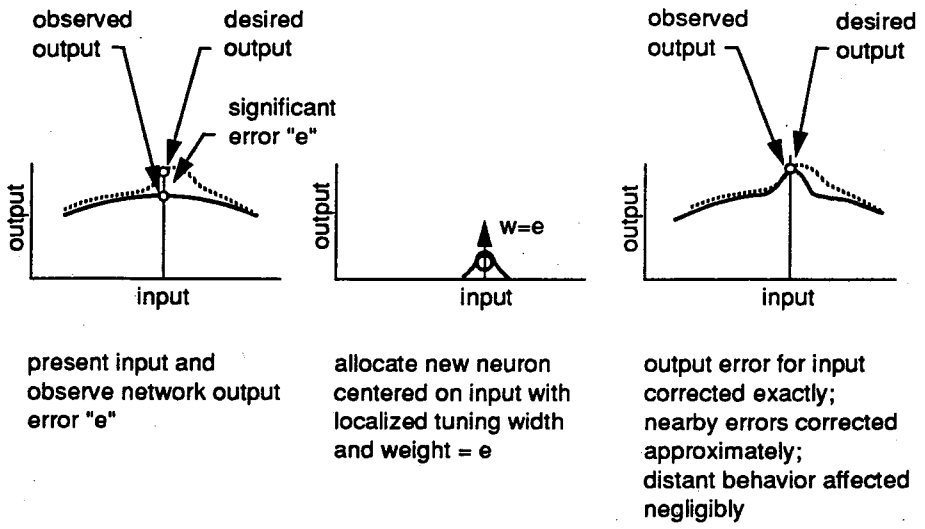

Figure A.5: Learning by allocating neurons.

vector is less than θ_d from the center of the nearest existing neuron, or no additional neurons are available.

Learning by allocating neurons is illustrated in figure A.5.

The distance threshold, θ_d is usually allowed to decrease according to some schedule to prevent fragmentation of the input space and to improve representational efficiency. Network performance depends critically upon the error thresholds and the algorithm for varying the distance threshold and determining the width of the neuron response functions.

Since resource-allocating networks have a single layer of neurons and compute good approximations because of the way neurons are allocated, parameter adjustment algorithms give good results, converge quickly, and tend not to become stuck in false minima. They also perform well at tracking drifting plants.

Resource-allocating networks using Gaussians are mathematically justified by the fact that Gaussians are universal approximators as mentioned above

[Hartman, Keeler and Kowalski (1990)].

A.6.4 The Cerebellar Model Articulation Controller

Resource-allocating networks are closely related to the Cerebellar Model Articulation Controller (CMAC) of Albus [Albus (1972), Albus (1975b), Albus (1975a), Albus (1981)]. In CMAC, which is based on a model of the cerebellum, a set of so-called association cells (neurons) is activated for each input according to a predefined mapping from the input sensory cells to the association cells. The total number of association cells is typically much larger than the number active for any particular input. The output of an active association cell is one because both the sensory and association cells are linear discriminant functions. Every active association cell has one weight coupling it to each output component. Each output component, then, is the sum of the weights of the association cells active for the input. The network can generalize because the predefined input mapping from the sensory cells to the association cells is constructed so nearby inputs activate many of the same association cells.

The network is taught by presenting example inputs and noting, for each input, the difference between the target output T_l and the observed output O_l for each component of l. The weights for that component associated with the neurons that are active for the input are all adjusted by the amount:

$$\delta = \eta \frac{T_l - O_l}{N}, \tag{A.19}$$

where N is the number of active neurons and $0 < \eta \leq 1$ is the learning rate. Controllers based on CMAC are able to learn smooth mappings of the type found in robot control quickly and accurately [Miller (1987)]. Ellison has shown that multidimen-

sional CMAC's learn by converging to well-defined limit cycles [Ellison (1991)].

A.6.5 Network Updating in Simulations

In simulating neural network behavior, the choice of network update scheme (the manner in which neurons are selected for state updating) is critical. Synchronous update (all neurons updated simultaneously) is unsatisfactory in recurrent networks because it can significantly affect network behavior, inducing spurious limit cycles [Amit (1989)] and affecting energy minimization [Hopfield (1982), Hopfield (1984)]. Synchronous updating is acceptable in feedforward networks because network dynamics is not an issue.

