

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 1995		3. REPORT TYPE AND DATES COVERED Final Technical Report 01 Jun 92 to 31 Oct 95
4. TITLE AND SUBTITLE EXPERIMENTS IN NEURAL-NETWORK CONTROL OF A FREE-FLYING SPACE ROBOT			5. FUNDING NUMBERS F49620-92-J-0329	
6. AUTHOR(S) Edward Wilson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Aerospace Robotics Laboratory Department of Aeronautics and Astronautics Stanford University Stanford, CA 94305			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NA 110 Duncan Ave, Suite B 115 Bolling AFB, DC 20332-6448			10. SPONSORING/MONITORING AGENCY REPORT NUMBER F49620-92-J-0329	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) <p>Four important generic issues are identified and addressed in some depth in this thesis as part of the development of an adaptive neural-network-based control system for an experimental free-flying space robot prototype. The first issue concerns the importance of true system-level design of the control system. A new hybrid strategy is developed here, in depth, for the beneficial integration of neural networks into the total control system. A second important issue in neural network control concerns incorporating a priori knowledge into the neural network. In many applications, it is possible to get a reasonably accurate controller using conventional means. If this prior information is used purposefully to provide a starting point for the optimizing capabilities of the neural network, it can provide much faster initial learning. In a step towards addressing this issue, a new generic "Fully-Connected Architecture" (FCA) is developed for use with backpropagation. A third issue is that neural networks are commonly trained using a gradient-based optimization method such as backpropagation; but many real-world systems have discrete-valued functions (DVF's) that do not permit gradient-based optimization. One example is the on-off thrusters that are common on spacecraft. A new technique is developed here that now extends backpropagation learning for use with DVF's. The fourth issue is that the speed of adaptation is often a limiting factor in the implementation of a neural-network control system. This issue has been strongly resolved in the research by drawing on the above new contributions.</p>				
14. SUBJECT TERMS			15. NUMBER OF PAGES 136	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	



**Department of AERONAUTICS and ASTRONAUTICS
STANFORD UNIVERSITY**

SUDAAR 666

**EXPERIMENTS IN NEURAL-NETWORK CONTROL
OF A FREE-FLYING SPACE ROBOT**

Edward Wilson

*Aerospace Robotics Laboratory
Department of Aeronautics and Astronautics
STANFORD UNIVERSITY
Stanford, California 94305*

Research supported by
NASA Coop-Agreement NCC 2-333-S18
Department of the Air Force Grant F49620-92-J-0329

March 1995

19971003 016

DTIC QUALITY INSPECTED 5

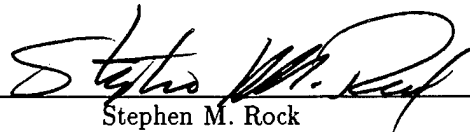
EXPERIMENTS IN NEURAL-NETWORK CONTROL
OF A FREE-FLYING SPACE ROBOT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Edward Wilson
March 1995

Copyright © 1995 by Edward Wilson
All Rights Reserved.

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



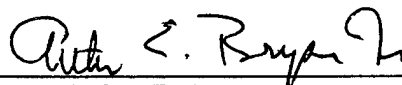
Stephen M. Rock
Department of Aeronautics and Astronautics
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Robert H. Cannon, Jr.
Department of Aeronautics and Astronautics

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Arthur E. Bryson, Jr.
Department of Aeronautics and Astronautics
Department of Mechanical Engineering

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Bernard Widrow
Department of Electrical Engineering

Approved for the University Committee on Graduate Studies:

2

Abstract

Because of their capabilities for adaptation, nonlinear function approximation, and parallel hardware implementation, neural networks have proven to be well-suited for some important control applications.

However, several important issues are present in many real-world neural-network control applications that have not yet been addressed effectively in the literature. Four of these important generic issues are identified and addressed in some depth in this thesis as part of the development of an adaptive neural-network-based control system for an experimental free-flying space robot prototype.

The first issue concerns the importance of true system-level design of the control system. A new hybrid strategy is developed here, in depth, for the beneficial integration of neural networks into the total control system. The basic philosophy is to borrow heavily from conventional control theory, and use the neural network as a key subsystem just where its nonlinear, adaptive, and parallel processing benefits outweigh the associated costs.

A second important issue in neural network control concerns incorporating *a priori* knowledge into the neural network. In many applications, it is possible to get a reasonably accurate controller using conventional means. If this prior information is used purposefully to provide a starting point for the optimizing capabilities of the neural network, it can provide much faster initial learning. In a step towards addressing this issue, a new generic "Fully-Connected Architecture" (FCA) is developed for use with backpropagation. This FCA has functionality beyond that of a layered network, and these capabilities are shown to be particularly beneficial for control tasks. For example, they provide the new ability to pre-program the neural network directly with a linear approximate controller.

A third issue is that neural networks are commonly trained using a gradient-based optimization method such as backpropagation; but many real-world systems have discrete-valued functions (DVF's) that do not permit gradient-based optimization. One example is the on-off thrusters that are common on spacecraft. A new technique is developed here that now extends backpropagation learning for use with DVFs. Moreover, the modification to backpropagation is small, requiring (1) replacement of the DVFs with continuously differentiable approximations, and (2) injection of noise on the forward sweep. This algorithm is applicable generically whenever a gradient-based optimization is used for systems with discrete-valued functions. It is applied here to the control problem using on-off thrusters,

as well as for training neural networks built with hard-limiting neurons (signums instead of sigmoids).

The fourth issue is that the speed of adaptation is often a limiting factor in the implementation of a neural-network control system. This issue has been strongly resolved in this research by drawing on the above new contributions: the FCA and an automatic growing of the network combine to allow rapid adaptation in an experimental demonstration on a 2-D laboratory model of a free-flying space robot. The neural-network controller adapts in real time to account for multiple destabilizing thruster failures. Stability is restored within 5 seconds, and near-optimal performance is achieved within 2 minutes. This performance is obtained despite the implementation on a serial microprocessor; implementation on parallel-processing hardware would provide dramatically faster performance.

To Susan

2

Acknowledgements

I have learned much from the professors on my reading committee. Each one is a distinguished leader in his field of work, and we are all lucky that Stanford University has succeeded in providing a focal point for attracting such excellent professors. The academic freedom Stanford provides is also appreciated – this has allowed me to broaden my education significantly, outside of the Mechanical Engineering department. I would also like to thank Stanford for providing the excellent weather we have had during my time here, as well as good proximity to the water and the mountains.

My principal adviser, Professor Stephen M. Rock, has provided excellent invaluable advice on how to organize and write a thesis, and how to present research work in general. He also helped with advice on getting started in windsurfing and beer brewing, both of which have provided many pleasant distractions on the long road to my Ph.D.

Thanks to Professor Robert H. Cannon, Jr. for providing an excellent research environment, giving me the freedom to pursue my research interests, and helping me keep my research on-track. This thesis owes much to the many hours he spent meticulously reviewing every aspect of the writing and content.

It was rewarding to learn optimal control theory from one of the principal inventors of the field, Professor Arthur E. Bryson. His dedication to academic excellence motivates me to strive for excellence in my work as well.

Professor Daniel B. DeBra has served as my Mechanical Engineering Program Advisor since I arrived at Stanford. He has been a supportive friend and impartial advisor for my research and career decisions since I have been here.

Hughes Aircraft Company and the Air Force Office of Scientific Research awarded me fellowships to support most of my Ph.D. studies. The National Aeronautics and Space Administration provided the bulk of the research equipment used, and one year of support as a research assistant. More importantly, NASA and the AFOSR have contributed to the lab over many years, allowing Professors Cannon and Rock to build the fine research program that allows me and other students to earn our Ph.D.'s.

I have enjoyed spending my time here at the Aerospace Robotics Laboratory, as my fellow students here have been both good friends and stimulating co-workers. Tim McLain was my lab partner for most of the classes I took here. I learned much from him then,

and have continued to enjoy discussing research with him. The patience, hard work, and integrity that is evident in his research attitude have motivated me in my work.

Marc Ullman and Stan Schneider contributed so much to the lab with their research several years ago that I sometimes wonder where we would be without them. Marc, along with Ross Koningstein, Vince Chen, Gad Shelef, and Godwin Zhang, designed and built the robot used in these experiments. I wish my car were as reliable as this fantastic piece of hardware. Stan, along with Vince, Marc, and Gerardo Pardo-Castellote, provided the excellent software development tool, ControlShell, used by every research project in the lab.

Speaking of computers, the patience of our lab computer experts, Steve Ims, Dave Meer, Denny Morse, and Kurt Zimmerman has been tested many times by my attempts to understand computers from a mechanical-engineering viewpoint. They have provided invaluable help, and I owe a special debt of gratitude to Kurt and Denny for helping to take the "hell" out of "ControlShell."

Thanks to Larry Alder, Brian Andersen, Bill Ballhaus, Bill Dickson, Steve Fleischer, Gordon Hunt, Warren Jasper, Kortney Leabourne, Jane Lintott, Rick Marks, David Miles, Eric Miles, Celia Oakley, Larry Pfeffer, Jeff Russakow, Glen Sapilewski, Diana Snyder, HD Stevens, Chris Uhlik, Rob Vasquez, Howard Wang, and Roberto Zanutta for their friendship and for contributing to the enjoyable research environment we have at the ARL.

My parents, Drs. Keith and Diana Wilson, have provided encouragement, education, and guidance that has been a strong influence in contributing both to my professional achievements and to my enjoyment of life.

Finally, my fiancée, Susan Fukuhara, has provided un-wavering support of me throughout my Ph.D. Her belief in me kept me going when things were uncertain or not going as planned. She also helped by always letting me know when I was working too hard and needed to put things back into perspective. Her love, patience, and understanding have been tested many times, and are always appreciated. I dedicate this thesis to her.

Contents

Abstract	v
Acknowledgements	ix
List of Figures	xv
1 Introduction	1
1.1 Motivation	1
1.2 Research Issues	2
1.3 Contributions	4
1.4 Background on Neural Networks	6
1.4.1 Biological Motivation	6
1.4.2 History of Neural Networks	8
1.4.3 Different Types of Neural Networks	9
1.5 Reader's Guide	11
2 Robot Control Application	13
2.1 Experimental System	16
2.1.1 Thrusters	17
2.1.2 Accelerometers, Angular-Rate Sensor	21
2.2 Thruster Mapping	27
2.2.1 Problem Definition	27
2.2.2 Cost Function	28
2.3 Solution Strategy, Mapping Methods	31
2.3.1 Thruster Mapping by Exhaustive Search	33
2.3.2 Direct Training of a Neural-Network Thruster Mapper	36

2.3.3	Indirect Training of a Neural-Network Thruster Mapper	37
2.4	Summary	38
3	Control System Overview	39
3.1	Control System Structure	40
3.1.1	Control System Design Considerations	41
3.1.2	Indirect Adaptive Control System	42
3.2	Cost/Benefit Analysis	43
3.2.1	Benefits of Neural Networks	44
3.2.2	Costs Associated With the Use of Neural Networks	45
3.3	Criteria For Valuable Application of Neural Networks	46
4	Fully-Connected Architecture	49
4.1	Background	51
4.2	Comparison with a Layered Network	52
4.2.1	Feedthrough Weights	53
4.2.2	Value of Cross-Talk Connections	57
4.2.3	Hidden-Neuron Interconnections	58
4.2.4	Learning Performance: FCA vs. Layered	60
4.3	Architecture Selection to Avoid Overfitting	62
4.3.1	Overfitting	62
4.3.2	Systematic Complexity Control	62
4.3.3	Other Complexity-Control Methods	67
4.3.4	Automatic Growing of the Network	68
4.4	Summary of Implementation Issues for the FCA	69
5	Gradient-Based Optimization for Discrete Systems	71
5.1	Problem Statement	72
5.2	Related Research	72
5.2.1	History of Neural-Network Training With Smooth Activation Functions	73
5.2.2	Neural-Network Training With Discrete-Valued Activation Functions	73
5.3	A New Training Algorithm: Approximation With Noisy Sigmoids	74
5.4	Intuitive Explanation	75
5.5	Application Considerations, Extensions	76

5.5.1	Selection of Noise Level	76
5.5.2	Discrete-Valued Functions Other Than Bi-Level Signums	78
5.5.3	Batch Learning	78
5.5.4	Optimization of Discrete-Valued Parameters	79
5.6	Application to Training Multi-Layer Signum Networks	79
5.7	Application to Thruster-Mapping Problem	81
5.8	Other Uses of Noise in Related Problems	84
5.9	Summary	85
6	Experimental Demonstration of Reconfiguration	87
6.1	System Overview	87
6.2	Trajectory-Following Performance	93
6.2.1	Trajectory-Following Performance: One Degree of Freedom	93
6.2.2	Trajectory-Following Performance: Three Degrees of Freedom	95
6.2.3	Trajectory-Following Performance: Summary	97
6.3	Control Reconfiguration Problem Definition	97
6.4	Identification of Failures	99
6.4.1	Identification Summary	99
6.4.2	Failure Detection	100
6.4.3	System Identification	102
6.4.4	Artificial Excitation	102
6.5	Neural-Network Training	104
6.6	Rapid Reconfiguration	105
6.7	Experimental Results of Reconfiguration	107
6.8	Summary of Experimental Results	111
7	Conclusions	113
7.1	Summary	113
7.1.1	System-Level Design Approach: The Superiority of Hybrid Control	113
7.1.2	Quick Adaptation – FCA	114
7.1.3	Gradient-Based Optimization for DVFs – Noisy Sigmoids	115
7.1.4	Experimental Demonstration of Reconfigurable Control System	116
7.2	Recommendations for Future Work	117
7.2.1	Integration of Neural-Network and Conventional Control	117

7.2.2	Optimal (Hybrid) Combination of Neural Networks with Conventional Control: FCA	118
7.2.3	Gradient-Based Optimization for DVFs	119
7.2.4	Thruster Mapping	120
A	Thruster-Mapping Cost Function	123
B	Accelerometer Specifications	125

List of Figures

2.1	Stanford Free-Flying Space Robot	14
2.2	Example Failure Mode	15
2.3	Photograph of Thruster Assembly	18
2.4	Photograph of Two Thruster Assemblies	19
2.5	Thruster Failure Modes – Reduction in Thrust Level	20
2.6	Thruster Failure Modes – Change in Thrust Direction	21
2.7	Accelerometer Photograph	22
2.8	Accelerometer Circuit	23
2.9	Accelerometer Mounting Locations	25
2.10	Translational and Angular-Acceleration Signals	26
2.11	Thruster Mapping, Problem Definition	27
2.12	Robot-Base-Control Strategy	28
2.13	Thruster-Mapping Methods	32
2.14	Possible Force Vectors with Eight Symmetrical Thrusters	34
2.15	Possible Force Vectors after Symmetric Transformation	35
3.1	Reconfigurable Control System – Block Diagram	40
4.1	Extra Connections Available with FCA	50
4.2	Weight-Matrix Representation to Highlight Benefits of FCA	52
4.3	Network Connection Activity During Training	56
4.4	FCA Subsumes Any Feedforward Layered Network	58
4.5	Training Performance Comparison	61
4.6	Training History, Performance on Test and Training Data	63
4.7	Complexity-Cost Function	65
4.8	Complexity Cost Reduces Overfitting	68
5.1	Training Algorithm	75

5.2	Effect of Noise Level on Sigmoid Output Distribution	77
5.3	A Multi-Layer Signum Network, Seen at Run-Time and During Training . .	79
5.4	Training with Noisy Sigmoids of a Multi-Layer Signum Network, Artificial Training Set	80
5.5	Training a Multi-Layer Signum Network, Thruster Mapping	81
5.6	Thruster Mapping, Indirect Training Method	82
5.7	Results of Indirect Training, Two Differentiable Thruster Models	82
5.8	Results of Indirect Training, Noisy Tri-Level Sigmoid Thruster Model . . .	83
6.1	Reconfigurable Control System – Block Diagram	88
6.2	Thruster Mapper – Signal Flow	89
6.3	Acceleration Sensors – Signal Flow	91
6.4	Neural-Network Training – Signal Flow	92
6.5	Single-Degree-of-Freedom Trajectory Following, Experimental Results . . .	94
6.6	Multiple-Degree-of-Freedom Trajectory Following	95
6.7	Multiple-Degree-of-Freedom Trajectory Following, Experimental Results . .	96
6.8	Example Failure Mode	98
6.9	Identification Process – Signal Flow	99
6.10	Experimental Results of Reconfiguration	106
B.1	Systron Donner 4310A Accelerometer - Dimensions	126
B.2	Systron Donner 4310A Accelerometer - Accuracy Specifications	127
B.3	Systron Donner 4310A Accelerometer - Physical Specifications	128
B.4	Systron Donner 4310A Accelerometer - Environmental Specifications	128

Chapter 1

Introduction

This dissertation presents generic theoretical and experimental investigations into the use of neural networks for control. As a significant “challenge problem,” a free-flying space robot prototype equipped with on-off gas thrusters was controlled well, despite major thruster failures, by using a new, hybrid neural-network-based reconfigurable control system. This research was conducted at the Stanford University Aerospace Robotics Laboratory (ARL) at Stanford University from 1990 to 1994.

1.1 Motivation

Due to their capabilities for adaptation, nonlinear function approximation, and parallel hardware implementation, neural networks have proven to be well suited for control applications. They have been used successfully by engineers in the chemical-processing industries [12] [62], steel industry [16] [17] [47] [54], and semiconductor-processing industry [17], as well as a number of research applications [20] [24] [38] [64] [67]. In some cases their learning abilities and inherent nonlinear nature allow them to solve control problems and provide performance unmatched by conventional methods. In other cases their distributed nature and resulting computational power allow them to implement known solutions more quickly and robustly than conventional serial processors.

Neural networks derive their advantage in solving very complex problems from the emergent properties that come with the massive interconnection of simple processing units. With good training techniques, the networks are capable of implementing very complex behaviors. For example, neural networks may be used to implement arbitrary mappings of inputs to

outputs, such as from sensor signals to actuator commands in a control problem. Further, since the mapping can be taught indirectly, neural networks are especially attractive for poorly understood systems – they can generalize from training inputs and then respond by interpolation in untaught situations.

Due to the distributed nature of the processing and their adaptive capability, networks are often robust to internal component failures. Even without re-training, the distributed processing gives the network the ability to withstand failure of several neurons without significant impact on the functionality. In addition to this, if on-line re-training is used, the remaining processors can adapt to account for the failure. Robustness is also contributed by the network's ability to adapt to changes in the environment, plant, performance criteria, etc.

These features of neural networks make them particularly attractive for control applications. Several of these features will prove useful in the control application presented here.

The central question is when – and how – will the incorporation of neural network components provide a clear, cost-effective advantage in real-time control?

One central goal of this research, then, is to study the use of neural networks for control, and to determine the characteristics of control applications that can benefit from the application of neural networks. In certain cases, the *merging* of neural-network technology with control-systems engineering can lead to the development of highly capable control systems. Much neural-network theory and control theory already exists such that significant advances in control capability could be produced simply through their astute *integration*.

1.2 Research Issues

Neural networks have proven themselves valuable in a number of control applications. See for example [20] [24] [54] [64]. There are, however, four important issues, that are often most germane in a real-world control application, that have not yet been addressed effectively in the neural-network literature:

1. For a given control need, should a neural network be used?
 - Does using a neural network provide a clear advantage over not doing so?
 - If it does, then to achieve that advantage optimally, just where in the control system should the neural network be used; and where should it not?

2. *A priori* knowledge is often available in the form of models of the system's key components and a preliminary control design (e.g. provided by "conventional" control design techniques). Is it possible to use this *a priori* information to improve greatly the performance (i.e. better initial performance, final convergence to a better solution) that the neural network can then enable?
3. Many control applications involve the use of discrete-valued devices. For example, thrusters often operate "on-off" rather than with analog-valued outputs. This presents a problem for backpropagation learning, since these discrete-valued functions are not continuously differentiable. Is it possible to modify backpropagation to accommodate the discrete-valued functions?
4. Speed of learning is very often important in real-time control applications. It is generally accepted that neural networks can run quickly during implementation (i.e. once the weights have been selected) due to the availability of parallel hardware; but the speed of learning (i.e. finding the weight values) is a separate, very critical issue. Can backpropagation-based learning be made fast enough to be feasible for rapid on-line adaptation?

A "challenge problem" was formulated to focus the study of these important issues: a reconfigurable neural-network-based adaptive control system was developed and experimentally demonstrated on a free-flying space robot prototype. In addressing this challenge problem, the issues were studied, neural-network developments were made, and a working reconfigurable control system was developed [69] [70] [71] [72] [73].

The experimental apparatus is shown in Figure 2.1. Specifically, the air-bearing-supported robot's position and attitude are controlled with eight on-off gas thrusters. The task was this: after the random, severe mechanical failure of a number of these thrusters, identify the new thruster-system characteristics, and reconfigure the control system to regain stability and near-optimal performance. This challenge problem is interesting not only for its practical applicability to space operations *per se*, but also – and even more pervasively – as an application that raises and focuses on several important fundamental generic issues in neural-network control.

The challenge problem addresses the first issue, since it is a fairly complex, yet realistic control problem. Also, the excellent experimental performance of a pre-existing conventional control approach is available for comparison; this is valuable for evaluating the performance

trade-offs between neural and conventional approaches. The application also helps to motivate the second issue, a desire to make use of *a priori* knowledge: an approximate solution can be calculated quickly before neural-network training begins. The desire to use this *a priori* information to accelerate learning is especially present here due to the need for rapid reconfiguration. The existence of on-off thrusters requires the development of a learning method to deal with discrete-valued functions, highlighting the third issue. Finally, the speed-of-learning issue is relevant, since stability must be regained quickly due to the limits enforced by the experimental implementation (i.e. the granite table is of limited size).

