THESIS
3

MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 01413 7719

This is to certify that the

dissertation entitled

TOWARDS A LEARNING SYSTEM FOR
ROBOT HAND-EYE COORDINATION

presented by

SALLY JEAN HOWDEN

has been accepted towards fulfillment
of the requirements for

PhD degree in Computer Science

John Weng
Major professor

Date 8/20/96

MSU is an Affirmative Action/Equal Opportunity Institution

O-12771

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
FEB 0 2 1998		

MSU is An Affirmative Action/Equal Opportunity Institution
c:\circ\datedue.pm3-p.1

Towards a Learning System for Robot Hand-Eye Coordination

By

Sally Jean Howden

A Dissertation

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Department of Computer Science

1996

Abstract

Towards a Learning System for Robot Hand-Eye Coordination

By

Sally Jean Howden

Through careful consideration of the Hand-Eye Coordination (HEC) problem, it can be viewed as the *process* of performing a sequence of transformations from an input space to an output space. Specifically, the entire process from eye to hand can be viewed as a mapping from *scene space* to *arm configuration space.* This single mapping may be broken into a *sequence* of mappings from one space to another. The sequence we have chosen to model is the following: scene space to image space; image space to camera coordinate system; camera coordinate system to arm/world coordinate system; arm/world coordinate system to arm configuration space. Additionally, an active vision system is incorporated which introduces an image space to head configuration space mapping.

Given the view that these subtasks are mappings from a N-dimensional input space to a M-dimensional output space, this research presents a unified framework by which the various subtasks of the HEC problem may be implemented. This framework uses a recursive partitioning algorithm to build a hierarchical tree classifier which uses a nearest neighbor classification based on the Voronoi tessellation as its decision making criteria. The resulting data structure is a *Recursive Partition Tree* (RPT), which is the heart of the framework. The topology of the RPT is not determined

a priori, or hand-coded. Instead the topology is allowed to develop during its construction, based on the given set of training samples and the order in which they are presented to the construction algorithm. Each node of the RPT represents a cell of the space which is further partitioned by its children via a *Voronoi tessellation.* Each leaf node corresponds to a training sample and stores the corresponding output. This general framework provides us with a method for systematically dealing with the complex relationship between the sensors and the manipulator. In the performance phase, given an input, the RPT is used to retrieve the desired output. The RPT results in a logarithmic average time complexity in the number of stored training samples.

Extensive simulations have been performed with two implemented modules. Experiments using a real setup demonstrate the ability of a system using RPTs to accomplish the stereo calibration, head-to-object space mapping, and point-to-point movement of the hand within the HEC task. Experiments using real data required that the current approach be simplified since collecting real data for training proved to be much more difficult and time consuming than generating simulated data. Nevertheless, promising results are obtained.

To an Awesome God—all glory and honor to Him
who is able to do all things.
To Edward Howden, Jr. and Nina Jean Howden,
the wonderful parents to whom He gave me,
for their undying love and support,
and the occasional kick in the pants.
To John Edward Howden,
the best "little" brother a sister could have,
for the love and the laughter,
and always being there when I need you.

COR MEUM TIBI OFFERO DOMINE PROMPTE ET SINCERE

ACKNOWLEDGMENTS

> Do you not know? Have you not heard?
> The Lord is the everlasting God, the Creator of the ends of the earth.
> He will not grow tired or weary, and His understanding no one can fathom.
> He gives strength to the weary and increases the power of the weak.
> Even youths grow tired and weary, and young men stumble and fall;
> but those who hope in the Lord will renew their strength.
> They will soar on wings like eagles;
> they will run and not grow weary,
> they will walk and not be faint.
>
> Isaiah 40:28-31 (NIV)

> "For I know the plans I have for you," declares the Lord, "plans to prosper you and not to harm you, plans to give you hope and a future."
>
> Jeremiah 29:11 (NIV)

There are so many people to thank; a few words on paper hardly seem adequate, but I will try. First, all glory and honor must go to God for paving the way and helping me to stay the course. Without His guidance, this would not have been possible.

To my thesis advisor, Dr. John J. Weng, I owe a debt of gratitude for his patience and wisdom; for being a sounding board; and for continuously pushing me to be the best researcher I could be. For the many long discussions which ensued during the development of this research I will always be grateful. His dedication to pushing the field of Computer Vision beyond the current limits, and the intensity and enthusiasm with which he pursues that goal have been both inspiring and challenging.

I would like to thank Drs. Anil K. Jain, William Punch, Lal Tummala and Jim Zacks for serving on my Ph. D. committee—their insightful comments and probing questions regarding my research are deeply appreciated. A special thank you goes to Dr. Anil K. Jain for introducing me to the field of computer vision and image processing; for bringing me into the Pattern Recognition and Image Processing Lab; and for guiding me in my first fledgling attempts at being a researcher. His door has always been open for honest discussions of research, as well as other, concerns. The support and research opportunities that he has provided for me have played a large part in my current success.

The faculty of the Department of Computer Science are some of the best, and I am honored to have studied under them. The importance of the support and advice given by Drs. George Stockman, Richard Enbody and the late Richard Dubes cannot be emphasized enough. And what can I say about Ruth Barton except that she has been more than a friend and mentor throughout these many years and my life would be greatly diminished if I had not known her.

The years have been filled with many good friends and able colleagues. Their keen minds have contributed to my growth as a researcher. Many have allowed me to use them as sounding boards when I needed to talk my way through some part of my research. These old (should I say former?) PRIP researchers include: Jinlong Chen, Shaoyun Chen, Jon Courtney, Yuntao Cui, Hansye Dulimarta, Farshid Farrokhnia, Patrick Flynn, Marie-Pierre (Dubuisson) Jolly, Greg Lee, Joe Miller, Aniati Murni, Tim Newman, Sharathcha Pankanti, Narayan S. Raja, Nalini Ratha, Steve Walsh, Marilyn Wulfekuhler, Yu Zhong. Still the following people must be singled out because I owe them a debt that can never be fully repaid: Deborah Trytten, for being my mentor and friend early on, and for showing me the ropes (and how not to hang myself with them!); Qian Huang, for not letting me give up, for

opening my eyes to take a better look at the world, and for sharing her life and her family with me; Chitra Dorai, for sharing with me in this final push to the finish line and helping me to keep my sanity through to the end; and Daniel Swets, for the honest and open advice, encouragement, moral support and friendship these last couple of years. I must also acknowledge the next generation of PRIP researchers, especially: Karissa A. Miller for allowing all of the PRIP system administration duties to be thrust upon her, and doing such an admirable job of keeping things running; Paul Albee and Laura Blackwood for taking over responsibility for the Macintosh; and Wey-Shiuan Hwang, for supplying software which helped to speed my research to its conclusion. This advice I leave you with: work hard; enjoy life; keep your goal clearly in mind; and never, never give up.

I truly appreciate all of the help and support I have received from the departmental staff, who always seemed to have the answers to my questions, or were willing and able to find the answer, no matter how strange the questions were. Cathy Davision, Kathy Harmon, Lora Mae Higbee, Brenda Minott, Linda Moore, Michelle Sidel, Tammy Syron, Tammy Underwood, and Beverly Wallace: Thank you all!

Over the years I have benefited tremendously from the fact that the Computer Science Department's computing resources have been so ably maintained by some of the best people in computing. I especially acknowledge the dedication of Frank B. Northrup and Lisa J. Lees for their wisdom, knowledge, forethought, and just plain hard work in keeping the department on the leading edge of computing technology. I hold many fond memories of the time I worked with the system administration staff, and especially thank the following for helping to keep the facilities in top working condition: Mark Brehob, Lily Cheng, Travis Doom, Edgar Kalns, Lynn Kubinec, Jay Kusler, Karissa A. Miller, Bill Moore, Sherry Moore, Chris Penney, Wenjian Qiao, Issac Tsai, Mark Urban-Lurain, Jennifer White. Thank you all for your camaraderie.

I also gratefully acknowledge the support of Distributed Computing Consultants, Inc., specifically Kevin and Miche Suboski, for their support over this last year, and allowing me to take the time I needed to complete my research and thesis.

To those friends who have lifted me up—whether in prayer, by an encouraging word or by a kind deed—I express my heartfelt gratitude. To list you all would take a second volume, yet I would mention the following few: Barb Czerny, Maureen A. Galsterer, Lisa J. Lees, Karissa A. Miller, Frank B. Northrup, and Gayle A. Northrup.

And finally I must acknowledge my family—for without them I would never have dared to reach for such a lofty goal. The love and support of aunts, uncles, cousins and grandparents is immeasurable. Yet nothing matches the steadfast love and support of my parents Edward Howden Jr. and Nina Jean Howden, and my brother John Edward Howden, who faithfully stood by me all of these long years, always expecting that I would indeed finish, even when I myself was doubtful. I tried to think of a gift I could give you to say thank you, but you already have the only gift that could suffice: my love. And so I dedicate this thesis, these years and the hard work with which they are filled—to you.

Table of Contents

LIST OF TABLES xii

LIST OF FIGURES xiii

1 Introduction 1
1.1 Manipulators with Visual Sensors 1
1.2 The Concept of Learning 5
1.3 The Task of Hand-Eye Coordination 6
1.4 The Active Vision System 7
1.5 Statement of the Problem 8
1.5.1 The Problem 8
1.5.2 Importance of the Problem 9
1.5.3 Desirable Properties 10
1.6 Overview of the Thesis 15
1.7 Major Characteristics of the Work 18
1.7.1 Complete System 18
1.7.2 Unified Framework 18
1.7.3 Recursive Partition Tree 19
1.7.4 Adaptability/Flexibility/Robustness Via Learning 19
1.7.5 Active Vision 20
1.8 Organization of the Thesis 20

2 Literature Review 22
2.1 Psycho-Physiological Background 22
2.1.1 Proprioceptive Control 24
2.1.2 Oculomotor Control 25
2.1.3 Visual and Proprioceptive Combined Control 27
2.1.4 Cognitive Control 30
2.1.5 Discussion 30
2.2 3D Object Recognition 31
2.2.1 Early Recognition Systems 31
2.2.2 Recent Popular Recognition Strategies 35
2.2.3 Neural Network Inspired Recognition Systems 39
2.3 Manipulator Control 42
2.4 Hand-Eye Coordination 46
2.4.1 Calibration 47
2.4.2 Image Space to Camera Space Calibration 48

2.4.3 Camera to End-Effector Calibration 49
2.4.4 Camera Space to World Space Calibration 50
2.4.5 World Space to Arm Joint Space Calibration 51
2.4.6 HEC Systems 52
2.4.7 Summary 63
2.5 Neural Networks 65
2.5.1 Biological Neural Networks 66
2.5.2 Artificial Neural Networks 67
2.5.3 Kohonen Maps 70
2.6 Active Vision 72
2.6.1 Purposive and Qualitative Active Vision 74
2.6.2 Vergence Control 77
2.6.3 Tracking 78
2.7 Decision Making 83

3 Framework for Learning and Performing Hand-Eye Coordination 88
3.1 Motivation 88
3.1.1 The Proposed HEC System 88
3.1.2 Generalization of Sub-Systems 92
3.1.3 The Framework's Module 94
3.2 Recursive Partition Tree 97
3.2.1 Definitions 98
3.2.2 Retrieval Phase 107
3.2.3 Training Phase 110
3.2.4 Discussion 119
3.3 Function Approximation 120
3.4 Some Properties of Framework's Performance 122

4 Experimental Results 125
4.1 Performance Study of the CSS and CAS Modules in Simulation 126
4.1.1 Experimental Setup for Simulation 127
4.1.2 Estimation of Transformation Matrix for Simulation 129
4.1.3 Experiments with the CSS Module in Simulation 130
4.1.4 Experiments with the CAS Module in Simulation 138
4.2 Experiments with a Real Setup 142
4.2.1 Experimental Setup 143
4.2.2 Software Communication 146
4.2.3 Camera-Centered Stereo System 148
4.2.4 Sensor Movement Control 151
4.2.5 Object Grasping 153
4.2.6 Temporal Sequences 154
4.2.7 Visual Guidance 156
4.3 Summary 158

5 Contributions and Future Work **160**
5.1 System Overview 160
5.2 Contributions 161
5.2.1 Unified Framework for Sensor-Actuator Coordinated Learning 161
5.2.2 Simple Object Recognition and Location Using the RPT Framework . 162
5.2.3 Manipulator Control in a Redundant System 162
5.2.4 Controllable Sensor in a Learning Framework 163
5.2.5 Addresses Several Modules in Hand-Eye Systems 163
5.3 Summary 164
5.4 Future Work 165

APPENDICES **167**

A Constructing an RPT Incrementally **167**

B Adoption Problem **172**

C Empirical Study of RPT Error **174**

D Investigation of Back Propagation Neural Network **177**
D.1 Experiment One: $f(x,y) = x^2 + 2.3y^2$ 178
D.2 Experiment Two: $f(x,y) = sin[(2x+y)\pi], -1.0 \leq x,y \leq +1.0$ 178
D.3 Experiment Three: $f(x,y) = sin[(2x+y)\pi], -2.0 \leq x,y \leq +2.0$ 181
D.4 Experiment Four: Number of Hidden Layer Nodes Increased 182
D.5 Experiment Five: Number of Training Cycles Increased 183
D.6 Experiment Six: Number of Hidden Layer Nodes Decreased 186

List of Tables

2.1 Cordo and Flanders differentiation between visual and kinesthetic control of arm movements, and correction of positional errors. 27
2.2 Table comparing HEC approaches in the literature. 64
2.3 Table comparing presented method and that of Schulten et al. 65

4.1 Table showing the success rate of the *approach grasp point* and *grasp* trials. 154
4.2 Table showing the success rate of the *approach grasp point*, *grasp*, *approach delivery point*, *pour* and *replace* trials. Ten training samples were used to train this system, and all ten samples were used in the re-substitution test. Ten different random samples were used for the random test. 156
4.3 Table showing the success rate of the approach grasp point and grasp when the simple vision module is used to locate the object and end-effector. 158
4.4 Table showing the time in seconds taken to locate an object (cup or end-effector) when using the simple vision module and when not using the simple vision module. 158

A.1 Example of spacing parameter for $L = 6, a = 2$. 168

List of Figures

1.1 A sequence of images showing movement of the PUMA 560 robotic manipulator and the Pan/Tilt Unit (foreground). 2
1.2 Components in an Hand-Eye Coordination System. 5
1.3 Hand-Eye Coordination as an intersection of Computer Vision and Robotic Manipulation. 6
1.4 Example of possibly unnecessary, large movement of arm due to static mapping from desired position to arm configuration. 11
1.5 Schematic of proposed system. 14
1.6 Concept of a general module for the framework. 16

2.1 Schematic of Sharma *et al.*'s approach. 59
2.2 An initial concept of a biological neuron. 66
2.3 Schematic of a McCulloch-Pitts neuron. 68
2.4 Schematic of a simple perceptron. 68
2.5 Example of a feedback neural network. 71
2.6 An example of a decision tree complete with costs for taking action, the utility for choosing a certain course of action, and the probability of each outcome occurring. 84
2.7 A hierarchical classification tree. The features $f_1, \ldots, f_6$ are used to decide between class $c_1, \ldots, c_6$. 85
2.8 An example of a classification tree based on non-numeric features. . . . 86

3.1 Schematic of proposed system. 89
3.2 Vector specifying direction from hand to object. 90
3.3 The organization of the module of the general framework. 95
3.4 Relationship between core and shell processes. 97
3.5 Example of the Voronoi tessellation of a bounded two-dimensional space, S^2. The dots represent three vectors within the two-dimensional space. In (a) the lines labeled a, b and c represent the distances between the three vectors, such that $a > T$, $b > T$ and $c > T$ where T is some specified radius. In (b) the dark lines represent the boundaries between the subregions assigned to each of the three vectors. The union of the three subregions define a partition of S^2. 100
3.6 Example of the Voronoi tessellation of a subspace of S^2. The small dots represent three new vectors within the subspace S^2, in addition to the vector represented by the large dot. 101

3.7 Example of a partitioning of space S^2 by partitioning each of the subspaces from the initial partitioning, via a Voronoi tessellation. 101
3.8 Example of a partitioning of space S^2 and its representation by a Recursive Partition Tree. At level $l = 0$ the entire space S^2 is represented by the root node of the RPT. 103
3.9 At level $l = 1$, S^2 is partitioned into three subspaces, each represented by a node at level $l = 1$ of the RPT. 104
3.10 At level $l = 2$ one of the subspaces into which S^2 was partitioned at level $l = 1$ is further partitioned into four subspaces, each represented by a node at level $l = 2$ of the RPT as a child of the node representing the larger subspace at level $l = 1$. Notice that the parent node is also represented at level $l = 2$. 104
3.11 The remaining subspaces of level $l = 1$ are further partitioned at level $l = 2$. 104
3.12 Schematics showing the training and testing processes. 105
3.13 Schematics of retrieval process given query vector Q. 107
3.14 Example of retrieval process for a two-dimensional space: location of query vector. 108
3.15 Example of retrieval process for a two-dimensional space: level $l = 1$. . . 108
3.16 Example of retrieval process for a two-dimensional space: level $l = 2$. . . 109
3.17 Example of retrieval process for a two-dimensional space: level $l = 3$. . . 109
3.18 Schematics of the level l construction process given the set of training samples $Q_1, \ldots, Q_n$. 112
3.19 The root node of the RPT represents the entire space S^2. The training samples are presented in batch in the order shown. 113
3.20 A is added at level $l = 1$. 113
3.21 B is added at level $l = 1$. 113
3.22 a and b are not added at level $l = 1$. C is added at level $l = 1$. 114
3.23 No other nodes are added at level $l = 1$. a is added at level $l = 2$. 114
3.24 b is added at level $l = 2$. 114
3.25 c is added at level $l = 2$. 115
3.26 d is added at level $l = 2$. 115
3.27 e is added at level $l = 2$. 115
3.28 α, β, χ, and δ are not added at level $l = 2$. f is added at level $l = 2$. . . 116
3.29 ϵ is not added at level $l = 2$. g is added at level $l = 2$. 116
3.30 No other nodes are added at level $l = 2$. α is added at level $l = 3$. . . . 116
3.31 β is added at level $l = 3$. 117
3.32 χ is added at level $l = 3$. 117
3.33 δ is added at level $l = 3$. 117
3.34 Finally, the last node, ϵ, is added at level $l = 3$. 118
3.35 ϕ is never added because it is closer to d than the deepest level's radius. 118
3.36 Q is not guaranteed to reach it's nearest neighbor, e, because at a higher level, Q is closer to C than it is to B. 119

4.1 Schematic of simulated camera system. 128

4.2 Relationship between sample and derived points. 129
4.3 Six additional points generated for training CSS module. Center point is the "input" point. The other six are generated so that they lie in the positive/negative X, Y and Z directions at a distance of R, where R is approximately equal to the radius of the neighborhood at the given level of the hierarchy. 131
4.4 The Normalized Error (NE) from experiments with the CSS module. Units are in millimeters (mm). (a) X, Y, Z combined error; (b) X, Y combined error; (c) Z error. 134
4.5 Average time in microseconds versus levels in the RPT for a random sample input to pass through the CSS network and find the nearest neighbor in the search space. 135
4.6 Average time in microseconds versus number of training samples for a random sample input to pass through the CSS network and find the nearest neighbor in the search space. 136
4.7 Average number of nodes visited by a random sample input as it descends CSS network. 136
4.8 Number of nodes in CSS network versus the number of training samples used to construct it. 137
4.9 Example of moving test points with camera system. 139
4.10 The Normalized Error (NE) from experiments with the CAS module. Units are in millimeters (mm). (a) X, Y, Z combined error; (b) X, Y combined error; (c) Z error. 140
4.11 Number of nodes in CAS network versus the number of training samples used to construct it. 142
4.12 Schematic of implemented system. 144
4.13 Setup for real experiments. 145
4.14 A snapshot of the experimental setup showing the PUMA 560 robotic manipulator and the camera system consisting of two Panasonic cameras mounted on a Direct Perceptions Pan/Tilt Unit. (a) Observer's view. (b) View of left camera. (c) View of right camera. 146
4.15 The grid used in the camera-centered stereo system experiment. (a) Observer's view. (b) View of left camera. (c) View of right camera. . . 149
4.16 The performance of image space to camera space mapping using a RPT trained with 365 samples and using the $KNDB$ interpolation algorithm to compute the output vectors. The results shown here are from testing the module with 200 samples. Various values for the interpolation parameter α are used. 150
4.17 The performance of the sensor control system using a RPT trained with 45 samples and using the $KNDB$ interpolation algorithm to compute the output vectors. The results shown here are from testing the module with 100 samples. The interpolation parameter is $\alpha = 3.0$. 152
4.18 Demonstration of a temporal sequence of five subtasks as seen by the left (top three rows) and right (bottom three rows) cameras. 155

4.19 Demonstration of a grasping task using the simple vision as seen by the left (row 1) and right (row 2) cameras. 157

A.1 Schematics of RPT Construction Method I. 170
A.2 Example of problems with skinny branches. 171

B.1 Demonstration of a node becoming invisible. (a) Before insertion of node C. (b) After insertion of node C. 173

C.1 Case I. 175
C.2 Case II. 175
C.3 Case III. 176
C.4 Case V. 176

D.1 Function: $f(x,y) = x^2 + 2.3y^2$, $x,y = 0.0, 0.1, 0.2, \ldots 0.910.0$; 200 training patterns; one hidden layer with 15 neurons, resp.; 50,000 training cycles 179
D.2 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-1.0 \leq x,y \leq +1.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; best results out of ten trials. 180
D.3 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-1.0 \leq x,y \leq +1.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; worst results out of ten trials. 180
D.4 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; best results out of ten trials. 181
D.5 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; worst results out of ten trials. 182
D.6 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 50,000 training cycles; best results out of ten trials. 183
D.7 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 50,000 training cycles; worst results out of ten trials. 184
D.8 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles. 184
D.9 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles. 185
D.10 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles. 186

D.11 Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 500,000 training cycles. 187

Chapter 1

Introduction

1.1 Manipulators with Visual Sensors

A robotic manipulator with visual sensors is a system which is capable of performing actions upon or with (manipulating) objects within its environment. More specifically, we are referring to the ability of a robotic manipulator to manipulate objects within its workspace using visual information. This is more commonly referred to as *Hand-Eye Coordination* (HEC). Figure 1.1 shows a sequence of images in which a HEC system is seen manipulating a set of blocks. Humans present an example of HEC in action when they use the information gathered by their vision system (the eyes) to guide the actions of their manipulation system (the arm and hand) in order to facilitate performing actions upon or with the objects they see in their environment. Furthermore, it is obvious that HEC can be done well by humans—even in very unconstrained, noisy, and unknown environments. There are many areas in which robotic manipulators are presently utilized, most notably in manufacturing settings. For example, manipulators are used in automobile factories to perform heavy lifting, welding/riveting tasks and parts picking. In order for such systems to operate

robustly they must be able to sense deviations in target position and adjust their movement to perform the desired task. Many of these systems rely on the accuracy of the system calibration and arm movement. Some systems receive no feedback on their performance, and those that do assume an accurate sensor calibration. If these assumptions are unmet, then the system is unable to complete its task. The work presented in this thesis seeks to use visual feedback in a fast and efficient manner to move towards systems which can operate in unconstrained, noisy and unknown environments.

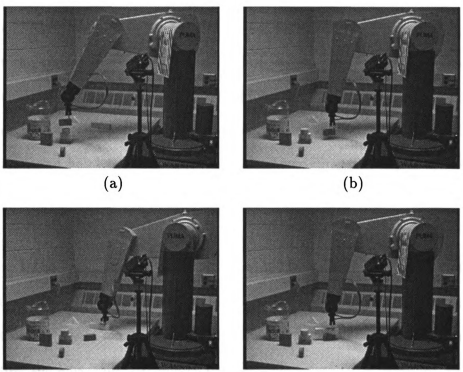


Figure 1.1: A sequence of images showing movement of the PUMA 560 robotic manipulator and the Pan/Tilt Unit (foreground).

A few basic tasks of HEC are: (1) locating an object within a scene; (2) moving the manipulator to the object; and, (3) once the object has been grasped, performing the desired manipulation. Many researchers in Computer Vision have concentrated

on performing the first task. On the other hand, many of the Robotics researchers have concentrated on performing the second and third tasks. Recently, though, researchers in both fields have been moving towards systems which incorporate the various techniques of each field so as to exploit the benefits of both to aid in the performance of both. Indeed, it seems reasonable to address ourselves to the task of investigating how the tasks of vision and manipulation are interrelated. How would the supposed interdependency change each of the tasks? What do we need from the computer vision system in order to perform the manipulation task? Why not use existing *pose* estimation systems? These systems have a goal of computing the 3-dimensional (3D) pose of an object[1]. Granted, we need object recognition in order to detect the object we are interested in, but do we use that kind of information when performing manipulation? That is, do we need the *exact* pose of the object? While this is a matter of debate, we propose that we need to use only the locations of the object and hand as seen within the two stereo images, relative to each other, to perform manipulation. This, then, changes how we design the vision system for our robotic manipulator.

During Hand-Eye Coordination, it is desirable that the object of interest be near the center of the eye's gaze. It is well known that the optics of common camera lens are in-exact and there exists radial distortion which corrupts to higher degrees the farther from the lens center. Humans, likewise, often perform a head/eye adjustment to place objects near the center of gaze. Once we have obtained data from the vision system it may be used to adjust head and/or eye position so that the object is near the center of gaze. This may take one or more data collection/adjustment steps before the object is *close enough* to the center of gaze. The reasoning behind this approach

[1] that is, the (x, y, z) parameters specifying the translation in the coordinate system, and the (θ, ϕ, γ) vector specifying the rotation parameters within the coordinate system

is that there is a higher acuity at the fovea than at the periphery of the eye, providing keener perception in that area of the sensing system.

As for head/eye adjustments, the arm/hand movement may not be a single movement, but rather a continuous, fairly smooth series of movements which begin with a fast, large, gross move followed by successively slower, smaller and more precise moves until the object is grasped by the hand. Such a series of movements may allow the entire system the time to make adjustments in the arm movement to compensate for miscalculations due to noisy, corrupt, or changing data. A single movement may require an extremely precise estimation of object position as well as extremely precise arm movements, which may not be practical.

Finally, once the object has been seized, the third task—that of manipulation—may be performed. This task requires high level control and goal planning, but would possibly consist of a series of *complex* steps—each consisting of the previous two tasks where the object is replaced by another object or a specified location. For example, placing an apple in a basket would consist of acquiring the apple as described above, and then locating and looking at the basket, followed by moving the hand—which now contains the apple—towards the basket (or more precisely a point at the inside bottom of the basket). Of course, more complicated tasks would require more complicated sequences of this type. Figure 1.2 contains a diagram of the components of a robotic manipulation system and how they fit into the design of such a system. There are several important issues to consider in designing a robotic manipulation system. These issues include: the concept of *learning*; the task of hand-to-eye coordination; and the use of an active vision system. Each of these issues will be discussed further in the following sections.

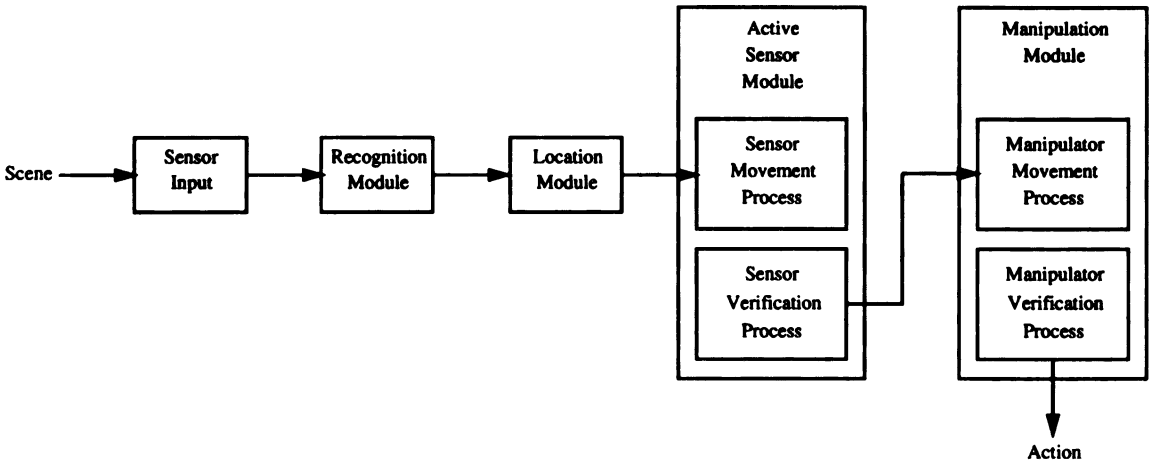


Figure 1.2: Components in an Hand-Eye Coordination System.

1.2 The Concept of Learning

Learning may be described as the process of acquiring knowledge. In humans, learning is a complicated process involving the architecture of the brain and central nervous system—a process we do not yet fully understand, though much work has been carried out in psychology and neural networks. The work in this thesis is concerned with the learning of the relationship between given stimuli or inputs, and the responses or outputs to which they correspond. This learning is performed through repeated presentation of the stimuli and their corresponding responses. Thus, given a set of possible inputs, and a set of possible outputs, the system learns which input corresponds to which output. Then, when presented with a specific input, the response is (hopefully) the correct output.

Learning provides a system with the ability to *generalize* its current knowledge (about the set of stimuli with which it is familiar) in order to apply it to additional stimuli. In this way, the system is able to adapt to changing conditions in its environment.

This concept of learning is missing in many research efforts at creating a robotic

manipulation system. The movements of the manipulator are often wired, or hand-coded, into the software. Unfortunately, this circumvents the desirable attributes of adaptability and generalization. If robotic manipulators are to be able to work in varying, sometimes unknown environments, and be able to manipulate many different objects, without constraint, then it is clear that there must be some mechanism of learning built into the system. This learning mechanism, whatever its form, will be essential in the development of the ability to coordinate the subsystems of vision and hand/arm movement.

1.3 The Task of Hand-Eye Coordination

Hand-eye coordination is the ability to use the information gathered by the eyes to guide the movement and activity of the hand/arm mechanism to perform a specified task [94], [78], [65], [66], [86], [69], [5], [15], as well as to use the task being performed to guide the sensing [7], [31], [19], [70], [81]. This task may be to take hold of an object, or move an already held object to a new location. From what the eyes *see*, the brain computes where the object is with respect to the body. Given that information, the correct signal must be sent to the motor mechanisms in the arm and hand such that the hand is placed in a position to grab the object. Turning our attention to a

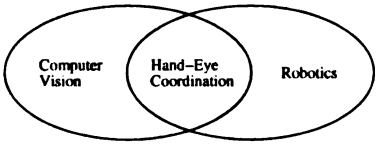


Figure 1.3: Hand-Eye Coordination as an intersection of Computer Vision and Robotic Manipulation.

robotic manipulator we see that the task of hand-eye coordination crosses the boundary between Computer Vision and Robotic Manipulation (Figure 1.3). Computer

Vision techniques are needed to gather relevant information from the environment such as whether or not the object of interest is visible, where the object is with respect to the body, how it is positioned, etc. What information is relevant depends on the manipulation task to be performed. However, without the ability to manipulate an object, the information obtained from the vision system would be irrelevant to a manipulation task. Thus, we see that the Robotic Manipulation techniques must also be addressed. So far, much of the research in these two fields has been done independently of each other. Only recently have researchers really begun to integrate these two areas and look at the hand-eye coordination task as a complete package consisting of a vision system and manipulation system intertwined with each other, and to some extent depending on each other. When an active vision system is introduced into the system, the task becomes still more complicated.

1.4 The Active Vision System

What do we mean by *active vision*? Active vision is the process of controlling the movement of the head/eye system so that what is visible, and hence the information gathered from the scene, is relevant and useful to performing the specified manipulation task. For example, if we wish to take hold of an object, then it is helpful if the eyes are positioned such that the object is near the center of gaze so as to minimize the amount of lens distortion. It is also helpful to be able to follow the object, while grasped by the hand, as it moves (a process commonly known as *tracking*). Furthermore, while it may not be necessary to see the hand during the initial phase of arm movement, viewing the hand and object simultaneously provides crucial information on the positions of hand and object relative to each other. This greatly minimizes the difficulty in performing the specified task. While it is certainly true that a human

could close their eyes and locate an object relying solely upon their sense of touch (assuming the object was within reach), this might not be the most efficient approach to emulate in artificial systems—both in terms of time and computational effort.

1.5 Statement of the Problem

Specifically we will address the topic of designing a robotic manipulation system which can perform point-to-point movement of the manipulator with the goal of grasping an object in an unknown position. We assume that stereo intensity images of the scene are available. All real time experiments are performed using a PUMA 560 robotic manipulator. A pan/tilt unit with a range of 360 degrees in the pan direction and a range of 160 degrees in the tilt direction is used to demonstrate that an active vision system can be controlled using the framework presented here. However, the active vision system itself is not incorporated into the current version of the system for performing point-to-point movement of the robotic manipulator.

1.5.1 The Problem

We assume that there exists a vision system which can detect a specified object within an image of a scene. Given this information, we present a framework for designing a system which can control a robotic manipulator to grasp that object through a series of movements based on the relative positions of the object and manipulator's end-effector within the stereo images. The following tasks were investigated in the course of this work:

- Incorporating the concept of learning into the system in order to gain adaptability and generalization.

- Designing a system that is able to cope with incomplete, inaccurate and/or noisy data—both in the input data and the output action of the system.

- Designing an experimental system which considers the requirements and abilities of both the visual as well as the manipulation systems.

- Incorporating an active vision system to assist in obtaining useful information from the visual system.

- Devising a general framework which facilitates the building of all system components in a uniform manner.

1.5.2 Importance of the Problem

Many of the previous approaches to HEC are rule-based. That is, the constraints, expectations and movements of the systems are hand-coded into the system. Additionally, the relationships between the various components of the system are often pre-computed and hand-coded into the algorithm. One side-effect of this type of approach is that whenever some component of the system changes, the relationship between it and the other components must be re-computed and coded. This is not an unlikely scenario. The imaging sensors may be replaced with new ones. The sensors might be moved to a different location. The active vision platform may move or simply become misaligned due to wear and tear; and likewise for the manipulator. The end-effector may be replaced with one which is shaped differently, or of a different size. Any, or all of these complications may occur. The result is a system which is inflexible and brittle, unable to adapt to small changes in the environment or incorporate any new information.

The approaches which do incorporate a learning mechanism usually map one or

more object positions into a *single* configuration[2] of the manipulator which is expected to place the hand at the desired location. That is, a single configuration represents an entire 3D patch of the visible space. During runtime, the manipulator moves into the configuration specified for the current target position. In order to obtain any degree of accuracy there must be a dense mapping from the visible input space to the configuration output space. Furthermore, this action usually consists of a single *absolute* movement. However, this type of movement requires a high degree of accuracy in all stages of the imaging process, in locating the object in the image, in mapping that location into the space of all possible manipulator configurations, and finally, in the actual movements of the manipulator. Expecting this high a degree of precision may not be practical.

Additionally, given that visual information may be corrupt and/or noisy, and that manipulation systems may not be as precise as needed to accomplish a given task, it is reasonable to assume that a mechanized HEC system will at times fail to reach the target position. It seems reasonable to incorporate a looping control into the HEC so that the system may re-evaluate, adjust, and continue moving until the goal is obtained. In this way, the system will not be dependent on making a single precision movement in order to accomplish its task.

1.5.3 Desirable Properties

In this section we discuss the properties which we desire our proposed system to exhibit. These properties are:

- Redundancy

[2]Here configuration implies a vector of angle values, one for each joint of the manipulator, each specifying the angle to which the corresponding joint should be moved

- Robustness/Resiliency
- Flexibility
- Lower Development Cost

Redundancy One of the issues that a HEC system must take into consideration is that there is an infinite number of configurations of the arm and hand which will place the hand at any single, reachable position. This redundancy in the parameter space is difficult to analyze. Most of the rule-based approaches consider this redundancy to be a problem to be circumvented; and thus, for each possible target position in space, a single arm configuration which reaches that position is hand-coded into the system. This strategy is further strengthened by the attempt to reach the targeted position via a single arm movement. If, however, we can devise a system capable of learning how to move the arm through a sequence of movements, then we can take advantage of the rich mapping from arm configuration space to target position. Additionally, we can avoid excessively large movements of the arm (see Figure 1.4 for an example).

Robustness/Resiliency Another issue is the requirement that the system be able to place the hand at the target position in spite of corrupt sensor data and inaccurate

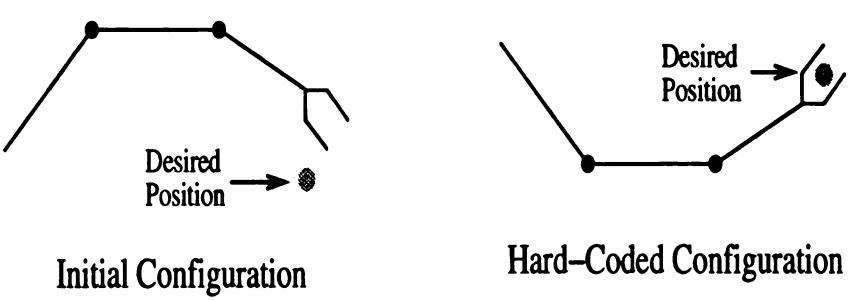


Figure 1.4: Example of possibly unnecessary, large movement of arm due to static mapping from desired position to arm configuration.

arm movements. There are two components to this issue: robustness and resiliency.

Robustness describes the ability of the system to deal with corrupt data. Data corruption comes in many forms. First there is the problem of taking finite samples of continuous data. A second problem is caused by low resolution of the input sensor. A third is the distortion caused by the non-linear optics of the camera system. Various types of distortion are discussed in [88] including: radial distortion, De-centering distortion, and prism distortion. In conventional approaches these sources of corruption must be analyzed and modeled by humans in order to be explicitly incorporated into the system. *Resiliency* describes the system's ability to recover if the final position of the hand is different from the object's position. This may be due to the object shifting position, or due to inaccuracies in the movement of the arm, or due to a miscalculation of that position. This issue is not addressed in most of the current systems. Rather, a single arm movement is made and the error between desired and actual position is analyzed; but no attempt is made to correct the positioning. Once again we need a system which is capable of moving the arm through a sequence of moves. Additionally, we need interactive feedback from the sensor system in order to keep track of the relative position of the object and hand.

Flexibility Many of the HEC systems are inflexible because they are tailored to work for a specific robot manipulator, in a specific setup, in a specific and fixed relation to a specific sensing system (see [66], [53] and [35] for examples). If a change is made to any part of the setup, i.e., different robot arm, different sensing position or sensing apparatus, etc., then the entire model often has to be re-analyzed and recoded. Furthermore, most HEC systems assume a static viewing position. The human system, on the other hand, is relatively flexible in nature. Humans are able to place their heads, and therefore their eyes, in many different positions/orientations with respect to an object, and still reach for and grab the object. Likewise, it is desirable to design a HEC system which has this flexibility built into it. Learning

provides two advantages over hand-coded rules. First, it provides the ability to learn multiple viewing configurations. Second, whenever the HEC system's setup changes, only a new training phase is required—something which can be accomplished without the aid of an expert.

Development Cost Another issue is that of development cost. Why is this an issue for most conventional systems? Given a target position, these systems make a one shot movement, and expect the hand to be accurately placed at the target position. Now, consider the complexities within the system. First there is the complexity of the distortion present in the input data. Second there is the complexity of the sensing systems which involves zooming, aperture, focusing, pan, tilt, and vergence parameters. Third there is the complexity of the redundant robot manipulator involving N joint parameters. Fourth there is the problem of modeling all possible errors in the system such as in joint values, etc. Finally, there is the complexity of incorporating all of these into a complete system. This makes the attaining of near–perfect performance of the system a costly endeavor for most conventional systems. The reason this is so, is that all of the analysis and modeling of the above mentioned complexities must be done *a priori* by humans and then hand-coded into the system.

Now, although it is important to understand the models involved in the above processes, we need to build a system which can be setup and used by the common person without the aide of an expert in the field(s). At the same time we need a system which can achieve a performance at least as good as, and hopefully better than, previous attempts at solving the HEC problem.

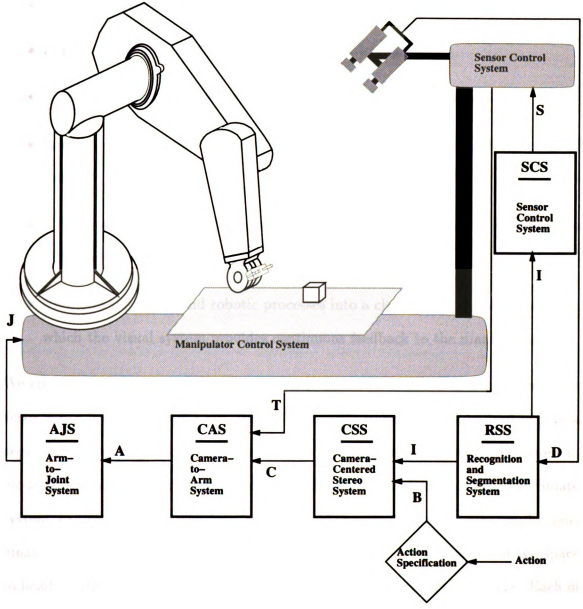


Figure 1.5: Schematic of proposed system.