A.6.6 Intuitive Motivation for Feedforward Neural Networks

The relationship of networks of sigmoidal neurons to practical computing can initially seem obscure, if not magical. Feedforward neural networks, however, are relatively easy to motivate intuitively in a way that is closely related to familiar mathematical ideas. The key is in considering *pairs* of neurons.

Consider, first, the sum of paired sigmoids in the high-gain limit, separated as in figure A.6 by an interval $\Delta > 0$. The sigmoids have equal and opposite gains and Δ points from the positive gain step to the negative gain step.[16] The result is a one-dimensional pulse of height two, which can be scaled to yield a unit pulse. Steps with range from -1 to +1 are shown in the figure. The use of steps with range 0

[16]Negative gains can be effected with negative input weights.

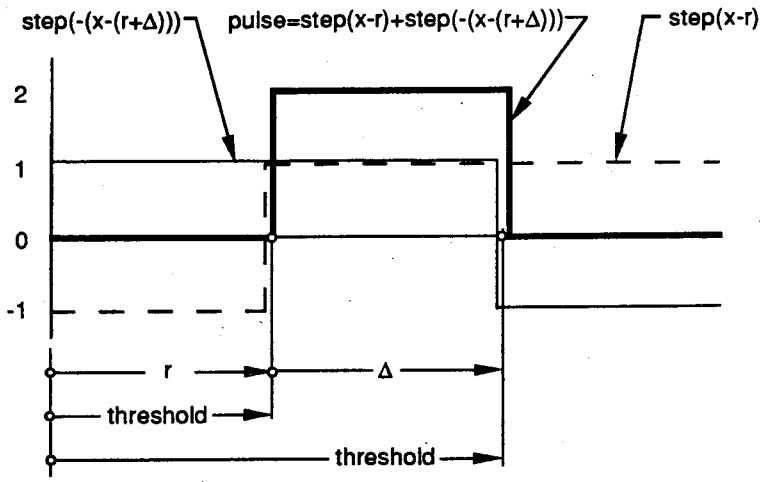

Figure A.6: Paired high-gain sigmoids. The sum of the high-gain sigmoids, which are step functions with opposite gains and thresholds separated by Δ, is a pulse of height two. The functions are shifted slightly for clarity.

to 1 will result in one plus a unit pulse. This can be shifted with a constant offset to yield a unit pulse. A unit pulse has value one inside its region of support and is zero elsewhere, so if its output is multiplied by the appropriate weight (the average functional value in the region selected by the pulse, for example) it can be an element of a functional approximation table. The elements would be like those making up a Riemann sum in integral calculus. Other neural circuitry, based on expanding and gating the linear regions of lower-gain sigmoids, could be added as well to interpolate between adjacent functional values.

Consider, now, lower gain sigmoids. It can be seen from figure A.7 that the sum of paired hyperbolic tangents which have opposite gains and thresholds separated by $\Delta = \pi$, is a reasonable approximation to a segment of $1 + \sin(x - r)$ over the region $-\pi \leq x - r \leq \pi$, providing the conditions on Δ are as above. The sum is close to zero outside the region. The sum of a pair of hyperbolic tangents with an offset of -1 therefore forms a computational module that generates an approximation to a segment of a sine. Such a module is illustrated in figure A.8a. By adjusting the thresholds of the component hyperbolic tangents, module domains may be shifted