1.3 Contributions

In addressing the research issues outlined above, the research reported in this thesis makes the following contributions to the fields of neural networks, automatic control, and robotics:

1. An adaptive neural-network-based thruster control system for a free-flying space robot is developed. This highly nonlinear complex control problem was solved in a very new way: by using a combination of conventional and neural network approaches, resulting in a "hybrid" control system. The balance between neural and conventional approaches will, in general, vary from one application to another. At issue is how to determine the correct balance on an application-by-application basis. To address this issue, systematic evaluation criteria have been proposed and demonstrated to aid in the system-level design.
2. A new "Fully-Connected Architecture" is developed for neural network control. This architecture is a generalization of the standard layered neural-network architecture. The value of the extra connections it offers is studied. Of particular importance for control, this new architecture allows for *direct pre-programming* of prior-known linear solutions. This benefit is used in the robotic application to reduce dramatically the time required for adaptation: a linear approximate controller is quickly calculated and implemented before training begins. The major hurdle for successful use of this architecture, excessive complexity, is addressed by the implementation of a systematic complexity-control method that manages the extra connections.

There are a number of possible advantages to using prior information. Since the network begins training with a reasonably good solution, initial performance is good;

and a better solution may result due to the better starting point for the nonlinear optimization. It also serves as a bridge to conventional control techniques. Optimizing the network from a starting point that is a direct emulation of a conventional controller may facilitate valuable understanding of what the network is doing.

3. A new algorithm was devised that now permits gradient-based optimization of systems with discrete-valued functions (DVF's). Gradient-based optimization of systems with DVF's is difficult because the gradient of the DVF is zero everywhere, except at the transitions, where it is undefined. The new algorithm works by forming a smooth, continuous approximation to the DVF, and then adding noise during training. It has been applied to a number of different applications; and each time, the value of noise injection is clearly demonstrated. Although originally developed for application to the on-off thruster control problem, this algorithm for gradient-based optimization for DVF's is broadly applicable. Three applications are:

- Training a neural network control system equipped with on-off actuators.
- Training neural networks built with hard-limiting neurons.
- Design optimization with discrete-valued design options (proposed, not yet implemented).

4. An experimental demonstration was performed, where the neural-network-based control system reconfigured itself rapidly in response to multiple, major, destabilizing thruster failures. Stability is restored within 5 seconds, and near-optimal performance is achieved within 2 minutes. This performance is obtained despite the implementation on a serial microprocessor; implementation on parallel-processing hardware would provide dramatically faster performance.

The experimental demonstration pulls together each of the above contributions: #1 led to the efficient system-level (hybrid) design that combines optimally the benefits of both conventional control and neural networks; #2 resulted in rapid recovery of stability, through the direct infusion of a linear approximate controller; #3 allowed the use of gradient-based optimization with this control problem. The ability to use gradient information at all dramatically improved the rate of adaptation (beyond what non-gradient-based methods could provide). These advances, combined with an

automatic network-growing method, increase the *speed* of the learning process to a point where it becomes a viable alternative for on-line adaptive control.

Each contribution is addressed individually and presented in Chapters 3 through 6 of this thesis.

1.4 Background on Neural Networks

A brief background on neural networks is presented here to familiarize the reader with the biological motivation, history, and mathematical foundation of artificial neural networks. More complete overviews may be found in [22] [29] [67].

1.4.1 Biological Motivation

Artificial Neural Networks are named after and motivated by the biological neural networks that allow phenomenal computing performance in humans and other living organisms. Despite the relatively slow computation rate of the individual human neuron, the human brain's sound and image recognition capabilities far exceed those of current computers. The naturally fault tolerant and adaptive nature of the parallel distributed processing model (both biological and artificial) make it well suited for ambiguous tasks or uncertain environments.

The following lists highlight the different characteristics and capabilities of computers and the human brain.

- Conventional Digital Computers:
 - Sequential instructions
 - Digital
 - Address memory
 - Speed measured in nanoseconds
 - Highly accurate
 - Not-necessarily fault tolerant

- Human Brain:
 - Massively parallel architecture
 - Analog
 - Associative Memory
 - Neuron response times on order of 1 millisecond
 - Less accurate than computers
 - Fault tolerant, naturally adaptive

Currently, conventional digital computers work by implementing a series of instructions, and provide highly accurate arithmetic and logic computations in cycle times on the order of nanoseconds.

Biological neural networks are difficult to study, and not completely understood. What is known is that computations are carried out in parallel, with thousands to billions (e.g. the human brain has roughly 10^{10} processing units (neurons) and 10^{14} connections (synapses)) of low-precision processors operating with rise times on the order of milliseconds. The neurons communicate by sending 100 mV impulses to other neurons. Since the magnitude of these pulses is fixed, information is encoded in the frequency of firing. By comparison, modern microprocessors have typically 10^6 to 10^7 transistors, but only one to four computations are executed at a time. This lack of parallelism is offset by the fast processing time on the order of 1-20 nanoseconds (50 MHz to 1 GHz clock rate).

Despite the slow processing of each individual neuron, the massive parallelism results in certain computing capabilities that are impossible with conventional sequential digital processors. Some of these capabilities that are most-nearly reachable with conventional processors are: vision processing, sound processing, pattern recognition, adaptive control, and planning. The key idea is that designing a computer with some attributes of the biological neural network, such as parallel computation and adaptive capability, may yield greater success in these areas than trying to push incrementally the state of the art in conventional computing hardware and algorithms.

The potential benefits of a parallel-distributed-processing approach create an incentive to cast a problem into a form that can use the computational capabilities of this architecture.

1.4.2 History of Neural Networks

When people began attempting sustained heavier-than-air flight, the first thought was to build an aircraft modelled after birds. Early ornithopters attempted to reproduce the flapping-wing motions that allow birds to fly. These designs failed. The first successful solution, by the Wright brothers in 1903, used instead a fixed wing to produce lift, with a wing-warping method to control the lift of each wing (similar to birds), but with an internal-combustion-engine-powered propeller for thrust. Most aircraft today resemble birds only slightly, in that they have a wing on each side of the fuselage, and the control system sits up front with the vision sensors. However, the propulsion system, control system, materials, etc. are very different. Using nature as a motivation was useful; but it has been important to incorporate the best engineering available, and not rigidly follow the biological model.

Similarly, one of the earliest ideas for building a computer was that it should be modeled after the human brain. Once biologists began to understand the basics about how the brain works on a microscopic level, early neural-network researchers modelled these neurons, and designed artificial neural networks.

However, before they understood how the brain worked, artificial computing systems had been built in the form of mechanical adding machines. These produced precise computations, one instruction at a time. As these mechanical linkages were replaced with electrical circuits, vacuum tubes, transistors, and finally an integrated circuit consisting of many transistors, the computational performance has increased dramatically, but the highly accurate and serial attributes have persisted. This development of conventional serial processors has continued in parallel with the development of neurally-inspired processors.

A sequence of major developments in neurally-inspired computing follows.

In 1943, McCulloch and Pitts modelled the neuron as a simple threshold device, and analyzed the computational capabilities of networks of these functions.

In 1948, Hebb proposed a way for neurons to change the effect they had on other neurons, forming the foundation for a model of learning.

In 1957, Kolmogorov's Theorem laid the mathematical foundation for neural networks. This theorem proved that networks of simple neuron-like processors are able to produce arbitrarily complex functions of their inputs [28]. This existence proof is described again in Chapter 4.

Around 1960, Rosenblatt invented the Perceptron, a simple neuron with binary output. An important feature of the Perceptron is the simple learning rule that is guaranteed to

converge to a solution, if one exists [43]. The functionality of the Perceptron is limited, as discussed again in detail in Chapter 5.

About the same time, Widrow and Hoff invented the LMS algorithm for training binary-output neurons [18] [67]. This algorithm was later applied extensively to adaptive filtering and control [68], and is the foundation of the backpropagation algorithm.

In 1969, Minsky and Papert proved the limitations of the Perceptron: 1. some input-output mappings are impossible (e.g. XOR ¹) with a single hidden layer, and 2. the number of Perceptrons (neurons) required grows faster than exponentially with an increase in problem complexity [32] [33].

In 1974, Werbos developed the backpropagation algorithm as part of his Ph.D. thesis in Economics [60]. Its discovery was not widely noticed until Rumelhart's publication in 1986 [46]. The backpropagation algorithm will be described again in Chapter 5.

In 1982, Hopfield developed networks for associative memory.

In 1984, Hinton developed the Boltzmann Machine, a type of Hopfield Network that uses an annealing learning process governed by Boltzmann statistics.

In 1986, Rumelhart developed the backpropagation algorithm for training networks with multiple hidden layers [46]. The hidden-layer neurons use continuously differentiable sigmoid functions to permit the backpropagation of error signals used for training. This was an important discovery, as it removed the first limitation of the Perceptron model. Although Werbos is often credited with development of the backpropagation algorithm, Rumelhart is credited with the development of it as a useful tool for neural-network training. The backpropagation algorithm can be traced back further to Bryson's work in the 1960s with multistage optimization for dynamic systems [6].

The neural network field has expanded greatly since 1986, as many researchers have added capabilities to the backpropagation algorithm and experimented with applications.

1.4.3 Different Types of Neural Networks

Two major families of neural-network types exist: memory-based and function-based.

Function-based networks include feedforward sigmoidal networks (used in this thesis), feedforward radial basis function networks, recurrent networks, and Adaptive Resonance Theory (ART) networks [14]. These networks work by attempting to form a *function* that

¹the EXCLUSIVE OR logic function, $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 1$, $f(1,1) = 0$.

“fits” the data, or training cases they are presented. The hope is that this function forms a generalization of the training data, and the network will perform well on new data.

Function-based networks such as backpropagation-trained feedforward sigmoidal networks can be thought of as a means of data compression. For example, if 1000 bytes of data are used to train a network whose weights can be described with 100 bytes, the data has been compressed. As with all data-compression methods, this one relies on finding and taking advantage of regularities in the data set – *generalizing*. If regularities do exist and are exploited successfully, the original data set may be reproduced to a high level of accuracy.

Memory-based networks include the Cerebellar Model Articulation Controller (CMAC) [2] [3], nearest-neighbor interpolation, probabilistic neural networks [51] [52], and Kohonen Learning Vector Quantization [27]. Rather than learn a generalizing function of the data, these methods store examples of the training data in memory (for example, input-output training patterns). When presented with a new input training pattern, nearby training patterns are recalled from memory and the output is a function of these patterns (e.g. a linear interpolation among the 5 nearest neighbors). The specifics of the processing during learning and recall vary among the architectures listed here.

Briefly, the tradeoff is that memory-based approaches learn very quickly since they simply remember each training input, but the recall can be much slower, since the nearest neighbors must be found and then an interpolation performed to produce an output. Function-based approaches train more slowly, as they must compress the data into the functional format created by the network topology, but have very fast recall. Also, the distinction between these groups is sometimes blurred, as some systems involve a significant amount of processing, but may be built around stored training examples.

From a controls perspective, function-based networks fit better with existing methods, providing a generic nonlinear control element. Function-based neural-network controllers have been used in many applications [16] [17] [38] [47] [54] [62] [67]. However, CMAC [2] [3] is one example of a memory-based neural network that has been used extensively in control applications [23].

Feedforward neural networks² built with sigmoidal activation functions (as described above) were used exclusively in this research. Due to their general function-approximation capabilities, it was clear that they would work well for this application. However, another reason for their use here is that they have been used successfully for a wide variety of

²Those employing no internal feedback.

applications, and do appear to hold much promise for neural-network-control applications in particular. Other neural-network architectures exist of course, with different characteristics that may prove to offer advantages depending upon the application.

Radial-Basis-Function (RBF) networks are similar in that they have a feedforward structure, but the activation function is different. A sigmoid forms a *hyperplane* (i.e. a point in 1-D space, a line in 2-D space, a plane in 3-D space, a 3-dimensional hyperplane in 4-D space etc.) that separates the mapping space into high and low regions with a transition region near the hyperplane. A radial basis function (typically a Gaussian function) produces an activation near a certain *point* in space (i.e. a line segment in 1-D space, a circle in 2-D space, a sphere in 3-D space, etc.). Statistical or iterative methods may be used to choose the centers of these radial basis functions, and the weightings of these basis functions may be calculated directly or iteratively. These can be significant advantages over sigmoidal networks for some problems that happen to fit well with the functionality offered by these networks – namely one- or two-dimensional mappings. However, a major problem with RBF networks is that large numbers of hidden units are required for high-dimensional input spaces. This can be understood by considering how the relative volume of a sphere of influence of a RBF decreases as the dimensionality of the space increases. The problems extending to high-dimensional input spaces provided a motivation to avoid RBFs in the study of general neural-network-control issues in this research. However, for a low-dimensional input space (3-D for this application, 6-D for a 6-dof robot), RBFs may be viable.

These and other different neural-network architectures have many common aspects (e.g. the issues of overfitting or system-level design), and therefore, many conclusions of the research here will be directly applicable to these different architectures.

1.5 Reader's Guide

This chapter has served as an introduction to the research that is presented in this dissertation. The remainder of this thesis is organized as follows:

In Chapter 2, the experimental equipment (i.e. the robot) that provides the “challenge problem” is described in detail, and the particular thruster-mapping problem addressed is presented.

In Chapter 3, the generic issue of neural network value to specific control problems is addressed. Criteria are presented that will aid the control systems engineer in the system-level design of each given control system, deciding which segments, if any, it will be beneficial to implement with neural networks.

In Chapter 4, the new concept of "Fully-Connected Architecture" (FCA) is presented. It is used with backpropagation, and is shown to have greater functionality than a standard layered network. Benefits of the FCA are outlined, with emphasis on its advantageous applicability for control.

In Chapter 5, a new method is presented that allows backpropagation learning with systems containing discrete-valued (and therefore not continuously differentiable) functions (such as the on-off thrusters). This enabling method requires only simple modifications to standard backpropagation, and extends to multiple layers of hard-limiting neurons or to the FCA with no need for modification.

In Chapter 6, the reconfigurable neural control system for the free-flying robot is presented. It draws upon each of the developments detailed above. Its good experimental response to drastic destabilizing changes in the thrusters verifies rather dramatically the viability of each of the new contributions made.

Chapter 7 concludes this dissertation with a summary of results and recommendations for future research.

Chapter 2

Robot Control Application

The control task addressed in this research is the control of position and attitude of a free-flying space robot using on-off thrusters. The challenge presented here is to (abruptly) damage mechanically a number of thrusters, and then have the control system autonomously and rapidly reconfigure itself in real time, so as to maintain good control throughout. Moreover, some thruster failures are strongly destabilizing, which places high demands on the speed of recovery. The experimental system is shown in Figure 2.1, and an example thruster failure mode is shown in Figure 2.2.

Control using on-off thrusters is a complex, nonlinear problem that is important for real spacecraft [63], and the nonlinear and adaptive capabilities of neural networks make them attractive for this application.

The robot used here has in fact previously been successfully controlled without the use of neural networks [56]. However, the (conventional) method relies on geometric symmetries in the thruster layout and does not scale well to thruster controllers with higher-dimensionality in the input and output spaces. A neural-network-based approximation method does scale well to higher-dimensional thruster controllers, and does not rely upon geometric symmetries, so it provides a structure conducive to reconfigurable control. Additionally, the neural approach offers computational flexibility, since the network can be designed with the desired speed/accuracy trade-off. If implemented in parallel hardware, it can be made to be extremely fast.

This challenge problem was chosen as an aid in highlighting and defining some of the relevant issues in neural network control. It also serves to facilitate discussion and explanation of the neural network control developments made in the course of this research.

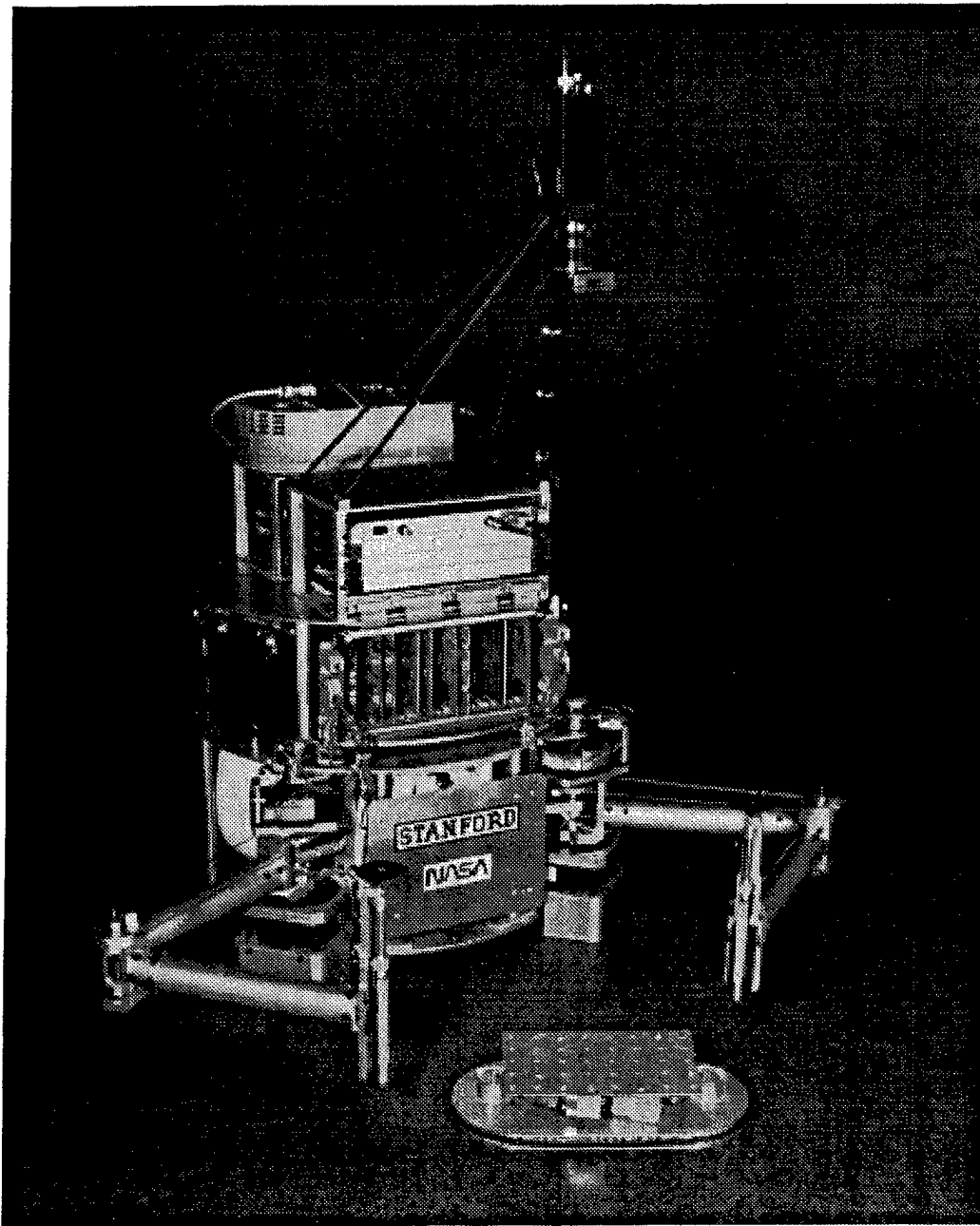


Figure 2.1: Stanford Free-Flying Space Robot

This highly autonomous mobile robot operates in the horizontal plane, using an air-cushion suspension to simulate the drag-free and zero-g characteristics of space. It is a fully self-contained planar laboratory-prototype of an autonomous free-flying space robot complete with on-board gas, thrusters, electrical power, multi-processor computer system, camera, wireless Ethernet data/communications link, and two cooperating manipulators.

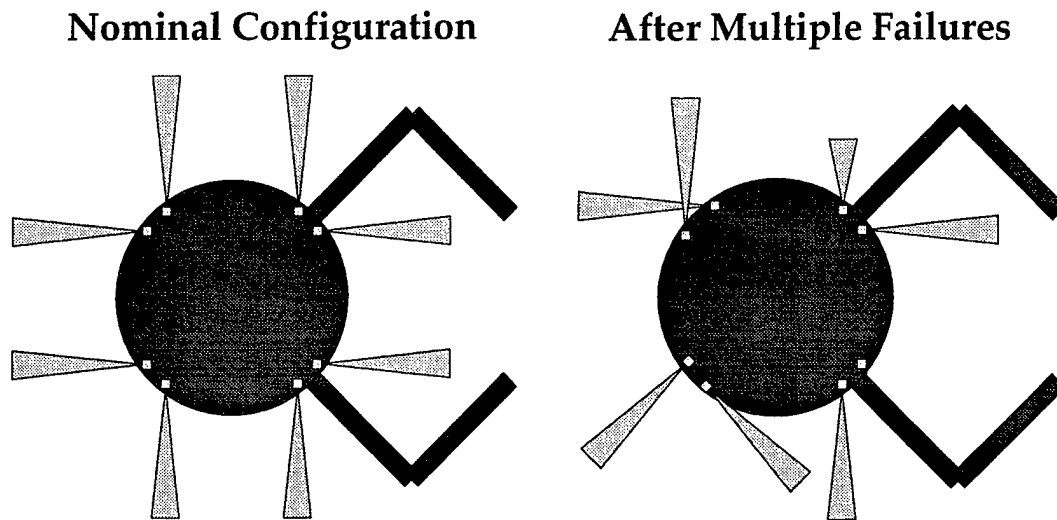


Figure 2.2: Example Failure Mode

Magnitude and direction of each of the eight thrusters is indicated by the length and direction of the lightly shaded triangles. Thruster failures were simulated mechanically with weaker thrusters and 90° and 45° elbows. Some of the elbows destabilize the robot by changing the sign of the thrust in the ψ direction.

The field of neural network control is vast, so the scope of this research has been limited to the use of feedforward neural networks¹ for a specific application. End-to-end development of a neural-network controller for a real, complex application highlights the truly important issues for this application, and these issues are relevant to other real-world applications. Where possible, information will be provided to allow extension of these developments to other applications.

Several specific attributes of the challenge problem common to other control applications include:

1. The complete control system is complex, involving the integration of several subsystems. Its level of complexity is similar to real-world control applications – it has requirements for high-level human interface, trajectory planning, system identification, and reconfiguration strategy, as well as low-level control.
2. Practical issues such as sensor integration, sample-rate selection, input/output control, and processor selection, are very much present.

¹That is, networks with no internal feedback, such as directly from network outputs to network inputs. These feedforward networks *will* be used as part of a feedback control loop.

3. Much relevant control theory exists, in addition to specific control knowledge regarding this control application.
4. Information about the changed plant will need to be extracted through an identification process, so the learning task is evolving continually.
5. Rapid adaptation is required to regain stability and prevent the system from damaging itself.
6. On-off actuators present a non-differentiable function that leads to problems with current learning algorithms.