1.6 Overview of the Thesis

Figure 1.5 shows a schematic of the proposed HEC system. The finished system will be a system which:

- is able to locate an object and end-effector in an image;
- is able to move the end-effector to a target object/position;
- is able to grasp the object of interest and move it to a target position;
- is able to pan and tilt the sensor system so that the object is near the point of gaze;
- is reasonably efficient in the number of arm movements performed and the total distance traveled;
- combines the visual and robotic processes into a closed-loop, coupled system in which the visual system provides continuous feedback to the manipulator.

We choose to view the entire process of HEC, from eye to hand, as a mapping from *scene space* to *arm configuration space.* This single mapping may be broken into a *sequence* of mappings from one space to another. The sequence we have chosen to model is the following: scene space to image space; image space to camera coordinate system; camera coordinate system to arm/world coordinate system; arm/world coordinate system to arm configuration space. Additionally, we include an image space to head configuration space mapping to handle the active vision component. Each of these mappings is represented by a module in Figure 1.5.

Given this viewpoint, we have developed the concept of a general module as shown in Figure 1.6. In this concept, a *module* has an input and an output, and a mechanism by which it computes the transformation from its input space to its output space. In

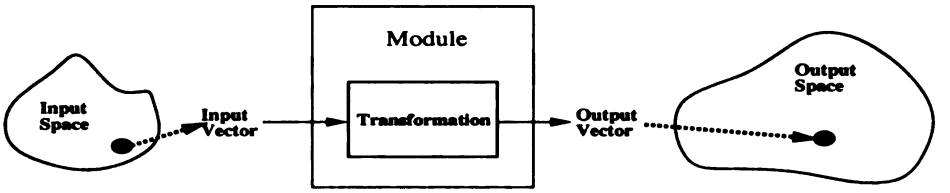


Figure 1.6: Concept of a general module for the framework.

our system, we choose to specify the input and output as vectors, and we compute the output as a function of the input:

$$V_O = \mathcal{F}(V_I)$$

where V_O is the output vector, $\mathcal{F}$ is the mapping function, and V_I is the input vector. Given this general concept we have developed a framework by which the system's modules can be implemented, trained and utilized in a generalized and consistent manner. At the heart of this framework is the *Recursive Partition Tree.* This tree structure hierarchically decomposes the input space from coarse (at the root node) to fine resolution (at the leaf nodes of the structure). Each leaf node represents a subregion within the input space. The leaf node stores the training sample associated with that subregion and its corresponding output vector. While each of these cells represents an $N - dimensional$ patch of input space, the network is not designed to map the entire cell into a single point in the output space as do many of the approaches reviewed in Chapter 2.

This framework provides us with the means of building a complete HEC system which exploits the redundancy inherent in the various sub-systems rather than circumventing them; is robust in the face of corrupt data; is resilient in the face of inaccurate arm movement; is flexible enough to handle changes in viewpoint; is able to execute a sequence of arm movements and accomplish the specified goal of

grasping an object using only the image locations of the object and the end-effector. The modules and hardware needed to realize a complete HEC system are shown in Figure 1.5. The Sensor Control System directs the movements of the active vision system. Control signals are received from the Sensor Control System Module (SCS). Current position parameters, T, are in turn sent to the Camera-to-Arm System Module (CAS). The Recognition and Segmentation System Module (RSS) receives stereo images, D, from the sensors, recognizes the desired object and extracts the object's image parameters—row, column, and disparity or $I = (r, c, d)$. These parameters are sent to the SCS and Camera-Centered Stereo System (CSS) Modules. The SCS uses the image parameters to learn the mapping from the image space to the sensor control parameters space so that the control parameters, S, which are output by the SCS to the Sensor Control System will move the sensors to position the object at the center of attention. The CSS uses the image parameters to learn the mapping from image space to camera space. The output of the CSS is the estimated $C = \left(x^C, y^C, z^C\right)$ position of the object with respect to the sensor's coordinate system. Given C from the CSS and T from the Sensor Control System as input (concatenated to form a single input vector) the CAS learns the mapping from the camera space to arm space. Thus the output of the CAS is the estimated $A = \left(x^A, y^A, z^A\right)$ position of the object with respect to the arm's coordinate system. A in turn is the input to the Arm-to-Joint System Module (AJS) which learns the mapping between the arm space and the joint configuration space. That is, given an input A, the AJS is responsible for outputting a vector of control signals which will move the end-effector of the manipulator to the point in space specified by A. The vector of control signals is sent to the Manipulator Control System at which time they are acted on. The control parameters may include velocity and acceleration parameters. These modules—how they are trained, tested and utilized—are discussed in further detail in Chapters 3, and 4. Unfortunately,

the implementation of the complete HEC system as described is beyond the scope of this thesis. However, several key subtasks are implemented. Simulated and real-time experimental results are presented in Chapter 4 which demonstrate the applicability of the framework to the HEC task.

1.7 Major Characteristics of the Work

1.7.1 Complete System

Most of the existing vision systems are built in isolation from the manipulator system, and vice versa. Then, these two systems are thrust together and expected to work well together. This type of approach leaves itself open to complications caused by incompatibilities between the two systems. The approach presented here is an attempt to view both systems in a systematic way which allows us to build a system in which the vision and manipulator sub-systems are viewed as a single, intertwined system.

1.7.2 Unified Framework

We present a view of the HEC process which allows us to generalize the sub-processes of which the HEC process consists. In our approach, each sub-process is represented by an entity, called a module. Each module has an input and output, and given an input computes the corresponding output. This allows us to develop a general framework by which we can build, train and use the various sub-systems in a systematic way.

1.7.3 Recursive Partition Tree

At the heart of the framework presented here is the internal structure by which the hierarchical partitioning of the input space for a given module is stored. This data structure is called the *Recursive Partition Tree* (RPT). The RPT is constructed based on a set of training data sampled from the input space, and provides a fast and efficient means of searching the sample set for the k nearest neighbors for a given query vector during the retrieval phase.

1.7.4 Adaptability/Flexibility/Robustness Via Learning

Most of the existing approaches to HEC are based on hand-coded rules. Due to this, they are not able to easily adapt to changes in the environment and the system itself. The result is a system which is brittle and easily breaks when such changes occur. Complicating this is that the changes may not be readily observable—such as changes due to wear on the manipulator joints. Furthermore, because of the hand-coded nature of these systems, they are not easily placed into different environments—such as changing the sensors or manipulator. This makes the system inflexible in situations where the environment or system needs to change often and/or quickly. Additionally, when knowledge is hand-coded into the system, the system will not be able to correctly handle any data which is even slightly different from what the system has knowledge about. In our approach we present a system which is capable of obtaining and organizing knowledge about its environment and its relationship to that environment on its own. By adopting a learning approach such as this, the system is able to adapt to changes, quickly and easily; is flexible enough to be placed in any new and unknown environment and be able to function appropriately (after a training phase) with little effort by the user; and is robust enough to perform its

assigned task even in the face of corrupt and noisy data.

1.7.5 Active Vision

We believe that a visual system which is capable of positioning itself to some extent with respect to the object(s) of interest must be an integral part of any HEC system if that system is to succeed in realistic scenarios. Thus, the system presented in this thesis has been designed to include an active vision system. In the current implementation this system is able to adjust itself in the pan and tilt directions only. However, the approach is general enough to handle more complicated movements which include such parameters as zoom and aperture.

1.8 Organization of the Thesis

The concepts, modules and ideas introduced in the above sections will be discussed in further detail in the remaining portion of this thesis. In Chapter 2 work in the psycho-physiological area and the major approaches to 3D object recognition are discussed. Also, a short survey of the work in neural networks is discussed because of its influence on designing the aforementioned framework for approaching this problem. The current literature on approaches to hand-eye coordination and more recently on active vision system is analyzed and discussed. Finally, the art of decision making using various tree approaches is briefly discussed. Chapter 3 introduces the framework for learning and succeeding at hand-eye coordination. The definitions relevant to the construction of the *Recursive Partition Tree*, the query algorithm and the construction algorithm are also presented in Chapter 3. We show how this framework is used to build the system modules discussed above. Simulation and real experimental results which demonstrate the applicability of the framework to the HEC task are presented

and discussed in Chapter 4. The contributions made by this research are presented in Chapter 5 and future work is outlined and discussed in Chapter 5.4.

Chapter 2

Literature Review

The following sections present a brief look at some historical and current endeavors in areas of research which are integrally related to and involved in the research of Hand-Eye Coordination in robotic systems. The following areas will be reviewed:

- Psychological work on vision guided manipulation in humans.
- Prominent work in three-dimensional object recognition.
- Research in robotics, specifically robotic manipulation.
- Major historical approaches to artificial neural networks, and the use of ANNs in guiding robot manipulators.
- Newly active research in using active vision to improve systems which depend on visual input.

2.1 Psycho-Physiological Background

How do we learn to use our limbs? How do we signal to our limbs what we want them to do? What guides our arms, for example, as we reach for an object? What

part does the visual data from our eyes play? Many researchers have attempted to isolate and analyze various functions of the brain including those functions specific to hand-eye coordination, yet much of how our brain and nervous system works is still a mystery. Even seemingly well established theories are not immune to being disproved. For example, until recently a theory called the *equilibrium-point hypothesis* was commonly accepted as an explanation of how the human brain controls the point-to-point movement of the limbs. This theory, first proposed in 1965 by Anatol Feldman, states that a limb is simply launched in the direction of a target point along an *equilibrium-point* trajectory. According to the theory, this trajectory is one which the brain can inexpensively approximate using geometric curves. Once the limb has been launched along the approximated trajectory, the muscles and reflexes are left to bring the limb back into a state of equilibrium. This state should occur such that the limb comes to rest at the desired position [71]. However, recent studies are presenting evidence that this theory may not hold after all. In [34] evidence is cited which indicates that it takes complex computations in order to compute a true trajectory to a given point. A study on voluntary multi-joint movement is performed which measures the stiffness of subjects' arms during limb motion. The results suggest that the human brain is probably in very tight control of the limb while it is moving, to the point of computing the values of the joint angles and the length limb muscles must attain. Gomi and Kawato believe that the brain actually uses a model of the limb in order to perform these computations.

A brief review of some other limb control theories is presented on the following pages. A common approach used to study how subjects perform hand-eye coordination is to somehow change the environment in which the hand-eye coordination occurs. An example of this is the process of *prism adaptation* in which a subject is forced to view the world through a prism which uniformly distorts the input data.

For example, the prism may make everything appear to the left or right of where it actually is. Then the subject is given tasks to perform while wearing the prism. Another approach hides the arm from view of the eyes and then the subjects are asked to perform tasks. This approach is used in an attempt to determine how much of the arm control is due to proprioceptive control independent of visual input.

2.1.1 Proprioceptive Control

The theory of *proprioceptive control* has as its hypothesis that corrections in arm trajectory are controlled only by information on the *felt position* of the arm relative to the body. That is, there exists some mapping from the body coordinate system to the arm coordinate system which provides clues to the body in terms of how the arm is positioned relative to the body. In [25] Efstathiou *et al.* present the results of experiments which tend to disprove the proprioceptive theory. In these experiments, the subjects were asked to point to visible and non-visible targets, before and after adapting to the distorting affects of a prism. In this case, the non-visible target was the arm which had not been viewed during the adaptation phase. The subjects were only allowed to see the *adapted* arm view during the actual adapting phase. Also, the subjects were never given any indication, visual or otherwise, as to how accurate they were or were not in their pointing. Efstathiou *et al.* performed three experiments. First, they analyzed the variance of the shifts made by subject's as they attempted to correct for the distortion between where the object was and where it appeared to be. According to Efstathiou *et al.*, if the proprioceptive theory holds, then the magnitude and direction of shifts in the after adaptation results of this experiment should have been equal for both the visible and non-visible targets. If this is not true, then the visual data must have an affect on the pointing process—in direct opposition to the

proprioceptive theory.

One thing to consider is that the non-visible target in Efstathiou *et al.*'s experiment is an arm which is attached to the body. Thus there is a common ground between the pointing arm and the arm being pointed at. This connection may act as a constraint on the pointing process as there is a known (by the body) relationship between the two arms. In the second experiment subjects were trained to reach for locations and learn these locations only knowing the felt position of the arm. The proprioceptive theory states that there should be a difference in pointing accuracy between the *before* adaptation results and the *after* adaptation results. Efstathiou *et al.* found no significant difference. Efstathiou *et al.* conclude that their results are more conducive to the theory that it is the relationship *between* the positions of the two arms, as felt by the subject, that is important and not the felt position of just the arm trained during the adaptation phase of the experiments.

2.1.2 Oculomotor Control

This theory is based on the belief that in order to make accurate limb movements it is **necessary** that the eyes have moved so as to place the target at the point of gaze. Abrams *et al.* propose that both retinal and extra-retinal information are used in order to accurately locate objects in space with respect to the observer's body [2]. *Retinal* information consists of the stimulus patterns on the retina and provide the location and movement of objects within view of the eyes. *Extra-retinal* information consists of the oculomotor commands which control the movement of the eyes and provides positioning of the eyes. Abrams *et al.* divide limb movement into three phases: *movement-preparation* phase; *initial-impulse* phase; and *error-correction* phase. During the movement-preparation phase the commands necessary

to start the limb movement are assembled. The initial-impulse phase consists of the initiation and execution of the limb movement. This movement may be considered to be ballistic in nature. That is, it is a rapid and continuous change in position which covers most of the distance to the target and is not modified once begun. The last phase consists of minimizing the error in the current limb position and involves discontinuous position and velocity changes.

In their work, Abrams *et al.* study the saccade movements executed by subjects in looking towards an object when it appears in their view. They find that the onset of the eye movement does not appear to depend in any systematic way on the target's position when the target position is known *a priori*, and the limb movement began very shortly after the eye movement. However, eye movement is much faster and so the eyes arrive at the target well ahead of the arm; and the eyes tend to remain on the target until the arm has arrived. In a second experiment, subjects are required to keep their eyes fixed on one target while moving their arms to another target. Results indicate that the farther away a target is located, the more dependent the system is on information received via visual feedback. Here *farther* indicates the distance between the position at which the eyes are fixated and the position to which the arm is reaching or pointing. Abrams *et al.* attribute this to the hypothesis that the longer the limb's movement, the more important retinal information is to the success of the process. Additionally, during the error-correction phase, the movement of the eyes appear to have no effect, but the availability of visual feedback does—when feedback is available, there is a greater tendency to *attempt* to correct errors. Further investigation reveals that the *nature* of corrections is affected by a combination of distance to the target and visual feedback. That is, for closer targets when there is no visual feedback, *each movement* in the change in limb position is nearly zero when no eye *movement* is allowed. These results tend to indicate that the information necessary to make error

corrections is not available under the given conditions. On the other hand, when visual feedback is available, and taking into account the total distance traveled during the correction phase, there is a larger total movement. However, this value is not affected by movement of the eyes. This would seem to indicate that it is visual feedback which provides error-correction information. Abrams *et al.* conclude that the limb control process appears to use extra-retinal information about eye position and movement to the extent that eye movement contributes to the initial-impulse velocity and distance, and eye position contributes to the accuracy of the error correction.

2.1.3 Visual and Proprioceptive Combined Control

Cordo and Flanders differentiate between visual and kinesthetic control of arm movements and correction of positional errors. Their distinction is summarized in Table 2.1 [21]. They hypothesize that both kinesthetic and visual control can be used to rapidly control the initial amplitude of the limb response, trigger that response, and correct errors in trajectory during that response.

Table 2.1: Cordo and Flanders differentiation between visual and kinesthetic control of arm movements, and correction of positional errors.

Kinesthetic	**Visual**
Information from muscle, joint and cutaneous receptors	Information from retinal patterns
Numerous types of receptors	Relatively homogeneous class of receptors
Efferent and *afferent* innervation (muscle, spindles)	Only *afferent* innervation
Movements referenced to internal environment (muscle, skeleton, etc.)	Movements referenced to external environment
Less distinct cognitively	Overt conscious sensation

Redding and Wallace propose a *directionality-of-guidance* theory which states that in hand-eye coordination either visual or proprioceptive guidance may occur—which kind is in control depends on the time at which the hand first becomes visible and the duration of its visibility [73]. In their hypothesis, if the hand is visible from the beginning of the movement and remains visible during the movement, the direction of guidance control is from *eye to hand.* If the hand is not visible until the end of the movement, then the direction of control is from *hand to eye.* By *direction* from A to B Redding and Wallace mean that A guides the movement of B. In the former case, the error between where the hand is seen to be and where it is felt to be, is assumed to be an error in the hand-to-head calibration and so proprioceptive adaptation occurs. In the latter case, the error is attributed to an error in the eye-to-head calibration and so visual adaptation occurs. This theory allows for *mixed* movements in which the hand becomes visible sometime after the beginning of movement but prior to the end of movement. In this case, adaptations are made in both the eye-to-head and hand-to-head calibration. Their experiments show that the earlier the hand becomes visible, the more dominant becomes the process of visual guidance in the hand-eye coordination process.

In [74] Redding and Wallace assume that there are two kinds of guidance *codes* used to control the hand-eye coordination process: codes based on position; and codes based on distance. Positional codes alone, as advocated by the proprioceptive theory, are not enough to accurately control the hand-eye coordination process since the system would not be able to correct the error induced by distortions in the visual information (such as are introduced by viewing the world through a prism). Thus the target would never be reached. In their model, one subsystem is guided by the other. The system which is guided is the one which recognizes a discrepancy between its expected and actual data (for example, in position). This system is then re-calibrated

using information from the other subsystem by gradually changing the mapping between the two systems. Redding and Wallace conclude that the direction in which guidance will flow is dependent on the availability of visual feedback. Specifically, if visual feedback is available early in the hand-eye coordination process, then visual control of the arm movement occurs and proprioceptive adaptation occurs (eye-to-hand direction). If feedback is not available until near the end of arm movement, then proprioceptive information is used to control the eye movement in tracking the invisible limb, and visual adaptation occurs (hand-to-eye direction).

Redding and Wallace's approach is a study of how subjects adapt to changes in the environment which affect, and/or constraints placed on the process of hand-eye coordination. Studies of this nature do not take into account that there is an intelligent, cognitive being involved. Thus hand-eye coordination is not necessarily merely a matter of instinct or coded response. In [29] Flanders *et al.* start from the premise that visual and kinesthetic control of the limb are integrated parts of the eye-hand system. They propose two simple models of *how* the two may be integrated: addition; and exclusion. If *additivity* holds, then the influence of the two kinds of control are added to each other to produce the total influence exerted on the limb. If *exclusion* holds, then one or the other is completely responsible for control of the limb. Their work is based on a study of the latency between when visual stimuli are presented and when that information appears to have an influence in controlling the movement of the limb; and similarly for kinesthetic stimuli. They determined that the two latencies are significantly different. Flanders *et al.* discover that when visual and kinesthetic stimuli are presented simultaneously, the magnitude of the limb's response is larger than when they are presented independently. This indicates that the two different kinds of stimuli are integrated by additivity.

2.1.4 Cognitive Control

In [87] Webster studied the relationship between three types of adaptation and control: cognitive; motor-kinesthetic; and oculomotor. Cognitive control involves giving feedback on the error subjects make in pointing towards an object and allowing them to attempt to correct it. This feedback is in the form of visual information. Webster's results indicate that cognitive control is used in learning hand-eye coordination in that there is an increase in pointing accuracy in both arms even when only one of the arms was trained to adapt to the distortion introduced by viewing the world through a prism. His results also show that motor-kinesthetic adaptation occurs because the improvements in pointing accuracy for the trained/adapted arm are, in general, larger than those for the un-adapted arm. From his experiments, Webster concludes that there is a dependent relationship between cognition and motor-kinesthetic adaptation. He suggests that proprioceptive adaptation may depend more on being able to see the error, than on actually seeing the limb.

2.1.5 Discussion

Many interesting insights into the human hand-eye coordination process are offered by the studies discussed above. Various work supports the hypothesis that the vision system does aid in the control of the arm [25], [21], [29] which is very important to our approach and to the concept of hand-eye coordination in general. In particular, there is evidence that visual feedback specifically contributes to the process of error-correction [2] (the difference between the expected and actual position of the hand). There is evidence that the human system uses some knowledge of the movement of the eyes in order to aid in guiding the arm towards a target object [2]. Finally, the results in [87] indicate that the learning of hand-eye coordination is not haphazard, but

controlled. This suggests a goal directed approach to learning hand-eye coordination. While the proposed research is not an attempt to mimic the way that humans learn and perform hand-eye coordination tasks, these research results and studies provide motivation for the way in which we approach the task of hand-eye coordination using robotic manipulators and mechanical sensing systems.

2.2 3D Object Recognition

In general, 3D object recognition systems are concerned with recognizing *what* is *where* in a scene using information given in the form of intensity images, range images, etc. The proposed research is concerned with the utilization of visual data to perform real world tasks. However, while the immediate focus of the proposed research is not on the acquisition of visual data, this task is certainly a major component of the proposed system. This being the case, a review of the major approaches to analyzing images using some form of *a priori* knowledge of the specific domain is certainly appropriate. These systems all fall under the banner of what is referred to by Haralick and Shapiro [40] as *knowledge-based vision.*

2.2.1 Early Recognition Systems

Visions—Hanson and Riseman

The *Visual Integration by Semantic Interpretation of Natural Scenes* system was introduced in 1978 by Hanson and Riseman [39]. This system is designed to extract 3D shapes from natural scenes. These shapes are represented as surface patches together with the edges separating distinct surfaces, represented as B-splines. The final object representations are built up in a hierarchical fashion from the lowest level of representation. At each level of the hierarchy a *knowledge source* constructs the next

level of representation from the data available at the current level. These knowledge sources combine to form the knowledge base of the system which, obviously, is also arranged in a multi-level hierarchy. Furthermore, the system uses a bottom-up as well as a top-down approach for hypothesis generation.

ACRONYM—Brooks

Brooks' ACRONYM system [13], [14] represents its objects in a coarse-to-fine hierarchical graph structure. Processing begins at the lowest level with the raw image data from which edge elements are extracted. These elements are joined to form ribbons and ellipses, the 2D shapes produced by the projections of 3D generalized cones. The 3D shapes are then *built up* from the ribbons and ellipses via the hierarchical graph where the links between the graph nodes represent symbolic constraints imposed on the properties of and/or relationships between the primitives. The constraints become stricter as the hierarchy is traversed from the coarse level to the fine level. A matching process is then performed between the constructed observation graph and the stored model-base which is stored in a hierarchically organized structure called the object graph. The matching process produces an interpretation graph. This process is repeated for each level of the hierarchy with the new information at each level of the interpretation graph being used to place further constraints on the model parameters to be matched during the next iteration.

While the ACRONYM system's use of a top-down feedback strategy and general symbolic constraint manipulation to determine and eliminate inconsistent interpretations is laudable, the rigidity of its model representation and the process used for building and matching the observation graph hinder its ability to recognize objects even with a small set of object models. Given a larger set of objects, the size of the corresponding object graph might result in a prohibitively expensive matching process of many iterations. Additionally, the system's performance is constrained by

the goodness of its segmentation of the image at the lowest level of processing. As many approaches do, ACRONYM depends on a *good* initial segmentation and has no means of improving that segmentation.

SPAM—McKeown, Wilson and McDermott

In the SPAM system designed by McKeown, Wilson and McDermott [67] the knowledge of the working domain, in this case aerial images of airports, is represented by a set of rules. These rules are used to *interpret* the primitives extracted from an image based on primitive features and relationships between primitives. These interpretations may lead to the combination or elimination of primitives as specified by the rules and verification process. In SPAM there are five categories of rules and five corresponding phases in which they are used:

1. *building phase rules* are used to initially classify/interpret regions in the initial segmentation of the image;

2. *local evaluation phase rules* essentially refine the initial segmentation, based on relationships between neighboring regions, by combining regions into larger or more encompassing primitives;

3. *consistency phase rules* are to ensure that adjoining primitives have been given interpretations that are consistent with domain-specific knowledge;

4. *functional area phase rules* start with the given consistent interpretations and attempt to combine regions into functional areas;

5. *global evaluation phase rules* use goal generation knowledge to discard regions with weak interpretations that have resulted in unlikely local interpretations given the global perspective.

The ability of the SPAM system to evaluate and attempt to correct its interpretation/segmentation of the input scene is a potent advantage over other approaches including the ACRONYM system. The disadvantage is that all of its domain specific knowledge is hand-coded into the system via the rules. Thus, any change in that knowledge has the potential of requiring a complete re-write of the rule-base. Otherwise, any airport which deviates from the given rules will be difficult to interpret correctly. Taking this argument one step further, for every different domain SPAM would require a completely different and independent set of rules governing the interpretation of a scene from that domain.

MOSAIC—Herman and Kanade

The MOSAIC system [41] uses general knowledge of urban building construction to interpret aerial images of urban scenes and construct 3D representations of the imaged scene. This approach differs from previously mentioned systems in that it uses a sequence of images of the scene including stereo pairs of images. Both the monocular and stereo images are used to construct wire-frame representations of the scene. *L junctions* are extracted from the stereo pairs and then the pixel gray-values and the depth values are used to evaluate a cost function in a constraint satisfaction scenario to find consistent matches. These consistent matches form the basis of the wire-frame construction. Junctions and line segments are extracted from the monocular images. A graph is constructed from the junctions based on their spatial relationships. These relationships are then evaluated based on the extent to which two junctions are connected by extracted line segments. The graph is pruned and extended as dictated by the above evaluations and a wire-frame constructed using domain-specific knowledge. Construction of the scene model begins with an initial representation, created from the initial image(s), and is progressively refined as new information is extracted from additional images. MOSAIC is capable of handling complex scenes in

an urban setting. However, strong and thus restrictive, assumptions are made about the contents of the scenes. Specifically, two constraints are that the buildings will be rectangular and the roofs will be flat. The roof restriction alone prevents the system from being successful in a suburban setting where house roofs are rarely flat. This lack of generality is due in part to the fact that the domain knowledge is hand-coded into the system and not learned by observation of the domain.

SCERPO—Lowe

Lowe's *Spatial Correspondence, Evidential Reasoning, and Perceptual Organization* approach [63] to 3D object recognition uses simple object features (line segments) that require no domain-specific knowledge to be extracted. These simple features, however, are sufficiently powerful to determine initial matches with the model features—from which a transformation is calculated—and allow for verification of the transformed model's match with the imaged object. Matching is performed on the basis of three perceptual organization relationships: proximity, parallelism and collinearity. However, prior to matching, line segments are grouped according to their relationships and then ranked based on the likelihood of the relationship having occurred accidentally (due to a specific viewpoint). Although this approach was not fully explored by Lowe, it offers the possibility of finding a really good match early in the process and significantly pruning the search space, in the best case.

2.2.2 Recent Popular Recognition Strategies

Constrained Search—Grimson and Lozano-Pérez

Grimson and Lozano-Pérez's [38] model-based approach employs a constrained search of an interpretation tree to generate hypotheses of matches between features (surface patches) extracted from input data and those stored as part of the model

database. The constraints are based on geometric properties of the features, such as angles between surface normals, or measured locations and orientations of planar surface patches. Each level of the interpretation tree represents the possible matches between a single data entity and all the model entities in the database. The geometric constraints provide a way to restrict the search down the tree. Specifically, any branch which produces a geometrically inconsistent arrangement of the data entity and the model entity pairings is immediately discarded. For example, consistency may require that the extracted data features fall within a certain range of values of the model surface features to which they are being paired. If the extracted values fail to do so, that branch is searched no farther. The advantage of this approach is the ability to prune the search space at the earliest sign of inconsistency, saving much wasted effort. The disadvantage is that if the model ranges are off by any fractional amount, a valid interpretation may be discarded. This model data is hand-coded into the system and there is no chance for updating that information based on experiential data. Thus, for a different environment the model data might be incorrect.

In [38], the original work was restricted to rigid objects composed of only planar surfaces. In [37], Grimson extends the work to families of objects represented by parameterized models. Whereas the original work only allowed for rotational and translational degrees of freedom, the extended work takes into account the effects of scaling, and stretching along a single axis, as well as allowing a limited number of movable parts (their example is a pair of scissors). In order to keep the interpretation tree search tractable in the case of parameterized families, Grimson's approach is to view objects as a collection of rigid parts. The tree is then searched for one of the parts. The hypotheses for the first part are then used to constrain the search for the next part, and so forth. Rather than using surface patches as in the original work, here Grimson uses edge segments.

Automatic Programming—Ikeuchi

The *bin-picking problem* is characterized as that of looking into a bin of parts that are all jumbled together, locating the part of interest, and grabbing that part out of the bin. One approach to this problem is proposed by Ikeuchi in [47] in which range data is used as input to the system. Similar to Grimson's work, this method is a model-based approach and employs an interpretation tree. In Grimson's method, the interpretation tree is used to make decisions about how to match the extracted data features with pre-stored model features. Ikeuchi's interpretation tree, however, is constructed and used differently. Grimson's tree is constructed by pairing extracted data features with known model features, whereas Ikeuchi's is constructed directly from the models. A model consists of the features computed for representatives of each class of topologically equivalent views of an object. Furthermore, Grimson's tree is used to detect and explore only consistent matches based on the extracted data. Ikeuchi's tree is used to *generate* consistent matches by dictating which features to extract and how these features are to be compared with the model features [40].

Evidence-Based Approach—Jain and Hoffman

Rather than performing direct matching between image data and object models, Jain and Hoffman [48] use the training data to generate *evidence rules.* Each rule specifies a list of features found in one or more views of an object model, the ranges of values that the features may take, and the number of times the features may/must occur. For a given input image, if evidence rule E_i is satisfied, then for each object model M_j, E_i specifies the degree of support provided by this *evidence* that M_j is the object being viewed. Rule E_i is *satisfied* if the list of features occur with values within the ranges, and as many times as allowed/required. After all E_i have been evaluated, a similarity measure is computed and the model with the most evidence is hypothesized to be the object in the image. Jain and Hoffman's approach has the

advantage of using a training phase to learn the set of rules rather than hand-coding a fixed and rigid set of constraints. This gives the system the flexibility of having an easily extendable model set. In their later work they generated 15 synthetic views of each object to populate the evidence-rule base automatically.

Geometric Hashing—Lamdan and Wolfson

The technique of geometric hashing is used in [58] to perform model database generation and object recognition. The model database in this case consists of a hash table whose objects are represented as geometric features and relationships. The keys into the hash table are based on the representation. During training, the keys are used to create the hash table. During the recognition phase, the keys are used as indices into the table to generate hypotheses of which object model is present in the scene. Lamdan and Wolfson use points and lines as their geometric primitives. In [80], Stein and Medioni extend the idea of using geometric hashing by using 3D curves, representing surface and depth discontinuities, and *splashes*, representing general shaped objects by their surfaces, as the geometric primitives.

Interpretation Tables—Flynn and Jain

In [30], Flynn and Jain construct interpretation tables based on invariant features of 3D CAD-generated object models. Surface types such as *planar*, *cylindrical*, and *spherical* are the primitives used, and invariant features, which in some way define a relation between two surface types, are determined. Hypotheses consist of correspondences between three scene primitives and three model primitives. This triple provides two invariant feature indices which are used as keys into the interpretation tables. An entry in a table is an interpretation of which three surfaces in a given model might correspond to the three scene surfaces. The invariant features are themselves constraints on the relationships between the two surface types they are *relating*. Thus, although several hypotheses may be generated for a given set of image surfaces,

there should be fewer than what would be generated in a constrained search of an interpretation tree for the same data.

2.2.3 Neural Network Inspired Recognition Systems

Cresceptron—Weng, Ahuja and Huang

In [92] [91], Weng *et al.* use a neural network called a *cresceptron* which has a hierarchical structure and is trained to recognize objects via unsupervised learning. The neural network allows the system to *learn* the important features needed for recognition, as opposed to being told *a priori* which features to use, and to adaptively change the importance assigned to a feature through time. This allows the recognition system to learn and adapt as its environment and the data with which it is working changes.

The network is able to grow by adding new neurons as new, previously unseen data is presented to it. This has the advantage over other schemes of not requiring retraining the system every time a new model is added. The hierarchical nature of the *cresceptron* provides the system with the attribute of generalization by creating a tolerance for small deviations in the viewed object features. This in turn facilitates the use of an unsupervised learning scheme in that it is not necessary to train the *cresceptron* with an exhaustive selection of all possible *views* of the objects. Rather, the topologically different views can be presented for training, and the small deviations in-between will be handled by the network. If something is presented that is so different that the network decides it is *new*, the network will adapt by growing to accommodate the new information. One disadvantage to this approach is that there is no explicit 3D modeling of the objects, so many views must be encoded within the system in the hope that any unseen, intermediate views will be recognized. The

advantage is that a learning scheme has the ability to generalize its knowledge.

SHOSLIF—Weng

The *Self-organizing Hierarchical Optimal Subspace Learning and Inference Framework* [89] example proposed by Weng is an attempt to overcome the restricted learning of current machine learning approaches. While hand-coded rules (as used in the previously mentioned approaches) are efficient, this is only true for the set of objects for which the database contains models. Unfortunately, enlarging the database by adding more and more models is not constructive as it creates indexing problems and often requires re-coding of the entire set of rules or constraints. The lack of generality creates a system which cannot handle unknown objects. The SHOSLIF example concentrates on using a self-organizing approach to automatically learn features and object models based on those features, and consequently organize themselves. In order for this approach to succeed, the object representation must be general enough to apply to virtually any object and any image. The SHOSLIF example uses a network to divide and represent the input space. The structure of this network is automatically built as a function of the learning phase. The network partitions the input space in a manner similar to a Voronoi diagram except that it is hierarchical in structure and the centers and extent of the cells are *recursively* updated during the learning process. The current version uses the Euclidean distance measure in determining the cell divisions. The hierarchy of the network is arranged in a coarse-to-fine manner in which a given region of the feature space is recursively partitioned into finer (and smaller) cells. The idea is that, the finer the cell division, the more specific recognition that can be obtained.

The SHOSLIF approach has the advantage of being robust in recognition. That is, it handles diverse and wide-spread domains and applications; and is also able to deal with *outliers*, and/or *bad* data. It has the potential of reducing the space complexity

required for storing the *database* by basing the cell distribution in the network on the population distribution of the input space. The indexing complexity is reduced because the search space is reduced at each level of the hierarchy (only one or a few of the network branches are ever searched). This in turn results in faster search times. Questions remain about the use of the SHOSLIF example as it now stands. For example, hyper-planes are used to decompose the feature space. However, this requires that the *classes* of objects be linearly separable, which may or may not be true.

Adaptive Resonance Theory 1 NN—Liang, Liao and Lin

Liang *et al.* propose a neural network approach to 3D object recognition [60]. This approach builds a database consisting of multiple views of polyhedral objects. There are two stages to the recognition—a coarse stage at which a subset of the database is selected, greatly reducing the search space; and a fine stage at which each of the possible matches are more thoroughly examined. The coarse stage consists of an *Adaptive Resonance Theory 1* neural network (ART-1) which self-organizes during the unsupervised learning phase. The input to the network are binary vectors representing the features which have been extracted from the 2D image data, one for each image. The method exploits the parallelism inherent in the task of matching a given feature vector simultaneously to all feature vectors in the model database. It is left to the ART-1 network to determine which components of the feature vector are most important or relevant to discriminating between models. During training, the network compares each new sample to the current set of models. If a pattern is found which is similar to the new sample, then the sample is assigned to that category. If no pattern is found to be similar enough, a new category is created. This approach has the advantages of being self-organizing, being able to handle a large database composed of multiple-views of objects, and being able to exploit the

parallelism inherent in such tasks. However, as is common to neural networks, it is difficult to describe exactly what features the network is giving the greatest weight, or what the network is using to discriminate between the entries in the database.

2.3 Manipulator Control

Many approaches to controlling robotic manipulators have been proposed. This section discusses a few examples in which no visual information is used. The following section discusses the approaches in the literature which use visual information to in some way aid in the control of these manipulators.

Configuration Control—Seraji, Long, and Lee

Seraji *et al.*'s work with seven degree of freedom manipulators takes advantage of the inherent redundancy in the trajectory of the arm in order to successfully track the trajectory [77]. Usually, the term *redundant* refers to the infinite number of joint configuration vectors which will position the end-effector at a *given point.* However, here redundancy indicates that there are an infinite number of joint configuration sequences which will move the end-effector along a *specified trajectory.* This redundancy is obtained by the addition of a seventh degree of freedom into the arm. In addition to the usual kinematic equations which specify the relationship between the joint configuration and end-effector position, the proposed configuration control approach uses additional kinematic functions which are task-related. These additional functions aid in the resolution of the trajectory redundancy, thus giving added control to the manipulator motion. These functions, in essence, act as constraints, dependent on the required task, on the motion which the manipulator is allowed or required to make as it follows the specified trajectory. This approach allows users to specify several constraints for a given task as well as place constraints on the basic set of kinematic

equations. The user is able to interactively assign weights to the constraints so that different constraints have precedence depending on the specific task being completed. Examples are given on how this approach could be used to control the movement of the elbow, obstacle avoidance, and optimal movement of all joints during trajectory. Here optimal implies that the total movement of all joints is minimal.

Neural Kinematics Net—Eckmiller, Beckmann, Werntges and Lades

Eckmiller *et al.* propose a set of artificial neural networks for learning the control mechanism for moving a four degrees of freedom manipulator along 2D trajectories[24]. These networks are called *Neural Kinematics Nets* (NKN) and they learn and maintain the different types of knowledge necessary to geometrically represent the inverse kinematics problem. The Position Model net keeps track of the current position and angle of each joint, as well as the distances between them. The Angle Model net indicates the direction of movement for each joints. Given the input from these two nets, the Contribution Selector Mechanism net decides the extent to which each joint will participate in moving the end-effector to the next location. The Angle Command Mechanism net uses the other three nets to perform movement of the manipulator. Viewing the manipulator movement as a geometrical relationship as opposed to the typical algebraic relationship, Eckmiller *et al.* have proposed a system which learns how to move the manipulator through a sequence of small steps towards the goal position rather than trying to achieve the goal in one sweeping movement of the manipulator. This has the potential for allowing corrections to errors in the current trajectory. Adding visual feedback to the system would allow it to track its movement in relationship to the goal position. Additionally, by breaking the system into several networks which each take on a part of the overall control task, the training of each network is simpler and the size of the network is smaller.

Trajectory Control—Bassi and Bekey

Bassi and Bekey propose a hierarchical, functional decomposition of the trajectory control problem [11]. They train a set of three artificial neural networks to learn the dynamics of the manipulator. Here the dynamics imply finding a trajectory which will move the manipulator to a specified position at a given velocity and within a certain amount of time. Each sub-network is responsible for learning a specific relationship which is essential to this control problem. This simplifies the problem because each sub-network learns a more simple mapping than if a single network was used to learn the entire mapping. The three relationships learned by these sub-networks are those between the linear torque and Cartesian acceleration, one between the quadratic Coriolis and centripetal torques, and one for the second-order acceleration term.

Semantic Networks—Agarwal

Agarwal presents a high-level specification for controlling robotic manipulators. This specification is in the form of *semantic networks* which represent the state of the robotic task and the operators which may be useful for solving it [3]. The low-level details of the mechanics necessary to perform the operations specified by the operators are not presented here and there is no feedback, either from external sensors or the robotic manipulator itself.

Decomposed Connectionist Architecture—Katić and Vukobratović

Katić and Vukobratović propose a method for decomposing the dynamics of the robotic manipulator in order to simplify the mapping functions which need to be learned by the system [52] decomposition, they claim, using neural networks as *black boxes* is impractical due to the high-dimensionality of the input and output spaces. The specific dynamics Katić and Vukobratović are interested in learning are the inverse kinematics. For each *sub-part* into which the dynamics are decomposed, a multilayer perceptron is trained to learn the mapping from its input to its output space.

Due to the decomposition, each sub-net has a lower-dimension input and output space than a single network would have without the decomposition. The result is that it takes fewer training samples and thus a shorter training time. This method still uses general information about the manipulators forward and kinematic dynamics.

Neuro-Adaptive Control—Zomaya, Suddaby and Morris

Zomaya *et al.* use a feed-forward network trained with a back-propagation learning algorithm based on *reinforcement learning* [96]. By using reinforcement rather than strictly learning by teaching, the network is able to adapt more easily to a changing environment and/or working conditions. The network learns by evaluating its own performance as it goes. This evaluation is based on a vector the network receives back about the current state of its environment. Zomaya *et al.* call this the *context vector.* The network evaluates the goodness of its output dependent on the current context in which it was generated. At each training step, the network generates an output, evaluates its performance, and updates its weighting functions.