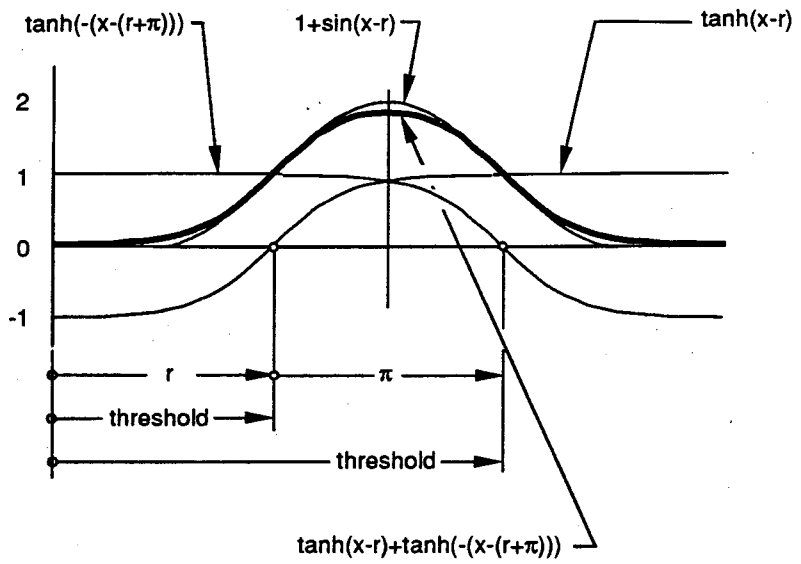

Figure A.7: Paired hyperbolic tangents. The sum of paired hyperbolic tangents with opposite gains and thresholds separated by π is approximately $1 + sin(x - r)$ over the region $-\pi \leq x - r \leq \pi$, and is approximately zero elsewhere.

to cover adjacent regions. Concatenating several appropriately shifted modules will yield longer segments of a sine or a cosine over an interval. Higher harmonics may be created by increasing the magnitudes of the input weights (hence the gains of the hyperbolic tangents) in integral steps. If the interval is $2L$, the input weights may be scaled by π/L to fit.

If sufficient modules of the proper harmonics required to cover the desired spatial and frequency ranges are concatenated, and the output weights from the sine and cosine modules are the appropriate Fourier coefficients, the network will give an approximation to the function corresponding to the Fourier coefficients. This is illustrated in figure A.8b. Such a network is basically composed of a single layer of sigmoidal neurons. The constant offsets of -1 can, in principle, be absorbed in the constant Fourier term. We know a Fourier series can be used to approximate any real-valued function with a finite number of finite discontinuities.