The complexity of the research task generates the requirement for a basic strategy in addressing this control problem: The system-level issues in items 1 through 4 are handled with a hybrid approach that involves an analysis at the system level of where the neural network can contribute, segments the problem, and makes full use of conventional control and system identification methods. To address issue number 5, a modified network architecture is developed to provide fast initial learning, and to allow initial infusion of a pre-calculable stabilizing controller. To address issue number 6, a new algorithm is developed to perform optimization with the on-off thrusters, while still allowing the use of gradient information to accelerate the optimization.

This chapter has three major sections:

1. The control application and experimental system (robot) hardware are described.
2. The thruster mapping problem at the center of the control application is defined.
3. A solution framework is presented, including three separate solution methods for the thruster mapping problem.

2.1 Experimental System

The experimental system used to study issues in autonomous navigation and control of free-flying space robots is shown in Figure 2.1. The design and construction of this robot are discussed thoroughly in [56]. In that work, Ullman designed and built the robot, and gave it the capability to intercept and capture a free-floating object autonomously. The only major hardware modification required to perform the experiments described here was

the installation of accelerometers and an angular-rate sensor. These sensors are used in the identification of the characteristics of each thruster after mechanical thruster failures occur. These minor hardware modifications allow the robot to sense the acceleration resulting from each of its thrusters, thus enabling the reconfigurable control system that is the focus of this application.

Operating in a horizontal plane, the mobile robot simulates the drag-free and zero-g characteristics of space: it exhibits nearly frictionless motion as it floats above a 2.74×3.65 meter (9×12 foot) granite surface plate on a 50 micron (0.002 inch) cushion of air. It is a fully self-contained planar laboratory-prototype of a free-flying space robot complete with on-board gas supply, eight cold-gas thrusters for propulsion, electrical power, multi-processor computer system, on-board camera, wireless Ethernet data/communications link, and two cooperating manipulators[56].

The robot has a mass of 70 kg, and is controlled with eight thrusters, each nominally producing 1 Newton of thrust. Position feedback comes from a pair of CCD cameras mounted to the ceiling above the robot. Two cameras are required to cover the total surface area of the granite table. The cameras detect a pattern of LEDs mounted to the top of the robot. A custom vision processing board processes the camera output, and produces position information at a 60 Hz update rate that is accurate to better than 1 mm. This $[x, y, \psi]$ vector is digitally filtered and differenced to produce a velocity vector. The processing is performed off-board and then communicated back to the robot via a Motorola Altair wireless Ethernet data/communications link.

The specifics of the control-system components are described in greater detail in Chapter 6. This section will focus on the hardware central to the reconfigurable control system: the thrusters and the accelerometers.

2.1.1 Thrusters

Central to the control system design are the actuators themselves, as shown in Figure 2.3. Eight on-off air thrusters are used to provide redundant actuation in all three degrees of freedom of the base. Each thruster produces about 1 N of thrust, and can operate effectively at rates up to 30 Hz. For the purposes of this control application, they can be modelled as pure on-off actuators, ignoring transient effects. However, the transient effects will be shown to impact selection of the sample rate and design of the filters used for the accelerometer signals.

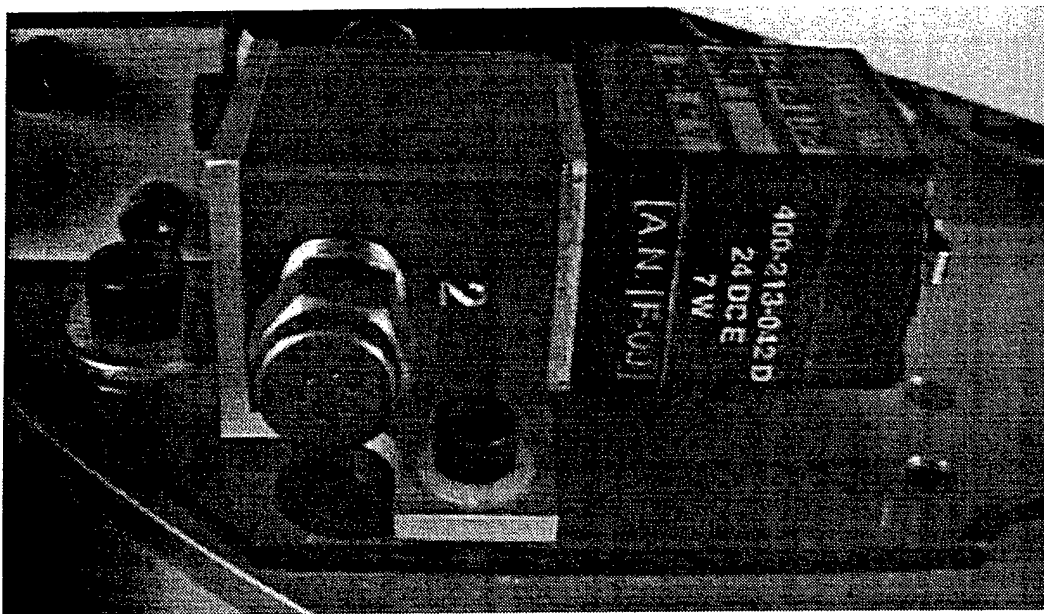


Figure 2.3: Photograph of Thruster Assembly

One of the eight cold-gas thruster assemblies is shown. The brass hexagonal plug with 6 holes is the thruster nozzle. The brass valve assembly is behind it, and the solenoid is to the right. The entire assembly is mounted with an aluminum bracket. Gas used is air at 690 kPa (100 psi) reservoir pressure, exiting to one atmosphere. The converging-diverging nozzles are designed with an exit velocity of Mach 2, resulting in one Newton of thrust per thruster [56]. The solenoid valve has a response time of about 5 ms.

The nominal thruster nozzles are described in [56]. The six converging-diverging openings in each nozzle were machined with a custom form tool. The expansion ratio of 1.7, reservoir pressure of 690 kPa (100 psi), and exit pressure of 101 kPa (14.7 psi) are designed to yield an exit velocity of Mach 2. Figure 2.3 shows an individual thruster assembly, including a solenoid valve that controls the flow through the nozzle. The solenoid, shown to the right of the valve, is spring loaded to stay closed, and opens fully in about 5 ms when current is applied. The valve has a choke point of about 1.65 mm (0.065 inch) diameter.

One of the pairs of thruster assemblies that is located at each of the four corners of the robot is shown in Figure 2.4. The nominal layout of all eight thrusters can be seen in the left side of Figure 2.2.

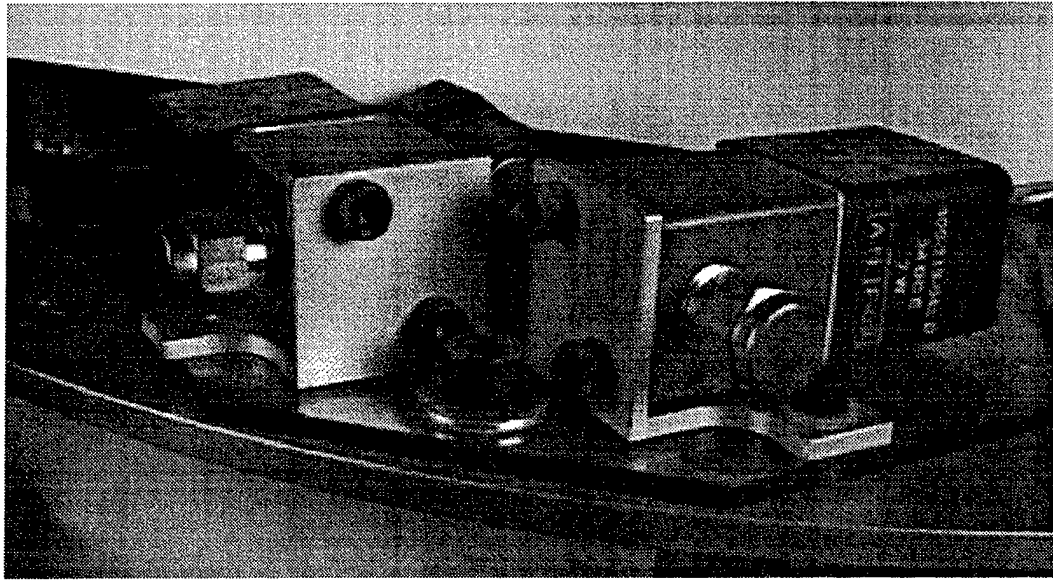


Figure 2.4: Photograph of Two Thruster Assemblies

To study control reconfiguration, a number of “failed” thrusters were built to simulate different failure modes. These failures include: zero thrust, reduced thrust, 45° misalignment, and 90° misalignment. The hardware used to simulate physically these failures is shown in Figures 2.5 and 2.6.

The use of a converging-diverging design resulted in a performance increase of 6.5% [56]. This may be significant for thrusters that are to be used every day, such as the nominal thrusters on this robot. However, the “failed” thrusters with off-nominal thrust characteristics were built with straight walls formed by drilling with standard bits ranging in diameter from 0.25 mm (0.010 inch) to 0.69 mm (0.027 inch). Thrusters were tested on the robot, measuring robot acceleration to determine the thruster strength.

It was not possible to build thrusters with greater thrust capability than about 1.2 Newtons by nozzle modification alone. As more air is required, the choke point in the valve causes a greater pressure drop across the valve, and less across the nozzle. As more openings were added, and the total nozzle area increased, thrust peaked at 1.2 Newtons with a 150% increase in area beyond nominal, and then declined. Obtaining greater thrust would require machining a larger valve orifice, or complete replacement of the solenoid-valve assembly.

Completely failed thrusters were simulated by nozzles with a single 0.25 mm (0.010 inch) diameter hole rather than being plugged completely. This resulted in about 0.025 Newton of



Figure 2.5: Thruster Failure Modes – Reduction in Thrust Level

Thruster failures are simulated by replacing the nominal thruster nozzles with mechanically altered nozzles. The first thruster has a single 0.25 mm (0.010 inch) diameter hole, and simulates a complete thruster failure. The second thruster has three 0.69 mm (0.027 inch) holes, simulating a reduced-strength thruster. The third thruster is a nominal thruster, with 6 converging-diverging holes.

thrust, which was 1/40th of nominal, and effectively zero. However, the presence of a small hole means the thruster can be heard to fire, allowing an observer a better understanding of the identification and reconfiguration process.

The volume of the chamber between the valve and the nozzle opening has a transient effect on thruster performance. When the valve opens, it takes a finite length of time for the pressure to rise to the steady-state pressure (which is defined by the reservoir pressure minus the pressure losses in plumbing and across the valve). Similarly, thrust continues after the valve closes, while the chamber empties. This effect may be seen in Figure 2.10.

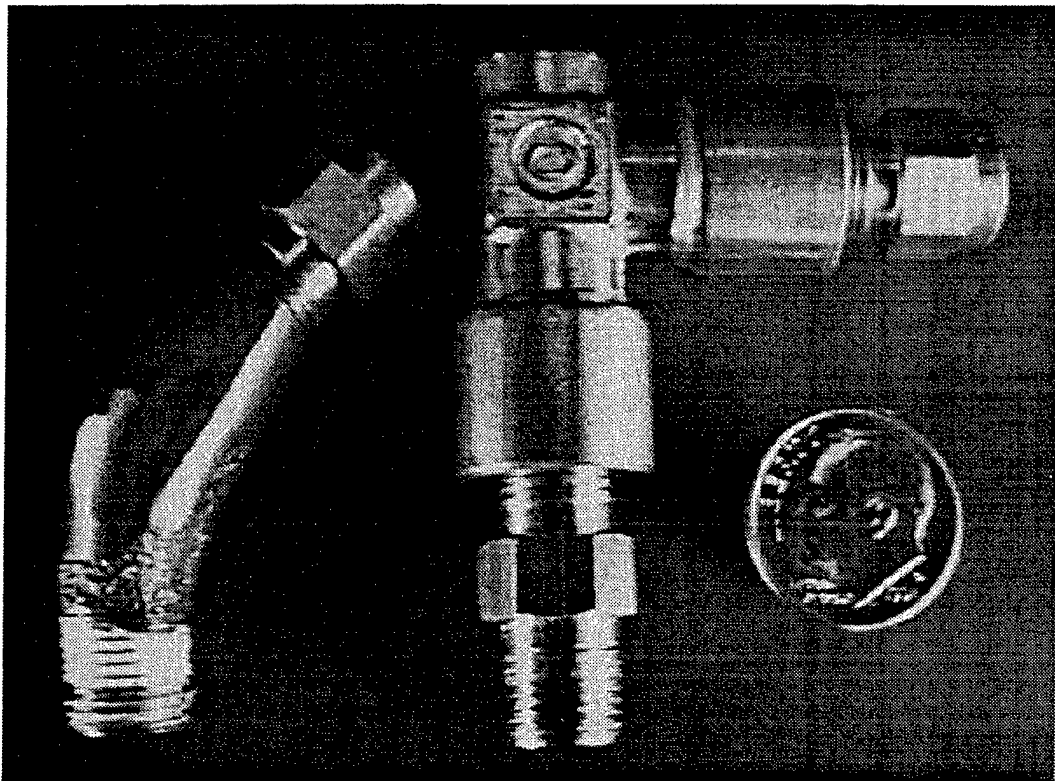


Figure 2.6: Thruster Failure Modes – Change in Thrust Direction

Thruster failures are simulated by adding elbows to change physically the direction of thrust. The 45° and 90° elbows simulate severe (and potentially destabilizing) thruster misalignments.

Since the 45° and 90° elbows used to simulate thruster failure increase the volume of this chamber, this effect is increased significantly to the point that it is greater than the sample period of 100 ms. Fortunately for the system ID process, thrusters tend to remain in the on position for several sample periods, so the transient effects can be tolerated.

2.1.2 Accelerometers, Angular-Rate Sensor

Accurate acceleration information is crucial to the identification process. Acceleration data are used to identify thruster failures and build a model of the robot for reconfiguration. Issues such as sensor noise, sensor placement, sample-rate selection, mechanical vibration, electrical noise, and thruster transient characteristics all contribute to the difficulty in obtaining accurate acceleration signals.

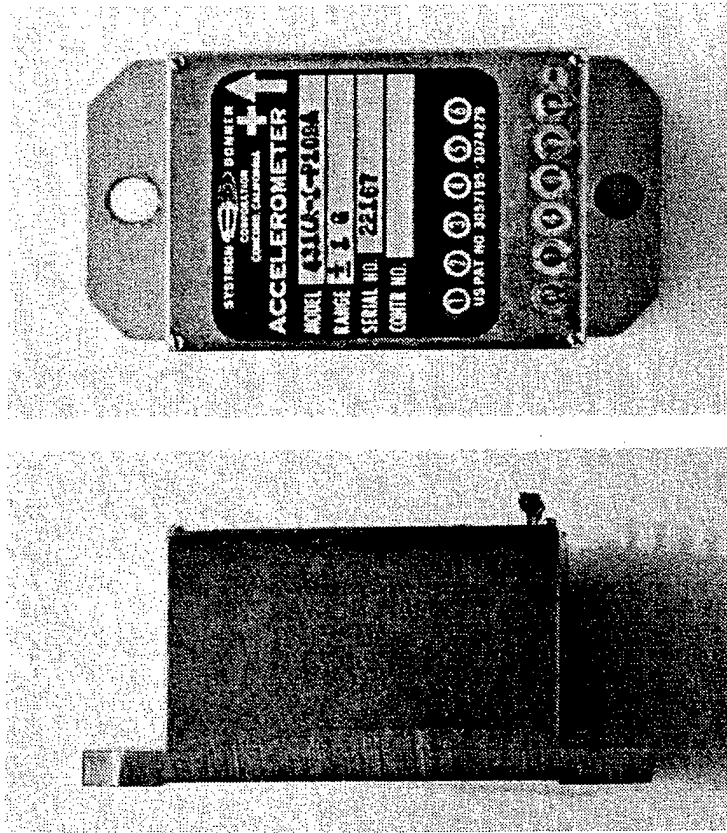


Figure 2.7: Accelerometer Photograph

Photograph of the Systron Donner 4310A Linear Servo Accelerometer. Actual size is as shown: overall length is 76.2 mm (3.0 inches). Two accelerometers are mounted to the robot base to measure translational accelerations.

Two Systron Donner 4310A Linear Servo Accelerometers are used. These accelerometers, shown in Figure 2.7, have a range of ± 1 g, and are accurate to better than 0.1 milli-g^2 . The accuracy of acceleration measurements is limited not by the accelerometers, but by the presence of extraneous vibrations. For example, the small cooling fan in the wireless Ethernet receiver at the top of the robot produces a 70 Hz vibration that is clearly measurable at accelerometer mounting positions on the robot base plate.

As with all Systron Donner accelerometers, the 4310A uses a force balance. A proof mass is suspended within the accelerometer, and moves slightly in response to acceleration, as depicted in Figure 2.8. This displacement is measured by a position detector, and a

²A full set of specifications is presented in Appendix B.

control circuit and torque coil are used to drive the displacement to zero. The control current used to keep the proof mass from moving is amplified and used as the accelerometer output signal.

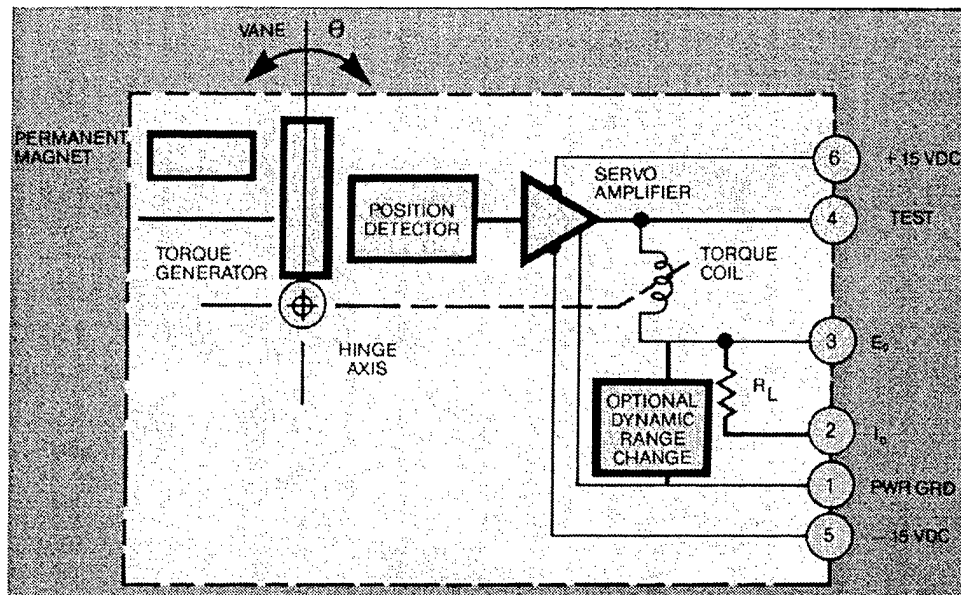


Figure 2.8: Accelerometer Circuit

The servo-control circuit contained within the force-balance accelerometer is shown. The control current used to keep the proof mass from moving is amplified and used as the acceleration signal.

A Watson Industries angular-rate sensor (model # ARS-C131-1AV) is used. This device, also called a tuning-fork gyro, vibrates a tuning fork and measures the Coriolis force on each of the beams as the fork rotates, thus producing the angular-rate signal. Accuracy is better than $0.1^\circ/\text{sec}$, but this needs to be differentiated to obtain angular acceleration.

The accelerometer signals and angular-rate signal pass through analog pre-filters with two critically damped poles at 75 Hz. They are then sampled by the A/D converter at a 200 Hz sample rate (while the control loop runs at 10 Hz). The accelerometer signals are digitally filtered with fourth-order Butterworth filters with poles at 25 Hz, and the angular-rate signal is digitally filtered with a second-order Butterworth filter with poles at 10 Hz. Angular acceleration is obtained by a first difference of the rate signal.

At this point, the filtered accelerometer signals are combined with the angular-rate and angular-acceleration signals to produce the base accelerations in $[x, y, \psi]$. The computations made to combine these signals are highly dependent upon sensor placement, so the sensors were placed to yield the highest possible accuracy, as shown in Figure 2.9.