Sensorimotor Learning—Salganicoff and Bajcsy

Salganicoff and Bajcsy [75] use super-quadric models to parameterize objects in a scene. This approach also relies on the concept of reinforcement learning as did [96], viewing the distribution of the reinforcement as a prediction surface. This surface is computed as a function of both the action attributes of the manipulator and the perceptual attributes of the sensors. Then for a given perceptual attribute, the surface is searched to find the action attribute which combined with this specific perceptual attribute converges to a peak on the prediction surface. This prediction surface is itself represented as a 2^k-tree where each node represents a k-dimensional hyper-rectangular volume. This k-dimensional pattern space is then searched associatively. Experiments are performed using a fixed camera giving a top view of an object whose centroid is extracted as the feature vector.

2.4 Hand-Eye Coordination

Many approaches to solving the robot manipulation problem have been proposed. Many previous approaches rely entirely on controlling the arm via data known *a priori* about the particular manipulator being used and the specific, often highly constrained, environment in which it is to operate. Some of these approaches make use of feedback from encoders on the joint actuators, which give explicit information about the joint's current position. One of the main problems involved in robotic manipulation is the requirement that the robotic system work in an unstructured and possibly unknown environment. In order to perform this task, the system must be able to learn about its environment in order to act within, and interact with, that environment. Data from a knowledge set about one environmental setup will, in general, not transfer easily to another setup. More recently, many researchers are turning to using visual systems to provide information about the manipulator, the environment and the manipulator's relationship to the environment, in order to guide the movement of the manipulator in its workspace. Data extracted from visual input is used in a variety of ways including computing various calibration parameters such as learning the mapping from the image space to the camera space [28], from the camera space to the end-effector space, [84], or from the camera space to the world space. [35]. Some researchers use visual data to learn the forward and/or inverse kinematic equations relating the world space and the manipulator [95]. Each of theses issues alone are challenging problems to resolve. If we combine them into a single system then we have what is commonly referred to as the *Hand-Eye Coordination Problem.* This term is applied to this research problem because of the functional similarity to the way in which humans cab use visual input from their eyes to guide the movement and use of their arm/hand to manipulate objects. In the case of robotic manipulators

with visual sensors, the robotic manipulator represents the *hand* and the visual sensor represents the *eye* in HEC. The following sections review the literature on robotic manipulation, including systems which have been proposed to tackle the HEC task.

2.4.1 Calibration

Much work in vision-guided robotic systems concentrates on solving the calibration dynamics of the camera and robot subsystems. In general, there are three types of calibration which must be solved in these systems.

First, there is the camera calibration. This involves computing the mapping from the perspective transformation of the camera lens to the image plane and depends on the properties of the camera itself. These properties include the focal length, the scaling factors on the x and y axis, and the image plane offset in the x and y directions.

Second, there is the calibration of the robot itself. This may involve determining the *forward kinematics*, the *inverse kinematics* or both. If the angle values of the manipulator are given, then the forward kinematics involves computing the (x, y, z) position of the end-effector in the world coordinate system. The forward kinematics problem has a solution which exists and is unique. On the other hand, if the desired position of the end-effector in the world coordinate system is given, then the inverse kinematic problem involves computing the vector of angle values for the manipulator's joints that will take the end-effector to the desired location. The solution to this problem is not guaranteed to exist, and if a solution does exist, there is a high-probability that it is not unique.

The third calibration that must be done is that between the camera and the robot manipulator, i.e., the relationship between the camera's coordinate system and the

robot's coordinate system. In general, the visual system is going to be used to *see* where the end-effector is, or where it is supposed to be positioned. In either case, given a point in the camera's image plane coordinate system, we need to compute the position of that same point within the robot's coordinate system, for example, so that we can then compute the joint angle vector that will move the end-effector to that position. In binocular systems, an calibration between the two cameras is also needed.

Once the above calibrations are known, given a target point in the world coordinate system, the point's position in the image plane can be computed, and then the position in the robot's coordinate frame can be computed, and finally, the joint angle values that will move the end-effector to that position can be calculated.

2.4.2 Image Space to Camera Space Calibration

Feddema *et al.* assume that the camera to end-effector and end-effector to manipulator transformations are known [28]. They are concerned with empirically estimating the transformation from the image plane to the camera coordinate system. To do this, they use an interactive visual feedback approach. At each camera location they extract features from the imaged object. Then, given a differential change in a feature's location in the image, Feddema *et al.* calculate the change in the pose of the object with respect to the camera. The features' positions within the *object* frame are assumed known from CAD models of the object. The problem reduces to selecting the best features for determining the pose of the object.

Feddema and Mitchell [27] use the above procedure in order to generate manipulator trajectories in feature space. They claim that this will allow for smooth motion of the manipulator in the presence of asynchronous or discontinuous visual updates

because the processing time constraints can be used when computing the path and speed of the movement/acceleration time. The advantage here is the use of continuous feedback as opposed to most visual-robotic systems in which the robot comes to a complete halt while waiting for the visual system to finish processing the current image. One serious drawback with this system is that the differential transformations are generated *a priori*, requiring users to have in-depth knowledge of robot kinematics and camera modeling.

2.4.3 Camera to End-Effector Calibration

Tsai and Lenz present a method in [84] for autonomously computing the calibration between the camera and the end-effector. In their work, the camera is rigidly attached to, or gripped by, the end-effector. The robot is required to have enough degrees of freedom so as to be able to rotate the camera around two axes and keep it focused on the stationary object being used for calibration. If this does not hold true, then the 3D geometric relationship can not be resolved uniquely. Tsai and Lenz decouple the hand/eye calibration from the manipulator calibration. Their target points consist of spots on a block whose exact positions are known *a priori*. The camera and manipulator calibration are assumed known. Then, the end-effector, along with the camera, is moved from one viewing position to another and the homogeneous transformation from an initial view point to a second view point is computed for the camera and for the end-effector. These transformations are then used to calculate the transformation between the camera frame and the end-effector frame.

2.4.4 Camera Space to World Space Calibration

In [35], Graf and LaLonde present an artificial neural network architecture which learns the inverse kinematics of the manipulator-to-camera calibration, as well as collision constraints, to generate collision-free trajectories which position the end-effector at the *point of gaze* of the camera. The sensory platform is physically mounted onto the robot body in such a way that the end-effector is clearly visible. The sensors consist of two cameras which move independently of each other. Additionally, there are joint sensors which output the current offsets of the links from their rest positions; and the eye motors each output displacement vectors which uniquely specify the direction and distance of the point of gaze along the eye's line of sight. Graf and LaLonde's approach requires three subnets: an obstacle map of fixed topology; and two Kohonen maps. One of the Kohonen maps, M_{arm}, learns the arm configurations. Neighboring neurons must represent neighboring configurations in order to insure smooth transitions between configurations; and only physically possible configurations should be represented in the map. The other Kohonen map, M_{eye}, learns the eye configurations. Each neuron in the obstacle map inhibits all neurons in M_{arm} that would cause a collision with the obstacle associated with the active obstacle map neuron. Each workspace point activates a neuron in M_{eye}, which in turn activates all neurons in M_{arm} which would result in the end-effector being positioned at the workspace point and which have not been inhibited by the obstacle map. Then a path is planned through the arm-space from the current end-effector position to one of the target positions. Training of the network consists in moving the end-effector to random spots in the workspace, tracking it with the eyes, and taking sensor samples along the way. In [36], Graf and LaLonde extend the above method to include an active camera platform. The network learns how to position, in a manner similar

to the above procedure, the "head" so that it has an obstacle-free view of the end-effector.

2.4.5 World Space to Arm Joint Space Calibration

Zhu and Leu [95] use a *cell state space* approach in order to plan the manipulator's optimal trajectory from its current state to the goal state, thus obtaining *optimal* control of the arm. The robot's *state* is specified by the joint angle values and their velocity. If the arm has N joints, then the system is modeled as N two-dimensional (2D) planes of cells with axes of joint angle value versus velocity. The *cells* of the plane are enumerated as one-dimensional (1D) vectors. The system state is then given as an N-vector where the i^{th} component is the index number of the corresponding cell of the i^{th} plane. Given the target, a set of equations is given which are iteratively solved *backwards* to the known starting position. At any position there are a number of possible "next steps" which may be taken, each determined by a unique pair giving the torque/force and time duration of the step (i.e.: a given torque/force and time duration determine a unique step). All possible (admissible) solutions are sorted to determine the optimal one. While this approach allows for a sequence of moves towards the target state which would allow for correction in the presence of inaccurate movement of the arm, there is no way of sensing whether there has been an inaccurate movement; that is, whether or not the arm is actually positioned where it "thinks" it is positioned. Given this observation, there is the probability that position error will build with each move towards the goal.

2.4.6 HEC Systems

Martinetz, Ritter and Schulten

In [65] and [66], Martinetz *et al.* propose an ANN approach to learning the inverse kinematics of a visual-robotic system which utilizes Kohonen's self-organizing topology-conserving maps. Their system incorporates two cameras which are neither attached to each other nor to the robotic manipulator. The network is a 3D lattice of neurons which take as input a 4-dimensional vector, $\mathbf{u}$, containing the position of the point of interest within the two camera image planes. The output consists of two parameters, the vector $\vec{\theta}$ and the matrix $\mathbf{A}$, which are the first two terms in the Taylor expansion of the $\mathbf{u}$ to joint-space transformation, $\vec{\theta}(\mathbf{u})$,

$$\vec{\theta}(\mathbf{u}) = \vec{\theta}_s + \mathbf{A}_s(\mathbf{u} - \mathbf{w}_s) \tag{2.1}$$

where s is the neuron activated by input $\mathbf{u}$. When a $\mathbf{u}$ is input, the neuron whose centroid, $\mathbf{w}_s$, is closest to $\mathbf{u}$ is activated. Each neuron represents a subregion of the 3D workspace specified by the centroid of that subregion.

Initially the $\vec{\theta}$, $\mathbf{A}$ and $\mathbf{w}$ are all assigned random values. During training, points are chosen from a random distribution within the workspace. The corresponding $\mathbf{u}$ are input to the network. The subregion whose centroid is closest to $\mathbf{u}$ is activated and its $\mathbf{w}_s$ is updated to reflect the new information. Additionally, all the neurons in its neighborhood are also updated to varying degrees. A neuron's neighbors are defined by the interconnections, or lattice, within the network. The degree to which the neighbors are updated depends on their distance from the activated neuron measured in Euclidean distance where a unit is defined as one *hop* on the lattice. The function determining whether or not a cell is updated is defined by a function of Gaussian shape with the activated neuron in the center of the distribution. The farther a neighbor

falls from the mean, the less it is affected by a new data sample. As mentioned in the discussion on Kohonen maps (Section 2.5.3), this is intuitively like pulling the activated function and its neighbors towards the input vector $\mathbf{u}$.

Once a neuron is activated the robotic manipulator is moved according to the values of the neuron's output parameters. There is a gross move, specified by $\vec{\theta}_s$, followed by a fine move specified by the linear equation in 2.1. The retinal images produced by the new positions of the end-effector are then used to update $\vec{\theta}$ and $\mathbf{A}$ for the activated neuron and it's neighbors.

Two things that Martinetz *et al.* do not address are the issues of singularities, where solutions to the inverse kinematic problem do not exist, and multiple solutions. In [53], Kieffer *et al.* address these issues in the context of Martinetz *et al.*'s method. Their simulations with a two-degree of freedom, planar robotic model produce the following results. When the network is presented with training data which contain some points outside the reach of the manipulator, no solutions are found for the unreachable points; but, correct solutions are still found for the reachable data points. Additionally, it is found that all possible solutions for a given point could be found by the network. Which solution is actually learned is dependent on the initial values of the $\mathbf{w}_s$ parameters.

Walter and Schulten

A modification of the algorithm in [65] is presented by Walter and Schulten in [86]. This modification introduces the *neural-gas* network which uses vector quantization in place of the Kohonen map used in the previous work. In this approach the neighborhood relations are built up dynamically through learning as opposed to the static neighborhood configuration of the self-organizing map. That is, during training, whenever a training sample falls into an existing cell, the set of that cells neighbors which are affected by the new training sample is determined by the neural-

gas algorithm. In [65] the set of neighboring cells is determined by their position in the lattice of cells of the network. In the neural-gas algorithm, all existing cells are ranked according to their nearness to the primary[1] cell. Then whether or not a cell's function is updated is determined by a function of the *closeness rank order k* and parameters determining the size of the neighborhood. This size can be chosen to decrease as the training progresses, essentially producing a coarse-to-fine approach in which, as the training proceeds, cells which are farther away from the primary cell are no longer updated by that cell. The training proceeds as described above for [65].

Papanikolopoulos, Khosla, and Kanade

Papanikolopoulos *et al.* use a camera mounted on a robotic manipulator to track a moving object [69], [70]. The object is constrained to move in a plane which is assumed perpendicular to the camera's optical axis. The depth of the plane on which the objects travel is known *a priori* in this approach. The proposed system is able to swiftly change the camera's orientation because all information is formulated with respect to the camera coordinate system, and not the world coordinate system. Given a feature point P_k in image k, the *sum-of-squared differences* (SSD) optical flow is calculated using multiple windows in image $k+1$ which are constrained to be within some defined neighborhood of P_k's position in image $k+1$. Each of these windows provides an optical flow measurement, and for each a confidence measure is calculated. The window with the highest confidence is chosen as the one specifying the true direction of the optical flow. This optical flow measurement is then used to control the manipulator's movement so that the neighborhood of P_k is kept stationary at the center of the image. Papanikolopoulos *et al.* call their approach *controlled active vision* because the motion of the robot-camera system is not accidental but specifically planned to maximize the visual information used in controlling the manipulator.

[1]The cell into which the new sample fell.

Their experimental results show that the choice of control scheme depends on the algorithms used to extract the visual information. Since the method requires that the camera be mounted on the robotic manipulator this method does not extend easily to setups in which the sensing system is unattached and/or moving independently of the manipulator. The main contribution of their technique is in presenting an approach for combining visual information and manipulator control to perform both tasks in a more useful manner, as opposed to other approaches which use the visual data only as further input to the control system.

Allen, Timcenko, Yoshimi and Michelman

In [5], Allen *et al.* present an approach to tracking and grasping a moving object. Their approach is motivated by psychological data on human reaching and grasping. Two of the psychological views on how skilled motor control of humans is organized are characterized by the individual work of Schmidt and von Hofsten. Schmidt's view is that there are generalized motor programs. Each skilled action of the human is composed of an ordered set of these programs. The disadvantage is that once begun, a program can not be interrupted, and so a mistake cannot be corrected until after a program has run its course. The advantage is that each program is of short duration. The view of von Hofsten is that there are two sensorimotor systems: one to handle *approaching*; the other to handle *grasping*. Schmidt's view maintains the visual and grasping activities as mutually exclusive tasks; von Hofsten holds that the reaching task is guided during its movement by the visual data. Thus, in von Hofsten's view, the combination of the motor and perceptual schemas produces a *coordinated motion*. Furthermore, von Hofsten takes the position that the visual schema used to track a moving object, as well as how this visual information is used to predict the intersection of the object and the hand, calculates the distance using the angle of vergence between the hand and object. Thus, to incorporate von Hofsten's approach,

it would be necessary to track both the object and the hand. In contrast, Allen *et al.* maintain that to monitor both object and arm would be computationally difficult. Therefore, their approach to visual tracking does not continuously monitor, visually, the grasping task; rather they use the visual data to calculate a final position and velocity to which the arm moves in order to intercept the moving target. This final position incorporates the knowledge that time will elapse as the hand is moving to the intercept position by predicting the object's forward position ahead of the position extracted from the visual data. The disadvantage to this approach is that it is rigid and unwieldy. That is, it assumes that the moving object will maintain a regular and repetitive movement that can be analyzed and then intercepted. If the object moves off in an unexpected direction, there is no correction ability since the visual data is only used initially, and the arm makes a single movement to place the hand in the intercept position.

In [6] Allen *et al.* use visual information from a stereo camera system to track a moving target, align the manipulator in a position appropriate for grasping that target, maintain an appropriate position, and finally grasp the object. This task is known as *dynamic grasping* because the moving object requires that the manipulator move in a way which is correlated with the object movement in order to effectively grasp the object. In this approach, the camera system is passive and the two cameras are separated by a relatively wide baseline. The object is initially located and tracked via computation of the optic-flow. Then the manipulator is moved into a trajectory which essentially tracks the moving object. Once this trajectory is stable, the arm is instructed to grasp the object. Due to delays in the image processing component and noise in the image information, it is necessary for this system to make forward predictions of where the arm will be at the time at which the visual system finally delivers the visual information. This is necessary because by the time the object's

position in a pair of images has been calculated, the object will have moved to a new position along its trajectory. This system only uses visual information about the moving target. No visual feedback about the manipulator is provided. Instead, the kinematic equations relating the manipulator's configuration space to the target's space is hand-coded into the system. Additionally, the calibration of the camera system is known *a priori.*

Sharma, Hervé and Cucka

In [43] Sharma *et al.* present an approach to HEC which learns the *Perceptual Kinematic Mapping* (PKM) from the robotic manipulator joint configuration space to the image parameter space. In order to compute this mapping exactly, it would be necessary to know the calibration between the camera and the pose of the manipulator. Instead, Sharma *et al.* extract qualitative features of the unknown PKM using visual feedback of the manipulator moving in the image. The control surface which is uniquely specified by the movement of the manipulator in the camera space is a hyper-surface in $\mathcal{R}^{2n}$ for an n degrees of freedom manipulator and a camera in a given position and orientation. Although the exact surface is not known, since neither the camera's position or orientation relative to the manipulator is known, its form is known. Control of the manipulator reduces to tracking it along the control surface, where each manipulator configuration specifies a point on the control surface.

In [79] Sharma *et al.* present an approach to dynamic manipulation which uses the visual tracking presented in [43]. In many systems much computational effort, either on- or off-line, is spent defining/calculating the forward and/or inverse kinematic equations and various calibrations of the system. In this approach, Sharma *et al.* attempt to avoid this effort by using qualitative properties of a topological surface defined between the robot position and the position of some observable image features. Visual input is used as continuous feedback to the system and is represented

in an image-based mode which requires more computations than a position-based representation in order to specify the task in the image space. On the other hand, an image-based representation does not suffer from the inherent non-linearities of the transformation or the uncertainties of the imaging process. Additionally, Sharma *et al.* have added a learning component to their system. Their approach is neither dependent on estimating the exact parametric representation of the kinematics, nor relies on the input/output of neural-net based approaches which ignores the underlying analytic structure of the kinematic mapping. Instead, Sharma *et al.* use a qualitative approach somewhere in the middle. In this way they keep from needing to define or calculate the precise calibrations between the individual components of the system. Figure 2.1 shows the schematic of their approach. Due to the qualitative nature of the task definitions and because the possibility of imprecision in the manipulator's movements, the *plans* for the arm trajectory will need to be updated in order to correct the trajectory and accomplish the goal. Sharma *et al.* envision a 2-stage, or single-correction, task planner which is not yet implemented. But this assumption is based on the requirement that the movement of the target is constrained, which may not hold true in an unstructured environment and, additionally, maintains a significant reliance on the precision of the manipulator and the visual sensing system.

Castanõ and Hutchinson

Castanõ and Hutchinson propose a system whereby visual feedback is used to control the two degrees of freedom which position their manipulator within a plane perpendicular to the visual sensor so that the end-effector's center is imaged at a specified pixel [15]. The final degree of freedom, the depth of the end-effector from the camera, is controlled via feedback sensors from the encoders of the manipulator's joints, and by forcing the end-effector to follow a straight-line trajectory along the projection ray which passes through the camera's focal point and is specified by

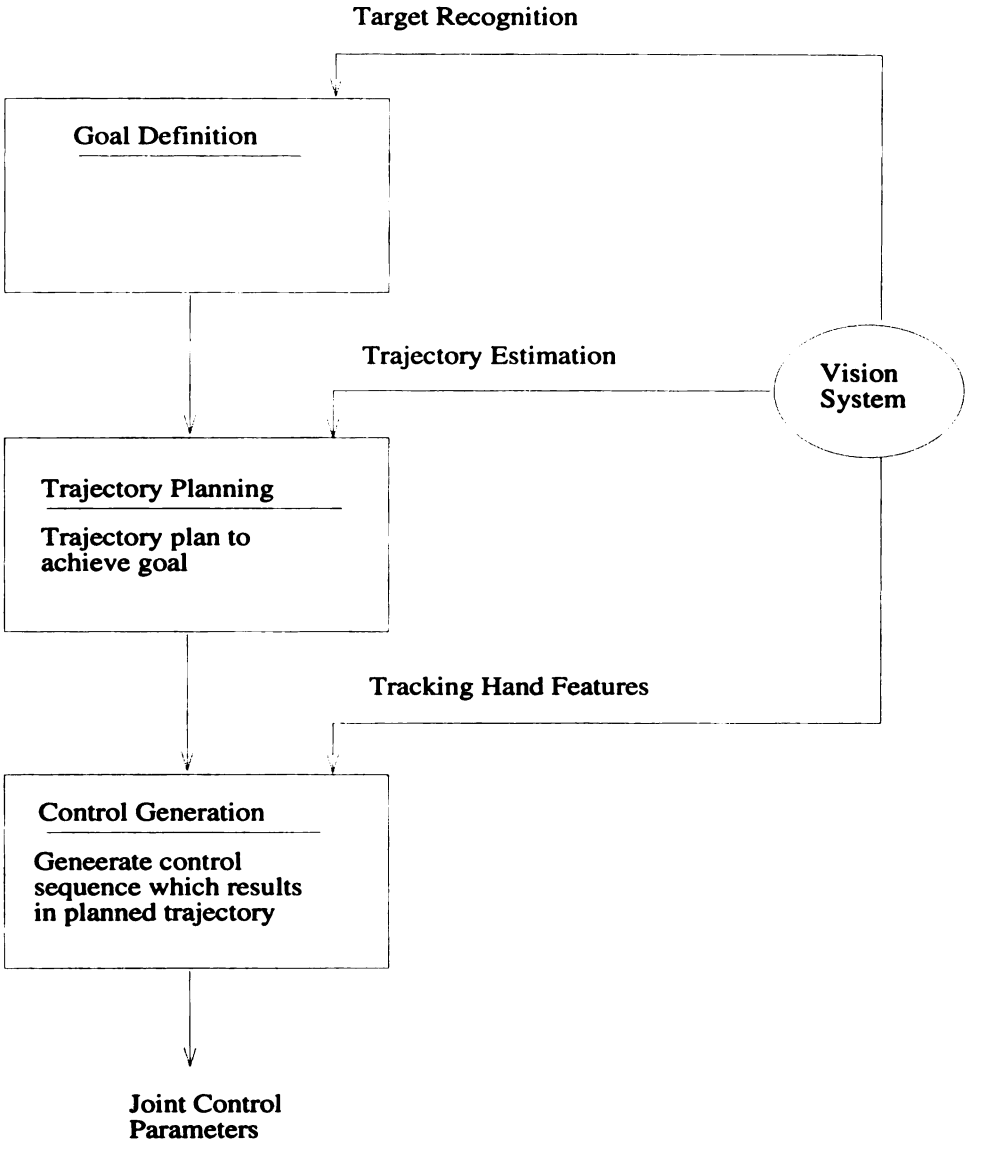


Figure 2.1: Schematic of Sharma *et al.*'s approach.

the image point to which the end-effector is first moved. Their approach requires hand derivation and hand-coding of various equations and Jacobian matrices. The equations are the projection equations which establish the calibration between the image space and the world coordinate system. Only the forms of the equations are derived. The actual values of the calibration parameters are estimated via a least squares approach given a set of image point coordinates and the world points which correspond to them. Additionally, the kinematic equations of the specific manipulator are encoded and the parameter values are computed based on the signals received from the joint encoders.

Espiau, Chaumette and Rives

In [26], Espiau *et al.* present an approach to visual servoing which involves designing tasks that are directly specified in relationship to the visual sensing system. In this case, they present the task of positioning[2] a camera, which is mounted on the robotic manipulator, somewhere within its environment. In their approach, this *task* is given as and initialized as, specifying an error between the position of the goal location as seen in the current image and the location's position in the "goal" image. Espiau *et al.* make an important assumption: that the visual data will show variations **only** in response to geometrical variations which occur in the environment. On the other hand, an all too common problem in image processing is the variation caused by changes in illumination. It is hard to visualize how this approach would handle image variations caused by a simple change in illumination, especially if this occurred in combination with other changes caused by camera or object motion. The main contribution is in trying to define general classes of tasks, which can be specified as the difference between an intermediate image and a desired image.

[2]Another task would be tracking a line, or moving object, etc.

Wijesoma, Wolfe and Richards

Wijesoma *et al.* reject the idea of using an eye-*in*-hand setup where the camera is mounted on the end-effector for two reasons [93]. First, this setup does not provide a workspace overview. The result may be that the system may miss important features which would aid it in locating both the object of interest and the manipulator within the workspace. Second, it therefore becomes necessary for the system to relocate objects, even ones whose positions were previous known. Given a static camera and a static target, this approach views HEC as the process of tracking the manipulator as it moves towards the target. The tracking is performed by comparing the positions of the arm and the target, where the arm's position is extracted from the current image and the target's position as it is located in the initial image is used. In order to track the arm, a marker is placed on the end-effector. The manipulator used in this presentation has two degrees of freedom, moving in a plane which is perpendicular to the camera's optical axis. The difference between the desired and current positions of the manipulator are considered errors within the Cartesian space in which the arm is moving, rather than as errors in the joint configuration space.

Geschke

In [33], Geschke presents a framework for specifying robotic tasks in such a way that the programmer has control over *what* and *how* sensory information is used to perform each task. Conventional controllers use the feedback from the sensors directly, leaving the programmer little control in how the visual information is used by the low-level servo processes. Additionally, the programmer can also specify the manipulator state information for each task. Most of the computations used in this method are specified in the Cartesian space. Given the position, orientation, force and torque, this approach uses the inverse of the Jacobian matrix to compute the desired state of the robot manipulator, map this information to joint space and send

the new signals (joint velocities and forces) to the servos. This system expects that a vision system exists which can locate and track any object when supplied with only that object's name, returning the 3D position of the object it sees. The framework provides the programmer with simple commands such as *Read*, *Tool*, *Object* and *Goto* with which to specify tasks.

Kuperstein

Kuperstein uses a *neural-like* network architecture and visual input to direct a robotic manipulator with three degrees of freedom to reach targeted 3D positions [57]. Given sensory and motor constraints the proposed approach self-calibrates by learning the visual-motor relationships necessary to perform its tasks. The system is controlled by maintaining consistency between the signals it sends to the manipulator and the signals sent to the visual system in order to position the sensors to observe where the manipulator is located. During training, random actions are sent to the manipulator which proceeds to move according to the specified actions. Then the sensor system is moved until the end-effector is visible. The pair of action specifiers—the one given to the manipulator and the one which finally positions the sensors so that the end of the manipulator is visible—are learned as a corresponding pair. A map is constructed whose weights are incrementally updated by each *learned* input-output pair. During run-time, the sensors are positioned so that the target location is visible, and the manipulator configuration which was learned to correspond to the current sensor configuration is expected to place the manipulator at the desired location.

Schrott

Schrott decomposes the 3D positioning of the robotic manipulator into three movements which are only 1D or 2D in complexity [76]. The target object is first recognized within the images received from the camera which is mounted on the end-effector so that the end-effector is also visible in the images. Then, the manipulator makes the

following sequence of movements towards the object's location: one in the xy-plane to align the end-effector with the object; a rotation about the z axis to position the grasping mechanisms parallel to the object; and finally movements along the z axis to place the end-effector over the object's center of gravity (as computed from the images). At the end of these movements the camera is positioned over the object. At this point, fine movements are made to position the end-effector to grasp the object. In order to do this, the intrinsic and extrinsic parameters of the camera are estimated from the visual information received as the camera approaches the object.

2.4.7 Summary

In most cases, when robotics researchers couple a vision system to their robotics system it is with the sole intent of using the visual feedback to calibrate the system. Once the calibration has been performed, the visual system is only used initially to *see* the object of interest. Then, once the target position has been calculated, by an object recognition process that is *assumed* to exist, the visual system is *turned off* and the robotic system operates based on the learned calibration. For example, a robot arm moves along a trajectory which has been determined based on the computed calibration. If the system dynamics change, for example, parts wear out and become imprecise, then the system must be re-calibrated. There is no visual feedback during operation and consequently, no dynamic adaptation to changes in the environment. A comparison of a few works in the hand-eye coordination area and the work presented in this thesis is presented in Table 2.2.

The system presented in this thesis is most similar to the work of Schulten et al. as presented in [65], [66] and [86]. Both works approach HEC as a mapping from a visual input space to a joint configuration space. However, there are several differences.

Table 2.2: Table comparing HEC approaches in the literature.

Researcher	Learning vs Hand-Coded	Iterative	Independently Moving Sensor	Stereo
Schulten et al.(89,90)	L	Y	N	Y
Walter/Schulten(93)	L	Y	N	Y
Sharma et al.(91,92)	L	N	N	N
Papanikolopoulos et al.(91,93)	HC	N	N	N
Castano/Hutchinson(92)	HC	N	N	N
Zheng et al.(91)	HC	N	N	N
Wijesoma et al.(93)	HC	N	N	N
Howden/Weng(93)	L	N	Y	Y

First, Schulten uses an artificial neural network approach whereas we use a recursive partition tree. Their network is based on the Kohonen map and imposes an explicit topology on the network. This results in a uniform distribution of the network's nodes in a three-dimensional grid. The topology of our tree is determined by the set of training samples used to construct it and the order in which those samples are presented to the construction algorithm. The resulting topology allows for a dense distribution of nodes in active regions of the input space, which is desirable for improving the accuracy of the computed output. Also, we use a hierarchical approach. Furthermore, Schulten's approach requires thousands of iterations to converge to the desired topology. Our approach is non-iterative because there is no need to converge to anything. Rather, our system simply remembers what it has observed. Our approach uses interpolation across k candidates in order to compute the output. Finally, as will be shown in Chapter 3, the framework presented in this thesis extends to any dimensionality in the input and output spaces. Table 2.3 contains a comparision between the work of this thesis and that of Schulten et al.

Table 2.3: Table comparing presented method and that of Schulten et al.

Researcher	Howden & Weng	Schulten, et al.
Method	RPT	Kohonen Self-Organizing Map
Speed in training	Fast: no iteration	Slow: many iterations
Speed in retrieving	Fast: logarithmic	Moderate: Fixed number of resolutions in search space
Active camera control	Yes	No
High dimensional input	Well suited to handle	Not well suited to handle due to fixed dimension of links between network nodes
Extension to automatic finding of features	Well suited and tested (Swets, Cui, Chen)	Not addressed
Extension to action sequence	Tested (Hwang)	Not suited because of limitation on input dimensions

2.5 Neural Networks

There are numerous ways to approach computer vision. One extreme approach advocates the structuring of computer vision systems exactly as the human visual system is structured. That is, computer vision has to be *done* in the same way that human vision is *done*. Unfortunately, we do not know *exactly* how the human vision system is structured. The other extreme holds that the important aspect of the human visual system is its *functionality*, i.e. *what it does*, and not how it does it. Researchers in the latter group are content to use whatever method will accomplish the goals they are attempting to achieve, irrespective of whether or not their adopted approach in any way resembles the human vision process. On the other hand, the former group of researchers is very much interested in determining, as much as possible, *how* the human visual system works; and then attempting to build systems which mimic the

human visual process in the hope of thereby attaining the same functionality. Of course, few researchers fall explicitly into one of these extremes—most fall somewhere in between, trying to find a middle ground which takes into account the best of both extremes. The following sections provide a brief summary of a historical view of the human neural network and research in creating artificial neural networks based on these views.

2.5.1 Biological Neural Networks

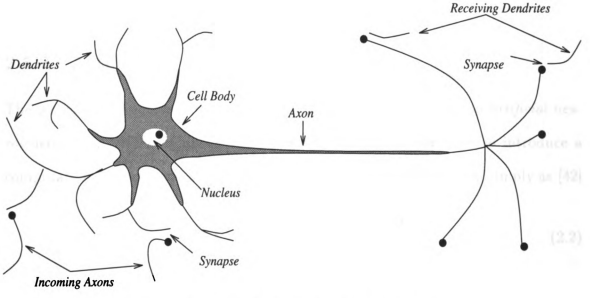


Figure 2.2: An initial concept of a biological neuron.

Some work in human vision has centered around the concept of the *neural network* of the human central nervous system. Figure 2.2 shows the concept of a *neuron* which motivated some of the following work in artificial neural networks. In this view the neuron is the brain's processing *unit*. It is a single cell whose input signals are brought to it on *dendrites*, and whose output signal is sent out via the *axon*. There are usually many dendrites attached to each cell. Each of the dendrites applies a *weight* to its incoming signal, signifying how pertinent the signal is to the

receiving neuron. Depending on the value of its weight, a signal may *excite* or *inhibit* the receiving neuron. The actual input to the cell is then the sum of the weighted signals of all incoming dendrites. The output of the neuron is a non-linear function (believed to be a sigmoidal function) of the weighted sum. The *signal* sent by a neuron is a chemical pulse passed chemically from the end of the axon, across a small gap called the *synapse*, to the waiting dendrites, there usually being more than one dendrite receiving the axionic signal. Several, specific configurations of neurons have been hypothesized/verified in the central nervous system, but all have the general characteristics described above.

2.5.2 Artificial Neural Networks

The technological equivalent of the human neural network is called an *artificial neural network* (ANN). McCulloch and Pitts were the first researchers to introduce a computational description of a neuron's activity. Their model is given simply as [42]

$$n_i(t+1) = \Theta\left(\sum_j w_{ij} n_j(t) - \mu_i\right) \tag{2.2}$$

where $n_i \in \{0,1\}$ is the state of the i^{th} neuron at time t; w_{ij} is the weight on the connection between the i^{th} and j^{th} neuron; μ_i is the threshold for neuron i; and $\Theta(x)$ is the *step function* defined as:

$$\Theta(x) = \begin{cases} 1 & if\ x \geq 0; \\ 0 & otherwise. \end{cases} \tag{2.3}$$

In general, a negative weight indicates an inhibitory signal and a positive weight indicates an excitatory signal. A schematic of this simple model taken from [42] is given in Figure 2.3.

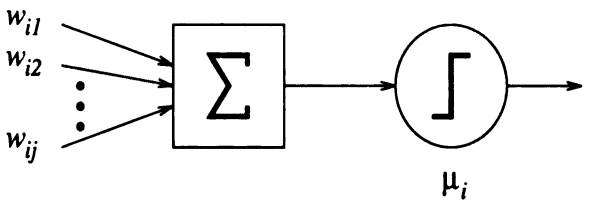


Figure 2.3: Schematic of a McCulloch-Pitts neuron.

The history of ANNs proceeded as follows ([97]). The McCulloch-Pitts model was improved when Hebb introduced his ideas on how neurons *learn* new, and adapt to changing, information. His model centered on the idea that the synaptic weights change to reflect the data being received. These ideas have since been associated with a weight updating rule which characterizes Hebb's hypothesis and is called the *Hebbian learning rule.* This rule allows a network of neurons to learn several patterns. If we assume that the neurons are binary (firing/not firing), and the input patterns are simple bit strings of -1s and 1s, then the rule is defined as [42]:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu \tag{2.4}$$

where w_{ij} is the weight between the i^{th} and j^{th} neurons; N is the number of neurons in the network; and ξ_k^μ is the k^{th} bit of the μ^{th} input pattern, out of p patterns.

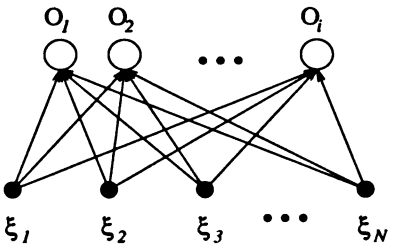


Figure 2.4: Schematic of a simple perceptron.

One of the first artificial neuron (AN) models was the *perceptron* introduced by Rosenblatt. The *simple perceptron* is shown in Figure 2.4 and consists of a single

layer of neurons, the only connections being those on which the input patterns are received. Perceptrons in general consist of N layers of neurons where connections exist from layer i to layer $i+1$ (see Figure 2.4). These networks are now called *feed-forward networks* because all connections proceed to the next higher level, i.e., there are no connections from a high-level neuron to any lower-level neuron, nor are there any connections between neurons within the same layer. Letting $g(h)$ represent the activation function of the output neurons, the output of the simple perceptron is given by:

$$O_i = g(h_i) = g\left(\sum_j w_{ij}\xi_j\right). \tag{2.5}$$

Simple perceptrons are limited in power by the fact that they are capable of computing solutions only for linearly separable problems. In particular, they do not work for the *exclusive-or* problem. Assuming a linearly separable problem, the following learning rule (similar to Hebb's) is used to update the connection weights [42]:

$$w_{ij}^{new} = w_{ij}^{old} + \triangle w_{ij} \tag{2.6}$$

where

$$\triangle w_{ij} = \eta\left(\zeta_i^\mu - O_i^\mu\right)\xi_j^\mu \tag{2.7}$$

where η is the *learning rate*; ζ_i^μ is the desired output; O_i^μ is the actual output; and ξ_j^μ is the input pattern.

The weight updating rule for the general multi-layer perceptron is called *error back-propagation* because the error between the actual output and the desired output is propagated backwards through the network and the weights are updated along the way. The mathematical definition of the updating rule depends on a gradient descent technique and is given in [42].

The perceptron was followed by the ADALINE and the *Widrow-Hoff learning rule*, which was more powerful than that introduced by Hebb. This rule for updating the connection weights is the same as given above for the simple perceptron, but was derived from an *energy function* using a gradient descent technique [42]. Given the *error function* [42]

$$E[\mathbf{w}] = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 \tag{2.8}$$

$$= \frac{1}{2}\sum_{i\mu}\left(\zeta_i^\mu - \sum_j w_{ij}\xi_j^\mu\right)^2 \tag{2.9}$$

the updating rule can be written as [42]:

$$\triangle w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} \tag{2.10}$$

$$= \eta\sum_\mu(\zeta_i^\mu - O_i^\mu)\,\xi_j^\mu. \tag{2.11}$$

The *neocognitron* was developed by Fukushima for use in visual pattern recognition, specifically in the recognition of handwritten characters. A main interest was in modeling the processes of the visual cortex.

2.5.3 Kohonen Maps

Kohonen, Anderson and others began looking at the connection between associative memory and neural networks. Specifically, Kohonen developed his *self-organizing, topology-conserving maps* [54], [55]. The architecture of these neural networks is defined by the connections between the neurons. The neurons are initially assigned random values which act as centroids of subregions in the input space. As training patterns are supplied the values are changed in such a way that the network assumes

a topology which approximates the distribution of the output space as determined by the input patterns. Intuitively, as each input pattern is presented to the network, the neuron whose subregion is closest to the pattern, S, is *pulled* towards the pattern. Additionally, the neighbors of S, within some distance, are pulled along. Given a good representation of input patterns, and enough iterations, the network is eventually "pulled into shape".

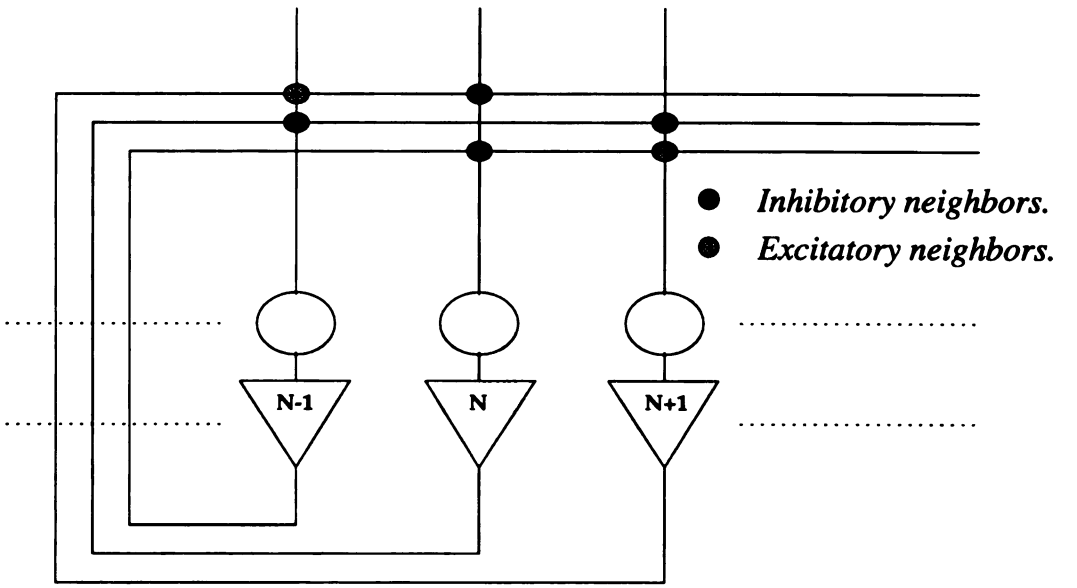


Figure 2.5: Example of a feedback neural network.