In a similar way, paired sigmoids approximate a Gaussian, and from the theorem

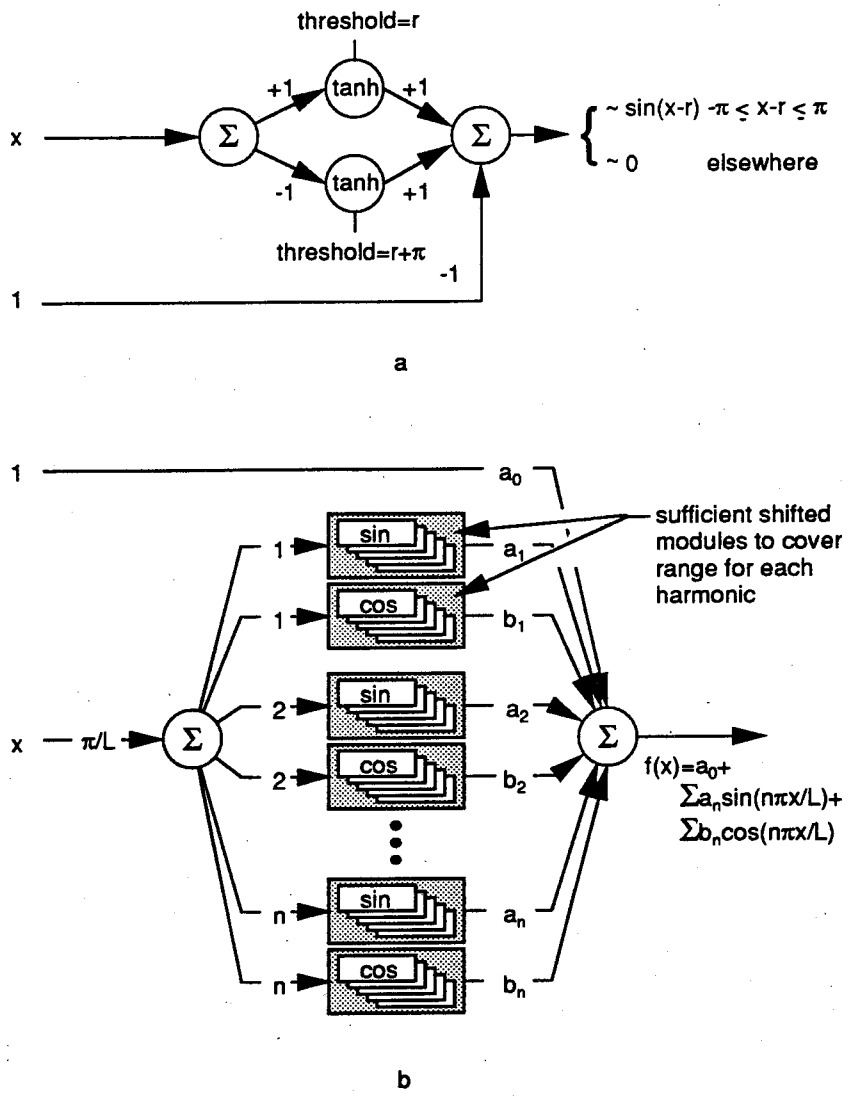

Figure A.8: Approximate Fourier series. The top figure, a), shows a computational module that approximates a segment of a sine. Concatenating several such modules by shifting thresholds will yield an approximation of $\sin(x)$ over an interval. Cosines can be obtained by shifting and concatenation as well. Different harmonics are obtained by adjusting the input weights in integral steps. The bottom figure, b), shows modules of different harmonics combined to approximate a function. The input is scaled to fit the interval in question. The thresholds and constant -1 offsets for the modules are not shown.

quoted above, we know that any real continuous function can be approximated with a single layer of Gaussian units.

In preliminary investigations, combining sinusoid modules into approximate

Fourier series was compared with randomly structured networks[17] in learning various functions such as triangles, pulses, and polynomials. The learning algorithm was back error propagation, and all input and output weights, including the thresholds, associated with each sigmoid were allowed to vary. Fourier coefficients were initially set at $1/n$, with n the order of the term. In these investigations, the structured approach converged much more rapidly and had significantly greater accuracy. This was also the case with resource-allocating networks using both Gaussian and other unimodal neurons as discussed in chapter 3.

The important thing to realize from this discussion is that considering opposing pairs of sigmoidal functions leads naturally to approximations of functions that appear in functional approximation schemes. This shows that sigmoidal neurons can be combined to yield reasonable functional approximations. It does not show that a given network will necessarily learn a good approximation.

The above discussion has considered functions over the real line. In higher dimensions, sigmoids can be composed multiplicatively to yield multidimensional unit pulses or multidimensional Gaussians. Sinusoidal eigenfunctions could also be composed multiplicatively if desired. Composition can be done with limited range amplifiers because sigmoid output can be scaled to be $\epsilon[0, 1]$. Multiplicative connections require imposing a greater degree of initial structure on the network. This structure will improve network performance since unstructured back error propagation networks perform poorly compared to more structured networks.

Finally, it is apparent that what feedforward networks actually do is functional interpolation. A network is taught with a finite set of training examples and is intended to provide good approximations for inputs that have not appeared in the training set. This is essentially interpolation [Poggio and Girosi (1989)].

[17]Networks with small random initial weights.

Appendix B

Implementing the CMAC and Virtual Neuron Learning Algorithms in Analog VLSI

This appendix explores, in a very high-level sense, implementing the adaptation algorithms of chapter 3 in analog VLSI hardware.

Gradient descent is difficult to implement because of the derivative computations and will not be discussed further. The CMAC and virtual neuron algorithms, on the other hand, require no derivatives and would be much simpler to implement, with CMAC being the simplest.

Calculating the common CMAC active neuron weight change Δw requires just that the componentwise output error ϵ_k be multiplied by the quotient of the learning rate λ and the sum of the active neuron outputs. The sum and componentwise error

can be calculated in parallel, and once the computations have stabilized, the quotient of λ and the sum can be calculated, followed by multiplication by the ϵ_k. Multiplying the ϵ's can all be performed in parallel since the quotient is common to all weight changes.