The accelerometers measure acceleration in one direction at one location on the robot base, so the basic task is to convert these acceleration signals into acceleration at the center of the base. If it were practical to locate both accelerometers with their proof masses exactly coincident with the robot mass center, one pointing straight ahead in $+x$, and the other pointing in $+y$, no compensation would be required. This is not practical, so the compensation requirements are:

1. Remove angular-acceleration effects (needed if the accelerometer measurement axis is not aligned perfectly with the center of mass (c.m.), i.e. has a tangential component).
2. Remove centrifugal-acceleration effects (needed if the proof masses are not located at the c.m. and the measurement axes have some radial component – e.g. these effects occur even when the robot spins about its c.m. with no acceleration of the c.m.).
3. Rotate translational-acceleration vector to robot frame (needed if accelerometers are not aligned with x and y axes)

In theory, the accelerometers could be placed anywhere on the base (as long as they are not perfectly parallel), and centrifugal and angular-acceleration effects could be subtracted by calculation. However, due to the differences in accuracy for each type of sensor, choosing the correct configuration will result in better acceleration measurements. Taking these factors into consideration it was found that:

1. Angular acceleration effects would be difficult to compensate due to a relatively noisy angular acceleration signal. For this reason, the accelerometers are aligned accurately with the c.m. of the robot, eliminating any angular acceleration effects.
2. The angular-rate sensor (ARS) provides a clean signal, so centrifugal acceleration effects can be accounted for by computation. However, the effect is proportional to the radial distance from the proof mass to the c.m., so the accelerometers are positioned as close to the c.m. as possible. The distance is 46.5 mm (1.83 inches). An additional complication is the saturation of the ARS. This usually occurs only

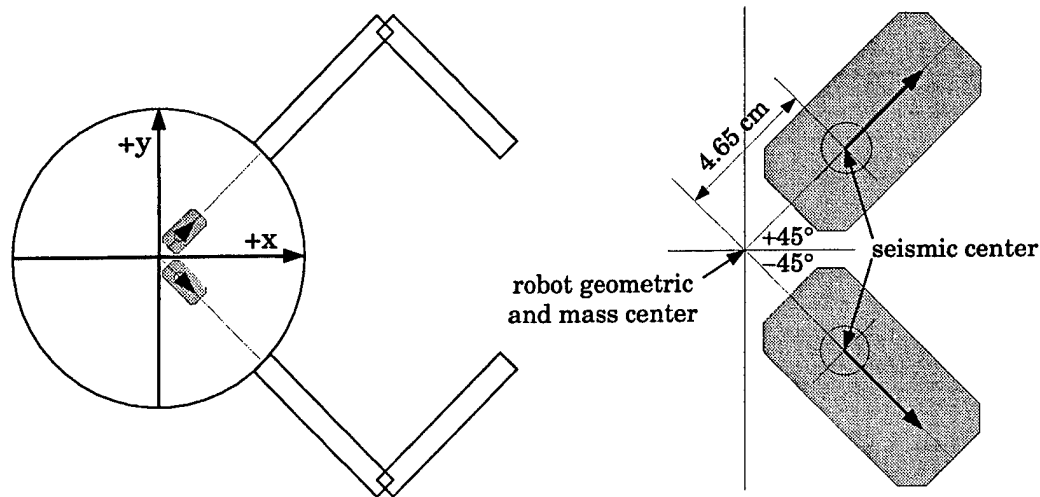


Figure 2.9: Accelerometer Mounting Locations

The accelerometers are mounted orthogonal to each other, with their seismic centers as close to the robot center of mass as possible, and aligned radially with the center of mass. This facilitates the removal of extraneous acceleration signals (i.e. from centrifugal and angular-acceleration effects) by minimizing their size and providing good sensors for their removal. For example, the angular-rate signal is cleaner than the angular-acceleration signal, so angular-acceleration effects are zeroed by alignment with the center of mass, while centrifugal effects are cancelled by calculation.

when the robot spins out of control, before reconfiguration, but some sensing is needed (both for centrifugal compensation and for angular-acceleration measurement). When saturation is detected, angular rate and acceleration are obtained by digitally filtering the vision-system position signal. The angular-rate sensor is used when possible, since it is one derivative closer to the measurement needed, and therefore less noisy.

3. Rotational transformation is accomplished with a 2×2 transformation matrix.

The resulting accelerometer mounting locations are shown in Figure 2.9. The calculations used to go from the sensors to the final acceleration signals are shown graphically in Figure 6.3.

This reconstruction of the acceleration vector is carried out at a 200 Hz update rate on-board the robot. Examples of dynamically corrected and filtered output from the accelerometers and angular-rate sensor are shown in Figure 2.10.

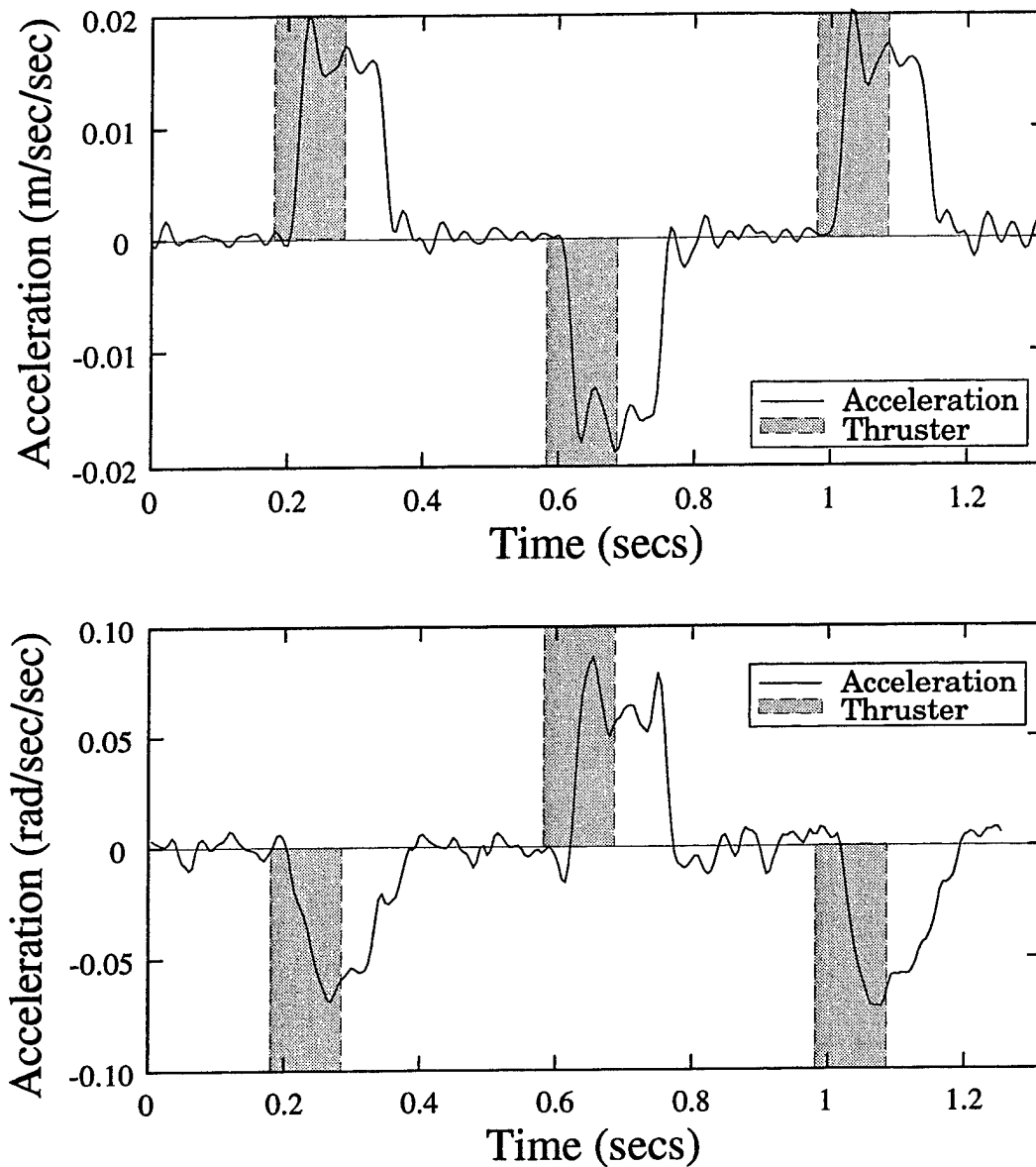


Figure 2.10: Translational and Angular-Acceleration Signals

Shaded areas indicate the sign and duration of a thruster pulse. 100 ms is the minimum-length pulse used for control. Lag is due to the transient response of the thruster and the effects of the analog and digital filtering. Acceleration persists for longer than the thruster pulse width due to the finite chamber size between the valve and nozzle in the thruster assembly. This data is still noisy after filtering, but leads to accurate identification when used with the linear-regression processes described in Chapter 6.

2.2 Thruster Mapping

2.2.1 Problem Definition

The three degrees of freedom (x, y, ψ) of the base are controlled using eight thrusters positioned around its perimeter, as shown in Figure 2.11. Each thruster produces both a torque and net force on the robot. This coupling, combined with the on-off nature of the thrusters, substantially complicates the control task.

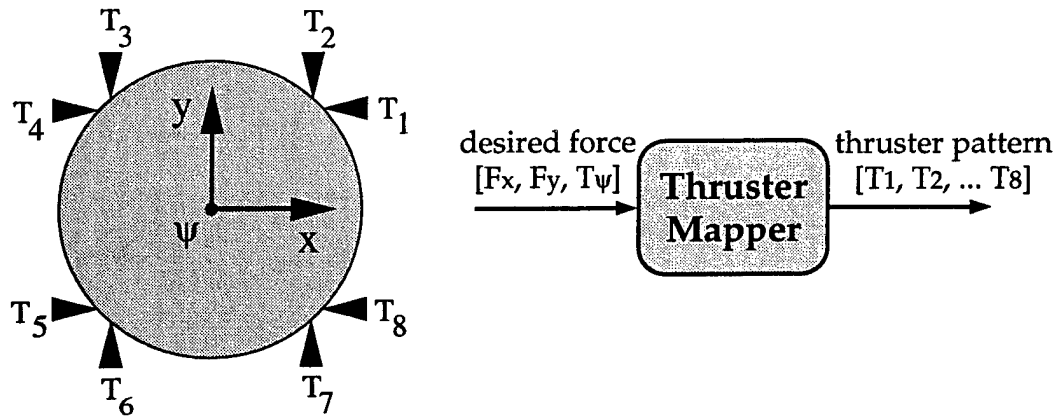


Figure 2.11: Thruster Mapping, Problem Definition

At every sample period, the Thruster Mapper takes a desired force vector, $[F_{x_{des}}, F_{y_{des}}, \tau_{\psi_{des}}]$, and finds the thruster settings, $[T_1, T_2, \dots, T_8]$, to minimize a specified cost function. The on-off thrusters and coupling between forces and torque make this problem difficult. This mapping is calculated several times per second, motivating the development of a nonlinear approximate solution that can run in real time. The thruster mapper must adapt to changes in thruster characteristics. Development of a neural network to implement this "Thruster Mapper" is the focus of this application.

The thruster mapping task, also shown in Figure 2.11, that must be performed during each sample period is to take an input vector of continuous-valued desired forces and torques, $[F_{x_{des}}, F_{y_{des}}, \tau_{\psi_{des}}]$, and find the output vector of discrete-valued (off, on) thruster values, $[T_1, T_2, \dots, T_8]$, that minimizes a specified cost function.

The robot-base-control strategy developed for this system is shown in Figure 2.12. The complete control system is described in detail in Chapters 3 and 6. A proportional-derivative control law produces a continuous vector of desired forces, F_{des} , based on position and

velocity information from the overhead vision system. The thruster mapper takes this force vector and outputs the pattern of thrusters to be fired on the robot.

Partitioning the controller into a “control module” (PD controller in this case) and a “thruster mapper” greatly simplifies controller design since both components can be designed independently. Smooth actuation is still possible due to the low thruster impulse, which results from high sample rate (10-60 Hz), low thrust (force per thruster, $F = 1$ N; torque per thruster, $\tau = 0.14$ N-m) and high mass (mass, $M = 70$ kg; moment of inertia, $I = 3.1$ kg-m²). This strategy was originally developed as part of a conventional control system for the robot [56].

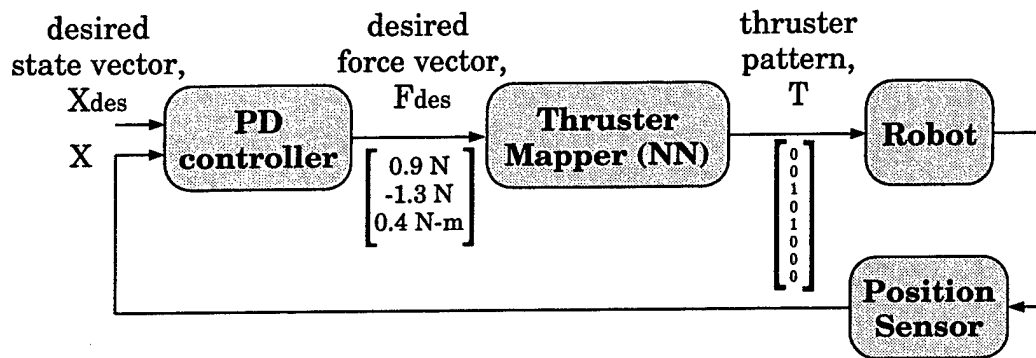


Figure 2.12: Robot-Base-Control Strategy

The control module treats the thrusters as linear actuators. The thruster mapper must find the thruster pattern producing a force closest to that requested by the base control module.

2.2.2 Cost Function

Since each thruster can output only full thrust (nominally 1 Newton) or nothing, the thruster mapper is not capable of exactly producing the requested force. The basic approach to this problem is to define a cost function, and then to find the thruster pattern, $[T_1, T_2, \dots, T_8]$, that minimizes this function. The specific search or neural-network functional mapping used to “find the thruster pattern” will be discussed in Chapter 3. In this research, a general cost function was used that incorporates the normalized force-error vector and the amount of gas used. This function is shown in Equation 2.1.

$$\min_{\mathbf{T}} J = \left[\left(\frac{F_{xerr}(\mathbf{T})}{F_{xnorm}} \right)^2 + \left(\frac{F_{yerr}(\mathbf{T})}{F_{ynorm}} \right)^2 + \left(\frac{\tau_{\psi err}(\mathbf{T})}{\tau_{norm}} \right)^2 + \alpha_{gas} \sum_{i=1}^8 T_i \right] \quad (2.1)$$

where,

- J = thruster-mapping performance cost
- \mathbf{T} = binary thruster values, $[T_1 \ T_2 \ T_3 \ T_4 \ T_5 \ T_6 \ T_7 \ T_8]$
- i = thruster number
- $F_{xerr}(\mathbf{T})$ = net force error in x-direction, $(F_{xdes} - F_{xact})$, resulting from \mathbf{T}
- $F_{yerr}(\mathbf{T})$ = net force error in y-direction, $(F_{ydes} - F_{yact})$, resulting from \mathbf{T}
- $\tau_{\psi err}(\mathbf{T})$ = net torque error about ψ -axis, $(\tau_{\psi des} - \tau_{\psi act})$, resulting from \mathbf{T}
- F_{xnorm} = normalizing factor for F_x
- F_{ynorm} = normalizing factor for F_y
- τ_{norm} = normalizing factor for τ_{ψ}
- α_{gas} = gas-weighting parameter

In matrix form, this can be expressed as Equation 2.2.

$$J = \mathbf{F}_{err}(\mathbf{T})^T \mathbf{N} \mathbf{F}_{err}(\mathbf{T}) + \alpha_{gas} \sum_{i=1}^8 T_i \quad (2.2)$$

where,

$$\begin{aligned} \mathbf{F}_{err}(\mathbf{T}) &= [F_{xerr}(\mathbf{T}) \ F_{yerr}(\mathbf{T}) \ \tau_{\psi err}(\mathbf{T})]^T = \text{force vector} \\ \mathbf{N} &= \begin{bmatrix} \frac{1}{F_{xnorm}^2} & 0 & 0 \\ 0 & \frac{1}{F_{ynorm}^2} & 0 \\ 0 & 0 & \frac{1}{\tau_{norm}^2} \end{bmatrix} = \text{normalizing matrix} \end{aligned} \quad (2.3)$$

If the robot were equipped with linear actuators (i.e. “proportional thrusters”), a vector of continuous-valued actual forces, $[F_{xact}, F_{yact}, \tau_{\psi act}]$, could be produced that exactly equalled the desired force vector, $[F_{xdes}, F_{ydes}, \tau_{\psi des}]$, requested by the controller (i.e. $J = 0$). However, a perfect mapping is not generally achievable with discrete-valued thrusters, and the weighting parameters selected in the cost-function define the distribution of error (i.e. translational force error vs. rotational force error vs. gas usage). Selection of the normalizing

factors and gas-weighting factor in Equation 2.1 define the cost function and the resulting optimal thruster mapping.

Throughout this thesis, the normalizing factors used are the nominal force and torque values produced by firing a single thruster. These values are indicated by $F_{thruster}$, force-per-thruster, and $\tau_{thruster}$, torque-per-thruster. With no weighting on gas usage, this results in the minimum-length force-error vector in normalized-force space. This is a simple, straightforward method that results in a good thruster mapper, and is used for analysis purposes in Chapters 4 and 5. This is shown in Equation 2.4.

$$\min_{\mathbf{T}} J = \left[\left(\frac{F_{x_{err}}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{F_{y_{err}}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{\tau_{\psi_{err}}(\mathbf{T})}{\tau_{thruster}} \right)^2 \right] \quad (2.4)$$

For the experimental implementation, discussed in Chapter 6, an additional practical issue is present: gas usage should be reduced if it can be achieved with minimal effect on force-mapping performance. To achieve this, an additional cost is placed on gas usage, so that if two candidate vectors produce similar size force errors, the more fuel-efficient one will be chosen. A good balance between control accuracy and gas usage is found with $\alpha_{gas} = 0.5$. This cost function is shown in Equation 2.5.

$$\min_{\mathbf{T}} J = \left[\left(\frac{F_{x_{err}}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{F_{y_{err}}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{\tau_{\psi_{err}}(\mathbf{T})}{\tau_{thruster}} \right)^2 + \frac{1}{2} \sum_{i=1}^8 T_i \right] \quad (2.5)$$

In minimizing the force error only, the thruster mapper does not consider the dynamics of the plant. It assumes that the F_{des} vector output by the controller feedback law is chosen carefully enough that it needs only concern itself with producing the closest matching F_{act} . In this application, the controller component is a simple proportional-plus-derivative controller (shown in Figure 2.12) that does not take into account the thruster limitations. Ideally, the controller component would be aware of thruster limitations, possibly leading to a merging of the control and mapping components. This complex nonlinear control problem is not addressed here, but a first step is proposed in the form of a modified cost function in Appendix A.

In summary, the cost function was chosen to be the length of the normalized force-error vector augmented by a cost on gas usage, where the normalization factors were the force-per-thruster, $F_{thruster}$, and torque-per-thruster, $\tau_{thruster}$. For neural-network analysis only, the cost function shown in Equation 2.4 was used. For experimental implementation, the function shown in Equation 2.5 was used, reducing gas usage. This thruster mapper trades

off force error for a reduction in gas usage, just as an optimal controller balances error with control effort.

Selection of the cost function defines the correct thruster pattern for any given $F_{desired}$ vector. The mechanics of how this correct vector is actually found (i.e. search or neural-network-based functional approximation) is described below.

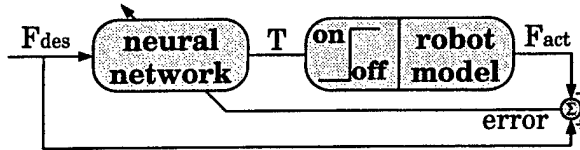
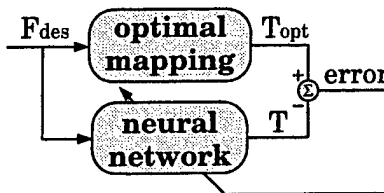
2.3 Solution Strategy, Mapping Methods

The reconfiguration strategy proposed in Figure 3.1 requires an "Indirect Training" approach, where the neural network attempts to find the best mapping based on the latest estimate of the plant model, and then adapts itself to optimize mapping performance. This indirect training approach is shown as the top part of Figure 2.13. The word "indirect" here refers to the lack of an optimal teacher, so the network adaptation is directed by experimentation (in simulation) with a model of the plant. As seen in Figure 2.13, the network's thruster pattern is passed through a model of the robot, and the resulting force vector is compared with the desired force vector, resulting in the error signal used to train the network (without the direction of an optimal teacher).

While "indirect learning" is the ultimate goal here, two other methods, "direct learning" and "exhaustive search," are developed as steps towards of this goal. All three methods are summarized in this section.

In the development of an indirect training procedure, several issues must be addressed, including neural-network architecture and optimization (also referred to as training, learning, or adaptation). To "separate variables," and permit the study of these generic issues separately, an intermediate step, "Direct Training," is introduced. This step, shown in the middle part of Figure 2.13, permits the development of neural-network architecture selection and optimization procedures which can then be carried over directly to the indirect training problem.

In direct training, the network is taught simply to copy an "optimal teacher," in this case the optimal thruster mapping. To obtain this optimal mapping, a search must be performed over all possible thruster combinations. Fortunately, when all thrusters are working correctly (i.e. before the reconfiguration due to thruster failures), symmetries exist that can simplify the search process. This non-neural-network approach is shown in the bottom part of Figure 2.13.

INDIRECT TRAINING**DIRECT TRAINING**

F_{des} : desired force,
 $[F_{xdes}, F_{ydes}, T_{\psi des}]$
 F_{act} : actual force,
 $[F_{xact}, F_{yact}, T_{\psi act}]$
 T : thruster values,
 $[T_1, T_2, \dots, T_8]$
 T_{opt} : T that minimizes
 the cost function
 error: signal minimized
 to train the network

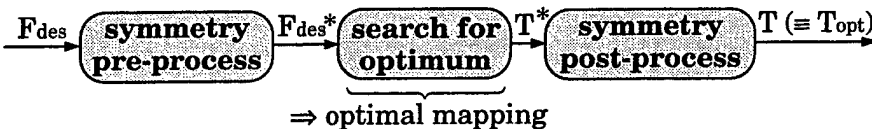
EXHAUSTIVE SEARCH (optimal solution)

Figure 2.13: Thruster-Mapping Methods

Indirect training (i.e. with no optimal teacher, adaptation is based upon performance with the robot model) is the ultimate goal, but direct training is used to study architecture and optimization issues, and an exhaustive search (symmetry-aided) is used to generate the optimal mapping required by direct training.

These three different techniques have also been used to make possible evaluation of performance and comparisons. Due to the discrete nature of the thrusters, even the optimal thruster mapper results in significant errors. This optimal performance level is used to evaluate the performance of the neural-network control system. Also, use of the direct training performance as a benchmark for evaluation of the indirect training performance allows study of the issues involved in indirect training.

Although the final goal is indirect training, the methods need to be developed in reverse order, i.e. (1) optimal search, then (2) direct training, then (3) indirect training. Each successive method builds upon knowledge gained in the previous step, as they work towards the final goal of indirect learning. The first step contributes the robot-base-control strategy, and an optimal solution to be used as a benchmark. The second step contributes understanding of architecture and optimization issues. The final step contributes a new learning

algorithm to accommodate the on-off thrusters. The result is a learning system that can be used in the reconfigurable control application. Segmenting the problem in this manner resulted in a "separation of variables," and allowed for concentration on one issue at a time.

These methods are presented in the order they were developed, since each one builds upon the previous step; but the final method, indirect training, is the one used in the reconfigurable control system required when thruster failures occur. Presenting the search method first also serves as a motivation for the neural-network approach, as the limited extensibility of this method is highlighted.