Other neural network (NN) architectures were introduced for associative memories, not the least of which was Hopfield's *recurrent* NN architecture. Hopfield's network is a uni-directional, fully-connected, feedback network [32]. A schematic is shown in Figure 2.5. The network is uni-directional in that values flow along the connections in one direction, namely, out of one neuron and into another. It is fully-connected in that the output of each neuron is available as input to every other neuron–but not itself. These networks have the advantage over the feed-forward networks of being able to learn, i.e. update their connection weights, in an *unsupervised* manner. The feed-forward networks did all of their adaptation "off-line" during the

training phase, and consequently only learn under *supervised* conditions. The rule used to update the connection weights of the Hopfield is the Hebbian.

The above mentioned ANNs have been used in computer vision in various ways. There have been systems to perform handwritten character recognition, implementations of clustering techniques, approaches to object recognition [92], and the control of mobile robots [72] and robotic manipulators [35], [65], [65].

2.6 Active Vision

Vision can be categorized into three broad paradigms: passive vision; undirected active vision; and directed active vision. In passive vision the camera is mounted in a static position which is unchanged [12], [56], [65], [17]. An example of undirected active vision would be that used by security systems in which the camera follows a static sequence of moves not based on anything being *seen*. In directed active vision the movement of the visual system may be considered to be governed in such a way as to achieve some defined goal [20], [68], [31], [19], [94], [78], [1], [81], [10], [28], [27], [84], [26]. Aloimonos called this *purposive active vision* [7]. Papanikolopoulos calls it *controlled active vision* [70]. In this thesis we will mean *directed active vision* whenever we use the simplified term *active vision* unless otherwise noted. It is to this type of active vision that the rest of this section addresses.

An active vision system has several parameters which can be *actively* controlled, including the six degrees of freedom for position and orientation, and the intrinsic parameters of focus, zoom and aperture. The major components to be considered for an active vision system include: attention; foveal sensing; gaze control; eye-hand coordination; and integration with robot architectures [9]. *Attention* involves determining what kind of processing to perform on specified *areas of interest* (AOI).

Foveal sensing takes advantage of the higher acuity near the fovea of the camera by providing high resolution sensing in AOI, and low resolution outside of the AOI. This reduces the computational expense compared to processing an entire image at high resolution, without sacrificing the detailed results which are available within the AOI. The concept of *gaze control* involves maintaining a stable fix on an AOI when either it or the vision system is in motion, *gaze stabilization*, as well as moving from one fixation point to another fixation point, *gaze change*. *Gaze control* directly affects a system's ability to break free from the limited field of view provided by a camera and is a major component involved in the process of object tracking.

The use of active vision raises several issues that must be addressed. One issue concerns deciding how the agent, environment and task will be represented in the larger system. This will certainly affect the way in which the active vision subsystem interacts with the other subsystems, such as a robotic manipulator. Another issue concerns the explicit representation of the uncertainties intrinsic to the vision and robotic systems. Additionally, accuracy issues must be addressed. How accurate do the various subsystems need to be in order to perform the required task(s)? What accuracy requirements do each of the subsystems make on the other subsystems? Finally, the task assigned to the system may affect the representations needed to most efficiently perform that task. Types of tasks include tracking objects as they or the camera move, locating objects (desired object is known *a priori*) within the scene, or noticing objects (reconstruct the scene or react to an object when it moves into the visible scene).

An active vision system provides several advantages over a passive vision system. Problems due to occlusion may be eliminated or at least reduced if the sensing system can be moved to a viewing angle which avoids the occlusion. Of course, this preferred viewing angle must be determined. The vision system can be positioned such that the

object of interest lies near the fovea where the visual acuity [3] is higher. The system can continuously track objects which are moving out of the current visible space. In the rest of the section, we review the literature concerning active vision.

2.6.1 Purposive and Qualitative Active Vision

In [7], Aloimonos presents essentially a new way of approaching/looking at the process of computer vision. He calls the old paradigm the *reconstructionist* approach. This approach, he claims, seeks to use computer vision to reconstruct the entire world, so that computer vision is an end unto itself. From this reconstructed world, all information necessary to complete any task in general can be obtained. Aloimonos proposes a *purposive* and *qualitative* paradigm, which approaches computer vision by first defining the task(s) that one desires to perform, and then designing the vision system in such a way that it performs the specific task. In this way, Aloimonos avoids needing to reconstruct the entire world, but rather, only those parts of it that are *specifically required by the task.*

Additionally, Aloimonos wants the modules defined for performing a task to be answering *Yes/No* questions. That is, he wants to use *qualitative* measures of "vision" to perform the given task, as opposed to the reconstructionist approach which uses *quantitative* measures. It seems, however, that many of the same quantitative computations that are used to answer the quantitative questions of the reconstructionist approach would need to be computed in order to answer the qualitative questions of the purposive approach. For example, while it may not be necessary to reconstruct the entire visible scene in order to determine whether an object was moving towards you, you would still need to make some form of quantitative calculations to make the

[3] Here acuity indicates the sharpness of the image, or the keenness of the perception of the scene in that area.

determination that the object is closer at each instance. So, the purposive approach is not *entirely* qualitative in nature.

Aloimonos states that the two most important goals of vision are navigation and recognition. What about reading or tracking? Where do these fit into Aloimonos' paradigm?

In this paper, Aloimonos states the following: "[A general vision system of the future] will consist of a large number of modules, each of which will be devoted to recovering a property of the world from a series of images. There will be the modules of shape from shading, shape from texture, structure from motion, etc. And all these modules will communicate and cooperate in building an accurate description of the environment (i.e. reconstructing it). Of course, this general system will have many more high-level modules which, using the results of the other modules, will perform planning and reasoning."[4]. This seems very similar to what we are already attempting to do in computer vision now, which Aloimonos calls the reconstructionist approach. It would appear that Aloimonos is not so much presenting a new way of *doing* computer vision as much as a new way of *approaching/thinking about* the computer vision process.

In [8] Aloimonos *et al.* discuss the advantages that use of an active vision system would bring to various problems in vision research. These areas include shape from shading, shape from contour, shape from texture, and structure from motion. Their hypotheses are based on the knowledge of the vision system's movements. Knowing the parameters of the movement, and being able to compute the transformations of the stimuli at the local neighborhood level, the possible values of the unknown scene parameters are highly constrained, and much more easily computed.

---

[4][7], pp 348

Surface Reconstruction—Abbott and Ahuja

Abbott and Ahuja combine depth cues from control of the vergence and focus parameters of the cameras, and stereo disparity in order to construct a map of the surfaces in a static scene [1]. They argue that using any of these three in isolation is insufficient to compute the large range of depths which is possible in a general scene. However, used together they provide a powerful system in which they compensate for each other's weaknesses. In this approach local surface patches are selected as targets. Focus and vergence control are used to fixate on a specific target. The focus control provides monocular depth cues. The depth from focus and vergence computations are compared. During this fixation process the focus and vergence parameters are adjusted slightly at each step until the resulting values are in agreement with each other. The result is that the same sample of the scene is visible within both of the cameras. Then vergence is controlled in order to register the two image centers. Finally, a coarse estimate of depth is computed and this coarse estimate is used to initialize the stereo process which computes the scene depth in the local area. These local surface patches are used to build the global surface map of the entire scene in a piecewise fashion. By controlling the camera parameters to point the cameras at specific locations in the scene, Abbott and Ahuja are able to reduce the complexity of the depth from stereo computation.

Locating—Clark and Ferrier

Clark and Ferrier propose a system in [20] to control the point of attention at which the camera is looking. That is, if a system is going to have active vision, then there must be a way for the system to determine where the sensor *should look*. Clark and Ferrier propose a system whereby the most salient feature of the target object is used to direct the visual system towards the object. If the target object is at the center of view, then it will be in better focus and the data will be more precise.

Reacting—Murray, McLauchlan, Reid and Sharkey

In [68], Murray *et al.* propose a system which *reacts* to motion detected within the image sequence being received from the visual system. In particular, Murray *et al.* are interested in detecting movement within the periphery of the image, saccading towards the moving object and then tracking it at the center of the image, that is, within the foveal region. Thus movement acts as a cue to the camera system as to where interesting things are happening and where it should consider looking next. Since it is common that more than one movement happens simultaneously it will be necessary to devise a scheme whereby one single movement, or a small loci of related movement, is given priority by the sensing system.

2.6.2 Vergence Control

Francisco

Francisco [31] uses fine vergence movements of the vision system around the point of fixation in order to estimate the relative depth. The approach requires no *a priori* knowledge of the intrinsic or extrinsic parameters of the camera system. The visual system is controlled with the objective of eliminating the disparity between a feature's position in the left and right images. The procedure employs an iterative approach whereby, for a given vergence angle, a micro-movement is made around the fixation point. This is repeated for a set of angle values. In this way, any object which intersects the set of sweeping planes can have its depth estimated by a *local correspondence operator* which responds to the continually changing vergence angle.

Ching, Toh, Chan and Er

Ching *et al.* use vergence control to concurrently detect the presence of occlusion and specularities within the image sequence being used to guide the vergence of

the stereo camera system [19]. They use a concurrent cross-correlation technique and multi-scale images. Essentially, this approach seeks to skirt the occlusion and specularities problems that static view cameras encounter. By actively controlling the camera system the cameras can be positioned so that the occlusion or specularity is eliminated or at least reduced in severity. Constraints on the values of the coefficients computed for the *multi-scale cross-correlation* (MSCC) algorithm are used to indicate the presence of an occlusion or specularity within the image in the neighborhood over which the MSCC is calculated.

2.6.3 Tracking

Papanikolopoulos, Khosla, and Kanade

Papanikolopoulos *et al.* use a camera mounted on a robotic manipulator to track a moving object [70]. The object is constrained to move in a plane which is perpendicular to the camera's optical axis. They hypothesize that active vision algorithms will perform better if visual information and manipulator control are combined to produce better tracking results. Given a feature point P_k in image k, the *sum-of-squared differences* (SSD) optical flow is calculated using multiple windows in image $k+1$ which are constrained to be within some defined neighborhood of P_k's position in image $k+1$. Each of these windows provides an optical flow measurement, and for each a confidence measure is calculated. The window with the highest confidence is chosen as the one specifying the true direction of the optical flow. This optical flow measurement is then used to control the manipulator's movement so that the neighborhood of P_k is kept stationary at the center of the image. Papanikolopoulos *et al.* call their approach *controlled active vision* because the motion of the robot-camera system is not accidental but specifically planned to maximize the visual information

used in controlling the manipulator. Their experimental results show that the choice of control scheme depends on the algorithms used to extract the visual information.

Zheng, Chen and Tsuji

In [94], Zheng *et al.* propose a new method to use an active camera system to guide a robotic arm in manipulating objects. Traditionally, manipulation approaches have been limited by the precision with which the internal world is modeled; the precision of the imaging system; and the precision of the robotic manipulator. Additionally, it is often the case that the sensing system is constrained to be mounted on the end-effector of the manipulator, or to be fixed in a stationary position. Zheng *et al.* propose that by using active sensing, the system will be able to deal with distance, occlusion and poor resolution—common problems among traditional approaches. With an active camera, the camera can be moved into a position which gives the best possible view of the manipulator and its target object.

Zheng *et al.* use block objects, mount the camera on its own manipulator, separate and independent from the arm which will be manipulating the objects. They define two areas of interest: the stationary goal and the moving target. Additionally, there are two global tasks: moving the manipulator to the target object; and moving the object (grasped by the manipulator) to the target position. In the first task, the stationary goal is the target object, and the moving target is the manipulator. In the second task, the stationary goal is the target position, and the moving target is the object. The two types of areas of interest must be *tracked* through the image sequences. The features used for tracking are a sparse sample of lines and points on the object and manipulator. Simple object recognition is done using color as the only feature; and the approximate position of the object is given.

At each camera position, the idea is to move the gripper so that its image is made to align with the projection of the goal position, L_g, which is defined to be

the vertical axis of the object coordinate system. First, an estimate of the map between the object and the camera coordinate system is calculated. Then, for the given camera position, the displacement, in the image, from the gripper to the object is calculated. A vector is defined which is orthogonal to the camera axis and in the direction of L_g; and the gripper is moved along this vector until it coincides with L_g. Then the gripper is made to make small movements until one of the image vectors falls onto the projection of L_g, namely l_g. In this way, the plane defined by L_g and the camera axis, E_1, is determined and fixed. Next, the camera position is changed. Then the gripper is moved again, except that this time its movement is constrained to be within the plane E_1. At this point, the gripper should be positioned over the object and the only movement left is in the vertical position.

Zheng *et al.* desire that the movement of the camera be such that an optimal aspect view of the object and gripper is obtained. To ensure this, the camera is first moved so that the fixation point and target position of the gripper are both lined up on the central column of the image. Then the camera makes a series of moves consisting of a horizontal movement orthogonal to the camera axis followed by a pan to place the fixation point back on the central column of the image. These small moves continue until the distance between the central column and the columnar location of the target position goes from increasing to decreasing. This point is called the *stopping position* of the camera.

One question that needs to be answered is how many small moves the gripper will need to make, worst case, in order to determine and fix the position of the plane E_1. This method has several desirable properties. Namely, that it uses an active camera system; it uses qualitative control of the camera motion rather than trying to precisely calculate quantitative measures; and it uses the relative positions of the objects and gripper. One of the problems is that there is no retention of any learned information,

and the same procedure must be re-done for each movement/manipulation.

Sharma and Aloimonos

Sharma and Aloimonos analyze the control of visual motion given a desired behavioral task [78]. In particular, they assume the task of *visual interception.* They propose that visual processes which are constrained by the task being performed will provide feedback which is robust and informative enough to succeed at the desired behavior. Their approach uses a qualitative description of the movement of the visual system, as opposed to quantitatively computing the exact position and velocity parameters. In the case of visual interception, Sharma and Aloimonos used small movements of the camera to acquire a sequence of images from which they compute the sign of the normal flow at image points whose spatial gradient is large enough to provide reliable information.

Stelmaszyk, Ishiguro and Tsuji

Stelmaszyk *et al.* use an active vision system to track a specific fixation point in the scene as their mobile robot moves through the environment [81]. The tracking is performed using visual feedback to keep the fixation point at the optical center of the camera. Then, feature points are extracted in a local neighbor of the fixation point and a map of the local scene is built, updated and revised with each subsequent image. The fixation point changes over time as the robot moves, with each fixation point chosen as a feature in the image which provides some cue to the robot, such as indicating an obstacle which needs to be avoided. The resulting map consists of vertical edges, which have appeared in three or more images. This map represents the 3D structure of the environment in the neighborhoods of the fixation points. The more often an edge appears in an image, the stronger its probability of actually being present in the environment. Two cameras are used: a passive one to locate all possible features in the entire scene; and an active camera which is mounted on a rotating

head (rotation in the pan direction) to keep the gaze positioned at the fixation point.

Barth, Ishiguro and Tsuji

Barth *et al.* use the fixation and saccade method in [81] to compute the ego-motion of the robot carrying the camera system [10]. Using feedback from a vision system aids the system in overcoming positional errors introduced by wheel slippage, etc. In their approach, the camera is controlled to fixate on a point in the image and then the optical flow is computed in the sequence of images seen while fixating. When ever the camera is not gazing in the direction of the *instantaneous translation vector*, there is a perception of motion parallax. *Motion parallax* is the observation that objects which are behind the observer's current gazing point are moving in the same direction as the observer, while objects in front of where the observer is looking appear to be moving in a direction opposite to the observer's motion. Given this observation, the optical flow patterns are divided into two groups moving in opposite directions. Then the camera is caused to saccade towards the direction in which the optical flow magnitude is greatest. In this way the camera system converges on the direction of instantaneous translation. Once this direction is reached, motion parallax will no longer be observed and thus the translational vector of ego motion will have been established. The rotational parameters are determined as equal to, but of opposite sign to the rotational saccades performed by the camera in converging on the instantaneous translation vector. This approach depends a great deal on *a priori* knowledge including the maximum length of the optical flow vector, the physical measurements of the camera system, and the intrinsics of the camera, in particular the focal length. A change in any of these would require re-coding of the algorithm. The advantage is that both the translation and rotational parameters are calculated at the same time rather than determining each individually, reducing the computation and time complexities.

2.7 Decision Making

Much of the work done in the pattern recognition (PR), artificial intelligence (AI), image processing (IP), and computer vision (CV) fields involves decision making:

- Which class does pattern p belong to (PR)?
- Which medication should be prescribed, given the patients symptoms (AI)?
- Is pixel p part of the background or part of an object (IP)?
- Given the available visual information, should the mobile robot continue moving straight ahead or turn (CV)?

In order to make a decision, we need information on which to base the decision. Each type of information is a *feature* of the decision making process; and each feature can typically take on one of a set of possible values. Additionally, a means by which the information is evaluated so as to come to a conclusion must also be specified.

A decision tree is a way to sequentially evaluate a set of features in order to make a decision. This is especially useful if the features' values have not all been obtained prior to the decision making process, since feature measurement can be expensive. By evaluating the features sequentially, it is possible that a decision can be made without evaluating all the features. The most advantageous approach therefore, is to analyze the features in order to rank them in such a manner that those features which provide the most discerning information are evaluated early in the process. In particular, a decision tree can be used to plan a course of action, as it is sometimes used in the AI community. In this case, each node of the tree is either a decision making node ("choose action A, B, or C") or an outcome node ("event E can have outcome A, B, or C") [16]. This represents a statistical approach to decision making since the cost of taking a certain action must be taken into consideration as well as

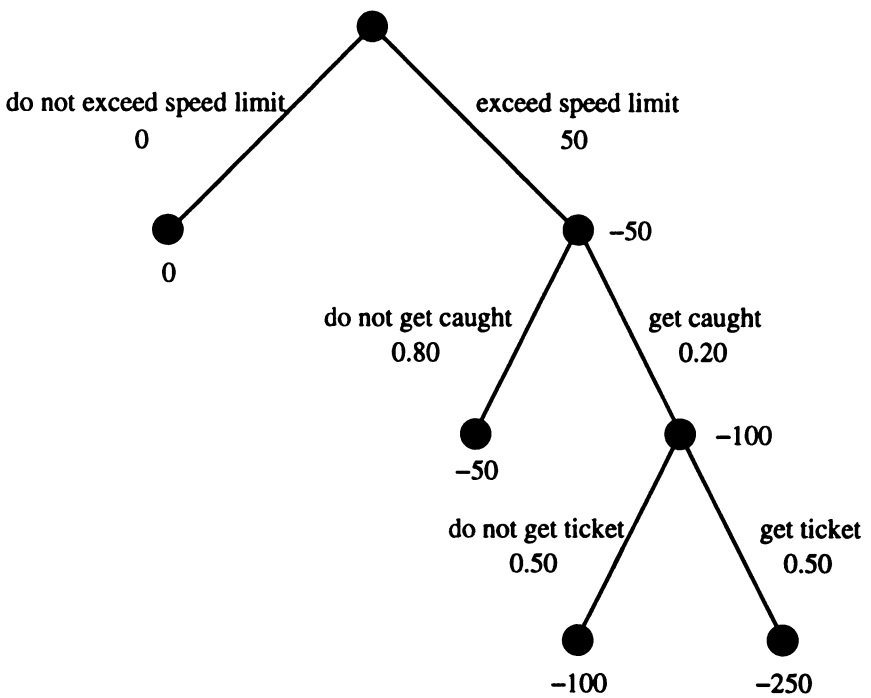


Figure 2.6: An example of a decision tree complete with costs for taking action, the utility for choosing a certain course of action, and the probability of each outcome occurring.

the penalty incurred if the wrong decision is made [23]. Also, what is referred to in [16] as the *utility* of possible outcomes must be evaluated in order to rank those outcomes (see [16] for details). This utility may be positive (a benefit) or negative (a loss) and is used to evaluate the utility of taking a particular course of action, given the probability of the outcome occurring.

A similar decision making approach used in PR is the hierarchical classification tree [50]. In this approach a pattern is presented as a vector of features. The internal nodes of a classification tree are decision points at which one or a few features of the test sample are used to decide which branch to follow down the tree in order to reach a leaf node. Each branch represents a non-empty, unique set of values which

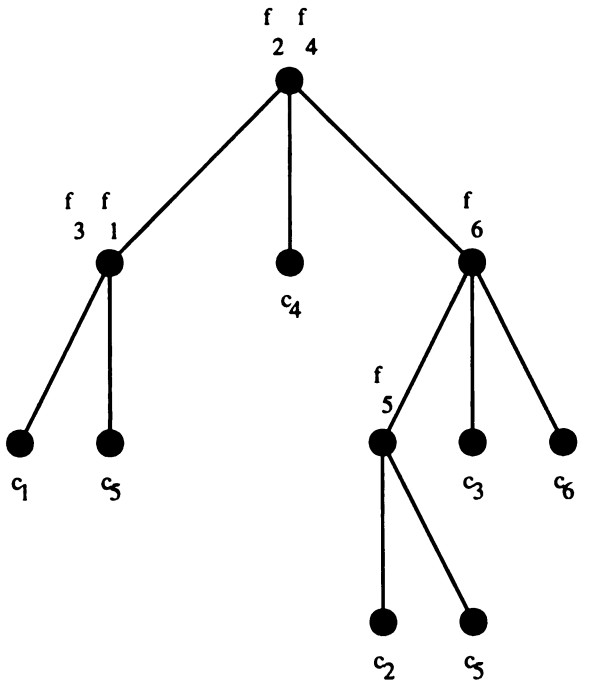


Figure 2.7: A hierarchical classification tree. The features $f_1, \ldots, f_6$ are used to decide between class $c_1, \ldots, c_6$.

the feature(s) may take. Which set the actual value(s) of the feature(s) fall(s) into determines which branch will be followed. The leaf nodes represent the classes into

which the test sample may be classified. Thus, once a test sample reaches a leaf node it has been classified. An example is shown in Figure 2.7.

The advantages of such approaches include a reduction in the number of, and therefore the computation of, features needed to perform the decision making; and higher processing speed since a given input pattern is compared to only a subset of the training samples. Additionally, this approach allows for features with non-numeric values such as color, size, height, etc. [64]. An example is shown in Figure 2.8.

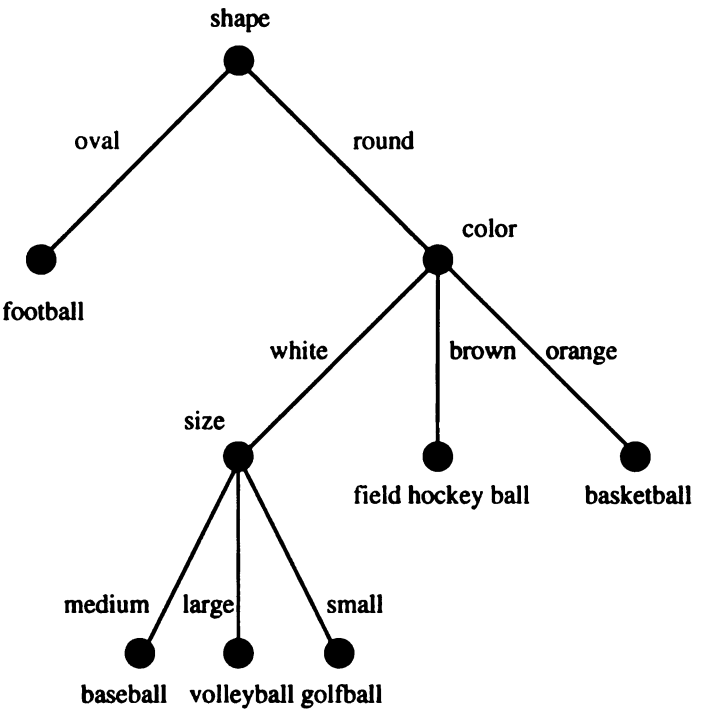


Figure 2.8: An example of a classification tree based on non-numeric features.

The work presented in this thesis takes a similar approach to decision making. Specifically, a recursive partitioning algorithm is used to build a hierarchical tree classifier. This classifier uses a nearest neighbor classification method based on the Voronoi tessellation as its decision making criteria. The approach in this thesis enjoys the advantages of reduced computational complexity and reduced computational time.

It differs in several respects. First, a nearest neighbor classification based on Euclidean distance is used, thus all features are used at every decision node. Second, all internal nodes may be considered decision nodes at which the *decision* is to decide which sub-region of the input space is to be explored further in the hopes of finding a training sample (or set of) which is near the test sample. Third, the "outcome" at a leaf node is determined via inter-node interpolation. Fourth, an *expected radius* is used to determine which training samples will participate in the decision making process at a given level of the tree. Finally, the training samples used to construct the decision making structure are chosen randomly, in an unsupervised fashion, and not because they will be particularly useful for decision making.

Chapter 3

Framework for Learning and Performing Hand-Eye Coordination

3.1 Motivation

3.1.1 The Proposed HEC System

The schematic of the proposed HEC system is repeated in Figure 3.1 for convenience. The proposed HEC system has the following sub-systems as seen in Figure 3.1.

Recognition and Segmentation System

The *Recognition and Segmentation System Module* (RSS) will provide the system with its recognition capabilities. The RSS will be responsible for recognizing and locating the manipulator's end-effector, hereafter referred to as *hand*, and the object of interest, hereafter referred to as *object*, within the left and right images, as well as the disparities between the two image positions. The RSS's input, D, will be

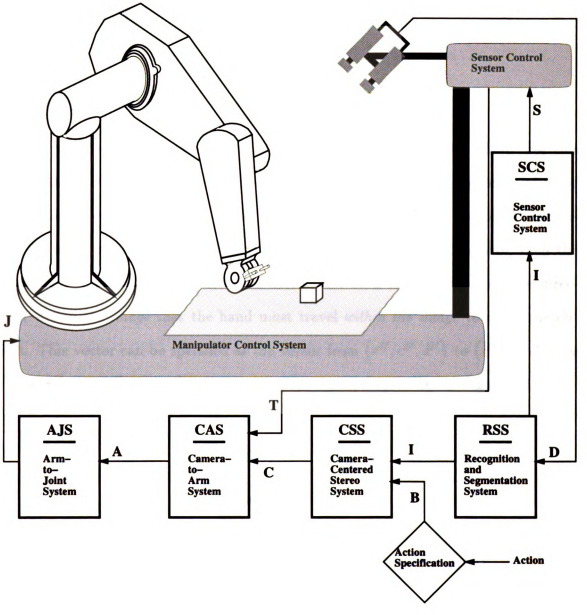


Figure 3.1: Schematic of proposed system.

a pair of intensity images, one from each sensor. The output from the RSS will be $I = \left(r^O, c^O, d^O, r^H, c^H, d^H\right)$ where $\left(r^O, c^O\right)$ is the object's position and $\left(r^H, c^H\right)$ is the hand's position specified in the rows and columns of the left input image. The $\left(d^O, d^H\right)$ pair specify the disparity between the object's column position in the left and right images, and the hand's column position in the left and right images, respectively.

Action Specification

The *Action Specification Module* (ASM) specifies the actions to be taken. For this thesis, that *action* will be to grasp the specified object. We envision a process whereby the action is encoded into the vector I producing an augmented vector I'. For example, if the manipulator and object image positions are given as above, then the desired manipulator movement would correspond to a vector indicating the direction *within the image* that the hand must travel *within the image* to reach the object. This vector can be specified as the vector from $\left(r^H, c^H, d^H\right)$ to $\left(r^O, c^O, d^O\right)$, or $\left(r^O - r^H, c^O - c^H, d^O - d^H\right)$. This concept is shown graphically in Figure 3.2.

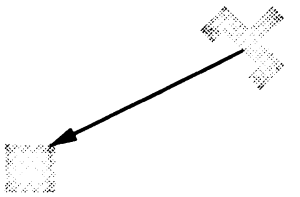


Figure 3.2: Vector specifying direction from hand to object.

Camera-Centered Stereo System

The *Camera-Centered Stereo System Module* (CSS) receives the vector I' as input. Given the target position of the hand in image space, $\left(r^O, c^O, d^O\right)$, the CSS has

the responsibility of mapping that target position to the corresponding position in the *camera coordinate system* space. That is, the CSS is charged with learning and maintaining the image to sensing system calibration. The CSS outputs the hand's target position specified in the camera coordinate system. The output is given as the vector $C = \left(x^C, y^C, z^C\right)$.

Camera-to-Arm System

The *Camera-to-Arm System Module* (CAS) receives the C and T vectors concatenated into a single vector C' as input. The vector T contains parameters from the sensor control system. These parameters are needed to incorporate an *active vision* system into our overall system. These parameters are those describing the pan, tilt, vergence, etc., of the binocular camera system. This is necessary because even if the sensors move, so that the scene *looks* different, and the object and hand appear in different positions *in the camera images*, the positions of the object and hand *relative to each other* in the arm/world space have not changed. Consequently, the required movement in the arm/world space to reach the object remains the same. Therefore, the position of the sensors with respect to the manipulator must be taken into account. For this thesis, T contains only the pan and tilt parameters, (α, β).

The CAS, then, is responsible for learning the camera system to arm/world system calibration. That is, the CAS must transform the object position as given in the camera coordinate system into the object's position in the arm/world coordinate system. To do this, it must take into account the position of the head (which affects the position of the eyes) as specified by the pan and tilt parameters. The output vector is $A = \left(x^A, y^A, z^A\right)$.

Arm-to-Joint System

The *Arm-to-Joint System Module* (AJS) receives the A vector as input. This network is responsible for learning the mapping between the target position, to which the manipulator is supposed to move, and the incremental angle adjustments of the manipulator joints needed to perform the move. An incremental angle adjustment is specified for each joint in the arm. These are output as vector $J = (\triangle\theta_1, \triangle\theta_2, \cdots, \triangle\theta_N)$ for an N-jointed arm. Additionally, J may contain the acceleration and velocity at which the arm is to perform the specified movement. The vector J will be sent to the manipulator control hardware.

Sensor Control System

The *Sensor Control System Module* (SCS) receives the I vector as input. Given the position of the object within the image space, it is the task of the SCS module to position the camera system (consisting of two cameras mounted on a pan/tilt head) so that the object is near the line specifying the focus of attention of the left camera. This module transforms the image space coordinates into the pan/tilt space. The output is the vector S specifying the pan and tilt parameters which will move the camera system into the desired configuration.

3.1.2 Generalization of Sub-Systems

Through careful consideration of each of the sub-systems listed above, we can view the process of HEC as a sequence of mappings from an input space to an output space. Given this view, we can envision a framework in which there is an entity for each mapping. In this framework each entity is responsible for learning the transformation from its input space to its output space. Each entity is also responsible for storing

this acquired knowledge and using it to efficiently perform the transformation upon receiving a member of its input space, and delivering the estimated member of its output space. Given this general view of an HEC system, we further envision a framework in which all entities are similarly constructed through training; have a similar internal structure; have uniform inputs and outputs; perform the exact same process in calculating the output; and are utilized within the system in exactly the same way.

This general framework provides us with a means of systematically dealing with the complex relationship between the sensors, manipulator, and environment without relying on the availability of accurate information about the sensory and mechanical systems or their relationship to the environment. Additionally, we expect a system built on this framework to provide enough flexibility that the system need not perform precise movements in order to accomplish its task—hand-to-eye coordination.

We have partially implemented and tested (in simulated and real setups) just such a framework. The following *general* assumptions are made:

- the function being estimated can be expressed as a mapping from an N-dimensional input space to an M-dimensional output space;
- The input and output space can be sampled in a correlated manner (that is, given a sample vector from the input space its corresponding output vector can be sampled);
- no occlusion;
- no intervening objects obstructing the path of the manipulator;
- the target position is visible to the camera system at all times throughout execution of the task;

- the gripper-identification point is visible to the camera system at all times throughout execution of the task.
- the target position is reachable by the manipulator (within the reachable workspace and not a singularity point).

The current *implementation* makes the additional assumptions that:

- only the cartesian coordinates of the target position are needed;
- no orientation of the end-effector is required to position end-effector;
- there exists a vision system capable of locating the target and gripper-identification points in the images.

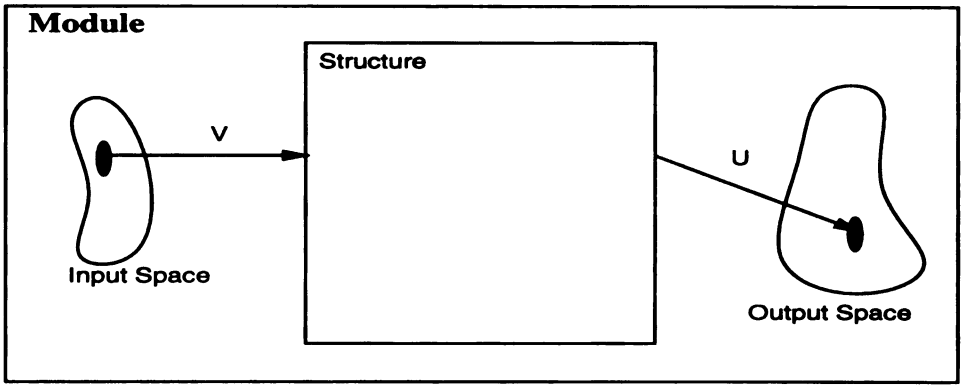
The following sections discuss:

- The Framework's Module
- Nodes of the RPT
- Functionality of an RPT
- Constructing an RPT via Unsupervised Learning.

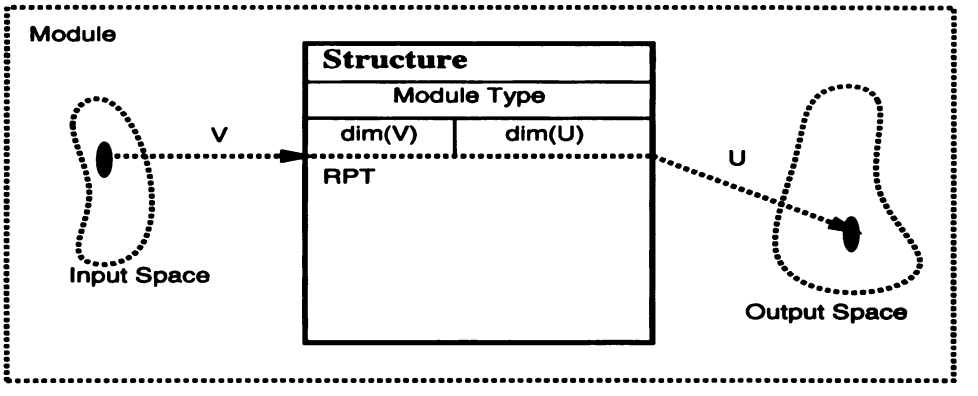
3.1.3 The Framework's Module

In the implemented framework, the *entity* responsible for each *mapping* is called the *module.* Figure 3.3 graphically illustrates the following important definitions.

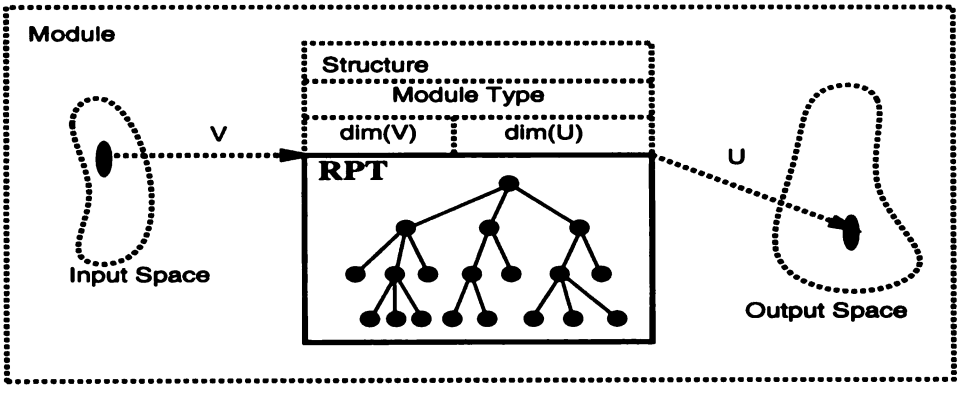
Module: As shown in Figure 3.3(a), we define the term **module** to include an input space, an output space, and the entity responsible for the mapping from the input to the output space. In our framework, the input and output are in the form of n and m dimensional vectors, respectively.



(a)



(b)



(c)

Figure 3.3: The organization of the module of the general framework.

Structure: As shown in Figure 3.3(b), we define the term **structure** to be the entity of the module which is responsible for the mapping between the module's input and output space. This structure contains "global" information about the module—such as an internal indicator of the type of mapping being performed, and the values of n and m for this module—used by the shell processes (defined on the following pages).

Network: As shown in Figure 3.3(c), the major portion of the structure is the *hierarchical neighborhood network.* This network is what the core processes are concerned with constructing, training and using. It is the RPT which organizes, learns and stores the knowledge of the mapping between the module's input and output spaces. The organization and learning are obtained during a training phase. The RPT, its organization, training and use are discussed in further detail in the following sections.

Core Processes: In order to facilitate the design and implementation of the framework it is necessary to understand the distinction we make between **core** and **shell** processes. The core processes are those which operate oblivious to the domain and range of the module and thus the specific mapping being learned. The only information the core processes are given are the dimensions of the input and output vectors. It is with the core processes that we are able to construct the internal networks, and learn and store the transformation between the input and output spaces of the various modules in a systematic manner. These core processes are available to, and used by, the shell processes to build the various modules of the HEC system[89].

Shell Processes: The shell processes then are those processes which know the mapping of each module, and the specific domain and range of the module. These are the processes responsible for obtaining the corresponding I/O pairs for training a specific module. These are the processes which determine how the various modules are put together to form the complete HEC system. Essentially, the shell processes are

high level, user designed processes for implementing a system composed of a set of modules. Figure 3.4 shows the relationship between the core and shell processes[89].

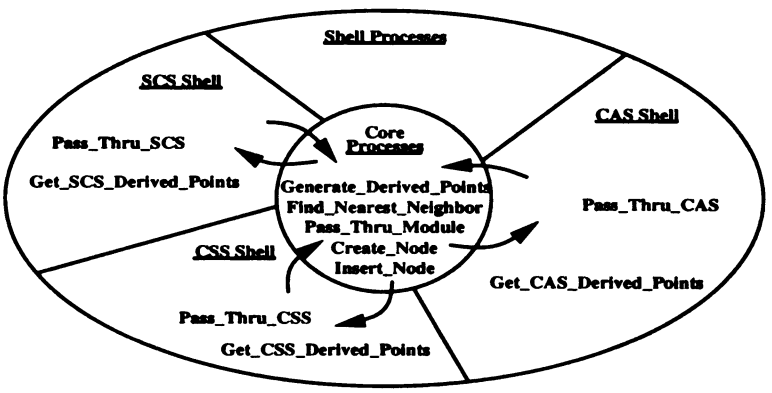


Figure 3.4: Relationship between core and shell processes.

3.2 Recursive Partition Tree

This section sets forth the procedure by which a vectorized input space is partitioned and a *Recursive Partition Tree* (RPT) is built. A brief explanation of how a RPT is used is presented here. A RPT is built during training using a set of pairs of training vectors where each pair consists of a vector sampled from the specified input space and its corresponding vector from the output space. Once a RPT exists, it is used to perform two functions. First, given a query vector from within the input space, the RPT is used to search the input space to locate the k training vectors which are near neighbors to the query vector. This search can be shown to occur in $\mathcal{O}(\log n)$ time given n training samples (see Section 3.4). The near neighbors all reside at leaf nodes of the RPT. These leaf nodes contain the information needed to map their

respective input space vectors to the corresponding output space vectors. Thus the second function performed using the RPT is to use the information in the retrieved leaf nodes to compute an output space vector for the query vector.

The first section is devoted to explaining the working definitions of the terms *partition*, *recursive partition* and *recursive partition tree* (RPT). The second section presents an example of how a RPT is used to quickly search the set of training samples to find a near neighbor for a given query vector. The third section shows how the input space is partitioned and a RPT is constructed based on the given training set. The fourth section briefly describes the KNDB and KNDB2 function approximation methods used to compute the output vector for a query vector given the k nearest neighbors found during the retrieval phase. The chapter concludes with a discussion on the RPT.

3.2.1 Definitions

In order to understand the following definitions, it is important to clarify what is meant, in this approach, by the term *nearest neighbor*. Given a training set $\mathcal{X} = \{X_1, \ldots, X_i\}$ where $X_i \in S^N \forall i$ and a vector $X \in S^N$, the nearest neighbor of X in $\mathcal{X}$ is generally defined to be that $X_i \in \mathcal{X}$ which minimizes a specified distance function $\mathcal{D}(X, X_i)$. However, based on this definition of nearest neighbor, the method described in the following sections is not guaranteed to find the nearest neighbor of X unless $X \in \mathcal{X}$. However, the method does find *near* neighbor(s) of X based on the method used to search the training samples, which can be considered to be *nearest* neighbor(s) based on this method. Therefore, throughout the rest of this paper, the term *nearest neighbor(s)* should be read to mean "nearest neighbor(s) *as determined by* the described method".