With the exception of division, and perhaps weight adjustment, analog VLSI circuit elements exist for all the operations that must be performed [Mead (1989)]. Since the factors are all positive, division can probably be handled using multipliers amplifiers, and feedback. Selecting the neurons and weights for modification can be accomplished by gating using thresholded neural outputs for the (latched) input. Signals for enabling and disabling learning would probably be provided externally since the network would be part of a larger system.

Implementing virtual neuron learning in hardware will be more complex. Calculating the weight changes involves determining the width of the virtual neuron, calculating its response at the site of each active neuron, determining the sum of the products of the virtual neuron response at each neuron and the neuron's response to $\mathbf{x}$, calculating the quotient, calculating the errors ϵ_k, and generating the weight change. This will require phased chip activity driven by timing or state mechanisms.

Active neurons are first selected by gating those with outputs over the threshold. The virtual neuron width is determined by selecting the neuron nearest the input vector $\mathbf{x}$, actively controlling its width (the original must be retained) with feedback till its response is 0.5, and multiplying the resulting width by h, the half-maximum fraction (which is an input), to yield the virtual neuron width. The nearest neuron is selected by temporarily setting the widths of all active neurons to the same value and selecting (latching) the one with the highest response to $\mathbf{x}$.

The virtual neuron width is then gated to all active neurons to determine each

neuron's response to $\mathbf{x}$ as if it were the virtual neuron. This determines the virtual neuron response at the site of each neuron. The product of each neuron's original output (which has to be stored locally along with the original) by the virtual neuron's response at its location is calculated and all such products are summed, thus determining the (positive) denominator of the quotient. The quotient of λ and the denominator sum can be computed using feedback as before. The quotient is then multiplied by the virtual neuron's response at each neuron site and the ϵ_k in parallel to determine the Δw_{kj}.

The necessity for storing original output and width at each neuron site can be eliminated by making each neuron a matched pair. One is used to calculate net outputs and retains its original width. The other has variable width and is used to determine the nearest neuron, the virtual neuron response at the neuron site, etc. The matched pair can share the same distance computation, and so should be relatively cheap to implement, especially if Gaussian neurons are used. Using paired neurons will also simplify chip timing and gating and improve parallelism.

If neurons have been allocated in a regular grid, as they might be on a chip, the virtual neuron width parameter might profitably be considered fixed and need not be determined. This would further simplify chip implementation. Random neuron locations could be handled by providing adaptable gates in the input lines to each neuron. These could be set when the neuron was allocated. It is apparent from figures 3.16, 3.18, and 3.20, however, that even when neurons are allocated without setting the output weights to eliminate output error at the allocation site, adaptation performance is quite good. That is, acceptable long-term network performance does not require the Platt allocation/correction algorithm, though it speeds up initial learning.

As in the case of CMAC, which requires the same kinds of operations, basic

analog VLSI tools exist for implementing the circuits discussed here, with the possible exception of division and weight adjustment. Division can conceptually be handled by variable-gain amplifiers and feedback. Weight adjustment via floating gates and other mechanisms is an active research topic.

We can conclude from this discussion that implementing the CMAC and virtual neuron learning algorithms in parallel analog VLSI hardware appears to be feasible.

References

Abu-Mostafa, Yaser 1986. Neural networks for computing? In Denker, John S., editor, *Neural Networks for Computing*, pages 1–6. American Institute of Physics, AIP. AIP Conference Proceedings no. 151.

Abu-Mostafa, Yaser 1989. Complexity in neural systems. In Mead, Carver, editor, *Analog VLSI and Neural Systems*, pages 353–358. Addison-Wesley, New York. Appendix D.

Albus, James S. 1972. *Theoretical and Experimental Aspects of a Cerebellar Model.* PhD thesis, University of Maryland, Department of Biomedical Engineering.

Albus, James S. 1975a. Data storage in the cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, pages 228–233, September.

Albus, James S. 1975b. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, pages 220–227, September.