2.3.1 Thruster Mapping by Exhaustive Search

The first implementation, *SEARCH*, used an exhaustive search at each sample period to find the thruster pattern that minimizes the force-error vector [56]. Symmetries are used to reduce greatly the search space, enabling it to run in real time at a 60 Hz sample rate. This solution method does not scale well for a three-dimensional robot, or when thruster failures are allowed, disrupting the symmetries. This provides the motivation for using a neural network: *the neural network is used to learn and implement an approximation to the optimal solution - one that can be computed in real time.*

The idea behind the exhaustive search is that there are a finite number of possible thruster combinations (in this case, with eight bi-level thrusters, there are $2^8 = 256$ combinations), so the thruster mapper can evaluate each possible combination, and choose the one that minimizes the specified cost function. This process must be executed at every sample period, so to speed up the process it is very helpful if the symmetries in the system can be exploited.

Search Simplification Using Geometric Symmetries

If the thrusters are all the same strength (the nominal configuration assumed in this example), firing two opposing thrusters (e.g. T_1 and T_4) will produce no net thrust. To eliminate these useless combinations, the eight on-off thrusters, $[T_1, T_2, \dots, T_8]$, may be considered as four backwards-off-forwards thrusters $[R_1, R_2, R_3, R_4]$, where, for example, R_1 represents the reaction force resulting from T_1 and T_4 . This reaction force representation can be used here to reduce the possibilities to $3^4 = 81$. Now the robot is considered to have 4 tri-level thrusters instead of 8 bi-level thrusters. This simplification is valid whenever two thrusters of equal magnitude are directly opposing.

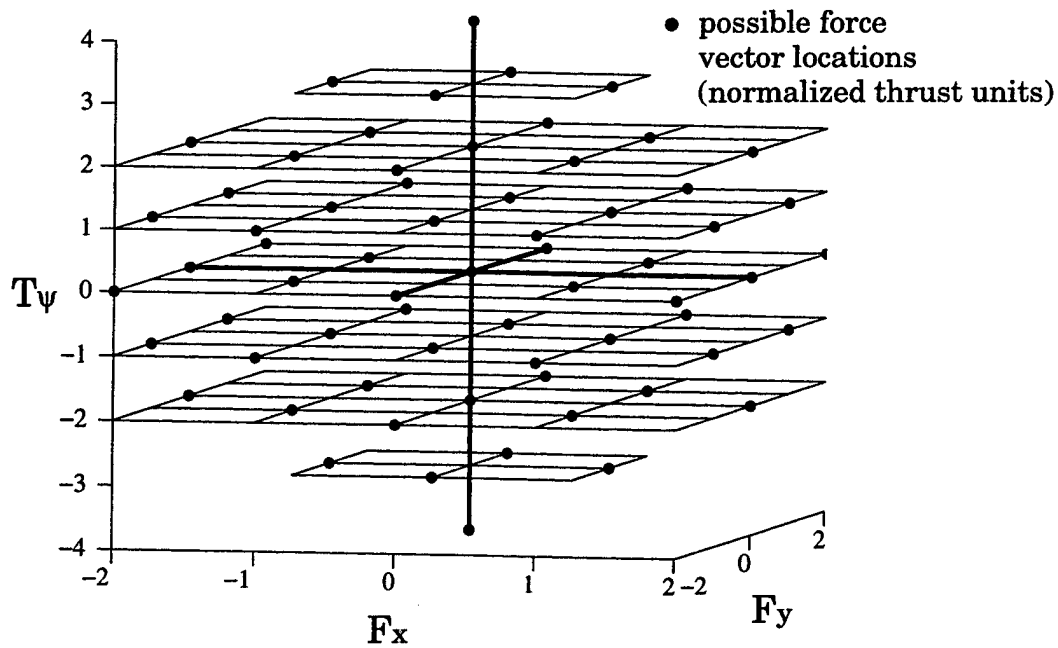


Figure 2.14: Possible Force Vectors with Eight Symmetrical Thrusters
Units are in normalized thrust units. Each of the 65 circles represents a force vector that is achievable with the nominal configuration of eight on-off thrusters. A simplified version of the thruster mapping problem is to find which of these circles is closest to the desired force vector. The problem is complicated by the additional desires to save gas and to accommodate for failed thrusters.

The next level of simplification comes about due to the redundancy in actuation capability. Elimination of redundant combinations (e.g. firing T_1 and T_2 produces the exact same net force vector as firing T_3 and T_8) reduces this number to 65. Since redundant combinations occur due to many thrusters having common strengths and regular positions, this simplification fails when these conditions are not met. These 65 remaining available thrust vectors are plotted in Figure 2.14.

Symmetries about the $x - y$, $x - \psi$, and $y - \psi$ planes allow us to consider candidates in the first octant only, reducing the search space to 16. The final symmetry is about the $x = y$ plane. This further reduces the number of candidate vectors to 11, resulting in the 11 locations shown in Figure 2.15.

The procedure to implement this symmetry-aided search is to take the desired force vector and use the symmetries mentioned above to transform it into the first half of the first octant in force space ($[F_{x_{des}}, F_{y_{des}}, \tau_{\psi_{des}}]$). This is done by taking the absolute values

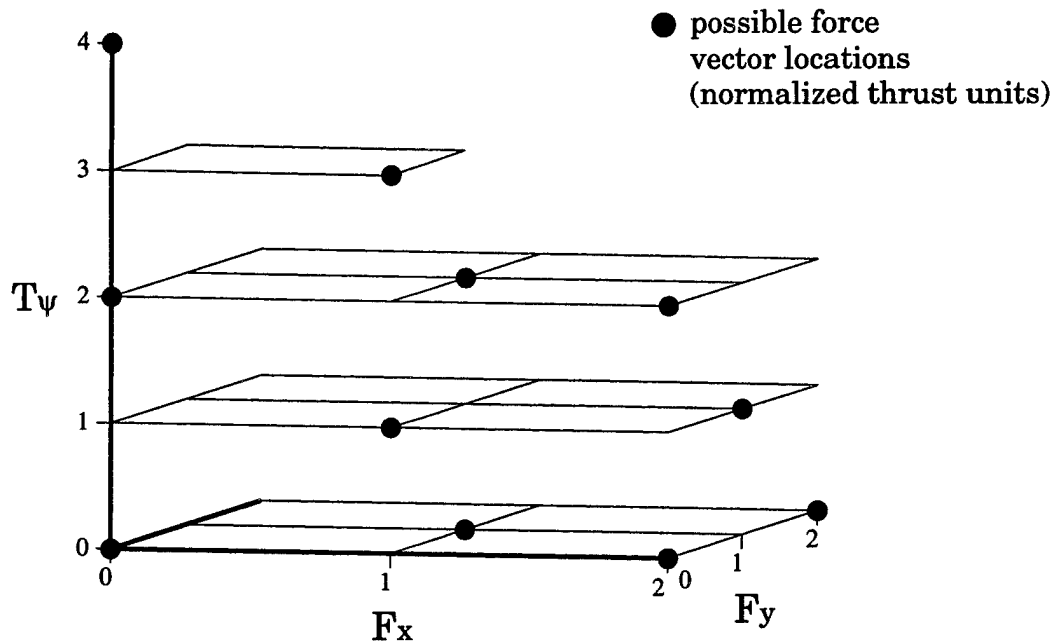


Figure 2.15: Possible Force Vectors after Symmetric Transformation

With all thrusters equal strength, and a geometrically symmetric layout, the 65 candidate thrust vectors can be reduced to 11 through symmetric transformation. This simplifies the search, allowing it to run in real time (this simplification is not possible when thruster failures occur).

of the vector components, and swapping the x and y components if necessary. Then this vector is compared to each of the 11 prototypes, resulting in 11 costs (perhaps a weighted cost function involving gas usage and force error), one for each of the 11 candidates. The candidate corresponding to the minimum cost is selected as the optimum. The thruster pattern associated with this candidate is then transformed to undo the symmetric transformations, bringing the force vector to the correct location in the full three-component force space. The resulting thruster pattern is implemented on the robot.

Reduction of the search space from 256 candidates in the general case to 11 in the fully symmetric case is critical to allowing the thruster mapper to run in real time. The amount of computation required to transform the F_{des} vector into this half-octant, search over 11 vectors, and then transform the minimum-cost R vector back to the one corresponding to the full 3-space input, then produce the T vector, is significantly less than if these symmetries were ignored and the search included 256 patterns.

Difficulties in Extending This Method

For a free-flying robot operating in three dimensions, the number of possible thruster combinations increases greatly (for example, with 24 on-off thrusters, there are $2^{24} = 16,777,216$ combinations). This is partially offset due to the number of symmetries also increasing (for a fully symmetric 3-D robot with 24 thrusters, there are 469 combinations that would need to be searched after complete symmetric reduction – a significant reduction, but still computationally demanding³).

Unfortunately, geometric symmetries may not exist, due to other spacecraft design constraints, or due to unanticipated thruster failures. In this case, the full number of thruster combinations would need to be searched to obtain the optimal solution. This situation motivates the use of a neural network for the thruster-mapping component: it is used to implement a nonlinear approximation to the optimal solution that can be computed in real time.

An alternative to developing a neural network to produce a function that approximates the result of the optimal search, is to use a sub-optimal search that can run in the time constraints imposed by the application. A simple example would be to limit the possible combinations to two thrusters firing at a time. In this case, only $24 \cdot 23/2$ (2 thrusters) + 24 (1 thruster) + 1 (no thrusters) = 301 combinations would need to be searched at each sample period. While this may make the problem tractable, mapping performance will be reduced drastically. Other sub-optimal search schemes may be developed that are more efficient than this simple example. One possible scheme is presented by Sperduti and Stork in “A Rapid Graph-based Method for Arbitrary Transformation Invariant Pattern Classification” [53]. This method was developed for an Optical Character Recognition application, highlighting the fact that this control application is similar to a pattern classification problem.

2.3.2 Direct Training of a Neural-Network Thruster Mapper

The search method described above defines the optimal solution to the thruster mapping problem. The next two methods are neural network approximations to this optimal solution. Since they are approximations, they will be sub-optimal, but can be designed to run in real time.

³An algorithm to automate derivation of the symmetric transformation functions has been developed by Kurt Zimmerman and Brian Kemper at the Stanford Aerospace Robotics Laboratory.

In the second method, *DIRECT TRAINING*, a neural network is trained to emulate the optimal mapping produced by the exhaustive search [71]. The network is repeatedly shown several desired force vectors along with the optimal thruster pattern chosen by the search algorithm. The weights in the network are adapted using backpropagation to make the network outputs match those produced by the search algorithm (the optimal solution).

This *DIRECT TRAINING* approach is useful primarily in that it allows the study of network architecture and topology issues before tackling the additional problems that come with indirect learning. Hence it serves as a stepping stone to the goal of indirect learning.

The approach also has potential advantages beyond that of an intermediate step. In particular, using a neural network as a function emulator may increase computational speed and system robustness very significantly due to the distributed, parallel nature of the computation.

The investigation of the network topology issues associated with this *DIRECT TRAINING* approach led to the Fully Connected Architecture, presented in Section 3. The FCA can also be used with the indirect training method described below.

2.3.3 Indirect Training of a Neural-Network Thruster Mapper

Once the topology issues have been investigated during the direct training exercise, the network architecture can be chosen. The topology of the network (i.e. the number of neurons and their interconnections) defines the functional complexity capacity of the network, whether it is trained directly or indirectly. With the architecture already selected to provide the required mapping accuracy, the next step is to focus on the training methods.

In the third method, *INDIRECT TRAINING*, a neural network is trained to find the optimal solution when presented with a model of the plant, but no optimal teacher. This required back-propagation of error through the discrete-valued thrusters, which in turn motivated development of the noise injection method to be presented in Chapter 5. This structure, shown in the top part of Figure 2.13, reveals that the thruster mapper is forming an inverse of the thruster model. Using a neural network to learn a plant inverse, and using this inverse in the forward control loop, is a common approach for neural-network control. As will be discussed later, the presence of non-differentiable hard limiters complicates the development of this inverse.

With this form of training made possible, the neural network control system is able to reconfigure itself quickly in response to even drastic changes in thruster characteristics. There is no longer a need to develop the search algorithm as an optimal teacher.

When evaluating mapping performance, the search method represents a lower bound, since it defines the optimal solution. Direct-training performance will be used as a benchmark for comparison with indirect training, since it represents the lower bound defined by the finite mapping complexity available with the chosen network topology.

2.4 Summary

The control application chosen to study neural-network control is reconfigurable thruster control of a free-flying space robot prototype, a capability compelled by major failures in the robot's thrusters. This chapter has described the experimental equipment used, the thruster mapping problem that is at the center of this control application, and the approach taken towards solution of the thruster mapping problem (that includes the use of three separate solution methods in building towards the final implementation). The remainder of this thesis develops a complete solution to this control problem, and presents advances in neural-network theory made to address this specific problem and the rather broad generic range of important real-world control problems that it represents.

Chapter 3

Control System Overview

This chapter presents an overview of the reconfigurable control system developed for the application described in Chapter 2. This is a complex control system, involving the integration of several components. As mentioned in Chapter 1, often the most important, and sometimes the most difficult aspects of a neural-network control application are the decisions about how to structure the control system and which components are to be neural-network-based.

Specifically, the first issue is to determine whether the application is one where neural networks can contribute efficiently better (and cheaper) control than is achievable without them. If they can, the second issue is to determine the optimal system architecture, that is to determine in just which segment(s) of the control system they should be used in order to do just that at minimal cost. This is the essence of astute hybrid control, a central contribution of this research.

In addition to presenting the system-level control system design, the reasons for choosing this structure are given. While this particular structure does not represent a general architecture for developing neural-network control systems, the new methodology that led to this structure *is* general, and can be applied to the development of a wide variety of neural-network control systems and neural network applications in general.

While this chapter discusses the overall control system and design considerations, Chapters 4 and 5 provide in-depth discussion of the specific neural-network issues encountered, and Chapter 6 provides a more detailed discussion of each of the control-system components.

3.1 Control System Structure

Figure 3.1 shows the overall system block diagram. The additions made here beyond the control system presented in Figure 2.12 include a user interface and an adaptive capability. These segments will be discussed in detail in Chapter 6. This chapter focusses on the system-level design considerations.

The objective is to control the position and attitude of the robot base, while subject to multiple, large, possibly-destabilizing changes in thruster characteristics. The plant is linear and well-modelled, except for the actuators, which are on-off thrusters that could have altered characteristics. An accurate vision system provides high-bandwidth position feedback, which is then digitally filtered and differentiated to provide velocity. On-board accelerometers and an angular-rate sensor are used to provide base-acceleration measurements.

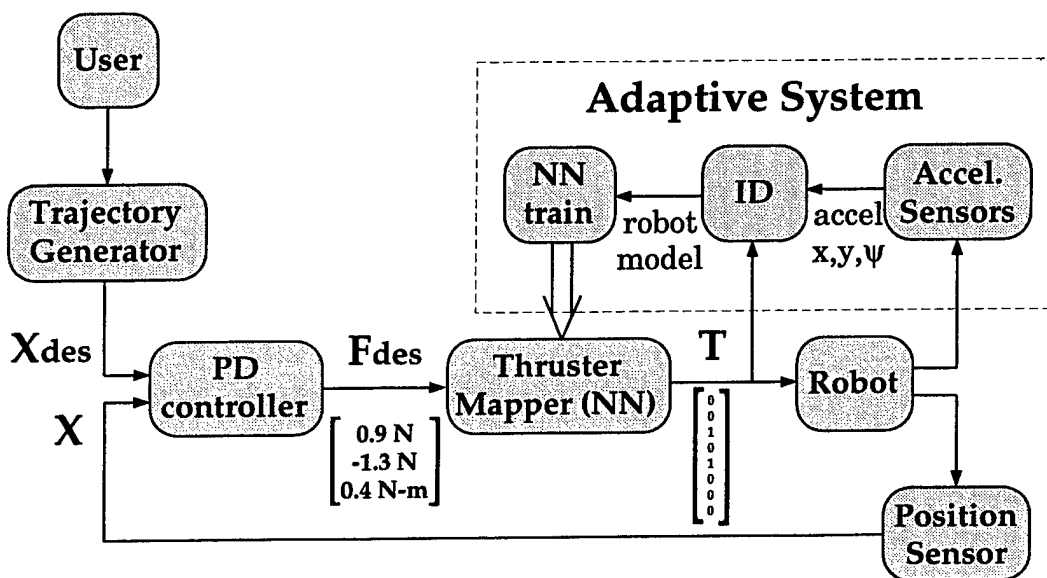


Figure 3.1: Reconfigurable Control System – Block Diagram

This control system is based upon a conventional indirect adaptive controller, such as a self-tuning regulator. Examples of the continuous-valued F_{des} vector and the corresponding discrete-valued T vector are shown. The ID block represents a recursive-least-squares identification of thruster strength and direction. This continually-updated model is passed to the neural network training block, shown in detail in Figure 5.6. The continually-updated neural thruster mapper is copied periodically into the active control loop.

3.1.1 Control System Design Considerations

Some control-system design considerations for this application include:

1. The robot is to be controlled by a human user at a high level, so path plan/-ning/traj/-ect/-ory generation is required.
2. The robot must reject disturbances and, at a low level, be robust to actuator and plant-model inaccuracies; so a robust feedback system is required.
3. Gas usage should be minimized where possible.
4. High-performance control is desired. The requirements for a free-flying space robot are different from those for a simple satellite control system. A robot is expected to carry out multiple-degree-of-freedom trajectory tracking with high control bandwidth. Satellites tend to spend their time regulating attitude to a fixed direction, or slowly slewing to a new direction. Satellite thruster-control systems are therefore usually designed for regulation performance and stability provability, at the expense of trajectory-following performance. For example, a satellite control system may look for the largest desired torque (roll, pitch, or yaw), and enforce a one-dimensional bang-bang control law in that degree of freedom only [63].
5. A non-adaptive conventional control system already exists.

Temporal issues that influence the control design include:

1. Control bandwidth is below 1 Hz.
2. Acceptable robot-base-control performance can be obtained with a 5 Hz thruster-update rate.
3. Accelerometer bandwidth extends from 0 Hz to greater than 500 Hz.
4. Extraneous vibrations exist from 30 Hz and up.
5. Thruster transient effects are on the order of 30 Hz and up.
6. During reconfiguration in response to thruster failures, stabilization is required within 15 seconds due to limited table area.

Items 2-5 lead to the selection of a thruster-control update rate of 10 Hz, but with a sensor sample rate of 200 Hz. Analog prefiltering, and digital filtering are performed on this over-sampled data to produce clean acceleration signals. The time limit imposed by item 6 provides sufficient time to begin, but not necessarily to finish building a model of the system. This leads to a design that has the adaptation running concurrently with the identification – there is not enough time to wait for the identification to converge.

3.1.2 Indirect Adaptive Control System

These system characteristics happen to fit well with a standard control structure known as “indirect adaptive control.” This refers to the use of sensor information to build a model of the system, and then to redesign a controller based upon the updated plant model. The “indirect” here refers to the intermediate step of building a model of the system. This is the structure shown in Figure 3.1.

The user issues desired-position commands to the robot via a graphical user interface. The current and desired position are used by a trajectory generator to calculate the path for the robot to follow, resulting in a trajectory vector, X_{des} , consisting of positions and velocities in the three degrees of freedom at each sample time. This desired state vector is input to a PD controller, along with the actual state vector, which is provided by the overhead vision system. The Proportional-Derivative controller can be used due to the simplicity of the plant (this is basically a $1/s^2$ plant, so no integral control is needed [8]), and the availability of a high-fidelity velocity signal. The PD controller output, F_{des} is sent to the Thruster Mapper, resulting in the thruster pattern, T . This T is then implemented on the robot.

This low-level portion of the control system, consisting of the trajectory generator, PD controller, thruster mapper, and position sensor, is always running, and does not have adaptive capability. The adaptive system is highlighted in Figure 3.1, and consists of three components: sensors, an identification process, and a controller redesign process. The accelerometers and angular-rate sensor produce a base acceleration measurement vector. These signals, along with the thruster firing signals, are used by the identification process to update a model of the robot’s thruster characteristics. This model is periodically sent to a control redesign process that generates an updated thruster mapper based upon the updated robot model. This updated thruster mapper is periodically copied to the thruster

mapper running in the control loop, as indicated by the double arrow. The control, identification, and controller-redesign loops are all running concurrently. Due to the possibility of a destabilizing failure, there is not enough time to wait to generate a new updated plant model before redesigning the controller.

So far, this structure makes no mention of neural networks. The factors involved in the decision of where to use neural networks are outlined below. In this application, a recursive least squares linear regression ID component was used, since identification of the thruster characteristics is a linear process. The algorithm used to obtain acceleration measurements was nonlinear, but could be derived analytically, so no neural network was used there either. A neural network was used for the thruster-mapping component since it is an inscrutable nonlinear function that requires adaptation. The control redesign process is therefore a backpropagation-based neural-network training algorithm.

The neural network is used precisely at the location where it is beneficial: the thruster mapper. If the robot were to remain perfectly symmetric, with no degradation, and it was restricted to in-the-plane motions with 8 thrusters, the symmetry-assisted search would work well enough, and no neural network would be required at all. In this application, the benefits of the neural network approach are required only if the symmetries are lost and adaptation is required.

The selection of this system architecture, and the following development of a neural-network-based reconfigurable control system present one specific example of a successful application of neural networks for control. However, the decisions of how to structure the control system, and where and how to use the neural network are more general: The lessons learned during the construction of this system may in fact be applied to any candidate neural-network control application. For example, although this application used an indirect adaptive control structure, the methodology that follows is not restricted to this architecture.

3.2 Cost/Benefit Analysis

To determine where neural networks can contribute effectively, the control systems engineer must consider the strengths of neural networks (nonlinear, adaptive, generic, unstructured, parallelizable) as well as the costs associated with these benefits (difficult to understand workings or prove stability, design is iterative, computationally complex). The cost/benefit

balance must be evaluated on an application by application basis. First at the system level, the system requirements and considerations of degree-of-nonlinearity, adaptation requirements, and computational complexity, etc., lead to a candidate system architecture. Then at the component level, this cost/benefit analysis is repeated, leading to the decision of what sort of subsystem will be used in each segment of the control system.

Before evaluating the applicability of neural networks for a control (or other) application, it is useful to examine, in more detail, the specific costs and benefits of neural networks, since these are what will be weighed in the design decision.