Partition

Let S^N be an N-dimensional space.

Definition: For all $X_i, X_j \in S^N$, $D_{ij} = \mathcal{D}(X_i, X_j)$ denotes the computed distance between vectors X_i and X_j, where $\mathcal{D}(\cdot,\cdot)$ is a defined distance measure.

One possible distance measure is the *Euclidean Squared Distance* defined as:

$$D(X_i, X_j) = \left[\sum_{k=1}^{N} (X_{ik} - X_{jk})^2\right]^{\frac{1}{2}} \tag{3.1}$$

Let T be a maximum distance threshold of D_{ij}. Then T represents the maximum distance allowed between X_i and X_j such that X_j is considered to be in the neighborhood of X_i. We call T the *radius* of the neighborhood. Let $\mathcal{X}$ be a subset of S^N.

Definition: The *Voronoi tessellation* [85],[4] of S^N, given the vectors in the subset $\mathcal{X}$, is the partition of S^N which results by assigning a subregion R_i to each vector X_i such that R_i consists of all $X_j \in S^N$ such that $\forall X_k \in \mathcal{X}, D_{ij} \ll D_{kj}$. That is, R_i consists of X_i and all the vectors $X_j \in S^N$ which are *closer* to X_i than to any other $X_k \in \mathcal{X}$.

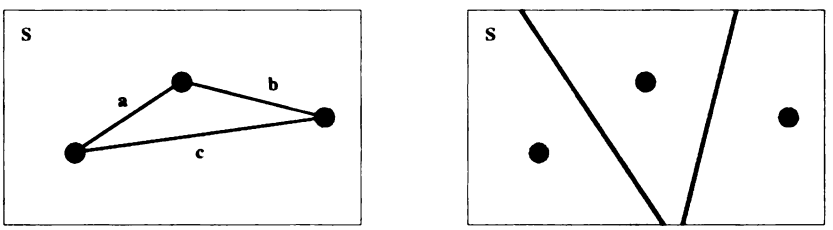
Let $V(\mathcal{X})$ denote the Voronoi tessellation of S^N given $\mathcal{X}$.

Definition: If $V(\mathcal{X})$ is a pairwise disjoint collection of subsets of S^N such that $S^N = \cup V(\mathcal{X})$, then $V(\mathcal{X})$ is a *partition*[51] of S^N.

Since $V(\mathcal{X}) = \cup R_i$ for all i, and the R_i are by definition pairwise disjoint subsets, $V(\mathcal{X})$ is a partition of S^N. Let $\mathcal{X}(T)$ denote any subset of S^N such that the following is true:

$$\forall X_i, X_j \in \mathcal{X}(T), D_{ij} > T \text{ if } X_i \neq X_j$$

That is, $\mathcal{X}(T)$ consists only of vectors which are farther than some radius T from each other. $\mathcal{X}(T)$ need not be an exhaustive set of such vectors in S^N. Let $V(\mathcal{X}(T))$ denote the Voronoi tessellation of S^N given subset $\mathcal{X}(T)$. Then $V(\mathcal{X}(T))$ also defines a *partition* of S^N. A graphical example for $N = 2$ is given in Figure 3.5. Notice that the 2-dimensional space in this example is bounded by a rectangle. In general, S^N may be unbounded. In that case, all subregion boundaries which do not intersect with another subregion boundary continues infinitely through S^N, and the resulting $V(\mathcal{X}(T))$ is also a partition of S^N. In this example, the sub-regions created by the Voronoi tessellation are polygons. In general, for an N-dimensional space, the sub-regions of the tessellation will be hyper polyhedrons.

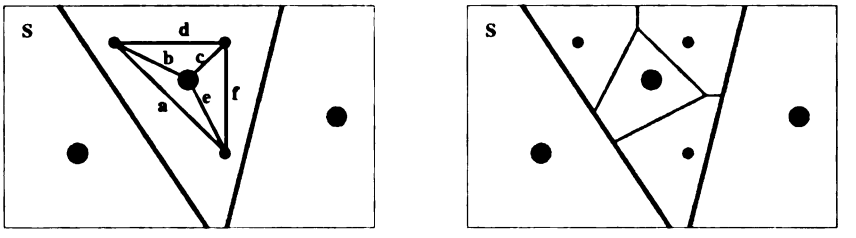


(a) (b)

Figure 3.5: Example of the Voronoi tessellation of a bounded two-dimensional space, S^2. The dots represent three vectors within the two-dimensional space. In (a) the lines labeled a, b and c represent the distances between the three vectors, such that $a > T$, $b > T$ and $c > T$ where T is some specified radius. In (b) the dark lines represent the boundaries between the subregions assigned to each of the three vectors. The union of the three subregions define a partition of S^2.

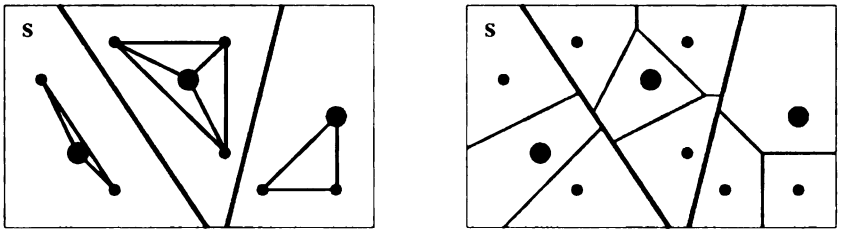
Recursive Partition

Let $T(n)$ be the function which defines the maximum distance radius for parameter n where $T(n+1) \ll T(n)$. Let $T = T(1)$ denote the radius used in the previous section. R_k is the N-dimensional subspace of S^N represented by $X_k \in \mathcal{X}(T(1))$ and defined by $V(\mathcal{X}(T(1)))$. Let $\mathcal{X}(k, T(2))$ denote any subset of R_k such that $X_k \in R_k$



(a) (b)

Figure 3.6: Example of the Voronoi tessellation of a subspace of S^2. The small dots represent three new vectors within the subspace S^2, in addition to the vector represented by the large dot.



(a) (b)

Figure 3.7: Example of a partitioning of space S^2 by partitioning each of the subspaces from the initial partitioning, via a Voronoi tessellation.

and $\forall X_i, X_j \in \mathcal{X}(k, T(2))$ where $X_i \neq X_j$, $D_{ij} > T(2)$. Then we can compute the partition $V(\mathcal{X}(k, T(2)))$ of R_k based on the Voronoi tessellation as described in the previous section. A graphical example for $N = 2$ is given in Figure 3.6. Now, for all $X_k \in \mathcal{X}(T(1))$, partition the subspace R_k as given above. The result is a second partition of S^N,

$$\mathcal{P}^2 = \bigcup_k V(\mathcal{X}(k, T(2)))$$

where $\mathcal{P}^1 = V(\mathcal{X}(0, T(1)))$ and $\mathcal{P}^0 = S^N$. The example in Figure 3.6 is completed in Figure 3.7. A sequence of partitions of space S^N can be recursively constructed by applying the above procedure. Specifically, given a sequence of radii

$$T(0), T(1), T(2), \ldots, T(l), \ldots, T(L-1), T(L)$$

where $T(0) = \infty$ and $T(l+1) \ll T(l)$, we can construct a sequence of partitions of S^N

$$\mathcal{P}^0, \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^l, \ldots, \mathcal{P}^{L-1}, \mathcal{P}^L$$

and $\mathcal{P}^{l+1}$ is constructed based on $\mathcal{P}^l$ just as $\mathcal{P}^2$ was constructed from $\mathcal{P}^1$ in the above example. Specifically, the subregions in P^l are further subdivided in P^{l+1}.

Definition: A sequence of partitions $\mathcal{P}^0, \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^l, \ldots, \mathcal{P}^{L-1}, \mathcal{P}^L$ as constructed above defines a *recursive partition* of space S^N of depth L.

where $\mathcal{P}^l$ is referred to as the partition of S^N at level l, and L represents the maximum level.

Definition: If every region in P^l is a proper subregion of a region in P^{l-1}, then partition P^l is nested in partition P^{l-1}[49].

The sequence of partitions of S^N described above defines a sequence of nested partitions

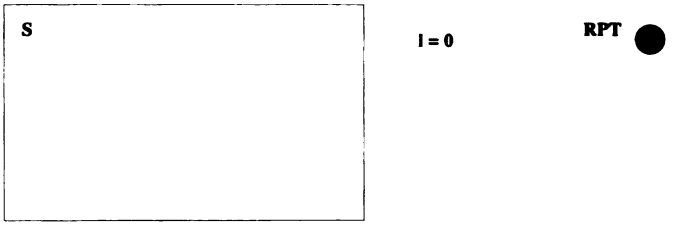
$$\mathcal{P}^0 \subseteq \mathcal{P}^1 \subseteq \mathcal{P}^2 \subseteq \cdots \subseteq \mathcal{P}^l \subseteq \cdots \subseteq \mathcal{P}^{L-1} \subseteq \mathcal{P}^L$$

where the symbol $\subseteq$ stands for *is nested in.*

Recursive Partition Tree

Definition: A *Recursive Partition Tree* (RPT) is a hierarchical representation of a recursive partition of an N-dimensional space S^N.

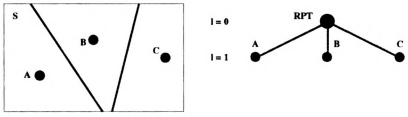
Each level l of the RPT represents a particular partition, $\mathcal{P}^l$, of S^N. In particular, letting $r(l)$ represent level l of the RPT, and $r(0)$ represent the tree root, then $r(0) \mapsto \mathcal{P}^0 = S^N$ and $r(l) \mapsto \mathcal{P}^l$ for all $l = 1, \ldots, L$. Each node at level $r(l)$ represents a vector X_k and the subspace R_k assigned to it in the partitioning of S^N at level l. The node's children (if it has any) represent the subspaces $R_i, i = 0, \ldots, n$ into which R_k is partitioned at level $l+1$ and their respective vectors $X_i, i = 0, \ldots, n$. Figures 3.8— 3.11 show the correspondence between partitions and RPTs for the



(a) (b)

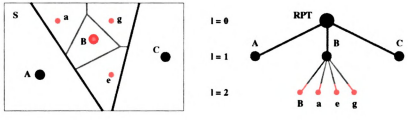
Figure 3.8: Example of a partitioning of space S^2 and its representation by a Recursive Partition Tree. At level $l = 0$ the entire space S^2 is represented by the root node of the RPT.

examples shown in Figures 3.5— 3.7. The examples given in Figures 3.8— 3.11 assume that for all l and for all k, $|\mathcal{X}(k, T(l))| > 1$. That is, at every level of partitioning,



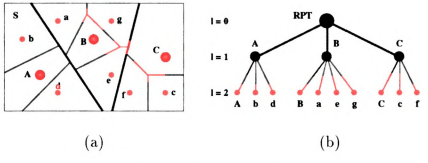
(a) (b)

Figure 3.9: At level $l = 1$, S^2 is partitioned into three subspaces, each represented by a node at level $l = 1$ of the RPT.



(a) (b)

Figure 3.10: At level $l = 2$ one of the subspaces into which S^2 was partitioned at level $l = 1$ is further partitioned into four subspaces, each represented by a node at level $l = 2$ of the RPT as a child of the node representing the larger subspace at level $l = 1$. Notice that the parent node is also represented at level $l = 2$.



(a) (b)

Figure 3.11: The remaining subspaces of level $l = 1$ are further partitioned at level $l = 2$.

every subspace of that level will be partitioned at the next level, until the final level is reached. In general, this is not true since for any X_k it is possible that at level l, $\mathcal{X}(k, T(l)) = \{X_k\}$. That is, X_k is the only member of $\mathcal{X}(k, T(l))$. For the current work it is assumed that if $\mathcal{X}(k, T(l)) = \{X_k\}$ becomes true at level l, then it is true for all levels $l, l+1, \ldots, L-1, L$.

It is important to note that an RPT need not be balanced, or full. Thus a leaf node may occur at any level of the hierarchy. The shape of the tree depends on the distribution and density of the training samples, and the order in which they are presented to the RPT construction algorithm. At higher levels, a leaf node may be responsible for a very large sub-space of the input space.

As shown in Figure 3.12, the RPT is constructed during the training phase when a set of training samples are presented to the construction module. During the query phase, the RPT is used to quickly and efficiently search the training samples to

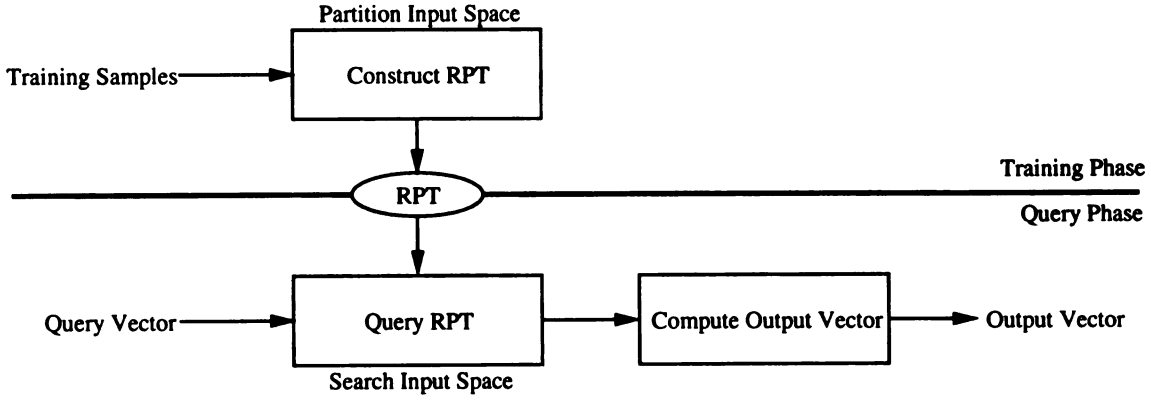


Figure 3.12: Schematics showing the training and testing processes.

locate the K training samples which are closest to the query vector, based on the defined distance measure and the described search method. Then, the output vectors corresponding to the K nearest neighbors, $\{Y_i \mid i = 1, \ldots, K\}$, are used to compute an output vector corresponding to the query vector. Specifically, given an N-dimensional

input space S^N and an M-dimensional output space O^M, let $f(\cdot)$ be the function which maps S^N to O^M. That is, for any $X \in S^N$,

$$f(X) = Y$$

where $Y \in O^M$ and Y is the output vector corresponding to X. Then the RPT combined with a method for computing an output vector for a given query vector based on the K nearest neighbors within the given training set is used to approximate the mapping function f by a function $\hat{f}$ such that for a given $X \in S^N$

$$\hat{f}(X) = f(X_1, \ldots, X_K)$$

where X_K is the Kth nearest neighbor of X within the given training set. The actual computation of the output vector may vary according to the current application. The simplest computation would be

$$\hat{f}(X) = f(X_1) = Y_1$$

where X_1 is the nearest neighbor to X and Y_1 is its corresponding output. However, this approach can lead to large errors if X is at the periphery of the sub-space represented by the training sample X_1. Another approach would be to interpolate over the sub-space represented by training sample X_1 in order to compute an output vector for X. A third approach, used in the experiments described in Section 4.2, is to interpolate over the set of K nearest neighbors found during the retrieval phase, $K > 1$. This approach is found to produce favorable results since it is better able to handle noise in the training set. Some results are described in Section 4.2.

The following sections show how the RPT is used during the query phase to retrieve the K nearest neighbors within the training set for a given query vector, and how the RPT is constructed.

3.2.2 Retrieval Phase

Given a query vector, Q, and a RPT, the vector within the set of training samples which is a near neighbor of Q is located as follows. Let $\mathcal{X}^1 = \{X_i \mid i = 1, \ldots, n\}$ be the set of nodes at level $l = 1$ of the RPT. Compute the distance $\mathcal{D}(Q, X_i)$ for all

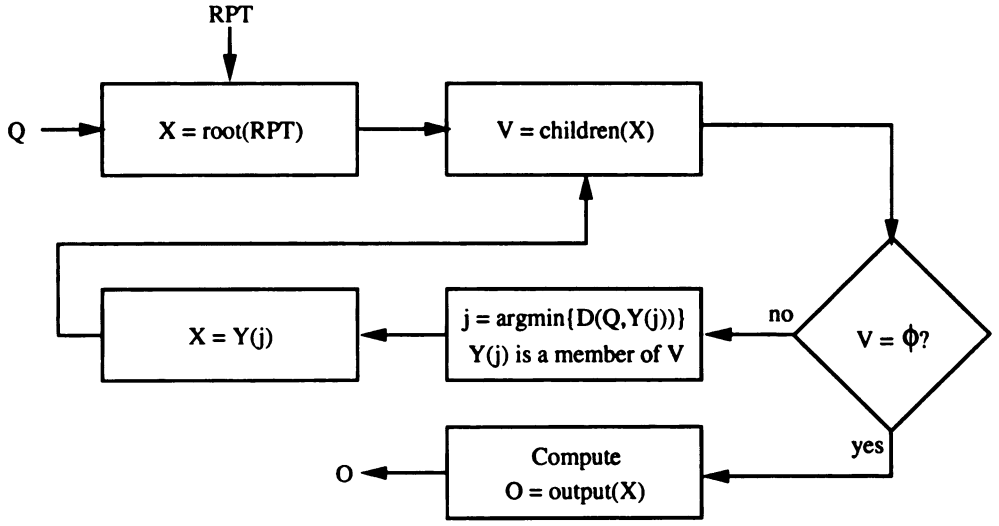
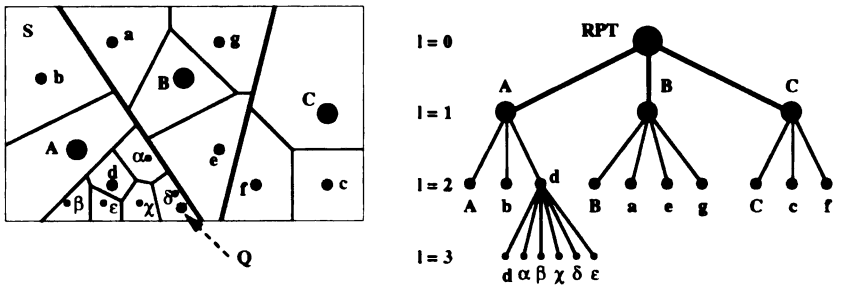


Figure 3.13: Schematics of retrieval process given query vector Q.

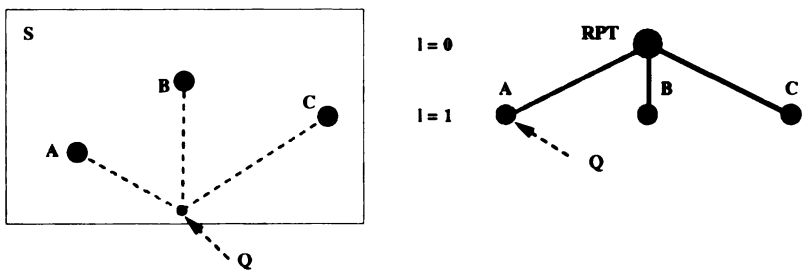
$i = 1, \ldots, n$. Let $D^1 = \min\{\mathcal{D}(Q, X_i) \mid i = 1, \ldots, n\}$. Then the vector X_i is Q's near neighbor at level $l = 1$. At level $l + 1$, only the nodes which are children of the near neighbor at level l, namely X_i, are searched for Q's near neighbor at level $l + 1$. Thus the search continues down the tree within the subtree rooted at node X_i. The descent down the RPT is continued until a leaf node is reached. The training sample associated with this node is a near neighbor to Q within the given set of

training samples used to construct the RPT. The algorithm for retrieving the near neighbor and computing the output vector is given in Figure 3.13. This algorithm does not guarantee that Q's near neighbor will be its *nearest* neighbor within the training set. (See the discussion at the end of this chapter.) A graphical example for a 2-dimensional space is given in Figures 3.14— 3.17. In this example, a



(a) (b)

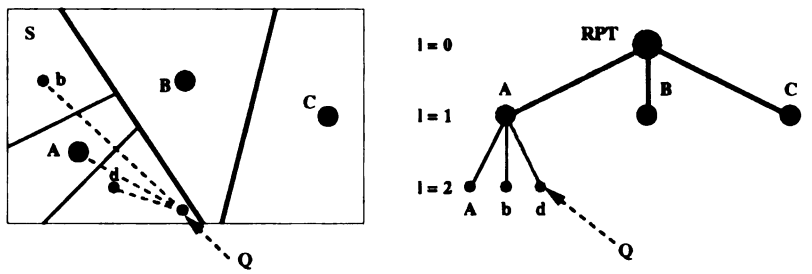
Figure 3.14: Example of retrieval process for a two-dimensional space: location of query vector.



(a) (b)

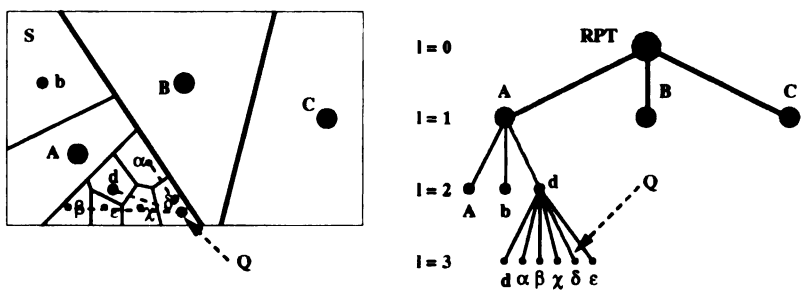
Figure 3.15: Example of retrieval process for a two-dimensional space: level $l = 1$.

vector Q is given as a query into the RPT as shown in Figure 3.14— 3.17. The goal is to find the vector $X_i \in \mathcal{X}$ which is the nearest neighbor to Q. First, the distances from Q to all training vectors corresponding to nodes on level 1 of the RPT are computed (Figure 3.15(a)). Let A represent the node at level 1 which is closest to Q (Figure 3.15(b)). Then the distances from Q to all training vectors



(a) (b)

Figure 3.16: Example of retrieval process for a two-dimensional space: level $l = 2$.



(a) (b)

Figure 3.17: Example of retrieval process for a two-dimensional space: level $l = 3$.

corresponding to nodes on level 2 of the RPT which are children of node A are computed (Figure 3.16(a)). Let d represent the node closest at level 2 closest to Q (Figure 3.16(b)). Finally, the distances from Q to all training vectors corresponding to nodes on level 3 which are children of node d are computed (Figure 3.17(a)). Let δ represent the node at level 3 which is closest to Q (Figure 3.17(b)). Then the vector δ is a near neighbor of Q. Finally, the output vector corresponding to δ is used to compute an estimated output vector for Q.

Optionally, rather than searching only the subtree rooted at the nearest neighbor's node at each level, the k nearest neighbors' subtrees can be searched and their k output vectors used to compute the output vector for Q. The next section describes the process used to construct the RPT.

3.2.3 Training Phase

Given a set of training samples $\mathcal{X} = \{X_i \mid i = 1, \ldots, n\}$ taken from a given input/output space pair, the RPT is constructed as follows. The order in which the samples are given determines the way in which S^N is partitioned as well as the shape of the RPT. Each node in the RPT is labeled with the vector, X_i, which corresponds to the subspace in R_i represented by that node in the partition of S^N for that node's level. Without loss of generality, the root node of the RPT can be set to any point in the space S^N, or to the first vector in the list of training samples. However, if the latter is chosen the resulting RPT may be very skewed if the vector is not in the vicinity of the center of S^N. Thus, the RPT is initialized at level $l = 0$.

For any level l, each training sample is considered, in the order given, for inclusion into the RPT at level l. Let Q be the current training vector. First, Q is used as a query vector into the existing RPT (in the manner given in the previous section) until

a leaf node, X_i is reached. If X_i is at a level less than $l-1$, then Q is not inserted at level l. If X_i appears on level $l-1$, then the distance $\mathcal{D}(Q, X_i)$ is computed and compared to the level l radius $T(l)$. If $\mathcal{D}(Q, X_i) > T(l)$, then X_i and Q are added as children of X_i at level l. Otherwise, a node is not added for Q at level l. If X_i is a leaf node at level l, then the distance $\mathcal{D}(Q, X_i)$ is computed and if $\mathcal{D}(Q, X_i) > T(l)$, then a node is added for Q at level l as a sibling of X_i—that is, a child of X_i's parent node.

Each level of the RPT is fully constructed from the given set of training samples before the next level is begun. This insures that there will be no adoption problem (see Appendix B for details) in the initial RPT. The algorithm for constructing level l of a RPT is shown in Figure 3.18. A graphical example for a two-dimensional space is shown in Figures 3.19— 3.35. In this example, vector A is added as the first node at level $l = 1$. B is added at level $l = 1$ because it is farther from A, its nearest neighbor already in the RPT, than the radius for level $l = 1$. a and b are not added at level $l = 1$ because they are closer to A than the level 1 radius. C however is far enough away from its nearest neighbor B, so it is added. Nodes are not added at level $l = 2$ for training vectors α, β, χ or δ because they are closer to their nearest neighbor, g, than the level $l = 2$ radius. Similarly, no node is added at level $l = 2$ for vector ϵ. Finally, all remaining nodes are added to the RPT, except for ϕ which is closer to its nearest neighbor, d, than the level $l = 3$ radius.

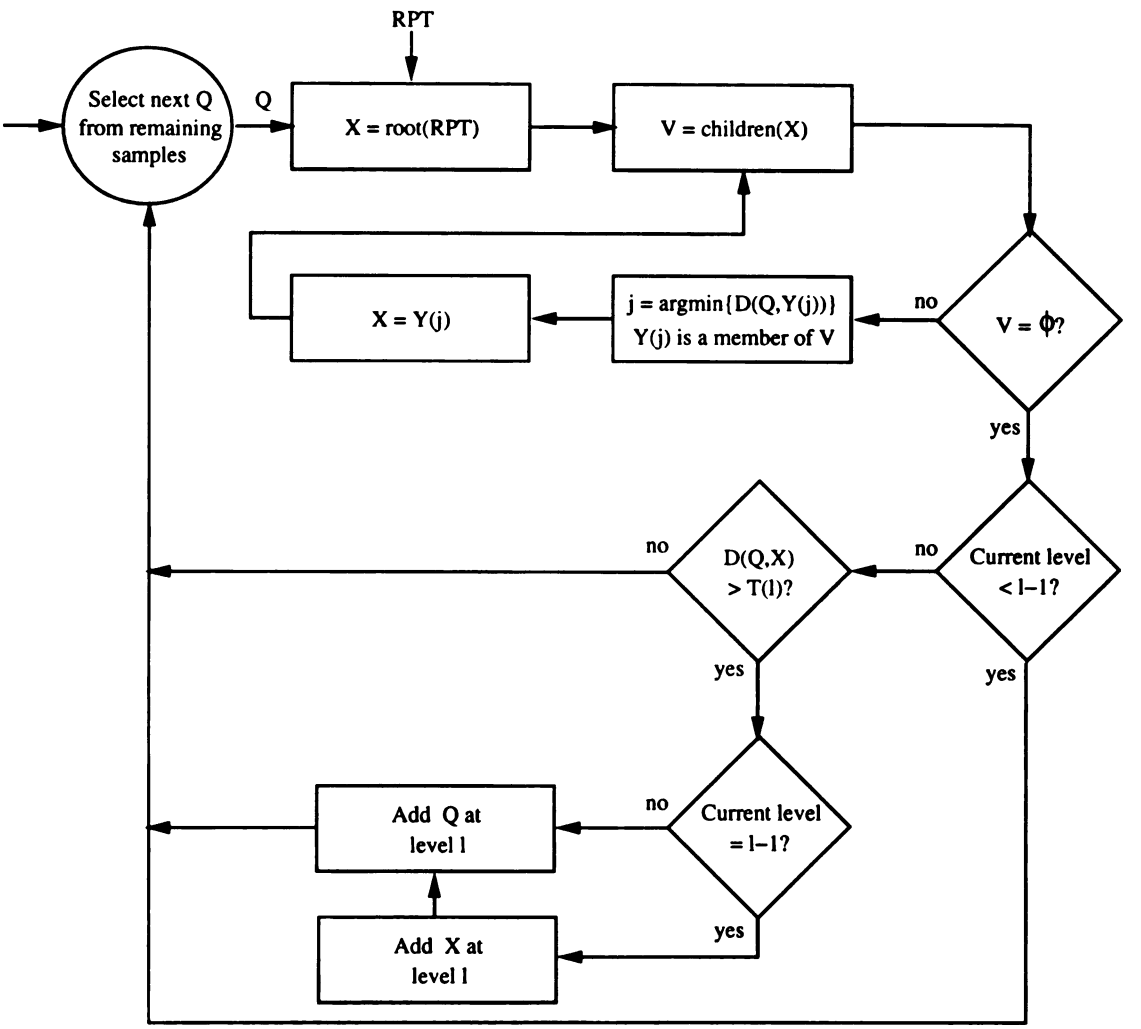


Figure 3.18: Schematics of the level l construction process given the set of training samples $Q_1, \ldots, Q_n$.

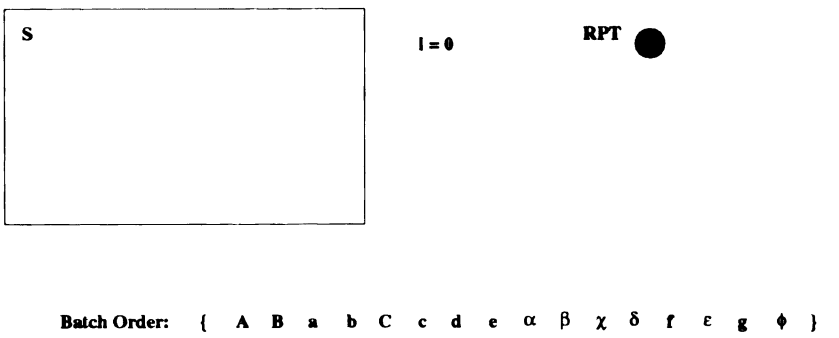


Figure 3.19: The root node of the RPT represents the entire space S^2. The training samples are presented in batch in the order shown.

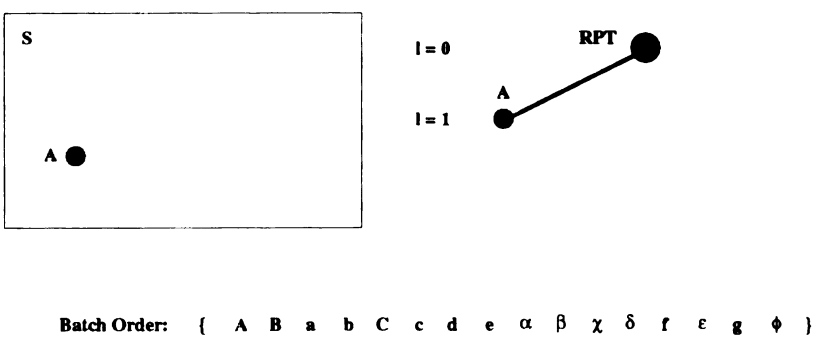


Figure 3.20: A is added at level $l = 1$.

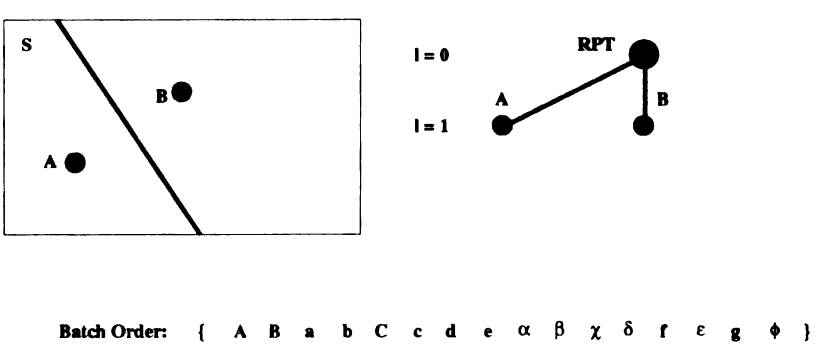


Figure 3.21: B is added at level $l = 1$.

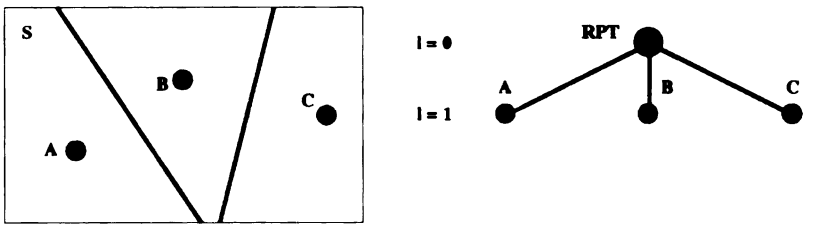


Figure 3.22: a and b are not added at level $l = 1$. C is added at level $l = 1$.

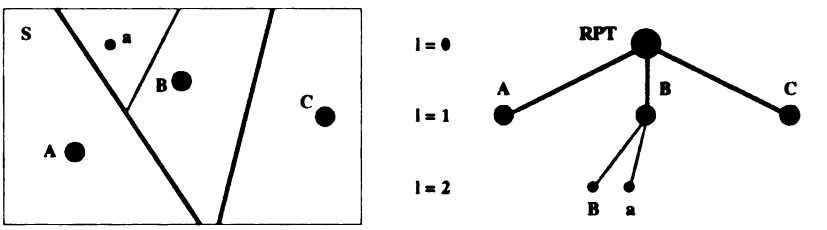


Figure 3.23: No other nodes are added at level $l = 1$. a is added at level $l = 2$.

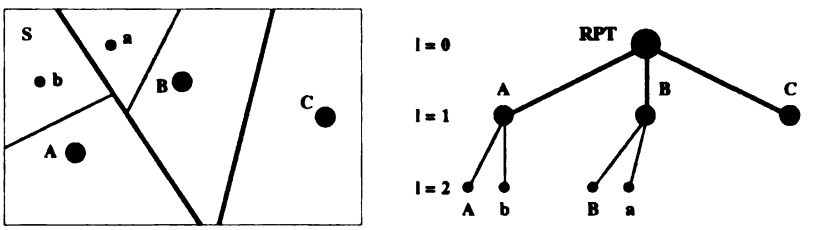


Figure 3.24: b is added at level $l = 2$.

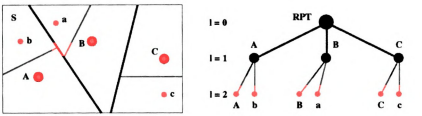


Figure 3.25: c is added at level $l = 2$.

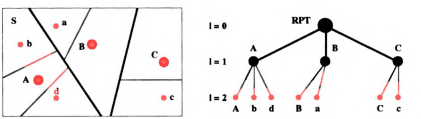


Figure 3.26: d is added at level $l = 2$.

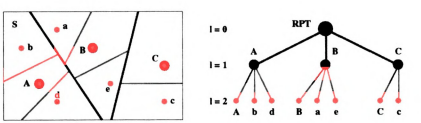


Figure 3.27: e is added at level $l = 2$.

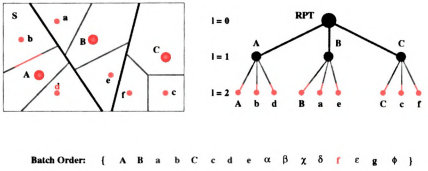


Figure 3.28: α, β, χ, and δ are not added at level $l = 2$. f is added at level $l = 2$.

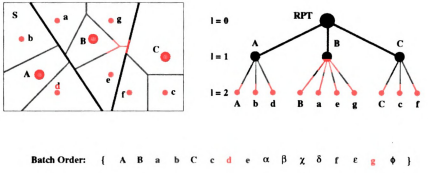


Figure 3.29: ϵ is not added at level $l = 2$. g is added at level $l = 2$.

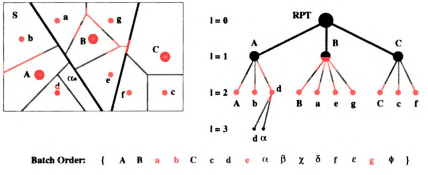


Figure 3.30: No other nodes are added at level $l = 2$. α is added at level $l = 3$.

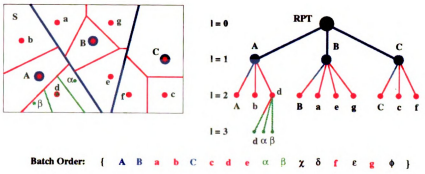


Figure 3.31: β is added at level $l = 3$.

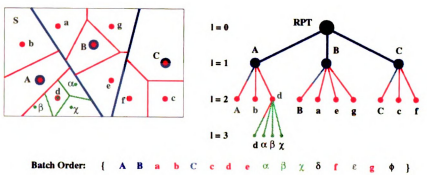


Figure 3.32: χ is added at level $l = 3$.

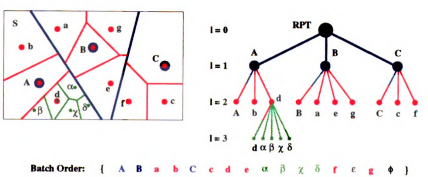


Figure 3.33: δ is added at level $l = 3$.

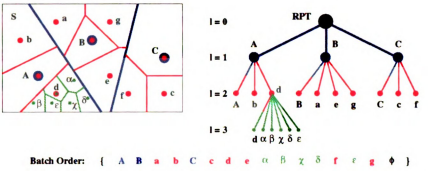


Figure 3.34: Finally, the last node, ϵ, is added at level $l = 3$.

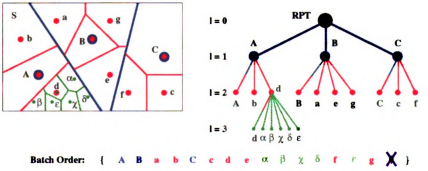


Figure 3.35: ϕ is never added because it is closer to d than the deepest level's radius.

3.2.4 Discussion

Given a query vector Q, let $e \in \mathcal{X}$ be the vector in the training set which is Q's nearest neighbor, and let $N(e)$ be the leaf node associated with e. There is no guarantee that Q will reach $N(e)$ during the query phase unless $Q \in \mathcal{X}$. A two-dimensional example is given in Figure 3.36. As shown in the figure, during training e was closer to B than it was to C and so became a descendant of B in the RPT. On the other hand, Q is closer to C than it is to B, so during the query phase Q traverses the subtree with C as its root. Thus Q will end up at $N(f)$ instead of at $N(e)$. In order

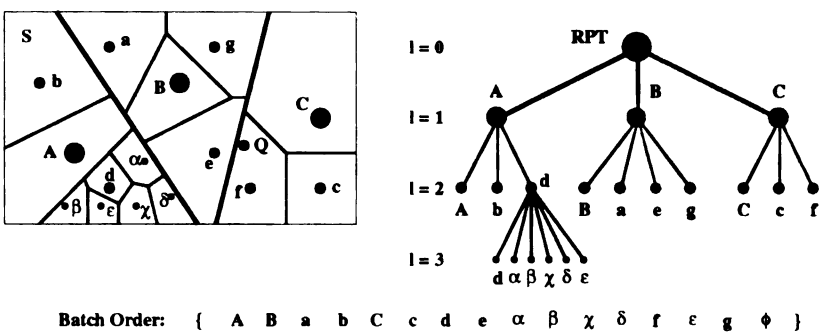


Figure 3.36: Q is not guaranteed to reach it's nearest neighbor, e, because at a higher level, Q is closer to C than it is to B.

to guarantee that each and every query vector reaches the leaf node representing its nearest neighbor, the search algorithm would have to search the entire set of training samples. If there are n training samples and m query vectors, this is an $\mathcal{O}(nm)$ search. The idea of the RPT is to decompose the search space in order to reduce the computational effort spent on searching for a nearest neighbor. The belief is that in using the RPT the system will be able to locate a training sample near enough to the query vector to produce a good estimate of the output corresponding to the query vector. Thus, it would seem that the denser the training data the larger the RPT and the closer a near neighbor could be found for the query vector, and the better

the estimate of the output vector. This is based on two assumptions. First, that the training samples provide a dense covering in often visited areas of the input space. Sparsely covered areas will result in large cells representing those areas at shallow levels of the RPT. Second, that as the number of nodes in the RPT increases, the accuracy of the *input-to-output* transformation should increase. Given that the N-dimensional input space can be divided by a piecewise linear boundary, we should be able to approximate the decision boundaries to any accuracy simply by adding more samples to the training set. This is expected because the estimation of the mapping is based on a smaller sub-space of the input space as the search proceeds down the RPT hierarchy and should represent a more accurate *picture* of that space. However, there is no guarantee that the error will *always* decrease as the size of the RPT increases, even when that increase is gained by adding additional training samples to the RPT (which gives a denser covering of the input space). An empirical study of two RPTs, one RPT created by adding a single new training sample to the training set used to construct the first RPT, tested with the same set of test data shows that the error *can* increase in some situations (see Appendix C for details of the empirical study).