Albus, James S. 1981. *Brains, Behavior, and Robotics.* McGraw-Hill (Byte Books), Peterborough, N.H.

Alexander, Howard W. 1961. *Elements of Mathematical Statistics.* John Wiley and Sons, New York.

Amit, Daniel J. 1989. *Modeling Brain Function: The World of Attractor Neural Networks.* Cambridge University Press, Cambridge.

Andersson, Russell L. 1988. Building fast, intelligent, robot systems. *ATT Technical Journal*, 67(2):73–86, March/April.

Arbib, Michael A. 1987. *Brains, Machines, and Mathematics.* Springer-Verlag, Heidelberg, 2 edition.

Atkeson, Christopher G. 1986. *Roles of Knowledge in Motor Learning.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September.

Backes, Paul G. and Tso, Kam S. 1990. Umi: An interactive supervisory and shared control system for telerobotics. In *PIEEERA*, pages 1096–1101, Cincinnati, Ohio, May.

Backes, Paul G. 1991. Generalized compliant motion with sensor fusion. In *Proceedings 1991 ICAR: Fifth International Conference on Advanced Robotics, Robots in Unstructured Environments*, pages 1281–1286, Pisa, Italy, June 19–22.

Backes, Paul G. 1992a. Dual-arm supervisory and shared control space servicing task experiments. In *Proceedings AIAA Space Programs and Technologies Conference*, Huntsville, AL, March 24-27. AIAA paper No. 92-1677.

Backes, Paul G. 1992b. Supervised autonomous control, shared control, and teleoperation for space servicing. In *Proceedings Space Operations, Applications, and Research Symposium*, Houston, August 4–6.

Baldi, Pierre and Heiligenberg, W. 1988. How sensory maps could enhance resolution through ordered arrangements of broadly tuned receivers. *Biological Cybernetics*, 59:313–318.

Beer, R.D., Chiel, H.J., and Sterling, L.S. 1991. An artificial insect. *American Scientist*, 79(5):444–452, September-October.

Brooks, Vernon B. 1986. *The Neural Basis of Motor Control.* Oxford University Press, Oxford.

Chiel, H.L.and Beer, R.D., Quinn, R.D., and Espenscheid, K.S. 1992. Robustness of a distributed neural network controller for locomotion in a hexapod robot. *IEEE Transactions on Robotics and Automation*, 8(3):293–304, June.

Churchland, Patricia Smith and Sejnowski, Terrence J. 1988. Perspectives on cognitive neuroscience. *Science*, 242:741–745, November.

Churchland, Patricia Smith 1986. *Neurophilosophy: Toward a Unified Science of the Mind/Brain.* Computational Models of Cognition and Perception. MIT Press, Cambridge, Massachusetts.

Craig, John J. 1986a. *Adaptive Control of Robotic Manipulators.* PhD thesis, Stanford University, Department of Electrical Enginering, Stanford, CA 94305, June.

Craig, John J. 1986b. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, Menlo Park, California.

Cronin, Jane 1987. *Mathematical Aspects of Hodgkin-Huxley Neural Theory.* Cambridge University Press, Cambridge.

Denker, John S. 1986. *Neural Networks for Computing.* AIP. AIP Conference Proceedings no. 151.

Distefano, Joseph J., Stubberud, Allen R., and Williams, Ivan J. 1967. *Theory and Problems of Feedback Control Systems.* McGraw-Hill, New York.

Dorf, Richard C. 1983. *Modern Control Systems.* Addison-Wesley, New York.

Duda, Richard O. and Hart, Peter E. 1973. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York.

Ellison, D. 1991. On the convergence of the multidimensional albus perceptron. *The International Journal of Robotics Research*, 10(4):338–357, August.

Engelberger, Joseph F. 1980. *Robotics in Practice.* Amacom. ISBN 0-8144-5645-6.

Franklin, Gene F., Powell, J. David, and Emami-Naeini, Abbas 1986. *Feedback Control of Dynamic Systems.* Addison-Wesley, New York.

Gennery, Donald, Litwin, Todd, Wilcox, Brian, and Bon, Bruce 1987. Sensing and perception research for space telerobotics at JPL. In *International Conference on Robotics and Automation*, pages 311–317. IEEE, IEEE Computer Society Press, March.