3.2.1 Benefits of Neural Networks

- **Nonlinear** – Since neural networks tend to be designed with an iterative gradient search, they can handle nonlinear internal and external (e.g. system to be controlled) components just as easily as linear ones.
- **General** – The most common neural-network architecture, the multi-layer perceptron (feedforward network with sigmoidal activation functions) has been proven to be capable of representing any MIMO function to an arbitrary degree of accuracy. This was presented by Hornik et. al. in “Multilayer Feedforward networks are universal approximators” [19]. This generality is important when neural networks are developed in software, but also for hardware implementation, where the ability to build multi-purpose ICs is valuable.
- **Unstructured** – Unlike a linear mapping or Fourier transform, there is no pre-specified structure to the computation a neural network can perform. The structure is developed during training as the network parameters are set, defining the strength (or existence) of connections between neurons.
- **Parallelizable** – Neural networks are designed to be implemented in parallel hardware. In most applications, they are developed in software, and implemented on serial-computing hardware, since that presents a more convenient development environment, and most of the effort is spent during the design and development phase. Hardware implementation then has the potential for vast improvements in processing throughput. An additional benefit of parallel hardware implementation is that the network is robust to partial processor failure. For example, in a space application,

if cosmic rays were to destroy a few of the neurons, it is unlikely that the output would be significantly affected, since the output is determined by contributions from thousands or millions of neurons. Additionally, the remaining neurons would be able to adapt to compensate for the damage.

3.2.2 Costs Associated With the Use of Neural Networks

- **Black Box** – The functionality of a neural network is defined by the connection strengths, i.e. a large number of parameters. This, coupled with the fact that they are nonlinear, means that it is difficult to understand what they do. It may be possible to verify the network's performance for a sufficiently large range of conditions, leading one to trust that the network will work well, but it is not easy to understand why the network does what it does (contrary to a simple linear controller, where it is often possible to study the gains or poles and zeros to form an understanding of the function of the controller, and perhaps why the automatic design process chose that function).
- **Stability Proofs** – Due to a neural network's nonlinearity and complicated structure, it is virtually impossible to develop rigorous stability proofs for it. This is a big concern for control systems that put high demands on stability, such as aircraft and spacecraft. One way to address this problem is to have a high-performance neural-network control system with a backup low-performance linear controller that has been proven stable. If instability were ever detected, control authority would be switched to the low-performance system.
- **Iterative** – Function-based neural networks, such as those described in this thesis, are not calculated in one step, but are developed through an iterative process known as training or learning. This takes time, and since it is a nonlinear optimization, convergence to a global minimum is not guaranteed. Fortunately, the local minimum is rarely significantly worse than the global optimum. The FCA, presented in Chapter 4, addresses both of these problems: by pre-programming in a linear solution, the initial training performance is as good as the best linear solution; also, starting the network close to a reasonably good solution makes it less likely that the optimization will terminate in an undesirable local minimum.
- **Computational Complexity** – The neural network may have excess neurons or connections, thereby offering more functional complexity than is needed. This results

in slower execution, and creates a susceptibility to overfitting (poor generalization). Fortunately, many network pruning methods are available that eliminate the excess complexity, but this remains a complicating issue.

The specific costs and benefits vary between different types of networks. For example, memory-based networks do not have the iterative cost mentioned above, as they just store all sensor information and recall the relevant information when needed. Also, some neural networks may be better than others for a specific problem – for example, MLPs vs. RBFs, as discussed in Chapter 1.

3.3 Criteria For Valuable Application of Neural Networks

Study of these costs and benefits, the focussed (experimental) experience with the robot application, and examination of other successful neural-network applications has led to the following summary. It is a concise list of criteria for an application where use of neural networks will be advantageous. The application should be:

- Nonlinear – The powerful *nonlinear* capability of neural networks comes at the significant cost of computational complexity, slow convergence speed, and lack of provability. If no advantage will be obtained from this capability, it should be avoided.
- Inscrutable – The fact that neural networks provide a general nonlinear function-approximation capability makes them particularly valuable for problems where the nonlinearity is *inscrutable*. If the exact form of nonlinearity is known (e.g. \sin , \cos , quadratic functions, etc.) it should be used explicitly; however, this may not be practical if the speed requirement calls for parallel hardware. For example, if 10,000 $\sin(x^2 + y^2)$ operations are needed at a 1 MHz update rate, parallel hardware is required, and it may not be feasible to custom design an Application-Specific Integrated Circuit (ASIC) for this application, where it may be feasible to train a neural network chip to emulate this function.
- (possibly) Requiring Adaptation – Since neural networks are generally trained iteratively based upon some form of error feedback, they are already set up for *adaptation* to changes in the plant or environment. Therefore adaptive capability can be added with minimal effort, enhancing their applicability in adaptive control situations.

or

- Requiring parallel hardware (processing speed) – The availability of parallel neural network hardware may make a neural-network approximation to even a known non-linear function (for which parallel hardware does not exist and is not efficiently implemented using a microprocessor or programmable logic device) highly advantageous.

An understanding of the alternative methods (statistics, linear adaptive control, etc.) is useful for determining whether the benefits a neural network can offer outweigh the costs for each application. It is common to see examples in the literature of neural network control systems used where a linear adaptive controller would have been easier to implement, and worked better. It is also common to see flawed justifications for neural control like “this is a difficult control problem that has not been solved using conventional methods, so we propose to use a neural network, (simply) because neural networks can do things conventional methods cannot.”

Once it has been determined that the application can benefit from the use of neural networks, these same principles should be used to determine which segments of the overall control system are advantageously implemented with neural networks and which are not. (This is the essence of the optimal hybrid system concept.)

In applying these principles to the robot control application, the conclusion is that a neural network will be beneficial. As mentioned in the previous section, the task is to develop an approximation to the optimal thruster mapping, which can be calculated optimally, but is too complicated to run in real time. This mapping is indeed both highly nonlinear and inscrutable, and does require adaptation in response to changes in the thruster characteristics. Limitations of the neural network approach for speed of reconfiguration, and training with the on-off thrusters, will be addressed with extensions to neural network theory in those areas.

2

Chapter 4

Fully-Connected Architecture

A number of issues are present in the thruster mapping control application discussed in Chapter 2 that are common to many neural network control problems.

- Prior information about the system exists, and it should be possible to exploit this information when generating the neural network.
- Initial learning speed is important if the neural network will be trained on-line.
- The neural-network topology (the number and connectivity of neurons) required to achieve an accurate mapping without over-fitting is unknown beforehand.
- Some of the control outputs (thruster values) influence one another (e.g., directly opposing thrusters should never fire together).

The most relevant of these features for the robot control application are the first and second ones. Reconfiguring in response to a destabilizing thruster failure places a high premium on speed of adaptation. The architecture presented here allows *immediate implementation* of a linear solution that is calculated using conventional methods. This provides a low-performance, but immediately-stable controller to use as a starting point in the optimization.

In this chapter, a general neural-network architecture that addresses these issues is suggested. This “Fully-Connected Architecture” is for feedforward neural networks that can be trained using backpropagation [46] [60], and refers to the structure shown in Figure 4.1. It was first presented by Werbos [61], and initially developed in a control context by Wilson and Rock [71].

The extra connectivity of this architecture, which is unavailable in a layered network, allows seamless integration of linear *a priori* solutions, communication among input and output neurons, and greater overall functionality than a layered network. The increase in parameters can exacerbate over-fitting problems, and a systematic complexity-control method is successfully demonstrated that lessens this problem.

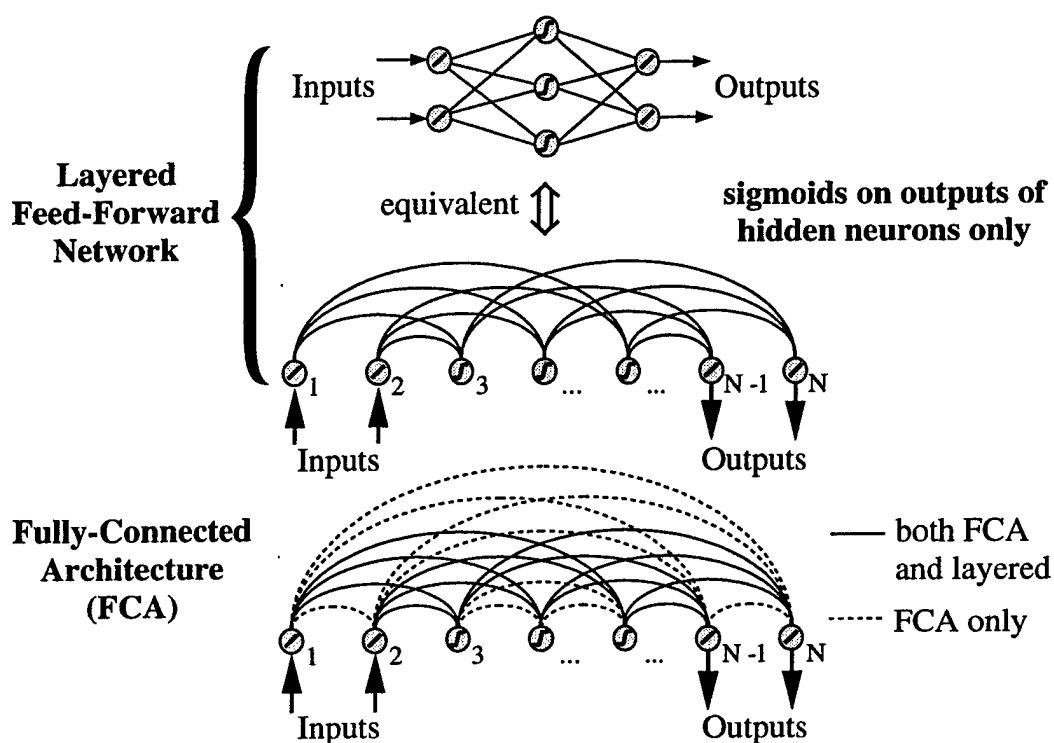


Figure 4.1: Extra Connections Available with FCA

This general feedforward architecture subsumes more-familiar single or double-hidden-layer architectures. Here, the FCA is shown to have all the connections of a single-hidden-layer network, and some extras as well. The network's neurons are considered to be ordered, beginning with the first input, ending with the last output, and having hidden units in between, perhaps interspersed among input or output units. Note that there is no longer a concept of layers. Backpropagation restricts information flow to one direction only, so to get maximum interconnections, each neuron takes inputs from all lower-numbered neurons and sends outputs to all higher-numbered neurons.

4.1 Background

In the literature, the term “fully-connected feedforward neural network” usually refers to a *layered* network, with an input layer, one or more hidden layers, and an output layer. “Feedforward” indicates that signals flow from the input layer, through hidden layers, and to the output layer in one direction only, which is required by the backpropagation algorithm. “Fully-connected” indicates that every input is connected to every neuron in the first hidden layer, and so on between successive layers. While this layered architecture may be particularly well suited for many applications and certain hardware implementations, a more general structure may be able to take advantage of the full capabilities offered by the backpropagation algorithm [46].

In this work, the term “fully-connected” will refer to the structure shown at the bottom of Figure 4.1. Instead of layers, a fully-connected network can be considered to have neurons that are ordered, beginning with the first input, ending with the last output, and having hidden units in between, perhaps interspersed among input or output units [61]. Backpropagation restricts information flow to one direction only; so, again, to get maximum interconnections, each neuron takes inputs from all lower-numbered neurons and sends outputs to all higher-numbered neurons. For example, the last output neuron takes inputs from all the hidden neurons, just as in a layered architecture; however, it now also takes inputs from each of the input neurons and previous output neurons.

The main benefit is not that it maximizes the connections-to-neurons ratio, but instead that, when combined with a systematic weight-pruning procedure, it allows a more flexible use of layering. There has been a recent trend in using not one but two hidden layers; the FCA is a generalization of that trend.

In the application addressed in this work, the extra connections are found to be useful when coupled with a procedure to control over-fitting. In particular, the 3×4 matrix in the upper right corner of the weight matrix shown in Figure 4.2 provides direct linear information flow from input to output (sigmoids are used only for the outputs of hidden neurons), and the 3×3 upper-triangular matrix in the lower right corner provides communication between outputs. While these functions could be provided with processing components in series or parallel with the network, the fully-connected architecture provides a seamless integration of these capabilities.

4.2 Comparison with a Layered Network

Figure 4.2 highlights the benefits of the extra connections that are unused in a single-layered network.

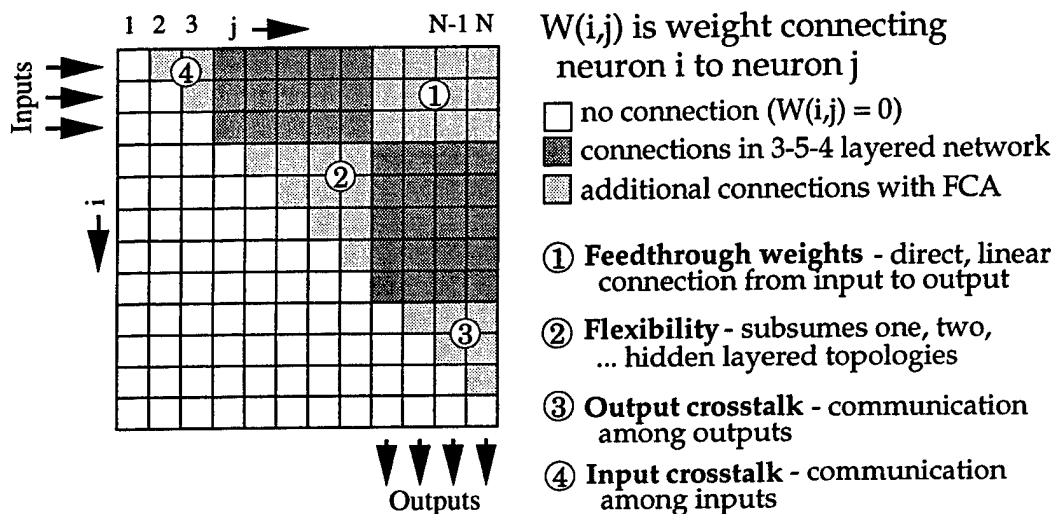


Figure 4.2: Weight-Matrix Representation to Highlight Benefits of FCA

- **Feedthrough Weights:** this segment, shown in region 1 in Figure 4.2, is a matrix that implements a direct, linear connection from inputs to outputs (sigmoids are used only on hidden units). This provides *fast initial learning* and allows *direct pre-programming* of a linear solution calculated by some other method. This is particularly important for control applications, where there is a large body of linear control knowledge that can be drawn upon to provide a good starting point. The FCA provides for seamless integration of linear and nonlinear components.
- **Flexibility:** since the FCA subsumes any number of hidden layers, when combined with a systematic weight-pruning procedure, the network topology (defined by the remaining connections) is set in a systematic manner based on gradient descent. The weights shown in region 2 of Figure 4.2 represent the flexibility of the FCA, in that the connections may be configured to provide one and two hidden layer topologies (in general, any feedforward network topology).

- Crosstalk among inputs and outputs: these connections, shown in regions 3 and 4 of Figure 4.2 may be valuable, i.e. one output may excite or inhibit another output, a feature unavailable with layered networks.

The “disadvantages” (i.e. issues which must be addressed) of the FCA include:

- Increased complexity: number of weights increases quadratically with the number of hidden units, versus linearly for a layered architecture. The extra weights increase susceptibility to over-fitting.
- Slower hardware implementation: updating must be one neuron at a time, versus one layer at a time for layered networks.

This general architecture makes full use of the backpropagation algorithm, while still allowing the use of modifications, such as the use of FIR connections in place of weights [57] or backpropagation through time [38]. Figure 4.1 shows the extra connections that are unused in a single-layered network. The question is whether the benefits of the enhanced functionality outweigh the increased computational load and susceptibility to over-fitting. This must be decided for each application. A more detailed description of each of these features of the FCA follows.

4.2.1 Feedthrough Weights

For the robot control application, the most important aspect of these connections is that they provide a means for directly pre-programming the network with a pre-calculated linear solution. This results in fast reaction to a destabilizing thruster failure. Initializing the network to a good linear solution may result in a better final solution, as described below. Another benefit is that the feedthrough weights make it easy for the network to implement a linear solution, so the FCA will work well when the actual solution has a strong linear component (a common situation) superposed with a nonlinear correction.

Motivation: Infusion of Prior Knowledge

Much is already known about how to find linear approximate solutions to many problems, both in control, and elsewhere. Often, the standard solution is a linear one, and there are many highly advanced, very powerful, *linear* design tools available. However, for many real-world problems, there are significant nonlinearities, and often the fallback procedure is to

use a linear controller designed for a linearized plant. Nonlinear design methods exist, but certainly not at the level of linear ones. One of the purported benefits of neural networks is that they address this problem with their adaptive, nonlinear approach [36].

Although a network often can be trained to solve a problem starting with no prior information, taking advantage of the (often abundant) linear theory can improve the learning rate and provide a better solution if properly presented to the network.

Beginning the network at a reasonably-good starting point can lead to a better final solution if it prevents the network from getting stuck in an unfavorable local minimum. This can also be useful as a learning guide. When Nguyen and Widrow trained the original truck backer upper [38], the initial learning runs were made with the truck pointed at, and a few steps away from the loading dock. After mastering this easy task, the initial conditions were made progressively more difficult, leading the control system through a gradual learning process. Backpropagation-through-time training for unstable systems like the truck can benefit greatly from some outside direction of the learning process. The teaching process used by Nguyen and Widrow, and linear initialization of an FCA network is another.

In general, it is possible to use existing linear control theory to form a linear solution to a problem (possibly a linearized version of a nonlinear problem). In many cases, this solution will in fact be a reasonable solution to the full nonlinear problem. The feedthrough portion of the weight matrix offers a direct vehicle to import and implement this linear solution as part of the neural network. Similar alternative techniques to building in knowledge include first training the (layered) network to emulate the linear solution, then adapting from there, or running the linear solution in parallel with the network. One benefit of the FCA approach is the seamlessness of the network-linear solution integration – it immediately becomes part of the network. Adaptation to this portion of the network can be turned off, use the same algorithm as the rest of the network, or use an adaptation algorithm based on linear theory.

Approximate Linear Solution: Thruster Mapping Example

A simple example of this situation exists here: the exact solution to the thruster mapping problem is highly nonlinear and complex, but there is a linear approximate solution that may be easily calculated. The feedthrough weights of the fully-connected network architecture simplify infusion of this *a priori* knowledge.

The *a priori* linear solution used here was found by assuming that the thrusters are capable of continuous-valued thrust output (a linearized version of this problem). The solution is simply a 4×3 pseudo-inverse of the 3×4 matrix that maps reaction forces, R (the set of four $[-1,0,1]$ thrusters), to base forces, F . Recognizing that the direct feedthrough segment of the fully-connected network provides exactly this computation (output = weight matrix \times input), it is possible to incorporate this *a priori* knowledge by putting the pseudo-inverse linear solution directly into that sub-matrix, as an initial condition for the weight matrix. This linear mapper is then rounded off to the actual thrust positions possible at run time $(-1,0,1)$.

The problem is more complex if thruster failures are allowed, and the one-sidedness of the thrusters is considered. For example, the linear approximate solution may request negative thrust from a thruster, which is not physically possible (certainly in space, and practically elsewhere). The approach taken here is to find when negative thrusts are requested and attempt to reassign these thrusts to positively-valued thrusters. This is done exactly when two opposing thrusters exist, but is inexact when an opposing thruster does not exist for each thruster. Since this provides only the starting point for adaptation, it is not critical that the linear approximate solution is optimal. A solution that considers one-sided continuously-valued thrusters is presented in [25]. This was developed for the Gravity Probe B satellite, which is unique in having proportional thrusters, rather than on-off thrusters¹.

Approximate Linear Solution: General Case

Alternatively, if a linear solution is expected to work well, but cannot be found through analysis, the network can find one adaptively. This involves zeroing all weights except the feedthrough ones, and using the standard backpropagation algorithm. At this point, with a linear problem, convergence will be very fast, as the cost function is parabolic (for direct supervisory training). This increase in initial learning rate can be valuable for certain real-time applications, both on start-up, and after a significant change in the system, where it is critical to find a stable solution very rapidly. Once the system is stabilized (if this is possible with a linear controller), the rest of the network can be freed up to deal with the nonlinearities.

It is not necessary to zero the rest of the weights when training the linear portion – the linear weights initially learn at a much greater rate than the others when all are

¹The satellite carries liquid helium that boils off slowly and must be expelled anyway.

subjected to the same training algorithm. This effect is explained below, and can be seen in Figure 4.3, which shows connection activation levels at various stages during training. If implemented on a serial processor, and speed is an issue, it may be useful to skip these extra computations during the initial learning phase, since they do not contribute much to the network performance.

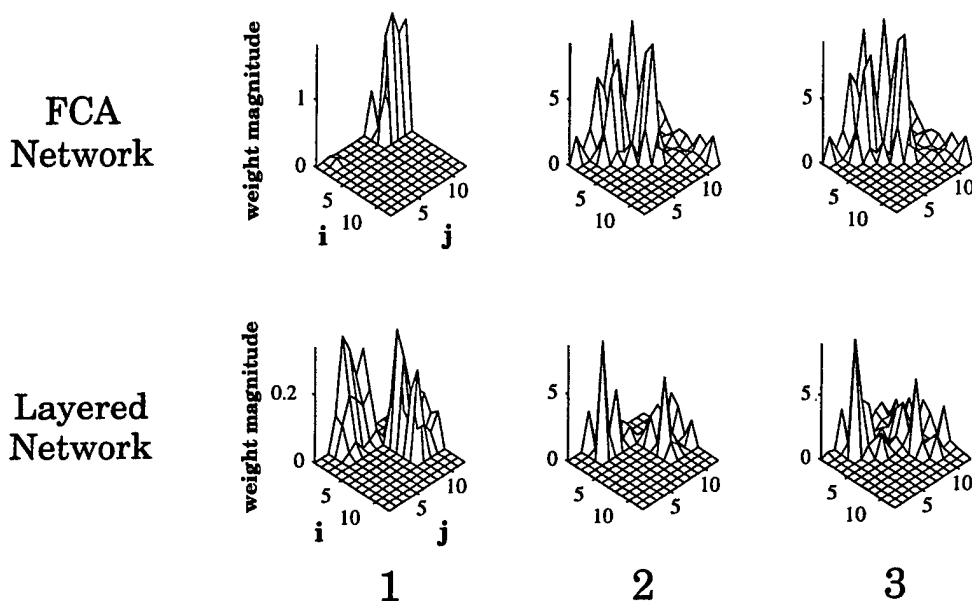


Figure 4.3: Network Connection Activity During Training

Mesh plots show the magnitude of network connections (weights). A weight matrix format is used, as in Figure 4.2. Fully-connected networks are in the top row, layered networks in the bottom row. First plot is after 25 epochs. Second plot, top, is after training with the feedthrough connections frozen to the linear solution. Second plot, bottom, is after training the layered network. Third plot, top and bottom, are the final solutions (local minima) after all weights were allowed to adapt.