3.3 Function Approximation

This section briefly describes the function approximation methods used to compute the output vector for a query vector given the $k \geq 1$ nearest neighbors found during the retrieval phase. Most of the experimental data described in Section 4.2 with the real setup were obtained using the *k-nearest-neighbor distance-based* (KNDB) function approximation as proposed in [90] and used in [45]. Let $\mathcal{X} = \{X_i \mid i = 1, \ldots, n\}$ be the training set of vectors sampled from the N-dimensional input space S^N. Let $\mathcal{Y} = \{Y_i \mid i = 1, \ldots, n\}$ be the vectors in the M-dimensional output space O^M such that

$f(X_i) = Y_i$. Here $f(\cdot)$ is the function mapping $S^N \longrightarrow O^M$. Given a query vector Q, a set of k nearest neighbors are retrieved from the RPT, say $\mathcal{Q} = \{Q_j \mid j = 1, \ldots, k\}$ with corresponding output vectors $\mathcal{O} = \{O_j \mid j = 1, \ldots, k\}$ where $Q_j \in \mathcal{X}$ and $O_j \in \mathcal{Y}$ for all j. The output vector for Q is then estimated by the KNDB approximation as

$$\hat{f}_k(Q) = \sum_{j=1}^{k} \frac{w_i}{w} O_j$$

where O_j is the output vector corresponding to Q_j, the jth nearest neighbor to Q, such that $Q_j \in \mathcal{Q}$. Here w_i is a scalar weighting function defined on the input vector Q such as

$$w_i = \frac{1}{\alpha^{|Q-Q_j|/(\epsilon+|Q-Q_1|)}}$$

and $w = \sum_{j=1}^{k} w_i$. A small ϵ is added to the denominator of the fraction to prevent division by zero. The parameter α is used to control how quickly the weight approaches zero.

It is important to note that the k nearest neighbors are retrieved from the RPT based solely on a global perspective of the input vectors. That is, the corresponding output vectors are not considered when deciding which of the training samples are the k nearest neighbors to the query vector. At the local level, given two vectors both within a small sub-region of the input space, one of the vectors may have a much stronger influence on the computation of the output vector than does the second vector. In order to compensate for this, a second approximation method, the *second-order KNDB* (KNDB2), was developed. In this method, the output vector for Q is computed as follows:

$$\hat{f}_k(Q) = \sum_{j=1}^{k} \frac{w_i}{w} \left(O_j + (Q - Q_j) * (\nabla f(Q_j))\right)$$

where Q_j and O_j satisfy the same conditions as for KNDB, $\nabla f(Q_j)$ is the gradient of Q_j, and the symbol $*$ indicates that the inner product of the two vectors should be computed. The two approximation functions are discussed in more detail in [45] and [44].

3.4 Some Properties of Framework's Performance

The Recursive Partition Tree method described in Section 3.2 exhibits several desirable properties. These properties deal with the convergence of the function computed by the RPT to the mapping function which is being approximated; the rate at which the function converges; placement of nodes within the tree during the training phase; and the time complexity for retrieving a near neighbor given a query vector. These properties are briefly reviewed in the following sections.

Convergence

Given an N-dimensional input space R^N and an M-dimensional output space R^M, let $f(\cdot)$ be the function which maps R^N to R^M. That is, for any $X \in R^N$,

$$f(X) = Y$$

where $Y \in R^M$ and Y is the output vector corresponding to X. Then the RPT combined with a method for computing an output vector for a given query vector based on the K nearest neighbors within the given training set is used to approximate the mapping function f by a function $\hat{f}$. Let $L = \{L_1, \ldots, L_n\}$ be a set of learning samples, where L_i is a random learning sample in R^N, and $\hat{f}$ is a function of L. Suppose $\hat{f}(X)$ is the value of $f(L_i)$ where L_i is the nearest neighbor of X in L.

Then Li and Weng [59] have proven that for any point $X \in R^N$, if f is differentiable and its Jacobian matrix is bounded, and the L_i are identically independently drawn according to a given distribution, then

$$\hat{f}(X) \longrightarrow f(X) \text{ with probability one}$$

which means that

$$\lim_{|L| \longrightarrow \infty} P\left\{X \mid \hat{f}(X) \neq f(X)\right\} = 0$$

where $\mid L \mid$ is the size of set L.

Convergence Rate

Currently, we are using the *k-nearest-neighbor distance-based* (KNDB) method for approximating the mapping function. This method is described in Section 3.3. Li and Weng [59] have shown that the KNDB approximation has a rate of convergence of $O\left((n)^{-\frac{2}{N}}\right)$, where $n \longrightarrow \infty$ is the number of samples in the training set and N is the dimension of the input space R^N.

Complexity

For a given application, the dimensionality, N, of the input space is fixed. Additionally, the expected radius, r_l, of a polyhedron at each level of the RPT is fixed. By letting V_i denote the maximum diameter of any polyhedron on level $l-1$, Swets and Weng [83] showed that the number of children any level $l-1$ node can have is bounded above by a constant κ where:

$$\kappa = \left(\frac{2 \max\{V_i\}}{r_l}\right)^N.$$

Furthermore, it can be shown that for any given set of n training samples the number of levels in the resulting RPT is $\mathcal{O}(\log n)$ [82].

Thus, during the retrieval phase, at each level l of an RPT we are exploring the children of at most k (k nearest neighbors) nodes. The maximum number of children that each of these k nodes can have is κ. Thus at each level l the query vector is compared to at most $k\kappa$ vectors. Since there are $\mathcal{O}(\log n)$ levels in an RPT constructed from n training samples, the number of comparisons for any query vector is $\mathcal{O}(k\kappa \log n)$. Thus the retrieval time for the RPT structure is $\mathcal{O}(\log n)$ (see [82] for a detailed proof).

Chapter 4

Experimental Results

The following sections describe the experimental results obtained using the method described in the preceding pages. This chapter contains the following sections:

1. A discussion on the modules implemented for the simulation experiments.

2. A report of the error analysis of the CSS module in simulation.

3. A report of the error analysis of the CAS module in simulation.

4. A report of the error analysis of a system consisting of a CSS and the CAS module trained using the CSS in simulation.

5. A report of the experimental data obtained from implementing the stereo camera calibration module using actual data.

6. A report of the experimental data obtained from implementing the sensor control module using actual data.

7. A report of the experimental data obtained from implementing the grasping task using actual data.

8. A report of the experimental data obtained from implementing a temporal sequence of five subtasks using actual data.

9. A report of the experimental data obtained from implementing a simple vision module and using it to guide the grasping task.

10. A discussion of the physical setup of hardware and software used in the real experimental setup.

11. A report on the implementation of an RPC message passing package for communicating between the three different types of hardware and software used in the real experimental setup.

Additionally, a discussion on an earlier investigation into the performance of a feed forward network with a back propagation learning algorithm can be found in the Appendix.

4.1 Performance Study of the CSS and CAS Modules in Simulation

To date, the CSS and CAS modules have been implemented. This section reports on the results of simulation and error analysis of the two implemented modules, the CSS and the CAS. When *noise-free* input data is used, both the training and testing input data are noise-free. Similarly, when noisy data is specified, both the training and testing input data sets have noise added, unless otherwise stated. The following sections discuss the experimental setup for simulation, the training and testing of the CSS and CAS modules and the setup for the real setup experiments. The following five sets of experiments are described in further detail, and the results reported, in

the following sections:

1. Training CSS with noise-free input data.

2. Training CSS with noisy input data.

3. Training CAS independently with noise-free input data.

4. Training CAS with noisy input data obtained from a CSS trained with noise-free input data.

5. Training CAS with noisy input data obtained from a CSS trained with noisy input data.

4.1.1 Experimental Setup for Simulation

In order to perform the simulations the camera system was modeled as shown in Figure 4.1. The camera system consists of a pair of cameras separated by a baseline of $b \approx 170mm$. The angle θ is chosen so that optical axes of the two cameras intersect at the point of origin of the world coordinate frame. The z-axis of the camera system frame is coincident with the z-axis of the world coordinate frame but in the opposite direction. When the pan and tilt parameters are both zero[1], the xy-planes of the world and camera system coordinate frames are parallel. The origins of the two frames are separated by a distance of $h \approx 744mm$ along the z-axes. b and h are chosen so that the entire workspace on the $(xy)^W$-plane is visible to each camera. The focal length of the cameras is set to $f_u = -1923$ *pixels* and $f_v = 1587$ *pixels*[88]. In simulation, the epipolar line constraint is assumed. Also, the augmented input vector given as input to the CSS module is equal to the original input vector, that is, $I' = I$.

[1] $(\alpha, \beta) = (0, 0)$

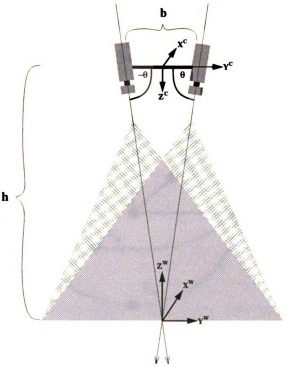


Figure 4.1: Schematic of simulated camera system.

4.1.2 Estimation of Transformation Matrix for Simulation

For the simulation data shown here, the mapping from input to output space is estimated by estimating a $m \times n$ transformation matrix where m is the dimension of the output vector and n is the dimension of the input vector. This matrix is a linear estimation of the function which maps the input space to the output space. Once an RPT has been constructed, it is traversed in LNR [2] order and at each leaf node the transformation matrix is estimated using the *least squares estimation* method. In order to estimate this matrix we obtain K *derived points* around the sample point. K is equal to twice the dimension of the input vectors for that RPT. Figure 4.2 shows an example of the relationship between the derived and sample points. The

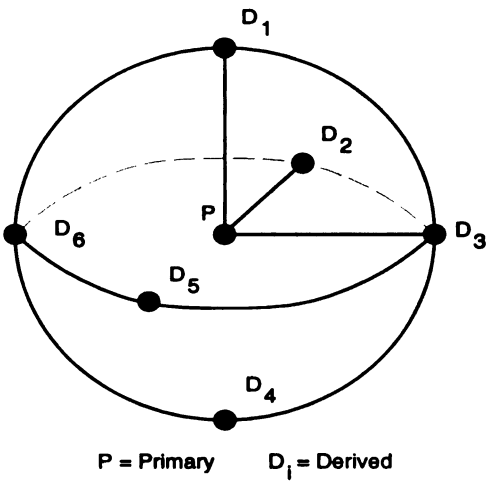


Figure 4.2: Relationship between sample and derived points.

transformation matrix, $\mathcal{T}$, takes the form shown in Equations 4.1 and 4.2.

$$U^D - U^P = \mathcal{T}\left(V^D - V^P\right) \tag{4.1}$$

[2] *LNR* indicates the search algorithm which specifies that at each node the node's *L*eft child is examined first, then the *N*ode itself is examined, and finally the node's *R*ight child is examined. Although each node can have several children, the tree structure is implemented as a binary tree where each node's left branch points to a list of its descendant nodes, and the right branch points to a list of its sibling nodes.

$$\begin{bmatrix} u_1^{D_i} \\ u_2^{D_i} \\ \vdots \\ u_m^{D_i} \end{bmatrix} - \begin{bmatrix} u_1^P \\ u_2^P \\ \vdots \\ u_m^P \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{22} & \cdots & t_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ t_{m1} & t_{m2} & \cdots & t_{mn} \end{bmatrix} \left(\begin{bmatrix} v_1^{D_i} \\ v_2^{D_i} \\ \vdots \\ v_n^{D_i} \end{bmatrix} - \begin{bmatrix} v_1^P \\ v_2^P \\ \vdots \\ v_n^P \end{bmatrix} \right) \tag{4.2}$$

Let $u^{iP} = u^{D_i} - u^P$ and $v^{iP} = v^{D_i} - v^P$. Then the above equation reduces to $u^{iP} = \mathcal{T} v^{iP}$, which in its expanded form is:

$$u_j^{iP} = t_{j1} v_1^{iP} + t_{j2} v_2^{iP} + \cdots + t_{jn} v_n^{iP} \tag{4.3}$$

For $i = 1, \ldots, K$ we use least squares estimation to calculate each row of $\mathcal{T}$'s components, using the following system of K equations:

$$\begin{bmatrix} u_j^{1P} \\ u_j^{2P} \\ \vdots \\ u_j^{KP} \end{bmatrix} = \begin{bmatrix} t_{j1} t_{j2} \ldots t_{jn} \end{bmatrix} \begin{bmatrix} v_1^{1P} & v_1^{2P} & \cdots & v_1^{KP} \\ v_2^{1P} & v_2^{2P} & \cdots & v_2^{KP} \\ \vdots & \vdots & \vdots & \vdots \\ v_n^{1P} & v_n^{2P} & \cdots & v_n^{KP} \end{bmatrix} \tag{4.4}$$

4.1.3 Experiments with the CSS Module in Simulation

CSS Training

The input to the CSS module is the position of a feature point in the left camera image and the disparity between the position in the left and right camera images (r, c, d). The output is the feature position in the camera system space $\left(x^C, y^C, z^C\right)$. In simulation, the CSS module is trained as follows. First, a random point $P^W = \left(x^W, y^W, z^W\right)^P$ is generated from a 3-dimensional Gaussian space centered around the origin of the workspace. This point is evaluated to insure that it is within the visible

field of *both* cameras in the system. Using the modeled camera system, $(r, c, d)^P$ and $P^C = \left(x^C, y^C, z^C\right)^P$ are computed. Then, six derived points, $D^i = \left(x^W, y^W, z^W\right)^i$ are generated around the *input* point, as shown in Figure 4.3, at distance R_l which is approximately equal to the maximum radius of the neighborhood around the input point at level l of the hierarchy. The generated points lie on the X, Y and Z axes of the coordinate system whose origin is the input point. The image positions, disparities and camera system coordinates are likewise computed for each of these six points. The

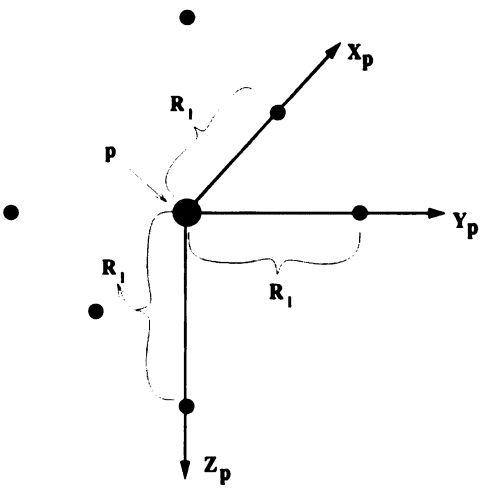


Figure 4.3: Six additional points generated for training CSS module. Center point is the "input" point. The other six are generated so that they lie in the positive/negative X, Y and Z directions at a distance of R, where R is approximately equal to the radius of the neighborhood at the given level of the hierarchy.

seven points are then used to compute a linear estimation of the transformation from the input space (image features) to the output space (camera system coordinates) as described in Section 4.1.2 where $V^I = (r, c, d)$ and $U^I = P^C$. Note that this system is over-determined in order that the entire neighborhood of the *input* point is well represented at the current level of the hierarchy (which determines the size, or extent, of the neighborhood).

CSS Testing

The CSS module is tested as follows. A seed is chosen for the random number generator and five CSS networks are constructed as follows. The first network, N_1, is constructed using $t_1 = 50$ training samples randomly generated in a 3D Gaussian space centered at the origin of the manipulator's workspace. Then for $i = 2..5$, the remaining four networks, N_i, are generated using $t_i = 2t_{i-1}$ training samples by adding t_{i-1} more training samples to the previously generated set of t_{i-1} training samples and then constructing N_i. Only training samples that are visible to the camera are used in the training sets. Furthermore, although the network will have more nodes than the number of training samples, not all of the training samples will be used to construct the network. Whether or not a training sample is placed in the network depends on the criteria as explained in Section 3.2.3.

Two such *sequences* of CSS networks are built. The difference between the two sequences is that in the first sequence the training samples are *perfect*; that is, both the input and output vectors are calculated to double precision accuracy and the row, column, and disparity values are not integer values. The second sequence we refer to as *noisy* because the components of the training samples' input vector are rounded off to the nearest integer. This sequence is closer to what the network will be trained with once we start using real training data received from images of actual scenes.

Each network, in both sequences, is then tested on three different test sets. In all cases, the first test set consists of all the primary points of that network. The second test set consists of the derived points used in the estimation of the transformation at the leaves of the network. Finally, the third set consists of 100 randomly generated samples. For each experimental set (network + test set), the *Normalized Error* (NE) of the difference between the *actual output* and the *expected output* is calculated and

plotted versus the number of nodes in the network.

Figure 4.4(a) shows that with as few as 200 training data, the average NE is below $1.0mm$ in all cases. Figure 4.4(b) and Figure 4.4(c) show a breakdown of the NE into the combined x, y error and z error, respectively. A comparison of these two graphs shows that most of the error occurs in the z direction. This is expected since the uncertainty is greatest in that direction. The NE is computed as follows:

$$NE_{xyz} = \frac{W_i \sum_{100}^{1} \left[(x_e - x_a)^2 + (y_e - y_a)^2 + (z_e - z_a)^2\right]}{T_W} mm \tag{4.5}$$

$$NE_{xy} = \frac{W_i \sum_{100}^{1} \left[(x_e - x_a)^2 + (y_e - y_a)^2\right]}{T_W} mm \tag{4.6}$$

$$NE_{z} = \frac{W_i \sum_{100}^{1} \left[(z_e - z_a)^2\right]}{T_W} mm \tag{4.7}$$

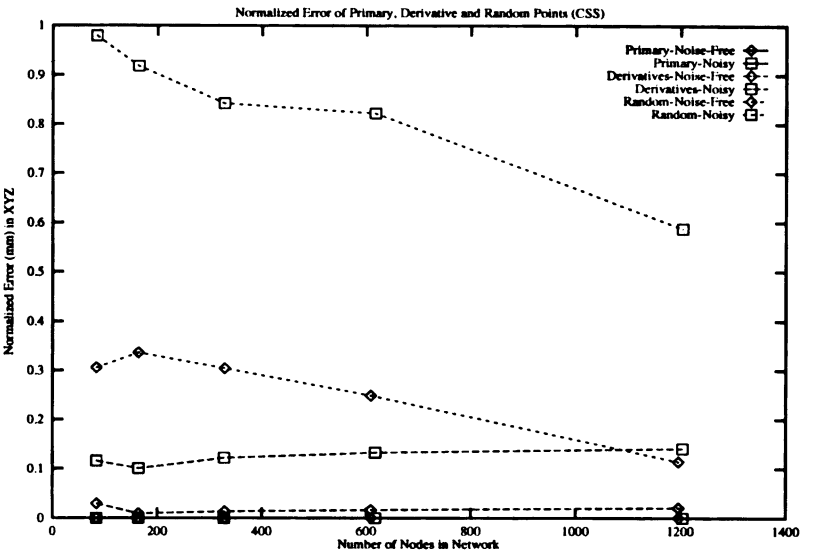
where each W_i is computed as:

$$W_i = \frac{1}{2\pi} e^{\frac{-1}{2}\left[\left(\frac{x_e - x_a}{\sigma_x}\right)^2 + \left(\frac{y_e - y_a}{\sigma_y}\right)^2 + \left(\frac{z_e - z_a}{\sigma_z}\right)^2\right]} \tag{4.8}$$

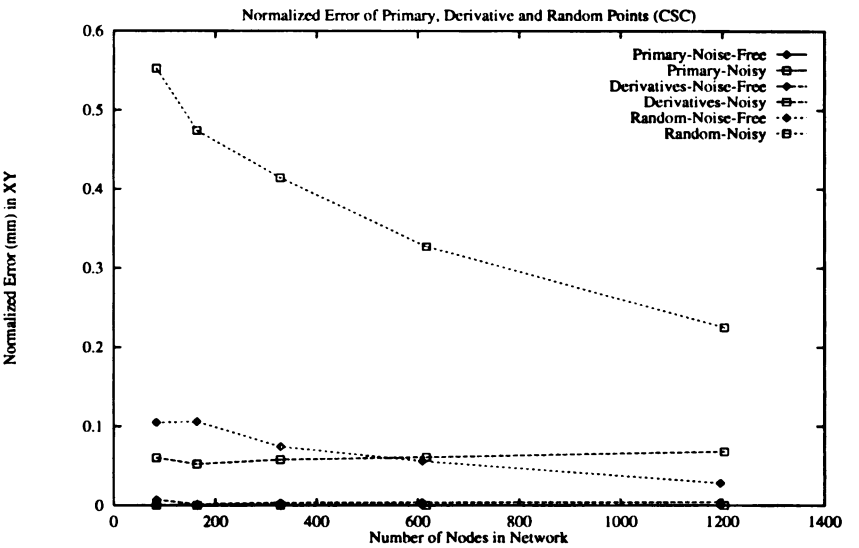
and T_W is:

$$T_W = \sum_{100}^{1} W_i \tag{4.9}$$

In Figures 4.5 and 4.6 are plots of the average time it takes a single input to pass through a CSS network and obtain an output vector. The time is given in microseconds and plotted versus the number of levels in the RPT (in Figure 4.5) and versus number of training samples (in Figure 4.6). As shown in the graph, on average it takes less than $400\mu secs$ to traverse an eight-level RPT with 800 training samples and find an input's nearest neighbor. Figure 4.7 shows the average number of nodes visited in the input's descent through the network. This number increases significantly

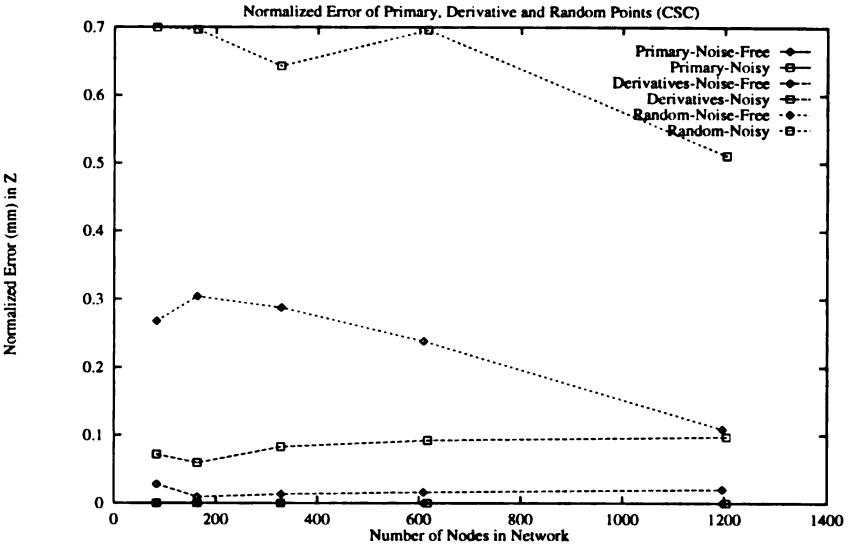


(a)



(b)

Figure 4.4: The Normalized Error (NE) from experiments with the CSS module. Units are in millimeters (mm). (a) X, Y, Z combined error; (b) X, Y combined error; (c) Z error.



(c)

Figure 4.4 continued...

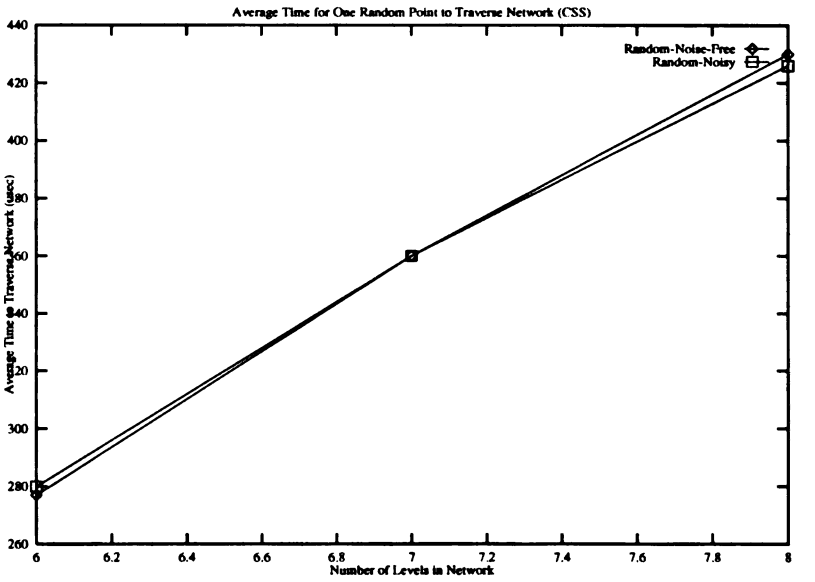


Figure 4.5: Average time in microseconds versus levels in the RPT for a random sample input to pass through the CSS network and find the nearest neighbor in the search space.

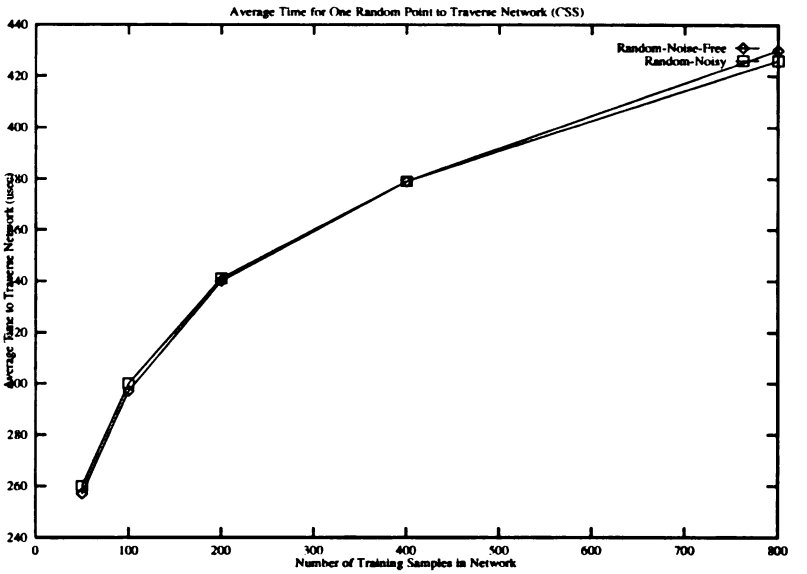


Figure 4.6: Average time in microseconds versus number of training samples for a random sample input to pass through the CSS network and find the nearest neighbor in the search space.

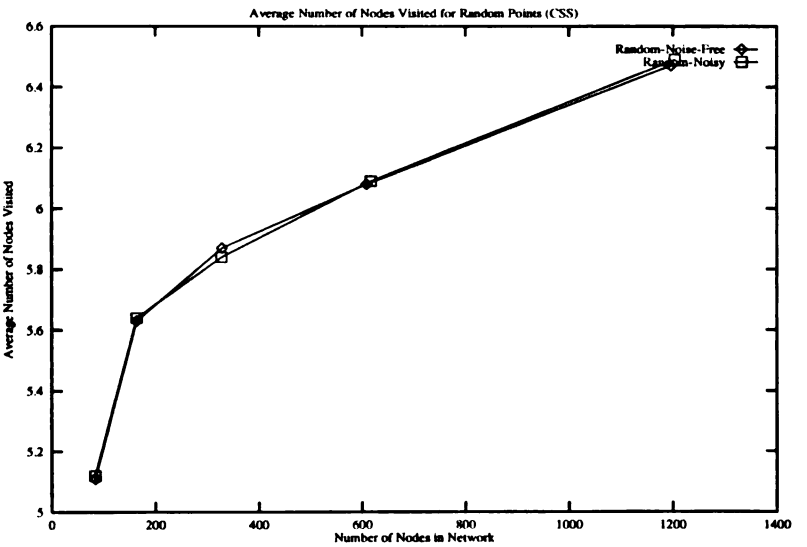


Figure 4.7: Average number of nodes visited by a random sample input as it descends CSS network.

from 100 to 400 nodes, more slowly from 400 to 600 nodes, and then there is only 0.5 more nodes traversed from 600 to over 1200 nodes. Figure 4.8 compares the number

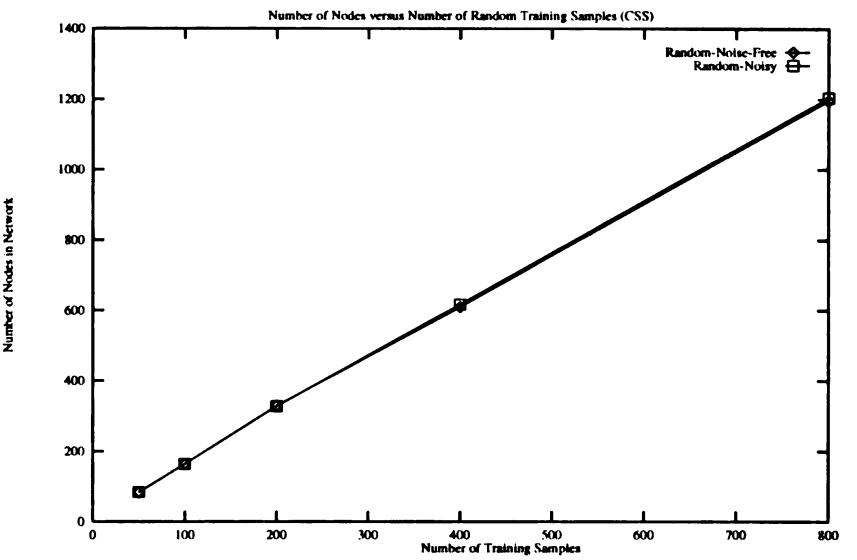


Figure 4.8: Number of nodes in CSS network versus the number of training samples used to construct it.

of nodes in a given CSS network versus the number of sample training input/output pairs used to construct it. The ratio of the number of training samples to the number of nodes in the network is approximately 1:2 under 300 training samples, and 2:3 between 300 and 800 training samples.

Together, these graphs indicate that the network tends to grow in breadth much faster than it grows in depth. However, the traversal time also indicates that the search through the input space is very efficient. This in turn indicates that the network is able to efficiently divide the search space in such a way that the search for a given input's nearest neighbor occurs within a reasonable amount of time.

4.1.4 Experiments with the CAS Module in Simulation

CAS Training

A CAS network is trained as follows. The input to the CAS module is the vector $X = \left(S^C P^C\right)$ where $S^C = (\alpha, \beta)$ is the vector corresponding to the pan and tilt signals, respectively, received from the camera system and $P^C = \left(x^C, y^C, z^C\right)^P$ is the vector of camera coordinates output by the CSS. The output is a vector $P^W = \left(x^W, y^W, z^W\right)$ which is point P's coordinates in the world coordinate frame. First, N training samples consisting of the $\left(x^C, y^C, z^C\right)$ components of the input vector are generated such that all points are visible to the camera system when it is positioned at $(\alpha, \beta) = (0, 0)$. Then, M (α, β) pairs are generated. Then for each (α, β) pair the camera system is rotated to that position and the N $\left(x^C, y^C, z^C\right)$ points are rotated with the camera system. This can be viewed as if the $\left(x^C, y^C, z^C\right)$ points are all rigidly attached to the camera system. Thus, for all (α, β) pairs, $P_i^C = P^C, \forall i$, but the P^W is calculated based on the camera system position determined by (α, β). That is, for each pair all input vectors are identical and fixed *in the camera system space* (see Figure 4.9 for an example). For this module, 10 derived points are generated around the primary point, in the same manner as the derived points are generated for the CSS module. These 11 points are then used to compute a linear estimation of the transformation from the CAS input space (camera system coordinates + camera system parameters) to the CAS output space (world system coordinates). The method described in Section 4.1.2 is likewise used.

CAS Testing

The CAS module is tested as follows. A network, N_1 is constructed with $t_1 = 16$ (α, β) pairs, and $N = 200$ $\left(P^C, P^W\right)$ input/output vectors. Then for $i = 2..5$, four

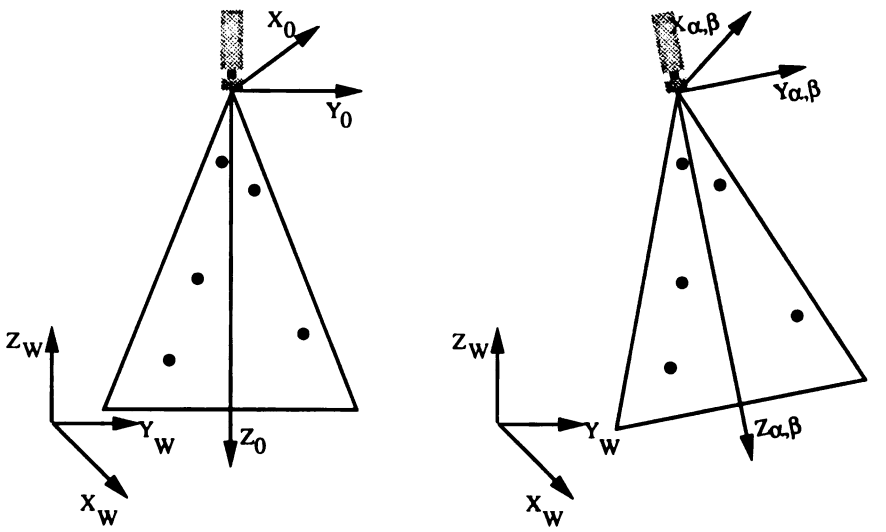
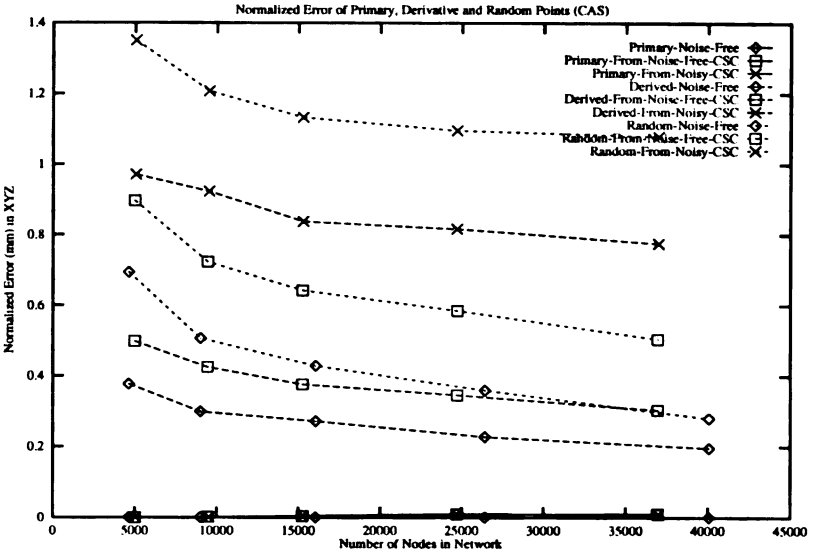


Figure 4.9: Example of moving test points with camera system.

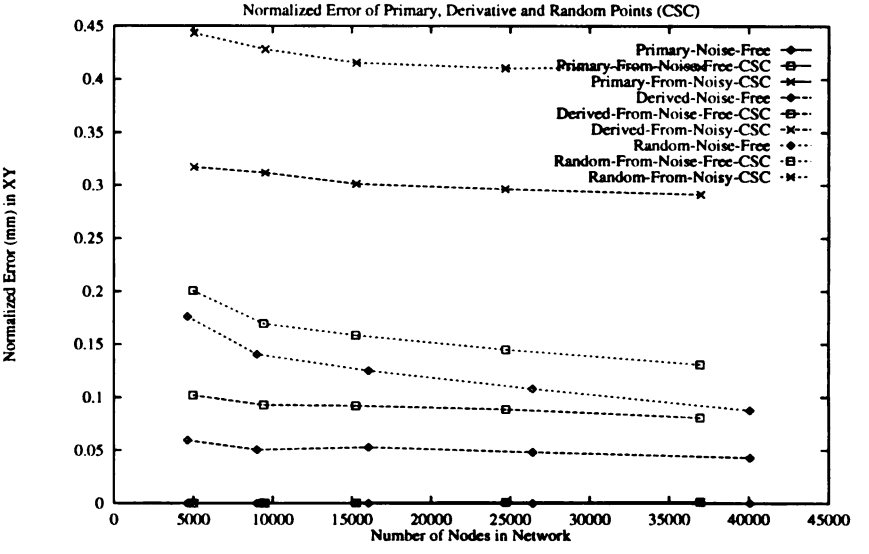
additional networks, N_i, are generated using $t_i = 2t_{i-1}$ (α, β) pairs by adding t_{i-1} more pairs to the previously generated set of t_{i-1} pairs used to construct the N_{i-1} network. The same set of $N = 200$ $\left(P^C, P^W\right)$ input/output vectors are used.

Three such *sequences* of CAS networks are built. The differences between the three sequences is that in the first sequence the training samples are all *perfect* as described in the CSS training section. The input (x, y, z) components of the second sequence are obtained as the actual output of a CSS network which itself is trained with perfect training samples. The networks of the third sequence are constructed in the same way as those of the second sequence except that the CSS network is trained with noisy training samples as described above.

As in the CSS experiments, each CAS network in each sequence is tested on three different test sets: the primary points, the derived points, and a set of 1000 random test samples. Figure 4.10 shows the plots of the NE of the difference between the *actual output* and the *expected output* for each CAS network versus the number of nodes in the network.

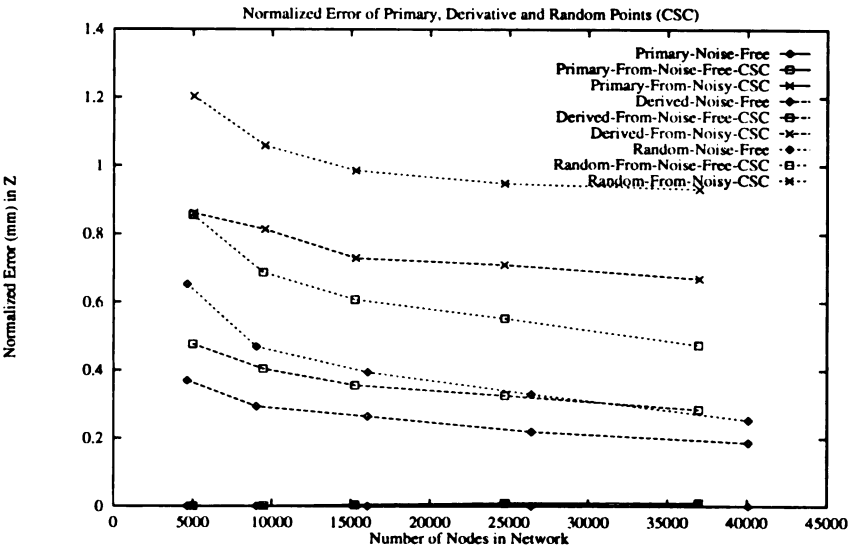


(a)



(b)

Figure 4.10: The Normalized Error (NE) from experiments with the CAS module. Units are in millimeters (mm). (a) X, Y, Z combined error; (b) X, Y combined error; (c) Z error.



(c)

Figure 4.10 continued...

The error is reported as a function of (x, y, z), (x, y) and (z) position in Figure 4.10(a), Figure 4.10(b), and Figure 4.10(a), respectively. The error calculation is the same as given in Equations 4.5 thru 4.9 for the CSS network.

The graph in Figure 4.11 is similar to that shown in Figure 4.8 for the CSS module. The only difference is that here we only plot the number of (α, β) training pairs used to construct the network. For each (α, β) pair, we used the same set of 200 (x, y, z) vectors for training. We observe that the CAS network is much bigger than the CSS network—at 3200^3 training samples, the network has approximately 5000 nodes. This size is necessary because the entire input space must be well represented, both the (α, β) subspace and the (x, y, z) subspace.

[3] 16 (α, β) pairs and 200 (x, y, z), for $16 \times 200 = 3200$ total training samples.

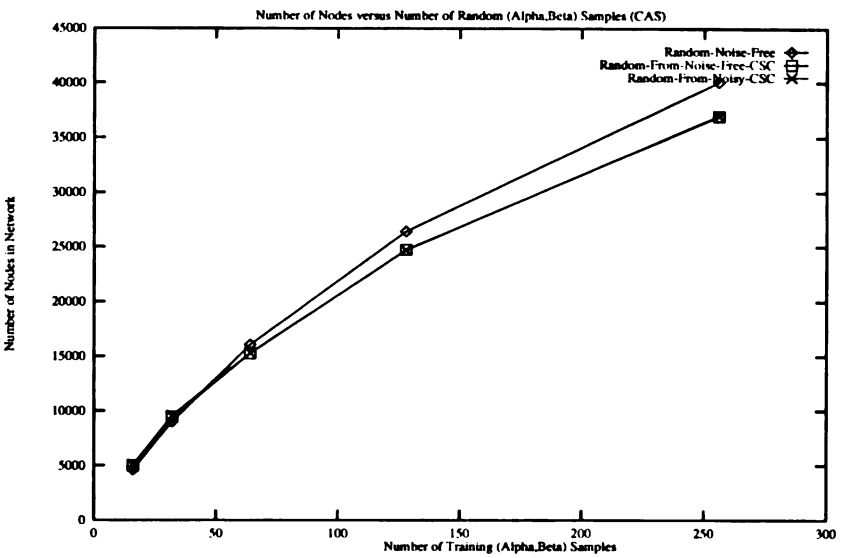


Figure 4.11: Number of nodes in CAS network versus the number of training samples used to construct it.