Goldberg, Ken and Pearlmutter, Barak 1988. Using a neural network to learn the dynamics of the cmu direct-drive arm ii. Technical Report CMU-CS-88-160, Carnegie-Mellon University, Pittsburg, Pennsylvania, August.

Goodwin, Graham C. and Sin, Kwai S. 1984. *Adaptive Filtering, Prediction, and Control.* Prentice-Hall, Englewood Cliffs, New Jersey.

Grossberg, Stephen 1982. *Studies of Mind and Brain.* D. Reidel Publishing Company, Boston, Massachusetts.

Handelman, D.A., Lane, S.H., and Gelfand, J.J. 1989. Integrating neural networks and knowledge-based systems for robotic control. In *International Conference on Robotics and Automation*, pages 1454–1461, Washington. IEEE, IEEE Computer Society Press.

Hartman, Eric J., Keeler, James D., and Kowalski, Jacek M. 1990. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2:210–215.

Hayati, Samad A. 1990. personal communication.

Hebb, Donald O. 1948. *The Organization of Behavior: A Neuropsychological Theory.* John Wiley and Sons, New York.

Hinton, David E., McClelland, James L., and Rumelhart, David E. 1986. Distributed representations. In Rumelhart, David E. and McClelland, James L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1*, volume 1, chapter 3. MIT Press, Cambridge, Masachusetts.

Hopfield, John J. 1982. Neural networks and physical systems with emergent collective computational capabilities. *Proceedings of theNational Academy of Science*, 79:2554–2558.

Hopfield, John J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA*, 89:3088–3092.

Hutchinson, S.A. and Kak, A.C. 1990. Spar: A planner that satisfies operational and geometric goals in uncertain environments. *AI Magazine*, 11(1):30–61, Spring.

Judd, J. Stephen 1990. *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge, Massachusetts.

Katz, B. 1966. *Nerve, Muscle, and Synapse.* McGraw-Hill, New York.

Kent, Ernest W. 1981. *The Brains of Men and Machines.* McGraw-Hill (Byte Books), Peterborough, N.H.

Koch, Christof and Segev, Idan 1989. *Methods in Neuronal Modeling.* Computational Neuroscience. MIT Press, Cambridge, Massachusetts.

Kuperstein, Michael 1987a. personal communication.

Kuperstein, Michael 1987b. Adaptive visual-motor coordination in multijoint robots using parallel architecture. In *International Conference on Robotics and Automation*, pages 1595–1602. IEEE, IEEE Computer Society Press, March.

Kuperstein, Michael 1988. Neural model of adaptive hand-eye coordination for single postures. *Science*, 239:1308–1311, March.

Laird, J.E., Yager, E.S., Hucka, M., and Tuck, C.M. 1991. Robo-soar: An integration of external interaction, planning, and learning using soar. *Robotics and Autonomous Systems*, 8:113–129.

Larkin, David 1993. Implementation of an adaptive controller. In *Proceedings, International Robot and Vision Show and Conference*, pages 27–35, Detroit, MI, April. Robotic Industries Association, RIA.

Lisberger, Stephen G. 1988. The neural basis for learning simple skills. *Science*, 242:728–735, November.

Ljung, Lennart and Soderstrom, Torsten 1987. *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, Massachusetts.

Lokshin, Anatole 1990. Telerobotics for space assembly and servicing–fy 90 final report. Technical Report JPL D-7875, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California 91109. see Appendix C: Hand-Eye Calibration.

Marr, David 1982. *Vision.* W. H. Freeman and Company, New York.

Mead, Carver 1989. *Analog VLSI and Neural Systems.* Addison-Wesley, New York.

Mel, Bartlett W. 1989. *Murphy: A Neurally-Inspired Connectionist Approach to Learning and Performance in Vision-Based Robot Motion Planning.* PhD thesis, Center for Complex Systems Research, University of Illinois, Urbana, Illinois, February.

Miller, W. Thomas III 1987. Sensor-based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, 3(2):157–156, April.