A weight must contribute significantly to the output before the resulting error signal will cause it to change significantly. If all weights are started small, the feedthrough weights learn fastest, since the input and output information provides an immediate error-gradient signal. Once these signals build up, the crosstalk weights receive strong learning signals and begin to adapt. Starting all weights with an initial condition of zero will allow the feedthrough and crosstalk weights to adapt, but all other weights remain at zero throughout the learning

because there is no error signal due to these weights to stimulate learning. It is common in network training to initialize these weights to some random values. Choosing the initial condition for the "random" weights is a problem in itself. The method presented by Nguyen and Widrow [37] has proven to be valuable in this application.

Once the feedthrough weights have been found, either analytically or adaptively, they can be frozen or allowed to adapt, depending on the problem. In the case of an analytical solution, an adaptive algorithm distinct from backpropagation may be appropriate.

4.2.2 Value of Cross-Talk Connections

In addition to the value of linear feedthrough connections, the upper triangular matrices contribute by providing the capability for crosstalk among outputs and among inputs. These weights allow one output to excite or inhibit a higher-numbered output. As a clear example for the thruster mapping problem, if the network were to have an output (0,1) for each of the eight thrusters, and during training, a penalty was put on gas use, the network could use this segment to allow the firing of one thruster to prohibit the firing of the opposite thruster (which would provide zero net thrust and waste fuel). This is so clear that in this case, it could perhaps most-easily be implemented by manually programming these weights, although the network would eventually learn this as well. The example illustrates the value of crosstalk between input and output neurons that is unavailable in a layered network. Another example would be the capability to select between redundant output patterns: if $[1 \ 1 \ 0 \ 0]$ and $[0 \ 0 \ 1 \ 1]$ both produce the same net force, they may both be equally likely to activate when that force is requested. This could result in either $[1 \ 1 \ 1 \ 1]$ or $[0 \ 0 \ 0 \ 0]$. The crosstalk would allow the network to use the first output to send it to either of the acceptable solutions, and avoid the ambiguity.

Crosstalk between all outputs would be nice, but backpropagation limits us to unidirectional information flow. This may make it important to select carefully the ordering of inputs and outputs. If more complicated, nonlinear crosstalk is desired, extra neurons may be placed between individual output or input neurons.

4.2.3 Hidden-Neuron Interconnections

FCA Generalizes the Concept of Hidden Layers

The FCA is a generalization of the feedforward layered network. It therefore subsumes layered networks with any number of hidden layers, i.e. it has all the functionality of a two or three-layered network. This can be seen in Figure 4.4, which shows how two and three-layered networks can be represented by the FCA.

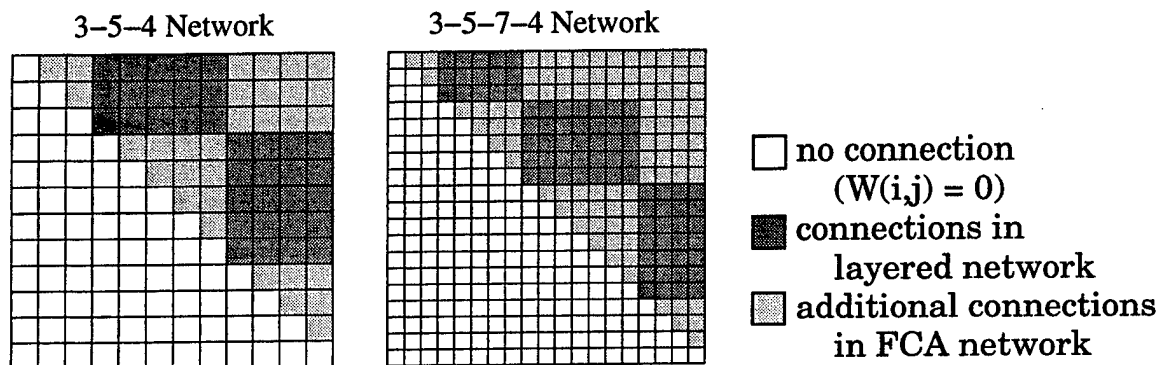


Figure 4.4: FCA Subsumes Any Feedforward Layered Network

The FCA is shown to include (as subsets) all the connections available in two or three-layered networks. In general, it subsumes any feedforward network topology. The matrix representation here is similar to that in Figure 4.2.

Since the FCA subsumes any number of hidden layers, when combined with a systematic weight-pruning procedure, the network topology (defined by the remaining connections) is set in a systematic manner based on gradient descent. The weights shown in region 2 of Figure 4.2 represent the flexibility of the FCA in that the connections may be configured to provide one- and two-hidden-layer topologies (in general, any feedforward network topology).

This flexibility is valuable, since often it is not known *a priori* which network topology is best-suited for the application. Coupled with a systematic network pruning method (presented below), the FCA allows for the network topology to be automatically chosen.

One Hidden Layer or Two?

The topology of a network can have a significant impact on the functional capabilities of the neural network. It is generally accepted that at least one hidden layer is necessary to

perform mappings that are not linearly separable. However, the decision to use one hidden layer, two, or more, is an active area of research [1] [5] [7] [13] [19] [21] [26] [30] [35] [41] [49] [50] [55] [58]. This section presents some background in this area. There is no consensus among the researchers – the number of hidden layers needed appears to vary from one application to another. Fortunately, since the FCA subsumes all layered networks, this issue is not so critical if the FCA is used with a systematic network pruning algorithm.

In “On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition,” A.N. Kolmogorov presents a mathematical proof regarding the functional complexity of neural networks. He shows that a one-hidden-layer network with $2n + 1$ hidden neurons (where n is the number of inputs), can implement any continuous mapping from n inputs to m outputs [28]. This is important, since it provides a mathematical foundation for the functional capabilities of neural networks, but there are two difficulties: (1) The nonlinear activation functions of each of the hidden neurons is not specified; (2) He does not show how to find the weights or nonlinear functions.

In “Multilayer feedforward networks are universal approximators,” Kurt Hornik, Maxwell Stinchcombe and Halbert White show that any function can be universally approximated to arbitrary accuracy using a neural network with only one hidden layer [19]. This requires that the network has “sufficient” hidden units, but no method for determining the number of hidden units is given. Additionally, there may be cases where a network with more than one hidden layer can implement the mapping more efficiently (using fewer neurons and connections, although more layers). This is more applicable than Kolmogorov’s work, since the authors worked with standard sigmoidal nonlinear activation functions.

In “Feedback stabilization using two-hidden-layer nets” [50], E.D. Sontag shows that while single-hidden-layer networks may be sufficient to implement direct input-output mappings, double-hidden-layer networks are required (to guarantee that it will work in the general case) to implement one-sided inverses of continuous mappings. This is especially important in control problems, where it is common to invert a plant model. This is the case in the thruster mapping, where the thruster mapper is an inverse of the thruster-to-force mapping defined by the thruster parameters.

In “Why two hidden layers are better than one” [9], D.L. Chester presents an example where a simple two-hidden-layer network is sufficient, but an infinite number of hidden neurons would be required if a single hidden layer were used.

In “Threshold circuits of bounded depth”[15], A. Hajnal presents problems requiring an exponential number of nodes in a single-hidden-layer network, but a polynomial number of nodes in a double-hidden-layer network.

As far as using one or two hidden layers for specific applications, different researchers have found success with both architectures. In [21] and [30], networks with one hidden layer are found to perform better than those with two hidden layers. In [26], [41], and [55], networks with two or more hidden layers are found to perform better.

Since the decision to use one or two hidden layers is simply not an issue with the FCA, the lack of consensus on this issue is not a major concern.

4.2.4 Learning Performance: FCA vs. Layered

Figure 4.5 compares learning histories (thruster mapping error on the training set) for the thruster mapping problem (with direct training) outlined in Chapter 2. Three networks are compared, each with 5 hidden neurons. Each was trained to emulate the optimal mapping (minimizing force error). Training a neural network is an iterative nonlinear optimization, and will usually produce a different result each time it is run, provided with a different initial condition. For this reason, results are presented as the average of several runs, each from a different initial condition of the weights. In this plot, each curve in the figure represents the average performance for ten different sets of initial weights.

This is the direct training problem mentioned in Chapter 2. Even though indirect training is the ultimate objective, in order to demonstrate the performance of the FCA, the direct training problem is studied here first. Direct training is much simpler, while still containing all of the architecture issues to be found in the indirect training problem.

Looking at the initial learning performance, the FCA network performs better than the layered network, due to the weight gradient being instantly available *via* the direct connection of inputs to outputs. As expected, the FCA network with the *a priori* linear solution built in provides the best early performance. Although the randomly-initialized networks catch up fairly quickly here, this initial head-start can be critical for a control application because it can mean the difference between stability and instability. This will be demonstrated later, in Chapter 6.

In the middle region, between 100 and 1000 epochs, the layered network performance surpasses that of the FCA, due to the reduced number of parameters, and simplified search space. However, after 1000 epochs, the greater functionality of the FCA network comes into

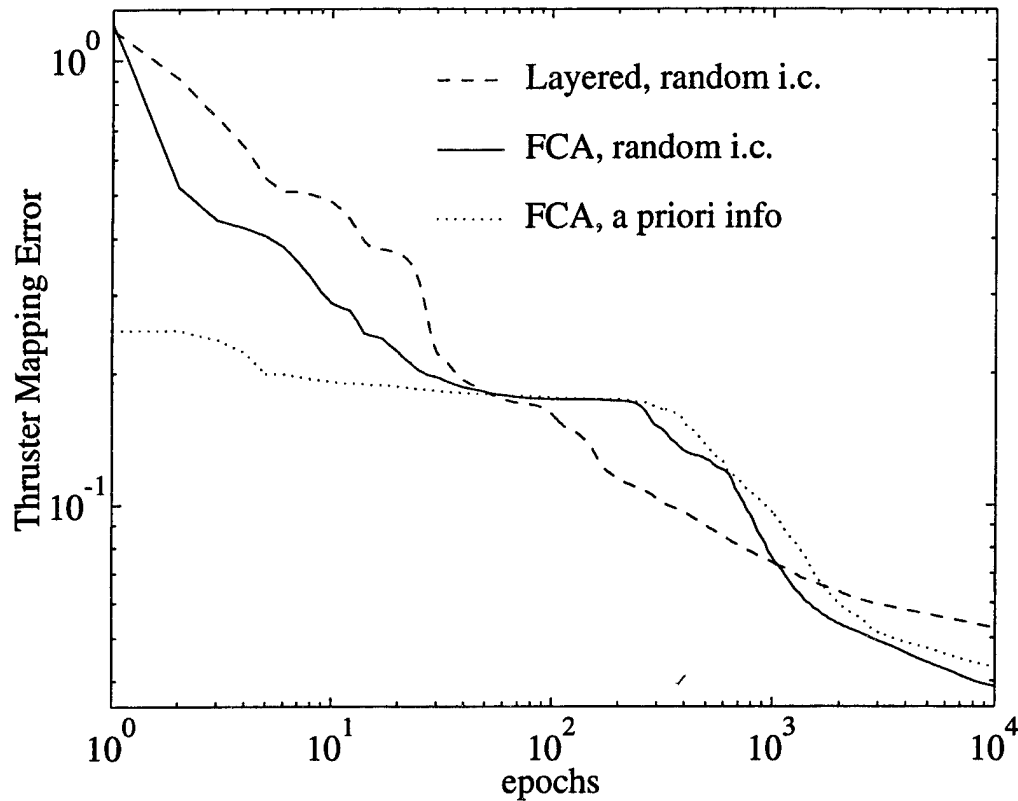


Figure 4.5: Training Performance Comparison

The fully-connected network (FCA) learns much faster at first, due to the linear connections. After the initial surge, the layered network passes it due to the reduced number of parameters and resulting faster learning. Towards the end, the fully-connected network's performance is significantly better – highlighting its extra capabilities. This is not surprising, since the FCA network subsumes the functionality of the layered network. The network initialized with the linear solution begins with significantly better performance.

play, and performance surpasses that of the layered network. This is of course expected since the FCA network has all of the connections of the layered network in addition to the extra ones described earlier. The FCA network with the *a priori* solution frozen in has slightly worse final performance, since the feedthrough weights are not adapted in this case.

4.3 Architecture Selection to Avoid Overfitting

4.3.1 Overfitting

The above has shown the potential value of the extra connections associated with a fully-connected neural network, both in faster initial learning and in better final performance. However, the high number of parameters, while increasing functionality, makes the network susceptible to over-fitting. A layered network with i inputs, h hidden neurons, and o outputs has $(i+o)h$ weights, while a fully-connected network has $((i+h+o)(i+h+o-1)/2)$ weights, not counting bias weights. More parameters to adapt means the network will be slower to train, and possibly susceptible to overfitting. This is an important concern with the FCA, and must be addressed.

A common method for evaluating the level of overfitting is to use a method known as “cross-validation.” In this method, a set of input-output data (known as the “test set”) is kept separate from the set of data used for training the network (known as the “training set”). Periodically, the network’s performance on the test set is evaluated. A decrease in test-set performance coupled with an increase in training-set performance indicates overfitting. At this point, the weights in the network have begun to adapt to the particulars of the training set (e.g. noise or lack of sufficient data), rather than forming a generalization of the full population from which the training samples are chosen.

Figure 4.6 shows how overfitting affects performance for different training set sizes. Overfitting becomes clear when the performance on the test set remains the same or worsens, while performance on the training set improves. It is common that during training, performance on test and training sets will improve until a certain point is reached when the network stops generalizing, and begins to fit the particular data set. Use of a “sufficiently-large” training set can reduce over-fitting problems, but this may not be practical due to a lack of data, or to an adaptation speed requirement that needs a faster solution than this data-intensive brute-force approach.

4.3.2 Systematic Complexity Control

When training function-based neural networks such as this FCA, the goal is to achieve good *generalization* by presenting the network with a large number of sample input patterns along with the desired outputs. The hope is that the parameters that define the functionality of the network will adapt to fit this training data, and will then respond correctly when

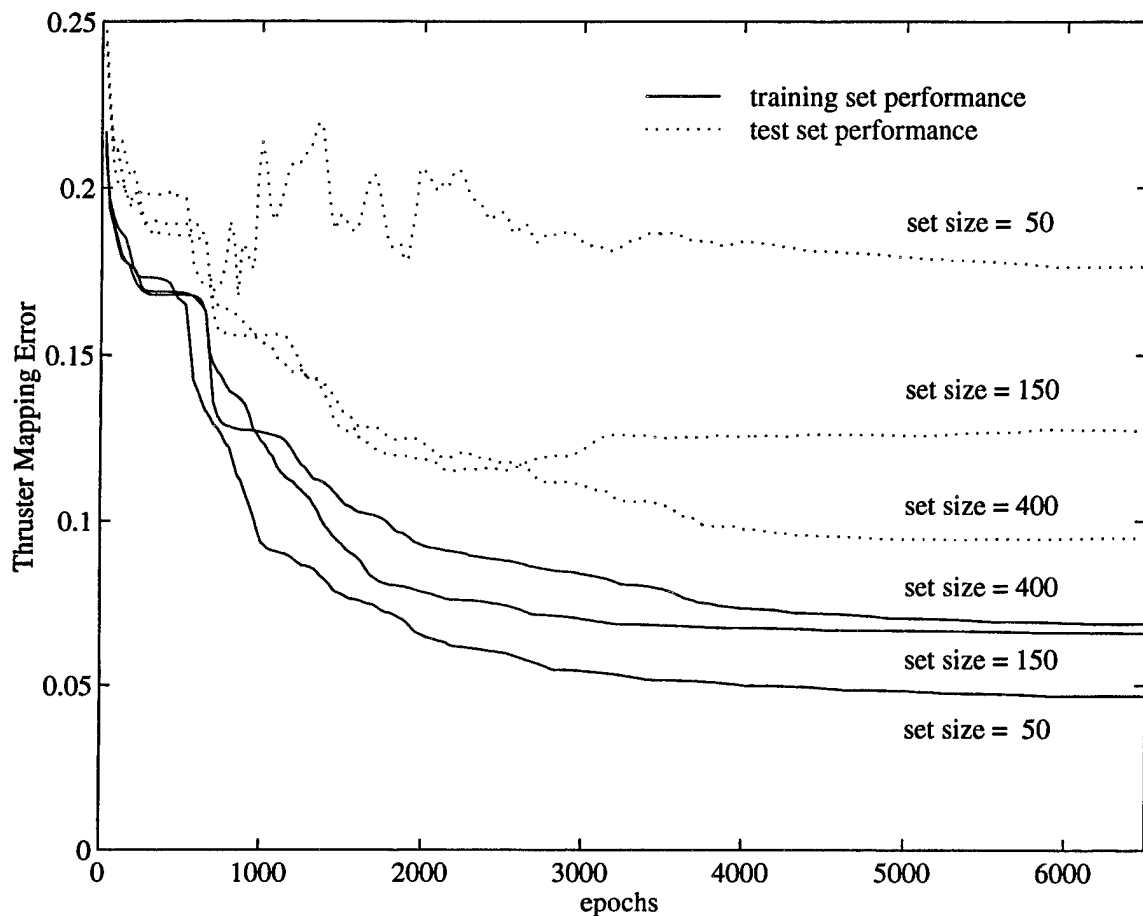


Figure 4.6: Training History, Performance on Test and Training Data

Overfitting, as seen by the divergence on training and test performance, is more of a problem for small training sets.

presented with new input patterns. The danger of overfitting arises when the network has an excess of parameters to fit: the danger is that these parameters will be used to fit the noise in the data and lead to poor generalization.

It is generally accepted that the fewer parameters used in the model, the less chance of excess functionality being used to fit noise, resulting in better generalization. The task now is to find out which connections are required to implement the desired mapping, and build a network using only those weights. The network architecture selection could be performed manually, but this would not be practical. For this problem, a network with feedthrough connections, weights corresponding to a layered network with five hidden neurons, and the

crosstalk connections on the outputs, would probably work well and not be susceptible to overfitting, given a good training set.

This heuristic approach may overlook some valuable extra connections, and may still result in overfitting, so a systematic network-pruning technique is desired. One that was found to be successful involves a modification of the cost function that the neural network is trained to minimize. This approach was first proposed by Rumelhart and Weigend [44] [59].

The cost function is augmented with a term that places a cost on the complexity of the neural network (complexity is defined by a mathematical function of the weight values). The neural network is then trained to minimize this new cost function, using the same gradient-based optimization methods as before.

This complexity-control structure is based on the following assumptions:

1. The best generalization is the least-complex one that still performs an input-output mapping with an acceptable error. Therefore, there is a user-defined parameter to determine this balance between complexity and mapping performance.
2. The complexity of a mapping is related to the *number* of connections between neurons. Therefore, the cost associated with each connection is zero when the connection is zero, monotonically increases as the weight magnitude increases, then plateaus at a large weight level. This way, the total complexity cost varies with the number of non-zero weights, rather than with the size of the weights. The relatively-small weights will be reduced towards zero, leaving the larger (and supposedly useful) ones unrestrained.

The complexity-control term is shown in Equation 4.1, and presented graphically in Figure 4.7.

$$J_{complexity} = \sum_{i=1}^N \sum_{j=i+1}^N \left(\frac{\left(\frac{w_{ij}}{w_0}\right)^2}{\left(\frac{w_{ij}}{w_0}\right)^2 + 1} \right) \quad (4.1)$$

where,

$J_{complexity}$ = complexity cost

i = number of neuron where connection originates

j = number of neuron where connection terminates

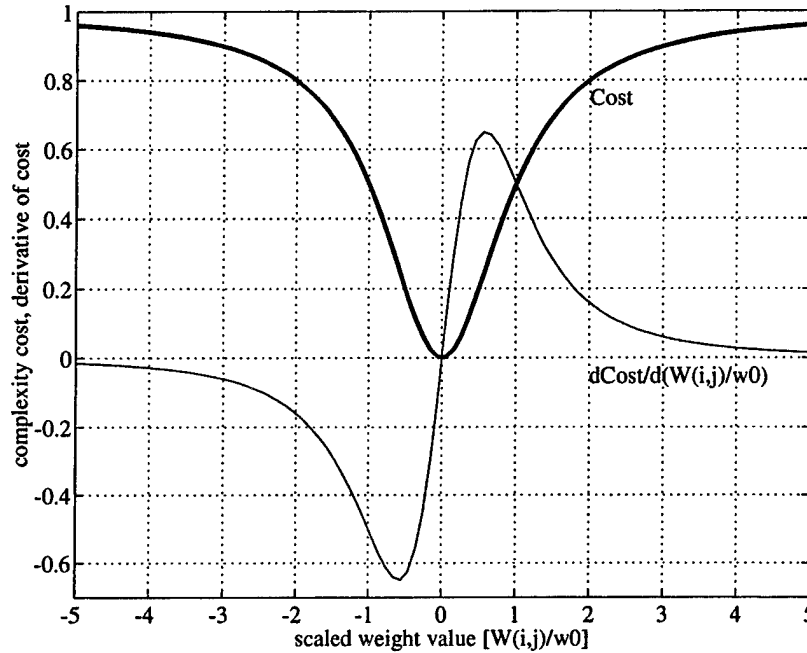


Figure 4.7: Complexity-Cost Function

Weights having values near zero cost little. Weights with high values (indicating that they contribute significantly to the network's function) cost λ , but the gradient is small, so there is little incentive to decrease them. Weights near the inflection point are small (they do not significantly affect network performance). The slope here is highest, so the network has the most to gain by decreasing them.