4.2 Experiments with a Real Setup

The following experiments demonstrate the ability of a system using RPTs to handle various mappings within the hand-eye coordination problem. Lack of a real-time recognition and image processing system required that data be collected manually. For the five-module system proposed in Figure 3.1 this ended up requiring an exorbitant amount of manual labor, which was not conducive to the allotted time frame. Therefore, the three-module system shown in Figure 4.12 is the one which was actually implemented for the real-time experiments presented in the following sections. The difference is only that the *Camera-Centered Stereo System*, *Camera-to-Arm System* and *Arm-to-Joint System* modules shown in Figure 3.1 has been replaced by a single module in Figure 4.12, the *Image-to-Joint System.*

In order to complete the large amount of work necessary for this thesis and [44]

and [45] in a timely manner, the load of all work common to both was divided[4] up so that no time was wasted in repeating implementation of common algorithms or performing common experiments. This required much consultation and discussion, but furthered the work of both researchers toward their respective goals. In all real experiments, for a given query vector Q, at each level of the RPT the subtrees of the K nearest neighbors are searched at the next level until at most K leaf nodes are reached. The $KNDB$ or $KNDB_2$ [46] algorithm is used to compute an output vector corresponding to Q. Figure 4.12 shows the implementation of the system used in these experiments.

4.2.1 Experimental Setup

In order to test our hypothesized system beyond the stage of simulation, the real setup shown in Figure 4.13 is used. The real setup consists of a PUMA 560 robotic manipulator connected to a Unimate Computer/Controller which houses a PDP 11/73. This controller has two connections to a Sun 4/330 with $40MB$ of ram and $140MB$ of swap space, running SunOS 4.1.2. One is a serial connection on an RS232C cable. This connection is used in place of a monitor to load and initiate the RCCL/RCI **Moper** control program on the controller. RCCL/RCI [62] [61] is a library of C routines which provide an easy way for user programs to send control signals to the PUMA via the controller. These signals are sent via a parallel connection which is interfaced with the Sun 4/330 via a Xycom XVME-240 board plugged into the Sun's

[4] Wey-Shiuan Hwang was responsible for implementing a GUI interface for the system; the $KNDB$ and $KNDB_2$ algorithms; and the corner detection algorithm. I was responsible for implementation of the RPT module for the simple vision system; the communication between the various types of hardware and software used in the real experimental setup; and all software used in constructing and searching the general RPT structure. All real experiments were performed cooperatively except that Hwang performed the Sensor Movement Control and temporal task sequence experiments independently. All simulation results reported in this thesis were performed exclusively by the author.

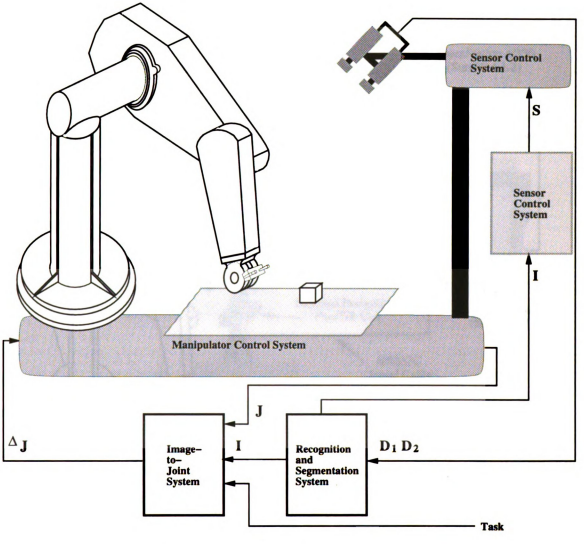


Figure 4.12: Schematic of implemented system.

VME bus, and interfaced with the Unimate via a DEC DRV11 board. Stereo images are obtained from two Panasonic Industrial Color CCD Cameras, model number GP-KR202. For these experiments $6mm$ lens are used. There are two SunVideo boards housed in a SparcStation 2 running Solaris 2.4 with $32MB$ ram and $64MB$ swap

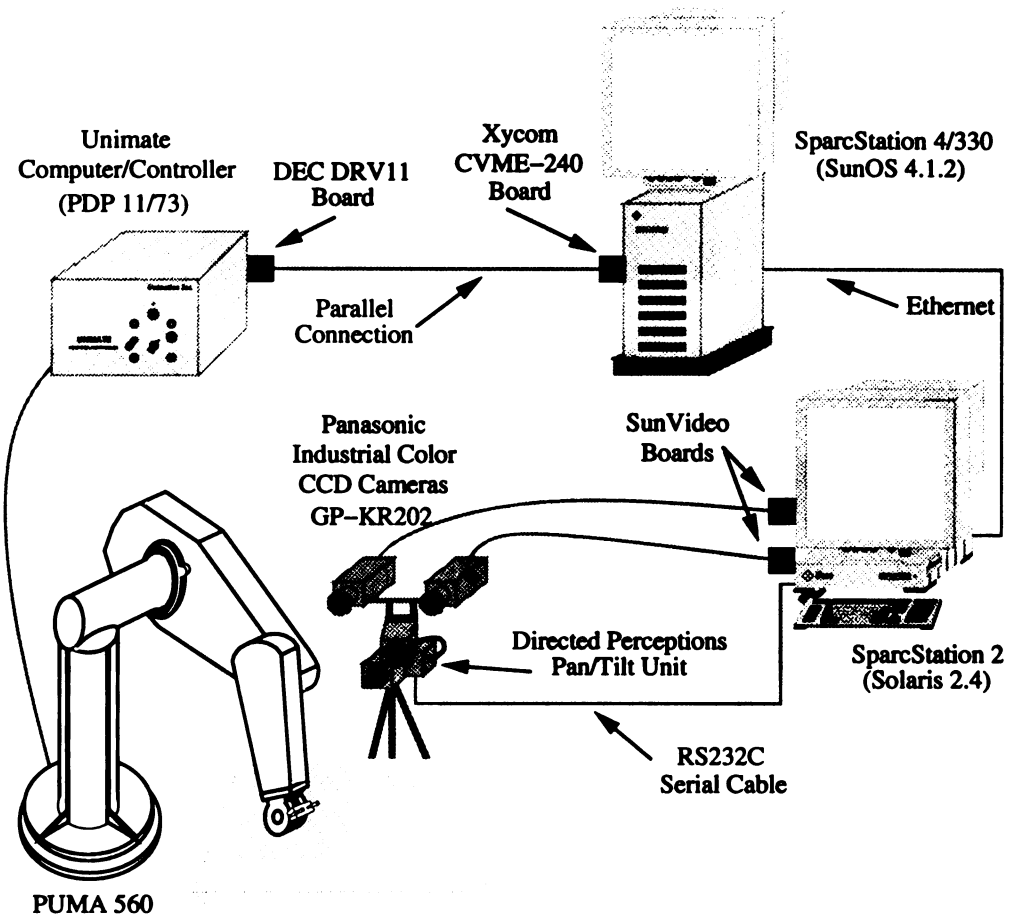
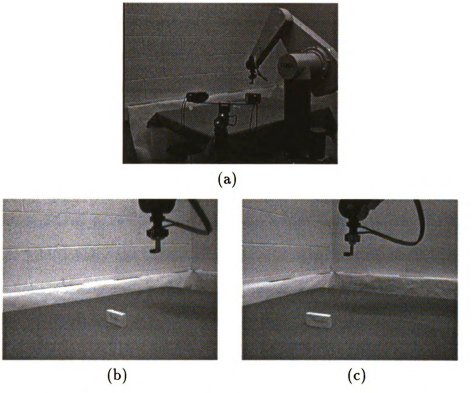


Figure 4.13: Setup for real experiments.

space. Each camera is connected to one of the SunVideo boards. The binocular camera system sets atop a Pan/Tilt Unit (PTU) from Directed Perceptions. The cameras are *connected* to each other and the PTU via an aluminum bar. The baseline distance between the cameras is approximately $12\frac{1}{2}$ inches. The PTU is connected to the Sun

4/330 via a second serial RS232C cable and is controlled via a simple set of signals. The Sun 4/330 and SparcStation 2 communicate via Ethernet. Figure 4.14 shows a snapshot of the real setup.



(a)

(b) (c)

Figure 4.14: A snapshot of the experimental setup showing the PUMA 560 robotic manipulator and the camera system consisting of two Panasonic cameras mounted on a Direct Perceptions Pan/Tilt Unit. (a) Observer's view. (b) View of left camera. (c) View of right camera.

4.2.2 Software Communication

Figure 4.13 shows the actual setup used to perform experiments with real data. As the figure shows, it is necessary to establish communication between three different pieces of hardware running three different operating systems:

- A PDP 11/73 controlling the PUMA 560 robotic manipulator.

- A SparcStation 4/330 running SunOS 4.1.2.

- A SparcStation 2 running Solaris 2.4.

The RCCL/RCI C routines provide the communication between the SparcStation 4/330 and the PDP 11/73. This software has not yet been ported to the Solaris operating system and thus must run on a SunOS machine. On the other hand, the SunVideo software only runs under Solaris. Therefore, the SunVideo boards used as frame grabbers have to be installed on a Solaris machine and any software which accesses them has to be run on a Solaris machine. The machine available at the time of experimentation was a SparcStation 2 running Solaris 2.4. However, in the future it would be desirable to move the image processing to at least a SparcStation 20 running Solaris 2.5. Additionally, the hand-eye coordination system is currently implemented under the Solaris 2.4 operating system and thus runs on the SparcStation 2.

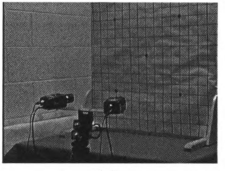
The communication provided by RCCL/RCI is a client/server message passing approach. In order to facilitate the communication between the implemented software on the Solaris 2.4 machine and the RCCL/RCI software on the SunOS 4.1.2 machine, a client/server message passing approach using the *Remote Procedure Call Language* (RPC) provided by the Sun operating systems is implemented. A client program was implemented on the Solaris machine. This client provides an interface between the coordination software and the RCCL/RCI software. The coordination software determines the next desired movement of the PUMA 560 in terms of the incremental joint movements which should be made. The RPC client routine is passed this information as an input parameter. The client then performs a remote procedure call to the server routine which is running on the SunOS machine. The server program consists of a set of functions which cause different actions to be performed by the PUMA 560. Each function calls the RCCL/RCI C routines needed to perform

its designated action. The client routine passes a record which indicates the type of action to be performed as well as vectors of single precision floating point values which provide the information needed to perform that action. As new actions are required, it is only necessary to add a function which calls the appropriate RCCL/RCI routines to accomplish the action. Currently, available functions include: park the manipulator; open the gripper; close the gripper; move to point (x, y, z) within the manipulator's coordinate system; move to joint values $(j_1, j_2, j_3, j_4, j_5, j_6)$; adjust joint values by $(\Delta j_1, \Delta j_2, \Delta j_3, \Delta j_4, \Delta j_5, \Delta j_6)$.

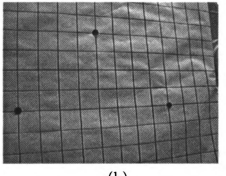
Discussion Establishing the RPC communication between the SunOS and Solaris operating systems was not straight-forward. The routine **rpcgen** converts RPC Language code into C code which implements the RPC protocol. However, the SunOS **rpcgen** and the Solaris **rpcgen** produce different code, namely, the latter produces ANSI standard C code, and the former does not. In particular, the way in which the formal parameters of functions are specified is different between the two, and must be dealt with in order for the client and server to communicate with each other. Once this difficulty was overcome, the coding for the RPC interface was straight-forward.

4.2.3 Camera-Centered Stereo System

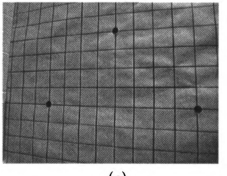
The function of the Camera-Centered Stereo module is to map the image coordinates (in pixels) of a selected point in space to its corresponding coordinates in the camera centered coordinate system. The input space is the row and column position of a specified point in the left camera image and the column-wise disparity between the left image position and the right image position of the point. In this experiment, a grid consisting of black lines on a white background is used to sample the input space (see Figure 4.15). This grid is set at various depths from the camera system, ranging



(a)



(b)



(c)

Figure 4.15: The grid used in the camera-centered stereo system experiment. (a) Observer's view. (b) View of left camera. (c) View of right camera.

from $85cm$ to $125cm$ at $5cm$ intervals. A set of vectors are sampled from the input space. These vectors correspond to the grid intersections in a $40cm \times 40cm$ vertical plane of the board, at each of the specified depths. There are $9 \times 9 = 81$ grid points within the given region, and 9 depths, or a total of $9 \times 81 = 729$ sample vectors. For each grid point, the (x, y, z) camera-system coordinates and the (r, c, d) image coordinates are recorded. A RPT is constructed to partition the input space. A set of 365 vectors are randomly chosen from among the sample set using a Gaussian distribution. The output vectors are stored in the corresponding leaf nodes. The

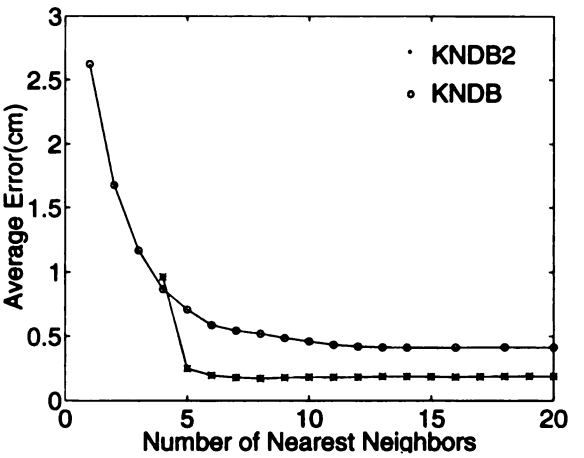


Figure 4.16: The performance of image space to camera space mapping using a RPT trained with 365 samples and using the $KNDB$ interpolation algorithm to compute the output vectors. The results shown here are from testing the module with 200 samples. Various values for the interpolation parameter α are used.

results in Figure 4.16 were obtained using the $KNDB$ and $KNDB_2$ interpolation algorithm (see Section 3.3) for various K values where K specifies the number of nearest neighbors used to compute the output vector for each query vector. In all, 200 test vectors were chosen randomly from the sample set and used as query vectors. Figure 4.16 shows the average error in cm between the computed output vector and the true output vector.

Discussion

This method provides a fast means of performing stereo mapping. For a given query vector it takes only a few mille-seconds, on average, to compute a corresponding output vector. Additionally, the computed output will be, on average, within $0.5cm$ of the true output (this will depend on the algorithm and the number of nearest neighbors used to compute the output) as shown in Figure 4.16.

The down side is that, currently, the training and test data are sampled manually. The manual approach is tedious and time consuming. However, while we acknowledge that an automatic or more simple manual approach to data collection is necessary for the camera-centered stereo method presented here, development of such a system is beyond the scope of this thesis and left for future work.

4.2.4 Sensor Movement Control

In one experiment, a sensor control module is constructed using an RPT whose input is the row, column and disparity vector of the target position in the camera images, and whose output is the pan and tilt values angles to which the head should move. For these experiments, the *head* is a pan/tilt unit from Directed Perceptions. The goal of the sensor control module is to position the head so that the target position is located at the center of the input images. This accomplishes the task of locating an object of interest on the fovea of the visual sensor. To train the sensor movement module, the grid in Figure 4.15 is used. A target point is marked at one of the grid intersections. Then the pan/tilt unit is moved so that the point is at the center of the images. The training pairs consist of the input row, column and disparity of that point, and the output pan and tilt angles required to place that point at the center of the images. An RPT is constructed using these training pairs. Figure 4.17 shows

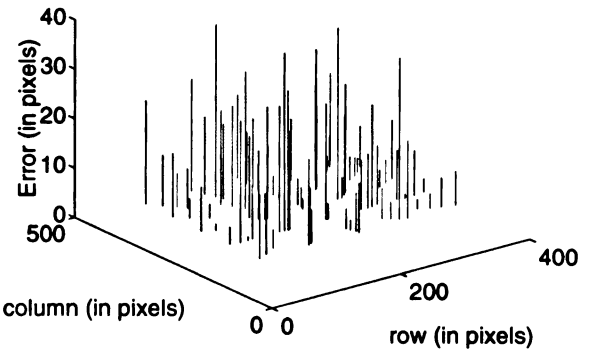


Figure 4.17: The performance of the sensor control system using a RPT trained with 45 samples and using the $KNDB$ interpolation algorithm to compute the output vectors. The results shown here are from testing the module with 100 samples. The interpolation parameter is $\alpha = 3.0$.

the results for this experiment. Here the $KNDB$ algorithm was used with $K = 5$. Training data was sampled at $10cm$ intervals in depth between and including $85cm$ and $125cm$; and at each depth grid intersections where chosen at $20cm$ intervals in both the x and y directions. This provides a 5×5 point grid at each depth for a total of 5×9 training samples. Test data was sampled at $10cm$ depth intervals but between and including $90cm$ and $120cm$. In order to test this module, the test data were given to the module, one at a time, the corresponding output was computed, and the pan/tilt unit was made to move to the new pan and tilt angles as specified by the module's output. The difference between where the target point actually fell within the image and where it should have appeared in the image after movement has been plotted in Figure 4.17. The average absolute error is 10.6 pixels.

Discussion

Even given very few training samples, the sensor movement control module is able to perform well in this experiment. It is not required that the target point be moved exactly to the center of the images, only that it be moved near it. The rational is that the closer to the center an object is, the less lens distortion will affect processing

of that object's image by other modules in the system.

4.2.5 Object Grasping

In the second experiment, a RPT is used in conjunction with the $KNDB$ [44] interpolation algorithm to guide a PUMA 560 robotic manipulator to grasp an object. In this experiment, the system is taught how to perform the *grasp* task via repetition of the task within a specified area of the manipulator's reachable work area. To train the system, the object, a blue cup, is placed in ten random positions in a $30cm \times 30cm$ plane positioned on a horizontal work surface. The *grasp* task is divided into two subtasks: one in which the manipulator *approaches* the object so that it is aligned in a proper position for grasping the object; and the second one in which it positions the end-effector around a specified part of the object.

The input space for the RPT consists of the type of subtask (approach or grasp), the row, column and disparity image coordinates of a vector from a point on the end-effector to a point on the object (thus indicating the separation between the current position of the end-effector and the target position), and the current values of the six joints. The output vector consists of the incremental values by which each of the six joints must move in order to place the end-effector at an appropriate position (depending on the subtask being performed). For this experiment, ten sequences were presented to the system as training data. Each sequence consisted of the following. The manipulator is moved into a pre-defined *park* position. The object is placed in a random position. The arm is moved into an appropriate approach position and a training input/output pair is saved. Then the arm is moved into an appropriate grasp position and a second training pair is saved. One such sequence is shown in columns 1-3 and row 1 and 3 of Figure 4.18 as seen from the left (a) and right (b) cameras.

Discussion

As Table 4.1 shows, the system is able to successfully perform the approach and grasp subtasks in all ten re-substitution and all ten random test trials. It should be noted that while visual feedback is presented to the system, no real vision system is used in this experiment. Instead, for the sake of expediency, image position of objects and end-effector are manually indicated using a workstation mouse and the graphical user interface developed in [44].

Table 4.1: Table showing the success rate of the *approach grasp point* and *grasp* trials.

	Approach Grasp	Grasp	Number of Trials
Re-substitution	100%	100%	10
Random Test	100%	100%	10

4.2.6 Temporal Sequences

The experiment presented in the previous section is actually a subset of a larger experiment in which the system is taught an entire temporal sequence of tasks in which it grasps a cup full of liquid and pours the contents into a second cup. This sequence consists of five subtasks: approach grasp point; move to grasp point and grasp cup; approach delivery position; pour contents; place empty cup back on work surface. As explained in the preceding section, the system is presented with ten sample sequences. A single RPT is constructed based on this training data. The system is then tested by re-substitution of the training samples as well as by ten randomly chosen test samples. The results are given in Table 4.2 (which includes data from Table 4.1). Note that the grasped cup and the *receiving* cup are both placed at ten different positions during training, within two different $30cm \times 30cm$ planes on the work surface. One temporal sequence involving all five subtasks is

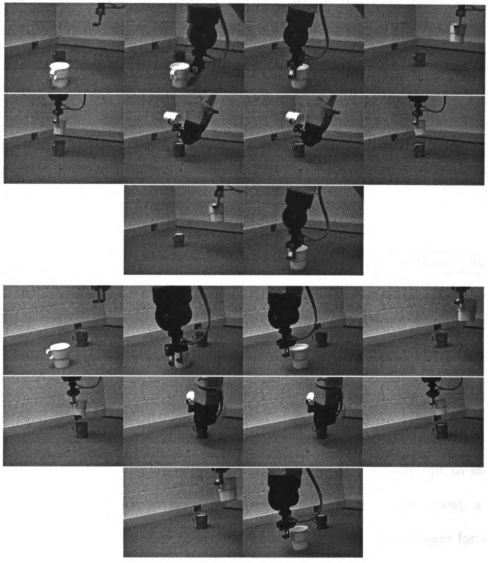


Figure 4.18: Demonstration of a temporal sequence of five subtasks as seen by the left (top three rows) and right (bottom three rows) cameras.

shown in Figure 4.18. Rows 1 and 2 show the sequence as seen from the left camera and rows 3 and 4 show the sequence as seen from the right camera.

Discussion

The success rate is a pass/fail discrete measurement for each subtask as described for the previous experiment. However, in the case of the pour subtask, a pass is recorded only if more than 50% of the liquid contents are successfully transfered from one cup to the other cup. In this experiment, two random pour trials failed based on this criteria. The other 18 trials all transfered at least 80% of the liquid.

Table 4.2: Table showing the success rate of the *approach grasp point*, *grasp*, *approach delivery point*, *pour* and *replace* trials. Ten training samples were used to train this system, and all ten samples were used in the re-substitution test. Ten different random samples were used for the random test.

	Approach Grasp	Grasp	Approach Delivery	Pour	Replace
Re-substitution	100%	100%	100%	100%	100%
Random Test	100%	100%	100%	80%	100%

4.2.7 Visual Guidance

Finally, a simple vision system is implemented for the last experiment to show the feasibility of this approach to hand-eye coordination. In this experiment, a simple corner detection method is used to search the input left and right images for the cup and the end-effector. In order to simplify the image processing task, the background scene is simplified and the input images are scaled to remove differences due to lighting conditions. The computation cost of searching two 640×480 pixel images for both the object and the end-effector is high. In order to reduce the search area of the corner-detection algorithm, a set of four RPTs are constructed—one for each combination of

{left,right} image with the set {cup,end-effector}. For these RPTs, the input space is the set of all possible 640 × 480 column vectors where each component can take a value in the range $0 - 255$. The training vectors are then constructed by stacking the columns of an image, one on top of the other. The output space is the row and column image position of the desired object (in this case, the cup or the end-effector). The RPTs used in this experiment are *MDF* RPTs based on the work reported in [90] and [83]. For this experiment each of the RPTs is trained with nine image sequences

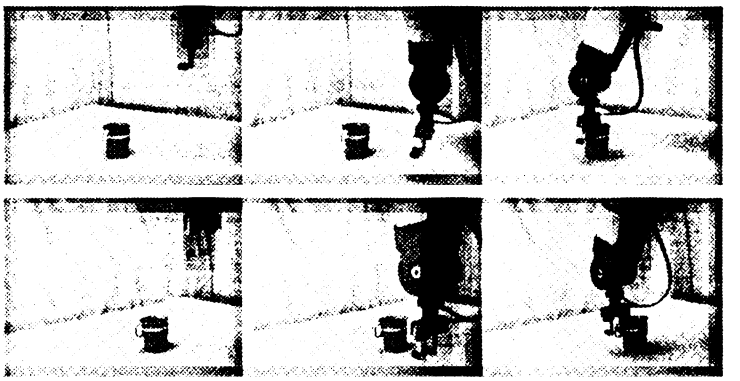


Figure 4.19: Demonstration of a grasping task using the simple vision as seen by the left (row 1) and right (row 2) cameras.

of the approach and grasp subtasks. As before, the system is tested by re-substitution of the training samples as well as by placing the cup at ten random positions within the work area. An example of one such sequence using this simple vision module is shown in Figure 4.19.

Discussion

As Table 4.3 shows, 100% of the re-substitution tests are successful. Also, 100% of the random approach tests are successful. However, two of the random grasp tests failed. These failures are a result of the MDF RPTs producing output positions which are 200+ pixels away from the actual position. The search area for the corner detection method is specified to be a $N \times M$ rectangle centered at the given output

Table 4.3: Table showing the success rate of the approach grasp point and grasp when the simple vision module is used to locate the object and end-effector.

	Approach Grasp	Grasp	Number of Trials
Re-substitution	100%	100%	10
Random Test	100%	80%	10

row and column position of the object. Thus, given the 200+ pixel error in the RPT output, the corner detection algorithm has no hope of ever locating the actual object within the image. The advantage given by using the RPTs can be seen in the overall

Table 4.4: Table showing the time in seconds taken to locate an object (cup or end-effector) when using the simple vision module and when not using the simple vision module.

	With RPT	Without RPT
Time to Search cup (secs)	37	164
Time to Search gripper (secs)	32	155

time saved in locating an object. As shown in Table 4.4, the time to locate an object without using the RPTs is 4.5+ greater than the time it takes to locate an object when the RPTs are used.

4.3 Summary

In summary, the modules which have already been constructed for the proposed complete system perform quite well in simulation, as shown in the preceding sections. The results shown were for single *runs*, that is, a single movement of the arm. Furthermore, although the system implemented for the real experiments differs from the originally proposed system, the results obtained are also quite good. However, while the simple vision module implemented for these experiments demonstrates the feasibility of the

system, steps need to be taken to integrate a general purpose, robust recognition module into the system. Additionally, all the pieces of the system demonstrated in the experiments above, need to be integrated into a complete system.

It is important to note that the implemented system does not handle occlusion—the target point and gripper-identification point must be visible to the camera systems throughout execution of the task. The system is unable to plan paths around objects in order to reach the target position; thus the path between the current end-effector position and the target position must be obstruction-free. The current system is limited to reaching positions via three dimensional Cartesian movement of the end-effector in a static orientation. It is expected that future implementations will be able to handle orientational changes as well. The current implementation is a low-level approach to HEC, and thus there is no high-level reasoning about the target object. This has the benefit that, given a high-level vision system as a front end which can identify a target position within the manipulator's workspace, the system can approach that position independent of the object. A disadvantage to the current implementation is that without high-level reasoning there is no orientational information available for planning the approach vector. Finally, the system is unable to correctly approach positions outside of the trained region.

This last point leads naturally to the question of whether or not an *ideal* training set can be defined for a given workspace and setup. As has been shown in the pattern recognition field and elsewhere, determining an ideal set of features is, at best, a difficult task. For the work presented in this thesis, the best that can be done is to compile a training set which most closely defines the desired application. This set may then be augmented, by adding additional training members to the existing data structure.

Chapter 5

Contributions and Future Work

5.1 System Overview

Through careful consideration of each subtask of the *Hand-Eye Coordination* (HEC) problem, we have viewed the *process* of HEC as a sequence of transformations from an input space to an output space. We choose to view the entire process of HEC, from eye to hand, as a mapping from *scene space* to *arm configuration space*. This single mapping may be broken into a *sequence* of mappings from one space to another. The sequence we have chosen to model is the following: scene space to image space; image space to camera coordinate system; camera coordinate system to arm/world coordinate system; arm/world coordinate system to arm configuration space; and image space to head configuration space. Each of these mappings is represented by a module in Figure 1.5.

Given this view, we have adopted a unified framework based on the *Recursive Partition Tree* (RPT) data structure presented in Chapter 3. In this framework the RPT is responsible for learning the transformation from its input to its output space and storing this acquired knowledge in a manner which facilitates the fast and

efficient retrieval of that data during run-time. During run-time, when presented with a query vector from its input space, the RPT is responsible for quickly determining the k vectors of its training set which are nearest to the query vector. Given this information, the output vector corresponding to the query vector is approximated via interpolation of the k output vectors corresponding to the retrieved training samples. This general framework provides us with a method for systematically dealing with the complex relationship between the sensors and the manipulator without explicitly modeling the relationship. Instead, the system is allowed to build whatever implicit models it needs to learn the observed action. As an example, in order to explicitly model the task of pouring a liquid from one cup into another would require analyzing the flow dynamics and fluid mechanics. This would be a complicated task given the large number of unknowns. The system presented in this thesis makes it possible to perform the task by showing the system how to handle the situation, and the system learns through observation. The contributions of this research are summarized below.

5.2 Contributions

5.2.1 Unified Framework for Sensor-Actuator Coordinated Learning

A unified framework for performing sensor-actuator coordinated learning has been introduced. This framework uses a recursive partitioning algorithm to build a hierarchical tree classifier which uses a nearest neighbor classification based on the Voronoi tessellation as its decision making criteria. The assumption is that the *task* can be modeled as a mapping from an input space to an output space. At the heart of this framework is the RPT data structure. The RPT models the density of the sample

space in that fewer nodes are allocated to areas of less density and more nodes are allocated to areas of high density, as represented by the training set. Thus areas of the workspace which are high traffic areas are well covered by the RPT.

5.2.2 Simple Object Recognition and Location Using the RPT Framework

A simple object recognition and location module has been implemented in order to guide the system during training by providing the image data required by the system. The same simple system was used during run-time to provide visual feedback. The RPT framework is used to reduce the search space for a simple corner detection algorithm. The resulting combined system is $4.4 - 4.8$ times faster than the corner detection algorithm used alone (see Table 4.4). In the current implementation, only a single iteration is performed, reducing the search space by one half. So, given images of 640×480 pixels, the search space is reduced to a 320×240 sub-region of the image. Adding additional iterations to further reduce the search area will yield an even greater reduction in processing time.

5.2.3 Manipulator Control in a Redundant System

The term redundant applied to robotic manipulators generally refers to the fact that there are an infinite number of configurations of the manipulator which will place the end-effector in a specified position and orientation. This in turn results in there being an infinite number of solutions for a given set of inverse kinematic equations. The robotic manipulator used for the experiments in this thesis has only six degrees of freedom and so is redundant only in terms of position, and not with respect to orientation. It takes at least seven degrees of freedom for true redundancy. However,

if a six degree of freedom manipulator is placed on a three degree of freedom mobile platform, then the integrated system is redundant in the usual sense of the term. The framework presented in this thesis is capable of implementing a manipulator control system for a redundant manipulator because the movements specified by the system are based on previously learned movements which are known to achieve a desired action which is at least similar to the action currently being requested. Thus, there is no need to hand-code a rule-based system which chooses an action from among the infinite number of possibilities supplied by the inverse kinematic equations of the system.

5.2.4 Controllable Sensor in a Learning Framework

A simple active vision system has been implemented using the framework. Active vision systems are typically complex and difficult to completely specify, involving parameters to control pan, tilt, zoom, aperture, etc. Calibration of such a complex system can be tedious. Although automatic control of the pan/tilt unit has not been implemented in this thesis, and while the current system only considers the pan and tilt parameters, the applicability of the framework in implementing an active vision system has been demonstrated. The framework is general enough that adding parameters or even additional sensory information only requires that an appropriate set of training data be supplied. The framework has been used to implement complex systems with hundreds of parameters with complex relationships in [82], [18] and [22].

5.2.5 Addresses Several Modules in Hand-Eye Systems

Several of the HEC subtasks have been implemented using the above mentioned framework. The implemented modules include: control of a pan/tilt unit used to

implement a simple active vision platform; stereo calibration; a simple object recognition and location system for training and for visual feedback during runtime; and a single image space to joint space system which combines the three systems originally proposed for the system.

5.3 Summary

The RPT framework has several advantages. By coupling an interactive vision component with manipulation we obtain a system which is able to constantly monitor the world and adjust the system based on what it is seeing *now*. Most robot navigation approaches have this feature. For example, in [72] the robotic system is able to navigate by performing simple curve following. To use this approach in a vision–manipulation context is much more challenging. Neither the recognition problem nor the manipulation problem is itself a trivial matter. Combining them together into a coupled system will be even more challenging, but may provide new insight into how to approach the two tasks. Many approaches to HEC are rule-based and hand-coded. Prime examples of this involve determining the equations and parameters for stereo calibration, and the forward and inverse kinematics for controlling the robotic manipulator. This research has introduced a unified framework for learning the mapping function from a given input space to a corresponding output space. At the heart of the framework is the RPT which is constructed via unsupervised learning given a set of training data sampled from the input space, and their corresponding output vectors. The topology of the tree is not determined *a priori* as in [65], [66] and [86] but allowed to develop based on the given set of training samples and the order in which they are presented to the construction algorithm.

5.4 Future Work

Currently, the system is at a *teaching* level—it is shown exactly how to do a particular task and the system itself generalizes what it knows, in so far as it can perform the same task in a different position. However, the degree of human interaction required during the learning phase needs to be reduced. Eventually, the system needs to reach the level of *autonomous* learning, which may still need some feedback, but to a lesser extent than what is currently needed. The method presented in this research is open to the possibility of performing autonomous learning. This learning would have as a base a set of explicitly instructed data on which to build. This research is a first step towards providing that explicit data. In order for systems to be useful in an unknown and dynamically changing environment it must be able to expand its knowledge. In order to be flexible and non-fragile, it must be able to do much of this on its own.

A robust object recognition and location system needs to be attached to the system in order to reduce the amount of manual labor required for training. In the current implementation, the Camera-Centered Stereo, Camera-to-Arm, and Arm-to-Joint system modules of the originally proposed system have been combined into a single module in order to reduce the amount of training data which has to be manually collected. The current system works for the tasks performed in this thesis (and [45] and [44]) because there is no need to obtain the three-dimensional coordinates of points in order to perform the desired tasks. The system implemented for this thesis provides a quicker and more efficient method of training for the given tasks. Given a system with a robust object recognition and location system, and a training algorithm based on feedback, it should be possible to implement the original system. It seems that the modularity of the original design will produce a system that is better suited for applications in which the three-dimensional coordinates of points are needed to

perform the specified tasks. This is because the Camera-Centered Stereo system would maintain the relationship between object locations as seen in the stereo images, and object locations with respect to the camera system's coordinate system. When the camera system is physically moved, this mapping would not be changed and so the Camera-Centered Stereo system would not require re-training as would the camera space to arm/world space mapping.

Given the simple vision system implemented for the current version of this system, it is not possible to tightly couple the vision system with the manipulation system. Thus, the system is limited by the lack of a general purpose, robust recognition system. If the method presented in this thesis is to be tested to its full potential, it will be necessary to integrate such a recognition system.

Finally, the current workspace for which the system has been trained is fairly small, again due to the need to reduce the amount of data to be manually collected. Once a more automatic data collection technique is implemented, the system needs to be trained to work on a larger region of its workspace.

Appendices

Appendix A

Constructing an RPT Incrementally

An RPT is a network of L levels. Each level consists of a number of nodes where each node represents a sub-space of the N-dimensional input space. A RPT is constructed *incrementally* when each training sample is visited only once during construction, as opposed to once per level of the RPT. In order to understand this algorithm for constructing an RPT we must define a few terms used in the decision making processes of the construction algorithm.

Learning: In this method our definition of learning is composed of two aspects: partitioning of the input space; and computing a linear estimation of the mapping function from the input space to the output space in each local subspace of the partition.

Spacing Parameter: At each level, l, of the network we define the maximum extent of the neighborhood via a distance parameter called the *spacing parameter*. Let V^N be a training sample being considered for inclusion in the RPT currently under construction. Let V^P be a primary point in that RPT. Then the phrase *maximum extent of the neighborhood* indicates the maximum distance allowed between V^N and

V^P at level l such that V^N may still be considered to be within the neighborhood of V^P at that level. The spacing parameter is currently defined as:

$$S(l) = a^{L+1-l},\ l = 1, \ldots, L \tag{A.1}$$

with $l = 0$ at the coarsest level and $l = L$ at the finest level of the hierarchy, and $a = \textit{one unit}$ where *one unit* indicates the smallest/finest unit size at level L. $S(0)$ is set to equal the maximum radius of the input space, which effectively covers the entire input space. An example of the spacing parameter is given in Table A.1 for $L = 6, a = 2$. Neighborhood: A node is a *level l node* if it defines a cell of size a^{L+1-l}.

L = 6, a = 2						
l:	1	2	3	4	5	6
$S(l) = a^{L+1-l}$:	$a^6 = 64$	$a^5 = 32$	$a^4 = 16$	$a^3 = 8$	$a^2 = 4$	$a^1 = 2$

Table A.1: Example of spacing parameter for $L = 6, a = 2$.

Let $\mathcal{L}(l)$ be the list of all level l nodes. In general, let $f : D \rightarrow R$ be a mapping from domain D to range R, and (x, z) be a node where $x \in D$ and $z \in R$. Let (y_0, z_0) be a sample node at level l. Then the neighborhood of y_0 is defined as

$$N(y_0, l) = \{x \mid \| x - y_0 \| \leq \| x - y \|, \forall x \in D, \forall y \in D \textit{ such that } (y, z) \in \mathcal{L}(l)\} \tag{A.2}$$

That is, the neighborhood of y_0 at level l consists of all $x \in D$ that are closer to y_0 than to any other level l node's primary point. Then given any $x \in D$, its nearest neighbor at level l is y_0, if $x \in N(y_0, l)$, i.e., x is in the neighborhood of y_0 at level l. For a given input x, denote node y_0, z_0 by $n(x, l) \in D$, i.e., the node representing the neighborhood to which x belongs at level l. Likewise, denote the output corresponding to x at level l given $n(x, l)$ by $z(x, l) \in R$. Then an RPT defines a mapping $g : D \rightarrow R$

such that:

$$g(x) = z(x, l) \tag{A.3}$$

Given a mapping $f : D \rightarrow R$ (where D is bounded), and a spacing parameter a, an RPT with L levels is constructed as shown in Figure A.1 using RPT construction method I and described as follows. Let *MOD* be the RPT under construction. Then for each training sample V^I, we descend the existing, partially constructed[1] network until the current branch ends or the distance between the nearest neighbor (along the current branch) at the current level exceeds the maximum extent of a neighborhood for the current level. If the maximum extent is exceeded then V^I is inserted into the network at the current level. If the branch ends at level $l = L$, then we discard V^I. If the branch ends at level $l < L$, then we compare the distance between V^I and the current leaf node, V^P, to the spacing parameter for level $l+1$, $S(l+1)$. If $Dist\left(V^I, V^P\right) > S(l+1)$, then both V^I and V^P are inserted at level $l+1$; otherwise, V^I is discarded.