Minsky, Marvin and Papert, Seymour 1969. *Perceptrons.* MIT Press, Cambridge, Massachusetts.

Morari, Manfred 1989. *Robust Process Control.* Prentice Hall, Englewood Cliffs, N.J.

Norman, Donald A. 1982. *Learning and Memory.* W.H. Freeman, New York.

Paul, Richard P. 1981. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, Massachusetts.

Platt, John 1990. A resource-allocating neural network for function interpolation. Preprint, Synaptics, Inc. 2860 Zanker Road, Suite 105, San Jose, CA 95134.

Poggio, Tomaso and Girosi, Federico 1989. A theory of networks for approximation and learning. Technical Report 1140, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Pomerleau, Dean A. 1990. Neural network based autonomous navigation. In Thorpe, Charles, editor, *Vision and Navigation: The CMU Navlab.* Kluwer Academic Publishers.

Popplestone, Robin J., Liu, Yanxi, and Weiss, Rich 1990. A group theoretic approach to assembly planning. *AI Magazine*, 11(1):82–97, Spring.

Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T. 1988. *Numerical Recipes in C*, chapter 14, pages 542–547. Cambridge University Press, Cambridge.

Psaltis, Demetri, Sideris, Athanasios, and Yamamura, Alan 1987. Neural controllers. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 551–558.

Raibert, M.H. and Horn, B.K.P. 1978. Manipulator control using configuration space method. *Industrial Robot*, 5:69–73, June.

Raibert, M.H. 1977. Analytical equations vs. table look-up for manipulation: a unifying concept. In *Proc. IEEE Conference on Decision and Control*, pages 576–579, New Orleans, LA. IEEE.

Ralston, Anthony and Rabinowitz, Philip 1978. *A First Course in Numerical Analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill, New York, 2 edition.

Rumelhart, David E. and McClelland, James L. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, Massachusetts.

Ruoff, Carl F. 1980. Pacs–an advanced multitasking robot system. *The Industrial Robot*, 7(2), June.

Ruoff, Carl F. et al. 1984. Autonomous ground vehicles: Control system technology development. Technical Report ETL-0375, U.S. Army Engineer Topographic Laboratories, Ft. Belvoir, VA, October.

Sejnowski, Terrence J., Koch, Christof, and Churchland, Patricia S. 1988. Computational neuroscience. *Science*, 241:1299–1306, September.

Shepherd, Gordon M. 1978. Microcircuits in the nervous systems. *Scientific American*, 238(2):93–103, February.

Shepherd, Gordon M. 1979. *The Synaptic Organization of the Brain.* Oxford University Press, Oxford, 2 edition.

Slotine, J. and Li, W. 1986. On the adaptive control of robot manipulators. In *Proceedings of the ASME Winter Annual Meeting*, pages 51–56, Anaheim, CA. ASME, ASME.

Slotine, J. and Li, W. 1991. *Applied Nonlinear Control.* Prentice-Hall, Englewood Cliffs, NJ.

Touretzky, David 1989. *Advances in Neural Information Processing Systems.* Morgan Kauffman, Palo Alto, California.

Varsi, G., Ruoff, C., Culick, F., and Burdick, J. 1992. Projected automation technology requirements 1992–2011. Technical Report JPL D-9306, Caltech Jet Propulsion Laboratory, Pasadena, California, May.

Waldrop, Mitchell M. 1988. Soar: A unified theory of cognition? *Science*, 241:296–298, July.

Wilcox, B. and Gennery, D. 1987. A Mars rover for the 1990s. *Journal of the British Interplanetary Society*, 40:484–488, October.

Wilcox, B., Tso, K., Litwin, T., Hayati, S., and Bon, B. 1989. Autonomous sensor-based dual-arm satellite grappling. In *Proceedings of the NASA Conference on Space Telerobotics*, pages 307–312, Pasadena, California, Jan. NASA, Jet Propulsion Laboratory. JPL Publication 89-7 V5.

Wise, Steven P. and Desimone, Robert 1988. Behavioral neurophysiology: Insights into seeing and grasping. *Science*, 242:736–740, November.