It is important to note that in this method every primary point has the *potential* of being represented at every level of the hierarchy *following* the level at which it was initially inserted into the network. However, these succeeding representatives are not added unless there is a definite sub-division at level $l+1$ of the sub-space represented by the node at level l. Thus, if $Dist\left(V^I, V^P\right) \leq S(l+1)$, then V^I is not inserted, and V^P is not duplicated at level $l+1$. The reason for this policy is illustrated in Figure A.2. At a given level l, suppose the node for V^P represents the entire sub-space whose boundaries are specified by the spacing parameter $S(l)$. This node is then responsible for computing the output U^I for any and all inputs V^I such that $Dist\left(V^I, V^P\right) \leq S(l)$. This assumes that V^P is a leaf node at level l. If every

[1] Each network consists of at least the root node

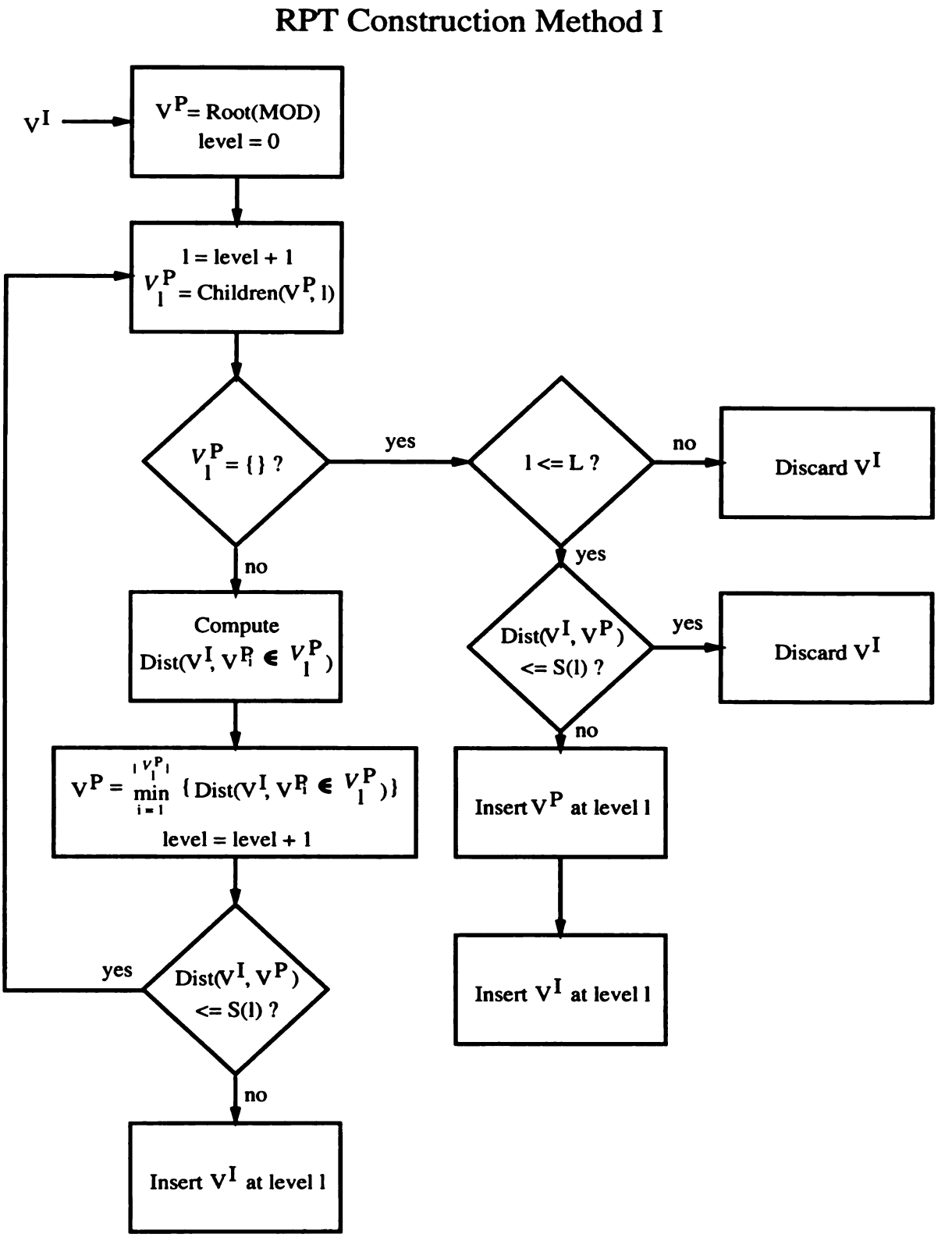


Figure A.1: Schematics of RPT Construction Method I.

time a primary point was inserted at level l we inserted the entire branch from l to L, then we could end up with a *skinny* branch as shown in Figure A.2(a). Then,

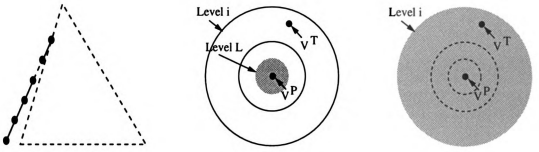


Figure A.2: Example of problems with skinny branches.

during testing, we might end up with a point V^T such that $V^T \in N\left(V^P, l\right)$, but $V^T \notin N\left(V^P, L\right)$. The transformation of each leaf node is estimated based on the neighborhood size defined for the level at which the leaf node appears. Thus, as shown in Figure A.2(b), the leaf node for V^P is only expected to produce good outputs for the inputs which fall in the smaller neighborhood indicated by the small shaded circle. V^T is not within that neighborhood. On the other hand, given the method described in the algorithm, the leaf node of V^P would appear at a coarser level of the hierarchy and thus be responsible for the neighborhood indicated by the larger shaded circle shown in Figure A.2(c), into which V^T does fall. It is not expected that leaf nodes at very coarse levels of the hierarchy will compute very precise outputs, but with the given methodology, their transformation mechanisms have at least been trained on the entire sub-space which they represent.

Appendix B

Adoption Problem

Constructing a RPT using a sequential approach (as presented in Appendix A) introduces an adoption problem. Constructing the network, as this method does, by randomly adding nodes at any level of the hierarchy may introduce *invisible nodes.* An invisible node is one which exists in the network but at some point during the construction becomes unreachable from that time on. Furthermore, this node will never be reached during run-time. In addition, all of its descendant nodes also become unreachable. A node A which is a child of a node B becomes invisible when a new node C is introduced at a higher level of the hierarchy and at that level, A's primary point is contained within C's neighborhood. This problem is due to the random order in which the samples are presented for training. If A's primary point had been presented after C's primary point, it would have become a child of C instead of B. Figure B.1 illustrates this idea.

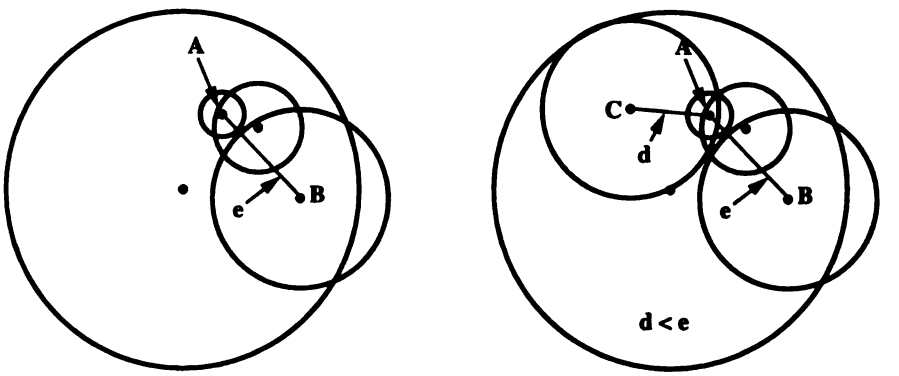


Figure B.1: Demonstration of a node becoming invisible. (a) Before insertion of node C. (b) After insertion of node C.

Appendix C

Empirical Study of RPT Error

Two RPTs are given: one network, T_4, consists of four nodes; the second network, T_5, consists of first network plus one additional node, for a total of five nodes. The same test data is presented to both networks. We are interested in any test data which *changes* leaf nodes from T_4 to T_5. Here *changes* indicates that during the test phase and after descending the network from root node to leaf node (using the algorithm outlined in Appendix A) the test data falls into the neighborhood of the newly added node in the input space. For this analysis, these test data will be referred to as *changlings*. First, we show the cases in which the error for a given changling decreases as expected. Then we will analyze the cases in which the error for a changling is found to increase and why this may have occurred.

Decreasing Error

Three cases are found in which the error for a changling decreases from T_4 to T_5.

Case I In both T_4 and T_5 the changling fell within the neighborhood used to estimate the transformation matrix. See Figure C.1.

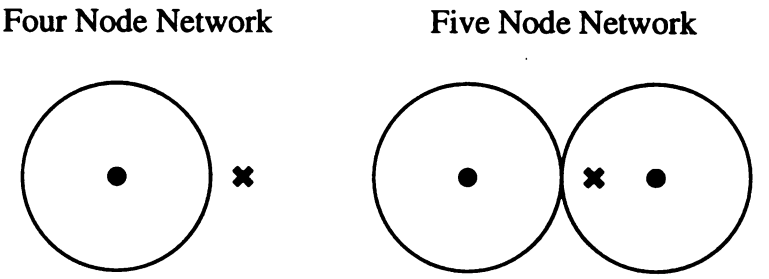


Figure C.1: Case I.

Case II In T_4, the changling was *between* two of the points used to estimate the transformation matrix. In T_5, the changling was *very* near one of the *derived points* (see Section 4.1.2). See Figure C.2.

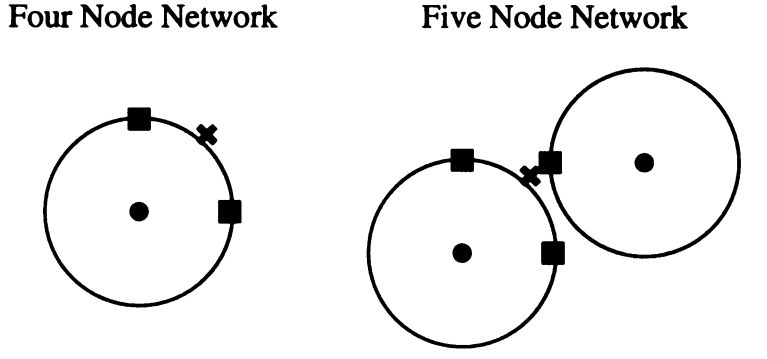


Figure C.2: Case II.

Case III In both T_4 and T_5 the changling was *nearer* the new node, but outside of that node's *estimation neighborhood.* See Figure C.3.

Increasing Error

Two cases are found in which the error for a changling increases from T_4 to T_5.

Case IV Same as Case III. See Figure C.3.

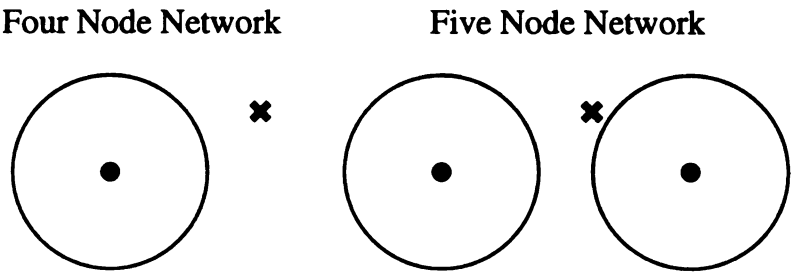


Figure C.3: Case III.

Case V The changling is near a *repeated* node, but outside of the smaller neighborhood even though it was within the parent's larger neighborhood. Here *repeated* indicates that the parent node was duplicated as its own child at the next finer level of the hierarchy and its transformation matrix was re-estimated at the lower level using a smaller neighborhood. See Figure C.4.

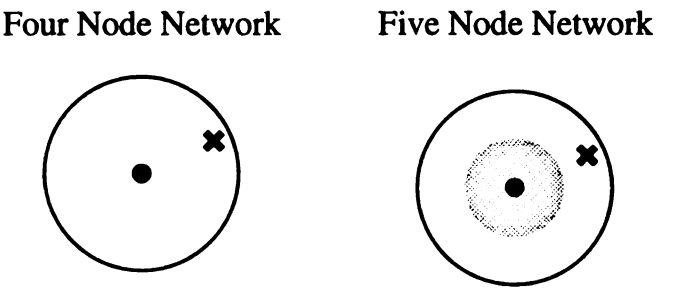


Figure C.4: Case V.

Appendix D

Investigation of Back Propagation Neural Network

Earlier work included an investigation into the ability of a feed-forward artificial neural network with a back propagation learning algorithm (FFBP) to correctly learn a simple smooth function. This was of interest because the desired movement of the robot arm can be modeled as a smooth, continuous function in space. So the idea was to see if a simple FFBP could learn the necessary control of the robot arm. Unfortunately, the FFBP did not perform well enough to inspire confidence in its ability to control a robot arm in motion.

In the following experiments, the *error* is the squared error calculated by the FFBP between the actual output and the desired output of the network. The training samples were all randomly generated in the specified space, and the testing samples, also used for display purposes, were taken as the vertices of a regular grid imposed on the function surface. The grid cells were 0.1×0.1 in size for all experiments. In each figure, the upper left image shows the ideal function, sampled at the grid points. The upper right image shows the surface recovered by the FFBP, at the sample points.

The lower left image shows the difference, in the mathematical sense, between the ideal function and the recovered function computed at the grid points. The lower right image shows the absolute difference between the two functions.

For all of the experiments, there were two nodes in the input layer and one node in the output layer. The number of hidden layers and nodes/hidden layer varied from one experiment to another. In Experiment One, we trained the network to learn the simple function $f(x,y) = x^2+2.3y^2$. The rest of the experiments were trained to learn the function $f(x,y) = \sin[(2x+y)\pi]$. In Experiment Two, $-1.0 \leq x,y \leq +1.0$, the hidden layer had 15 nodes and was allowed 50,000 cycles. In Experiment Three the range of x and y was doubled to $-2.0 \leq x,y \leq +2.0$. In Experiment Four the number of nodes in the hidden layer was increased to 30. In Experiment Five the number of allowed training cycles was increased tenfold to 500,000. Finally, in Experiment Six the number of nodes in the hidden layer was reduced back to 15, while the number of cycles was held at 500,000. The results are reported on the following pages.

D.1 Experiment One: $f(x,y) = x^2 + 2.3y^2$

The results of Experiment One are shown in Figure D.1. The FFBP had one hidden layer with 15 nodes. The FFBP reported an error of 0.004797 after 50,000 iterations with 200 training samples. As expected, the error is small for this relatively simple function.

D.2 Experiment Two: $f(x,y) = sin[(2x+y)\pi], -1.0 \leq x,y \leq +1.0$

The results for Experiment Two are shown in Figures D.2 and D.3. The FFBP had one hidden layer with 15 nodes. Ten trials were conducted in this experiment

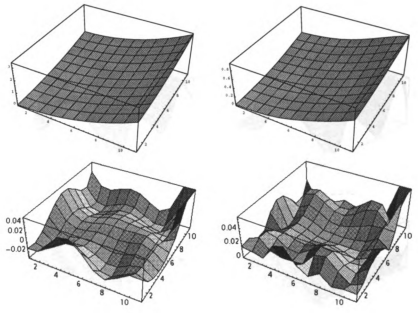


Figure D.1: Function: $f(x,y) = x^2+2.3y^2, x, y = 0.0, 0.1, 0.2, \ldots 0.910.0$; 200 training patterns; one hidden layer with 15 neurons, resp.; 50,000 training cycles

using the same 200 training samples. For each trial the network was cycled for 50,000 iterations. Figure D.2 shows the *best* result for Experiment Two, where best indicates that of the ten trials the result in Figure D.2 had the lowest reported error. The error reported by the FFBP for this run was 0.021662. Figure D.3, on the other hand, shows the *worst* result for Experiment Two, where worst indicates that of the ten trials the result in Figure D.3 had the highest reported error. The error reported by the FFBP for this run was 0.055287. Even for this slightly more complicated function the worst result produced an absolute difference between the desired and actual functions of approximately 0.6, except at the corners of the boundaries where there was not enough data to give an accurate picture of the function at those points.

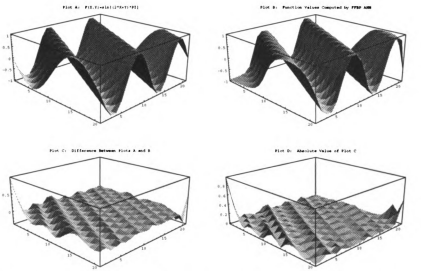


Figure D.2: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-1.0 \leq x,y \leq +1.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; best results out of ten trials.

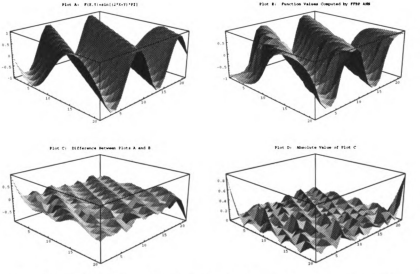


Figure D.3: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-1.0 \leq x,y \leq +1.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; worst results out of ten trials.

D.3 Experiment Three: $f(x,y) = sin[(2x+y)\pi], -2.0 \leq x,y \leq +2.0$

The results for Experiment Three are shown in Figures D.4 and D.5. The FFBP had one hidden layer with 15 nodes. Ten trials were conducted in this experiment using the same 200 training samples, but different initial values for the connection

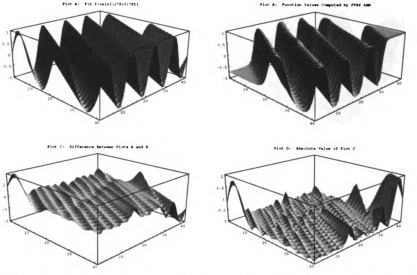


Figure D.4: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; best results out of ten trials.

weights. For each trial the network was cycled for 50,000 iterations. Figure D.4 shows the *best* result for Experiment Three, where best indicates that of the ten trials the result in Figure D.4 had the lowest reported error. The error reported by the FFBP for this run was 0.036592. Figure D.5, on the other hand, shows the *worst* result for Experiment Three, where worst indicates that of the ten trials the result in Figure D.5 had the highest reported error. The error reported by the FFBP for this run was 0.209956. For this more complicated function we start to see problems. It is obvious from Figure D.5 that the worst result is very bad indeed. Even the best result shows peaks in the *center* of the function patch that have a difference of 1.5.

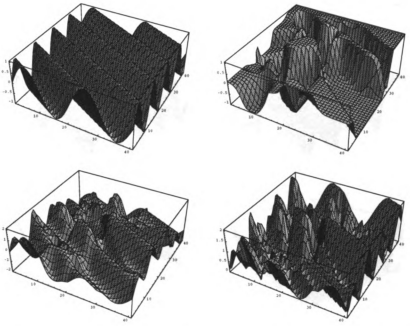


Figure D.5: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 50,000 training cycles; worst results out of ten trials.

D.4 Experiment Four: Number of Hidden Layer Nodes Increased

Since the results in Experiment Three were not as good as hoped for, it was decided to try for better results by changing various parameters of the experiment. In Experiment Four the number of nodes in the hidden layer was increased. The results for Experiment Four are shown in Figures D.6 and D.7. The FFBP had one hidden layer with 30 nodes. Ten trials were conducted in this experiment using the same 200 training samples, but different initial values for the connection weights. For each trial the network was cycled for 50,000 iterations. Figure D.6 shows the *best* result for Experiment Four, where best indicates that of the ten trials the result in Figure D.6 had the lowest reported error. The error reported by the FFBP for this run was 0.039215. This is comparable to the best result in Experiment Three. Figure D.7, on

the other hand, shows the *worst* result for Experiment Four, where worst indicates

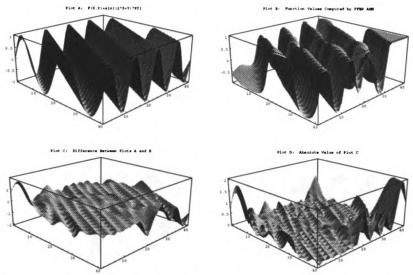


Figure D.6: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 50,000 training cycles; best results out of ten trials.

that of the ten trials the result in Figure D.7 had the highest reported error. The error reported by the FFBP for this run was 0.194288, which is also comparable to the worst result in Experiment Three. Increasing the number of nodes in the hidden layer seems to have had no significant effect on the results.

D.5 Experiment Five: Number of Training Cycles Increased

Experiment Five has the same setup as Experiment Four, except that the number of cycles through which the network iterates is increased to 500,000. That is, let's see if the network just needs more time to converge. Only three trials were run for Experiment Five. All three results are shown in Figures D.8, D.9 and D.10. The FFBP had one hidden layer with 30 nodes. Ten trials were conducted in this

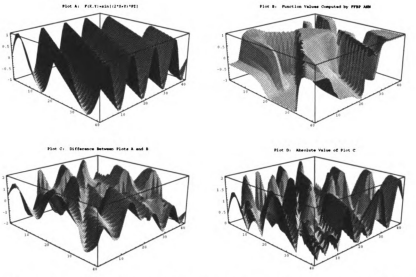


Figure D.7: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 50,000 training cycles; worst results out of ten trials.

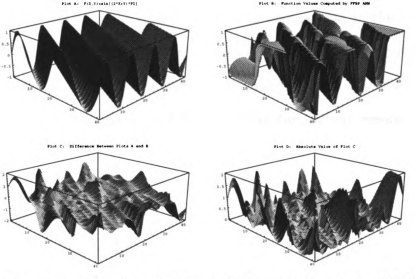


Figure D.8: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles.

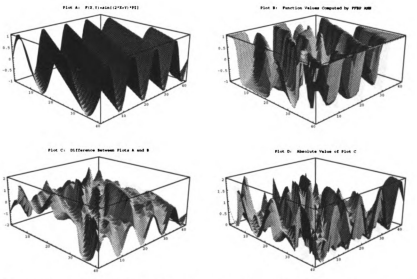


Figure D.9: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x, y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles.

experiment using the same 200 training samples, but different initial values for the connection weights. For each trial the network was cycled for 500,000 iterations. Figure D.8 shows the *best* result for Experiment Five, where best indicates that of the three trials the result in Figure D.8 had the lowest reported error. The error reported by the FFBP for this run was 0.034006. This is comparable to the best result in Experiment Four and is even a little better, but still not what is desired. Figure D.9, on the other hand, shows the *worst* result for Experiment Five, where worst indicates that of the three trials the result in Figure D.9 had the highest reported error. The error reported by the FFBP for this run was 0.045196, which is better than the worst result for Experiment Four by approximately 0.15. The result in Figure D.10 had a reported error of 0.038566. The range of errors in this experiment has narrowed, and the overall error has decreased. So, giving the network more time seems to have helped to lower the error; but looking at Figures D.8, D.10 and D.9, one can see that the results are still not very good. Increasing the number of iterations seems to have had a small, but still insignificant effect on the results.

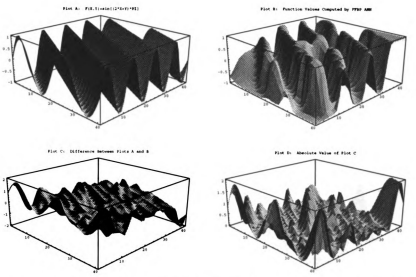


Figure D.10: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 30 neurons, resp.; 500,000 training cycles.

D.6 Experiment Six: Number of Hidden Layer Nodes Decreased

For the sake of completeness, Experiment Six goes back to having only 15 nodes in the hidden layer as in Experiment Three, but allows the network to cycle 500,000 times as in Experiment Five. The results for Experiment Six are shown in Figure D.11. The FFBP had one hidden layer with 15 nodes. One trial was conducted in this experiment. The network was cycled for 500,000 iterations. Figure D.11 shows the result for Experiment Six. The error reported by the FFBP for this experiment was 0.044859. This is comparable to the worst result in Experiment Five, but is a little worse. Increasing the number of iterations for a network with fewer nodes in the hidden layer seems to have had no significant effect on the results.

What can we conclude from these results? The large variance between the best and worst results in Experiments Three and Four show that the network does not perform consistently from one time to the next. Even with an extremely high number

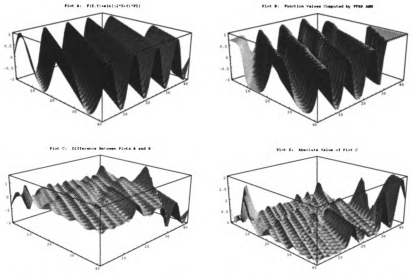


Figure D.11: Function: $f(x,y) = \sin[(2x+y)\pi]$; $-2.0 \leq x,y \leq +2.0$; 200 training patterns, one hidden layer with 15 neurons, resp.; 500,000 training cycles.

of iterations allowed, the FFBP is not able to converge to a stable global minimum. This kind of behavior will not work well for learning to control a vision-guided robot manipulator where the action of the manipulator is a highly complex function and must be consistent and smooth at all times.

Bibliography

Bibliography

[1] Abbott, A.L. and N. Ahuja. Surface Reconstruction by Dynamic Integration of Focus, Camera Vergence, and Stereo. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 532–543, Los Alamitos CA, 1988. IEEE Computer Society Press.

[2] Abrams, R.A., D.E. Meyer and S. Kornblum. Eye-Hand Coordination: Oculomotor Control in Rapid Aimed Limb Movements. *Journal of Experimental Psychology*, 16(2):248–267, May 1990.

[3] Agarwal, K.K. Solving Problems in Robotics with Semantic Networks. *IEEE Trans on PAMI*, PAMI-5(2):213–217, Mar 1983.

[4] Ahuja, N. Dot Pattern Processing Using Voronoi Neighborhoods. *IEEE Trans on PAMI*, PAMI-4(3):336–343, May 1982.

[5] Allen, P.K., A. Timcenko, B. Yoshimi and P. Michelman. Trajectory filtering and Prediction for Automated Tracking and Grasping of a Moving Object. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, volume 2, pages 1850–1856, Nice, France, May 1992. IEEE Computer Society Press.

[6] Allen, P.K., A. Timcenko, B. Yoshimi and P. Michelman. Automated Tracking and Grasping of a Moving Object with a Robotic Hand-Eye System. *IEEE Trans on Robotics and Automation*, 9(2):152–165, Apr 1993.

[7] Aloimonos, J. Purposive and Qualitative Active Vision. *Int'l Conf on Pattern Recognition*, pages 346–360, 1990.

[8] Aloimonos, J., I. Weiss and A. Bandopadhyay. Active Vision. *Int'l Journal of Computer Vision*, pages 35–54, 1988.

[9] Attendees. Promising Directions in Active Vision. In *NSF Active Vision Workshop*, pages i–60, Aug 1991.

[10] Barth, M., H. Ishiguro and S. Tsuji. Determining Robot Egomotion from Motion Parallax Observed by an Active Camera. In *Int'l J Confs on Artificial Intelligence*, pages 1247–1253. ijcaii, Aug 1991.

[11] Bassi, D.F. and G.A. Bekey. High Precision Position Control by Cartesian Trajectory Feedback and Connectionist Inverse Dynamics Feedforward. In *Int'l J Conf on Neural Networks*, volume 2, pages 325–331, Jun 1989.

[12] Braunegg, D.J. MARVEL: A System That Recognizes World Locations with Stereo Vision. *IEEE Trans on Robotics and Automation*, 9(3):303–308, Jun 1993.

[13] Brooks, R.A. Symbolic Reasoning among 3-D Models and 2-D Images. *Artificial Intelligence*, 17:285–348, 1981.

[14] Brooks, R.A. Model-Based Three-Dimensional Interpretations of Two-Dimensional Images. *IEEE Trans on PAMI*, PAMI-5:140–150, 1983.

[15] Castanõ, A. and S. Hutchinson. Hybrid Vision/Position Servo Control of a Robotic Manipulator. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1264–1269, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[16] Charniak, E. and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading MA, 1987.

[17] Chen, S. and J.J. Weng. Autonomous Vehicle Navigation Using Scene Recognition. Technical Report CPS-94-29, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1994.

[18] Chen, S. and J.J. Weng. Incremental Learning for Vision-Based Navigation. In *Int'l Conf on Pattern Recognition*, Aug 1996. To Be Published.

[19] Ching, W.-S., P.-S. Toh, K.-L. Chan and M.-H. Er. Robust Vergence with Concurrent Detection of Occlusion and Specular Highlights. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 384–394, Los Alamitos CA, May 1993. IEEE Computer Society Press.

[20] Clark, J.J. and N.J. Ferrier. Modal Control of an Attentive Vision System. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 514–523. IEEE Computer Society Press, 1988.

[21] Cordo, P.J. and M. Flanders. Sensory Control of Target Acquisition. *Trends in Neurosciences*, 12(3):110–117, Mar 1989.

[22] Cui, Y. *Image Sequence Analysis: Motion and Structure Estimation with Transitory Sequences and Recognition of Hand Signs.* PhD thesis, Michigan State University, 1996.

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

[24] Eckmiller, R., J. Beckmann, H. Werntges and M. Lades. Neural Kinematics Net for a Redundant Robot Arm. In *Int'l J Conf on Neural Networks*, volume 2, pages 333–339, Jun 1989.

[25] Efstathiou, A., J. Bauer, M. Greene and R. Held. Altered Reaching Following Adaptation to Optical Displacement of the Hand. *Journal of Experimental Psychology*, 73(1):113–120, 1967.

[26] Espiau, B., F. Chaumette and P. Rives. A New Approach to Visual Servoing in Robotics. *IEEE Trans on Robotics and Automation*, 8(3):313–326, Jun 1992.

[27] Feddema, J.T. and O.R. Mitchell. Vision-Guided Servoing with Feature-Based Trajectory Generation. *IEEE Trans on Robotics and Automation*, 5(5):691–700, Oct 1989.

[28] Feddema, J.T., C.S.G. Lee and O.R. Mitchell. Automatic Selection of Image Features for Visual Servoing of a Robot Manipulator. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 832–837, Scottsdale AZ, May 1989. IEEE Computer Society Press.

[29] Flanders, M., P.J. Cordo and J.G. Anson. Interaction Between Visually and Kinesthetically Triggered Voluntary Responses. *Journal of Motor Behavior*, 18(4):427–448, Dec 1986.

[30] Flynn, P.J. and A.K. Jain. 3D Object Recognition Using Invariant Feature Indexing of Interpretation Tables. *CVGIP: Image Understanding*, 55(2):119–129, Mar 1992.

[31] Francisco, A. Relative Depth from Vergence Micromovements. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 481–486, Los Alamitos CA, May 1993. IEEE Computer Society Press.

[32] Freeman, J.A. and D.M. Skapura, editor. *Neural Networks: Algorithms, Applications, and Programming Techniques.* Addison-Wesley, Reading MS, 1992.

[33] Geschke, C.C. A System for Programming and Controlling Sensor-Based Robot Manipulators. *IEEE Trans on PAMI*, PAMI-5(1):1–7, Jan 1983.

[34] Gomi, H. and M. Kawato. Equilibrium-Point Control Hypothesis Examined by Measured Arm Stiffness During Multijoint Movement. *Science*, 272:117–120, Apr 1996.

[35] Graf, D.H. and W.R. LaLonde. A Neural Controller for Collision-Free Movement of General Robot Manipulators. In *Int'l Conf on Neural Networks*, volume 1, pages 77–84, San Diego CA, Jul 1988. IEEE Computer Society Press.

[36] Graf, D.H. and W.R. LaLonde. Neuroplanners for Hand/Eye Coordination. In *Int'l J Conf on Neural Networks*, volume 2, pages 543–548, Washington DC, Jun 1989. IEEE Computer Society Press.

[37] Grimson, W.E.L. Recognition of Object Families Using Parameterized Models. In *Proc of the First Int'l Conf on Computer Vision*, pages 93–101, London England, Jun 1987.

[38] Grimson, W.E.L. and T. Lozano-Pérez. Localizing Overlapping Parts by Searching the Interpretation Tree. *IEEE Trans on PAMI*, PAMI-9(4):469–482, Jul 1987.

[39] Hanson, A.R. and E.M. Riseman. *VISIONS: A Computer System for Interpreting Scenes.* Academic Press, New York NY, 1978.

[40] Haralick, R.M. and L.G. Shapiro. *Computer and Robot Vision – Volume II.* Addison-Wesley, Reading MS, 1993.

[41] Herman, M. and T. Kanade. Incremental Reconstruction of 3D Scenes from Multiple, Complex Images. *Artificial Intelligence*, 30:289–341, 1986.

[42] Hertz, J., A. Krogh and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City CA, 1991.

[43] Hervé, J.-Y., R. Sharma and P. Cucka. Toward Robust Vision-Based Control: Hand/Eye Coordination Without Calibration. In *Proc of the 1991 Int'l Symp on Intelligent Control*, pages 457–462, Arlington VA, Aug 1991. IEEE Computer Society Press.

[44] Hwang, W.S. Performing Temporal Action with a Hand, Eye and Head System Using SHOSLIF Approach. Master's thesis, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1996.

[45] Hwang, W.S., S.J. Howden and J.J. Weng. Performing Temporal Action with a Hand-Eye System Using the SHOSLIF Approach. In *Int'l Conf on Pattern Recognition*, Aug 1996. To Be Published.

[46] Hwang, W.S., S.J. Howden and J.J. Weng. Performing Temporal Action with a Hand-Eye System Using the SHOSLIF Approach. Technical Report CPS-96-17, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1996.

[47] Ikeuchi, K. Generating an Interpretation Tree from a CAD Model for 3D-Object Recognition in Bin-Picking Tasks. *Int'l Journal of Computer Vision*, 1:145–165, 1987.

[48] Jain, A.K. and R. Hoffman. Evidence-Based Recognition of 3-D Objects. *IEEE Trans on PAMI*, 10(6):783–802, Nov 1988.

[49] Jain, A.K. and R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall Advanced Reference Series, Englewood Cliffs NJ, 1988.

[50] Jain, A.K. and R.C. Dubes. *Pattern Recognition.* The Benjamin/Cummings Publishing Company, Inc., East Lansing MI, 1994. Course Packet.

[51] Johnsonbaugh, R. *Discrete Maathematics.* Macmillan Publishing Company, New York NY, 1984.

[52] Katić, D. and M. Vukobratović. Decomposed Connectionist Architecture for Fast and Robust Learning of Robot Dynamics. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 2064–2069, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[53] Kieffer, S., V. Morellas and M. Donath. Neural Network Learning of the Inverse Kinematic Relationships for a Robot Arm. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 2418–2425, Sacramento CA, Apr 1991. IEEE Computer Society Press.

[54] Kohonen, T. Self-organized formation of topologically correct feature maps. In J.A. Anderson and E. Rosenfeld, editor, *Neurocomputing*, pages 509–521. Springer-Verlag, Berlin Germany, 1988.

[55] Kohonen, T. The Self-Organizing Map. *Proc of the IEEE*, 78(9):1464–1480, Sep 1990.

[56] Kosecka, J. and R. Bajcsy. Cooperation of Visually Guided Behaviors. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 502–506, Los Alamitos CA, May 1993. IEEE Computer Society Press.

[57] Kuperstein, M. Adaptive Visual-Motor Coordination in Mulitjoint Robots Using Parallel Architecture. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, volume 3, pages 1595–1602, Raleigh NC, Apr 1987. IEEE Computer Society Press.

[58] Lamdan, Y. and H.J. Wolfson. Geometric Hashing: A General and Efficient Model-Based Recognition Scheme. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 238–249, Tarpon Springs FL, Dec 1988. IEEE Computer Society Press.

[59] Li, G. and J.J. Weng. The SHOSLIF and Some Properties. Technical Report CPS-96-26, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1996.

[60] Liang, C.-C., F.-Y. Liao and W.-C. Lin. ART-1 Neural Network for Reducing Search Space in 3-D Object Recognition Using Multiple Views. In *Int'l J Conf on Neural Networks*, pages 1–21. IEEE Press, 1990.

[61] Lloyd, J. and V. Hayward. *Multi-RCCL User's Guide.* McGill Research Centre for Intelligent Machines, McGilll University, Montreal, Quebec, Canada, July 1992.

[62] Lloyd, J. and V. Hayward. *RCCL/RCI System Overview.* McGill Research Centre for Intelligent Machines, McGilll University, Montreal, Quebec, Canada, July 1992.

[63] Lowe, D.G. Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31:355–395, 1987.

[64] Luger, G.F. and W.A. Stubblefield. *Artificial Intelligence and the Design of Expert Systems.* The Benjamin/Cummings Publishing Company, Inc., Redwood City CA, 1989.

[65] Martinetz, R.M., H.J. Ritter and K.J. Schulten. 3D-Neural-Net for Learning Visuomotor-Coordination of a Robot Arm. In *Int'l J Conf on Neural Networks*, volume 2, pages 351–356, Jun 1989.

[66] Martinetz, R.M., H.J. Ritter and K.J. Schulten. Three-Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm. *IEEE Trans on Neural Networks*, 1(1):131–136, Mar 1990.

[67] McKeown, D.M., A.H. Wilson, Jr. and J. McDermott. Rule-Based Interpretation of Aerial Imagery. *IEEE Trans on PAMI*, PAMI-7:570–585, 1985.

[68] Murray, D.W., P.F. McLauchlan, I.D. Reid and P.M. Sharkey. Reactions to Peripheral Image Motion Using a Head/Eye Platform. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 403–411, Los Alamitos CA, May 1993. IEEE Computer Society Press.

[69] Papanikolopoulos, N.P., P.K. Khosla and T. Kanade. Adaptive Robotic Visual Tracking. In *Proc of the 1991 American Control Conference*, pages 962–967, Jun 1991.

[70] Papanikolopoulos, N.P., P.K. Khosla and T. Kanade. Visual Tracking of a Moving Target by a Camera Mounted on a Robot: A Combination of Control and Vision. *IEEE Trans on Robotics and Automation*, 9(1):14–35, Feb 1993.

[71] Pennisi, E. Tilting Against a Major Theory of Movement Control. *Science*, 272:32–33, Apr 1996.

[72] Pomerleau, D.A. ALVINN: An Autonomous Land Vehicle in a Neural Network. In Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 305–313. Margan Kaufman, 1989.

[73] Redding, G.M. and B. Wallace. Effects on Prism Adaptation of Duration and Timing of Visual Feedback During Pointing. *Journal of Motor Behavior*, 22(2):209–224, Jun 1990.

[74] Redding, G.M. and B. Wallace. Effects of Pointing Rate and Availability of Visual Feedback on Visual and Proprioceptive Components of Prism Adaptation. *Journal of Motor Behavior*, 24(3):226–237, Sep 1992.

[75] Salganicoff, M. and R.K. Bajcsy. Robotic Sensorimotor Learning in Continuous Domains. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 2045–2050, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[76] Schrott, A. Feature-Based Camera-Guided Grasping by an Eye-in-Hand Robot. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1832–1837, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[77] Seraji, H., M.K. Long and T.S. Lee. Motion Control of 7-DOF Arms: The Configuration Control Approach. *IEEE Trans on Robotics and Automation*, 9(2):125–139, Apr 1993.

[78] Sharma, R. and Y. Aloimonos. Visual Motion Analysis Under Interceptive Behavior. In *Proc of the 1992 Conf on Computer Vision and Pattern Recognition*, pages 148–153, Champaign IL, Jun 1992. IEEE Computer Society Press.

[79] Sharma, R., J.-Y. Hervé and P. Cucka. Dynamic Robot Manipulation Using Visual Tracking. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, volume 2, pages 1844–1849, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[80] Stein, F. and G. Medioni. Structural Indexing: Efficient 3-D Object Recognition. *IEEE Trans on PAMI*, 14(2):125–145, Feb 1992.

[81] Stelmaszyk, P., H. Ishiguro and S. Tsuji. Mobile Robot Navigation by an Active Control of the Vision System. In *Int'l J Confs on Artificial Intelligence*, pages 1241–1246. ijcaii, Aug 1991.

[82] Swets, D.L. *The Self-Organizing Hierarchical Optimal Subspace Learning and Inference Framework for Object Recognition.* PhD thesis, Michigan State University, 1996.

[83] Swets, D.L. and J.J. Weng. Hierarchical Discriminant Analysis for Image Retrieval. Technical Report CPS-96-17, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1996.

[84] Tsai, R.Y. and R.K. Lenz. A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration. *IEEE Trans on Robotics and Automation*, 5(3):345–358, Jun 1989.

[85] Tuceryan, M., A.K. Jain and Y. Lee. Texture Segmentation Using Voronoi Polygons. In *Proc of the 1988 Conf on Computer Vision and Pattern Recognition*, pages 94–99, Washington, DC, Jun 1988. Computer Society Press.

[86] Walter, J.A. and K.J. Schulten. Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot. *IEEE Trans on Neural Networks*, 4(1):86–95, Jan 1993.

[87] Webster, R.G. The Relationship Between Cognitive, Motor-Kinesthetic, and Oculomotor Adaptation. *Perception & Psychophysics*, 6(1):33–38, 1969.

[88] Weng, J., P. Cohen and M. Herniou. Camera Calibration with Distortion Models and Accuracy Evaluation. *IEEE Trans on PAMI*, 14(10):965–980, Oct 1992.

[89] Weng, J.J. On Comprehensive Visual Learning. In *Proc of the NSF/ARPA Workshop on Performance vs. Methodology in Computer Vision*, pages 152–166, Seattle WA, Jun 1994.

[90] Weng, J.J. and S. Chen. SHOSLIF Convergence Properties and MDF Version of SHOSLIF-N. Technical Report CPS-95-22, Michigan State University, Department of Computer Science, East Lansing, MI 48824, 1995.

[91] Weng, J.J., N. Ahuja and T.S. Huang. Learning Recognition and Segmentation of 3-D Ojbects from 2-D Images. In *Proc of the IEEE Int'l Conf on Computer Vision*, Germany, May 1993. IEEE Computer Society Press.

[92] Weng, J.J., T.S. Huang and N. Ahuja. Cresceptron: A Self-Organizing Neural Network Which Grows Adaptively. In *Int'l J Conf on Neural Networks*, Baltimore MY, Jun 1992. IEEE Computer Society Press.

[93] Wijesoma, S.W., D.F.H. Wolfe and F.J. Richards. Eye-to-Hand Coordination for Vision-Guided Robot Control Applications. *Int'l Journal of Robotics Research*, 12(1):65–78, Feb 1993.

[94] Zheng, J.U., Q. Chen and S. Tsuji. Active Camera Guided Manipulation. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 632–638, Sacramento CA, Apr 1991. IEEE Computer Society Press.

[95] Zhu, W.H. and M.C. Leu. Optimal Trajectory Planning of Robotic Manipulators by Cell State space Approach. In *Robotics Research*, pages 233–239, New York NY, Dec 1989. The American Society of Mechanical Engineers.

[96] Zomaya, A.Y., M.E. Suddaby and A.S. Morris. Direct Neuro-Adaptive Control of Robot Manipulators. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1902–1907, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[97] Zurada, J.M. *Introduction to Artificial Neural Systems.* West, St Paul MN, 1992.

MICHIGAN STATE UNIV. LIBRARIES
31293014137719