N = total number of neurons

w_{ij} = weight denoting the connection strength from neuron i to neuron j

w_0 = weight normalization parameter

(4.2)

Selecting the scale factor effectively sets the cutoff point for weights – it determines where the inflection point of the complexity cost function occurs. This defines the transition from a nearly-parabolic (for $w \ll w_0$) cost surface to one that asymptotically approaches (for $w \gg w_0$) a flat surface (i.e., with zero gradient). For $w \ll w_0$, weights are very-strongly driven to zero, whereas for $w \gg w_0$, the gradient is near zero, and weights are not restricted significantly.

Selecting a high w_0 will result in a nearly-parabolic cost function that keeps all weights from growing too large. In the parabolic section, the gradient acting against each weight is roughly proportional to the magnitude of that weight.

Selecting a low w_0 will have the effect of shutting off completely some of the weights, while not affecting the others. This parameter is selected iteratively by the user.

The complexity-control term is added to the total cost function with a weighting parameter, λ , as follows in Equation 4.3.

$$J_{total} = J_{performance} + \lambda J_{complexity} \quad (4.3)$$

where,

$$\begin{aligned} J_{total} &= \text{total cost function to be reduced by gradient-based optimization} \\ J_{performance} &= \text{network-performance cost function, such as shown in Equation 2.5} \\ J_{complexity} &= \text{network-complexity cost function, shown in Equation 4.1} \\ \lambda &= \text{complexity-cost weighting parameter} \end{aligned} \quad (4.4)$$

The weighting parameter, λ , is set by the user on an application-by-application basis to achieve the desired balance between performance optimization (e.g., thruster-mapping performance) and complexity minimization (i.e., to reduce overfitting problems). The parameter can be adjusted iteratively by observing performance on test and training data sets, such as those shown in Figure 4.8.

Equations 2.5 4.1 and 4.3 are combined, resulting in the total cost function shown in Equation 4.5.

$$\begin{aligned} J_{total} = & \left[\left(\frac{F_{xerr}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{F_{yerr}(\mathbf{T})}{F_{thruster}} \right)^2 + \left(\frac{\tau_{\psi err}(\mathbf{T})}{\tau_{thruster}} \right)^2 + \frac{1}{2} \sum_{k=1}^8 T_k \right] \\ & + \lambda \sum_{i=1}^N \sum_{j=i+1}^N \left(\frac{\left(\frac{w_{ij}}{w_0} \right)^2}{\left(\frac{w_{ij}}{w_0} \right)^2 + 1} \right) \end{aligned} \quad (4.5)$$

where,

$$\begin{aligned}
J_{total} &= \text{total cost function to be reduced by gradient-based optimization} \\
F_{x_{err}}(\mathbf{T}) &= \text{net force error in x-direction, } (F_{x_{des}} - F_{x_{act}}), \text{ resulting from } \mathbf{T} \\
F_{y_{err}}(\mathbf{T}) &= \text{net force error in y-direction, } (F_{y_{des}} - F_{y_{act}}), \text{ resulting from } \mathbf{T} \\
\tau_{\psi_{err}}(\mathbf{T}) &= \text{net torque error about } \psi\text{-axis, } (\tau_{\psi_{des}} - \tau_{\psi_{act}}), \text{ resulting from } \mathbf{T} \\
F_{thruster} &= \text{normalizing factor for } F_{x_{err}} \text{ and } F_{y_{err}}, \text{ force per nominal thruster} \\
\tau_{thruster} &= \text{normalizing factor for } \tau_{\psi_{err}}, \text{ torque per nominal thruster} \\
\mathbf{T} &= \text{binary thruster values, } \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 \end{bmatrix} \\
k &= \text{thruster number} \\
\lambda &= \text{complexity-cost weighting parameter} \\
i &= \text{number of neuron where connection originates} \\
j &= \text{number of neuron where connection terminates} \\
N &= \text{total number of neurons} \\
w_{ij} &= \text{weight denoting the connection strength from neuron } i \text{ to neuron } j \\
w_0 &= \text{weight normalization parameter}
\end{aligned} \tag{4.6}$$

The benefits of this method may be seen in the training histories shown in Figure 4.8. The network had five hidden neurons; and without any sort of complexity reduction, overfitting is clearly a problem, given the reduced training set. With the addition of the complexity term, overfitting was controlled, resulting in comparable performance on test and training sets.

The complexity control function and training histories for a fully-connected network with 5 hidden neurons are plotted in Figure 4.8. Without complexity control, overfitting becomes clear at around the 4000th epoch, as the performance on the test set worsens, while performance on the training set improves. With the addition of the complexity term, overfitting is controlled, as performance histories on test and training sets no longer diverge.

4.3.3 Other Complexity-Control Methods

Many systematic network-pruning techniques have been proposed and used successfully in certain applications. For, example "weight decay" uses a cost function like $\lambda(w_{ij}^2)$ to try to

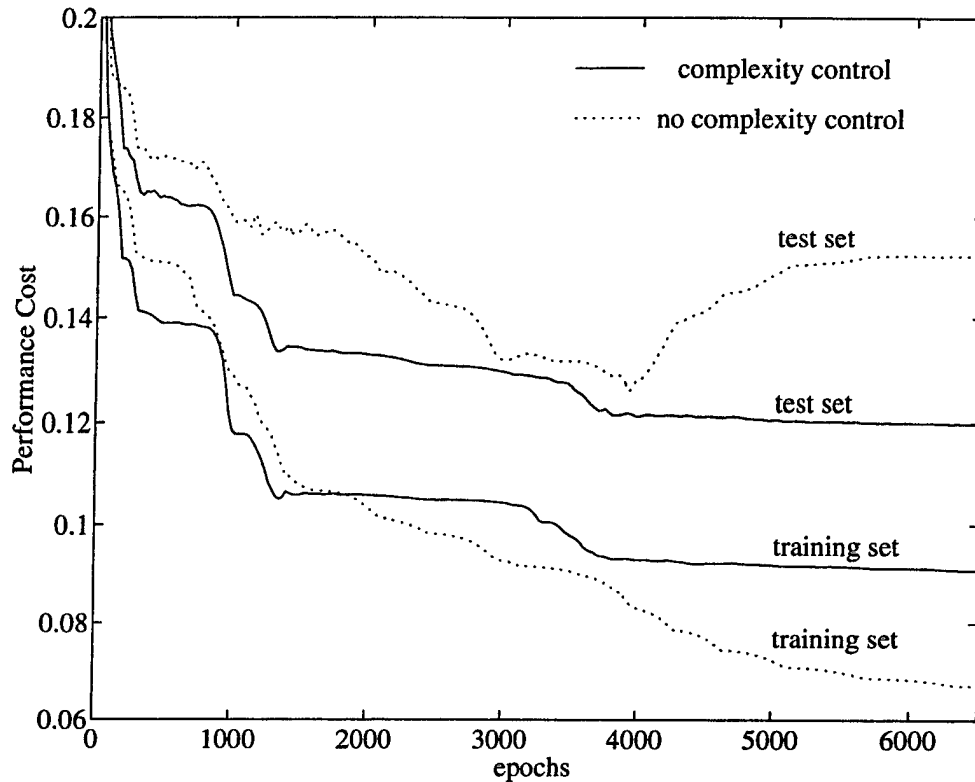


Figure 4.8: Complexity Cost Reduces Overfitting

In the case with no complexity control, the divergence of network performance on the test and training sets indicates overfitting. Addition of a complexity cost term is successful in controlling overfitting. Although training performance is worsened, performance on the test set is improved, which is of course the desired outcome.

reduce all the weights [39] [40]. Other methods completely eliminate connections or neurons in an iterative process [48] or with a genetic algorithm [65]. A survey of pruning methods is presented in [42]. The method used here has been shown to be effective in this application, but other methods may work as well or better at improving generalization performance.

4.3.4 Automatic Growing of the Network

The above-mentioned complexity control method works by selecting a network topology, and then trimming the excess connections to achieve the desired complexity. An alternative

is to begin with a small network, and add neurons to achieve the desired functionality. One advantage of growing a network is the potential for an increase in learning speed. With fewer hidden neurons, very quick learning takes place since fewer computations are required (this would not be true for a parallel hardware implementation, but is true for the more common serial implementation). Additionally, fewer training patterns are required (to avoid overfitting), further reducing the number of computations required during training.

Growing the network is not a new concept, it is similar to the Cascade Correlation network proposed by Scott Fahlman, in which the network is grown one neuron at a time [11]. This has been found to have benefits beyond the reduction in required computation: reduction of the "moving-target" problem², and reduction of susceptibility to getting stuck in local minima. The network adapts until performance asymptotically approaches an optimum; then a neuron is added. These extra degrees of freedom are often sufficient to break the network out of a local minimum. In Cascade Correlation, the previous hidden neuron weights are frozen while the weights for the new neuron are adapted. This simplification of the search space reduces the moving-target problem. It can reduce computation if batch training is used, and the previously-calculated neuron activations are stored in memory.

In the real-time implementation required for the robot-control application, the network is grown automatically. Beginning with a small number of hidden neurons and a small training set, the initial learning rate is high. As network performance plateaus (measured by a sustained cessation of improvement in test set performance), hidden neurons are added, a small batch at a time. As the number of hidden neurons increases, the network performance approaches optimality, but at the expense of slower training. This approach fits well with the control application, since rapid stabilization and coarse optimization are important, while rapid attainment of near-optimal control is not so critical.

4.4 Summary of Implementation Issues for the FCA

The above has outlined the features of the new FCA developed in the present research, and of a number of issues in using it effectively. The specific use of complexity control, network growing, and the extra connections offered by the FCA, will vary from one application to another. The implementation issues for the robot-control application are outlined here.

²This refers to the weights changing directions and back-tracking throughout the training while the network approaches a final solution. While this is not necessarily bad, it can slow down learning.

- The FCA's feedthrough weights are the most important feature, as they provide near-immediate stabilization. Some implementation requirements, such as the use of parallel hardware, or the use of software optimized for vector-processing on a serial computer, can place a high cost on the use of hidden layer interconnections. In such a case, these connections may need to be eliminated.
- The use of automatic growing has been found to produce a significant improvement in initial learning rate. Since the added coding requirements are minimal, this technique should be used whenever there is a requirement for fast initial learning.
- Complexity control is simple to implement, and has been shown to reduce overfitting problems, so its use is recommended.

Many modifications to backpropagation that claim to improve learning speed have been proposed in the literature. Backpropagation is an algorithm for efficiently calculating the derivatives of the weights with respect to a cost function in a neural network. Once this gradient estimate is obtained, any of the several existing gradient-based optimization methods may be used. Some algorithms specific to neural networks have been developed that attempt to take advantage of some features specific to neural-network optimization.

The simplest implementation multiplies this gradient estimate by a fixed parameter to calculate the change in weights. More complex implementations adapt this learning rate parameter, or add a "momentum" term that sums past gradient estimates to filter out high-frequency noise and integrate low frequency trends. Several other methods, such as conjugate gradient, Levenberg-Marquardt, Quickprop, and other second-order gradient optimization schemes have proven successful in certain applications. However, the benefits of each algorithm appear to be somewhat application-dependent.

For the robot-control application, batch learning, adaptation of the learning rate (in this case, a matrix of independent learning parameters is used – adapted independently for each weight), and use of momentum are used to accelerate learning. For the thruster mapping application, this combination of enhancements to backpropagation has been found to provide the best trade-off between simplicity of implementation and rate of adaptation.

Chapter 5

Gradient-Based Optimization for Discrete Systems

The previous chapter dealt with direct training, and led to the development of the neural-network architecture used to implement the thruster mapping. To allow indirect training, where the learning signal (error) is generated based on the robot-model output (rather than on an optimal teacher), the error must be backpropagated through the robot model. The discontinuity introduced by the use of the robot's on-off thrusters presents a significant obstacle, and makes absolutely necessary the development of the new training method presented here. The solution to this problem is, in turn, a general algorithm for performing gradient-based optimization for systems with discrete-valued functions.

The discrete-valued functions did not cause a problem for direct training, since in that case the discrete values are supplied as the output patterns in the training set (e.g. $[1.87 \ -0.76 \ 0.11]$ gets mapped to $[0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0]$). The fact that the target outputs are restricted to 1's or 0's does not affect the training. However, for indirect training, the on-off actuators are represented by discrete-valued *functions* that are used as a forward model in the backpropagation training.

In this chapter, a new technique for backpropagation learning for systems with discrete-valued functions is presented. It is applied to the on-off thruster control problem described in Section 2, as well as to the generic problem of training multi-layer signum networks [69] [70] [73].

5.1 Problem Statement

Optimization methods that use gradient information often converge much faster than those that do not. Use of the backpropagation algorithm [46] [60] to get this gradient information for training neural networks has made them useful in many applications; however, backpropagation's requirement of continuous differentiability, not only for the network itself, but for anything through which the error is backpropagated (e.g. the plant model in a control problem), limits its applicability.

This is a significant limitation since there are many applications where discrete-valued states arise. For example: on-off thrusters commonly used in spacecraft (the example used in this research); other systems with discrete-valued inputs and outputs; and neural networks built with signums (also known as hard-limiters or Heaviside step functions) rather than sigmoids. Signum networks may be preferred to sigmoidal ones due to hardware considerations.

In cases like these, one choice is to use an alternative method not restricted to continuously-differentiable functions, such as unsupervised learning, simulated annealing, or a genetic algorithm; but these are usually significantly slower to train, because they do not use gradient information.

Also, it is common for a problem to be well-suited for gradient-based optimization, except for the presence of discrete-valued functions. The neural-network thruster mapping is a prime example: a neural network (differentiable) produces an output that is discretized (with a non-differentiable function) and then passed through a model of the robot-thruster system (differentiable) before the performance can be evaluated and used for training. Except for the DVF, this problem is well-suited for gradient-based optimization. Rather than go to a completely different solution strategy, it is desirable to introduce a modification to gradient descent that will accommodate the non-differentiable functions. This sort of a situation is rather common when DVFs are involved – the DVFs often represent a small portion of the overall system, but the problem they present for gradient-based optimization is formidable.

5.2 Related Research

This problem is related to a similar problem that has received some attention in the field of neural networks: training multi-layered networks of hard-limiting neurons. The algorithm

presented here will be shown to be applicable to this problem. This section presents a historical background and related research directed towards training signum networks.

5.2.1 History of Neural-Network Training With Smooth Activation Functions

Before the task of training a network built with DVFs is examined, it is useful to consider the history of the feedforward neural network, and why the sigmoid function was chosen in the first place.

Learning algorithms for single-layer signum networks date back to 1960, with Widrow's ADALINE (ADAPtive LInear NEuron) [67] and Rosenblatt's Perceptron [43]. Unfortunately, neither of these methods extends directly to multiple layers. Minsky's proof of the functional limitations of single-layer Perceptrons [32] [33] combined with this lack of a learning algorithm contributed to a reduction in interest in neural networks in the 1970s.

In 1974, Werbos [60] presented the backpropagation algorithm for the first time. While the mathematics of the algorithm may be traced back to work in 1969 by Bryson on optimal control [6], Werbos developed the algorithm for a number of applications, including neural networks built with sigmoidal activation functions in the hidden layer. Unfortunately, this work was largely unnoticed until its rediscovery and publication by Rumelhart in 1986 [46]. The key extension that allowed training of networks with hidden layers was the replacement of the signum with the sigmoid. This allowed Bryson's work with multistage optimization for dynamic systems to be applied to gradient-based optimization with the now-differentiable neurons. It is understood that use of a sigmoid in place of a signum is computationally more expensive, without providing significant added functional complexity; however, the use of a function that is continuously-differentiable allows for the application of the efficient gradient-based optimization methods developed by Bryson.

5.2.2 Neural-Network Training With Discrete-Valued Activation Functions

MADALINE (Many ADAPtive LInear NEurons) Rule I was a two-layer network (one hidden layer) that had a trainable first layer, but the second layer was a fixed logic operation, such as OR, AND, or MAJ (majority) [18]. In MADALINE Rule II, Winter [74] used

a heuristic approach which had limited success at training a two-layer network of hard-limiters (ADALINEs). These methods may be classified as “error-correction rules” rather than “steepest-descent rules” (gradient-based) [67].

In recent research aimed at using gradient-based learning for multi-layer signum networks, Bartlett and Downs [4] use weights that are random variables, and develop a training algorithm based on the fact that the resulting probability distribution is continuously differentiable. The algorithm is limited to one hidden layer, requires all inputs to be 1 or -1, and needs extra computation to estimate the gradient.

Another method is to approximate the discrete-valued functions with linear functions or smooth sigmoids during the learning phase, and switch to the true discontinuous functions at run-time. This is similar to the original ADALINE, where the neuron was trained on its linear output, but in operation, this output passed through a signum function [67]. This method may work in cases where the behavior of the system with sigmoids is close enough to that of the real system; however, this assumption is very often unreliable.

5.3 A New Training Algorithm: Approximation With Noisy Sigmoids

The method of noise injection is introduced by applying it to the training of a single hard-limiting neuron, as shown in Figure 5.1. Although this neuron could be trained with the ADALINE or perceptron learning rules, those methods do not extend to multiple layers. The method presented here does not have this significant restriction.

The first block diagram in Figure 5.1 shows the neuron as it appears at run time: a dot product with hard limiter. For simplicity in bookkeeping, the input, X , and weight, W , vectors are augmented to include the threshold bias for the output function. The next two diagrams show the neuron during training, where the signum has been replaced by a smooth sigmoid function. The input, X , is propagated through the forward sweep, finally resulting in an error, ϵ , and a cost. The derivative of this cost is calculated and propagated through the backward sweep, resulting in a $\partial \text{cost} / \partial X$ to be propagated to more units upstream, and a $\partial \text{cost} / \partial \text{net}$ to be used in calculating $\partial \text{cost} / \partial W$, which is used in the weight-update algorithm.

This is almost the same as training a standard neuron with backpropagation; the only difference involves the injection of zero-mean noise, N , immediately before the sigmoid.

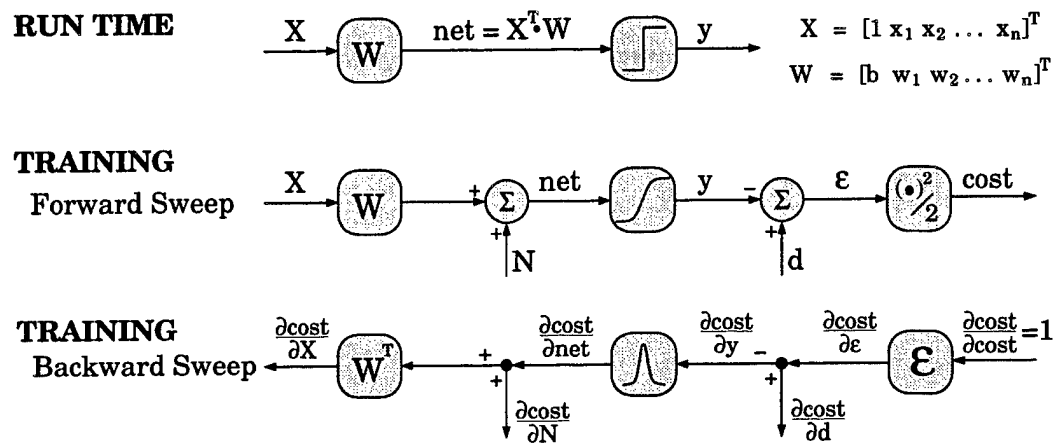


Figure 5.1: Training Algorithm

During training, replace discontinuous signums with sigmoids, and inject noise before the sigmoid on the forward sweep. The backward sweep calculation is the same as standard backpropagation.

While the mechanics of the backward sweep are identical, different weight updates result because the forward sweep resulted in a different error.

Note that the noise injection does not corrupt the calculation of $\partial \text{cost} / \partial W$ (just as the desired signal does not). Using an unmodified backward sweep is not only the simplest thing to do, it does precisely the right calculations for estimating the weight gradient.

What makes this method useful is that as the noise level increases to cover the sigmoid's transition region, adaptation with the resulting $\partial \text{cost} / \partial W$ leads to a set of weights that work well for the signum network.

To summarize, the training algorithm is:

- Replace the hard-limiters with sigmoids during training
- Inject noise immediately before the sigmoids on the forward sweep
- Use the exact same backward sweep as with standard backpropagation

5.4 Intuitive Explanation

Without addition of noise, the network may train using sigmoid output values in the sigmoid transition region (roughly -0.8 to 0.8) that will be unavailable at run time. Simply rounding

off at run time may introduce significant errors. For example, in a hypothetical cost surface, a value of 0.4 may be optimal, but if forced to choose between -1 and 1, a value of -1 may be better.

The problem is much more apparent when the DVF outputs are recombined, such as with the output layer of a network built with hidden signum units. This also occurs when the robot-thruster physical parameters combine to produce a three-element force vector based upon the binary eight-element thruster vector.

The goal of noise injection is to move neuron activations away from the transition region, so that roundoff error will be small when the discrete-valued functions are replaced. For this reason, the standard deviation of the noise is chosen to be higher than the width of the transition region of the sigmoid.

Figure 5.2 shows how the neuron output distribution changes as the noise level increases: with no noise, only a single output can result; but as noise increases to cover most of the transition region, the output distribution approaches that of a hard-limiting function. Differentiability is maintained, however, so that gradient information will be available to speed up learning. Since the noise has pushed the distribution to approximate a hard-limiting nonlinearity, when the hard-limiter is re-introduced at run-time the performance degradation will be small.

5.5 Application Considerations, Extensions

5.5.1 Selection of Noise Level

One concern is the attenuating effect of the derivative-of-sigmoid function. When back-propagated through many layers of near-saturated sigmoids, the error signal is attenuated and may lead to slow learning. To handle this problem, it may be necessary to be gradual in increasing the noise level; slowly push the outputs from the linear region to the hard-limits, rather than all at once. However, since all the experiments presented here had a single layer of discontinuity, no such gradual increase was required.

For training networks with simple bi-level sigmoids, once the noise reached a sufficient level (roughly 0.5 and 3 in two different applications), there was no degradation if it were increased beyond that level. The only possible drawback is the attenuation effect mentioned above. The required noise level varies in different applications depending upon how sharp

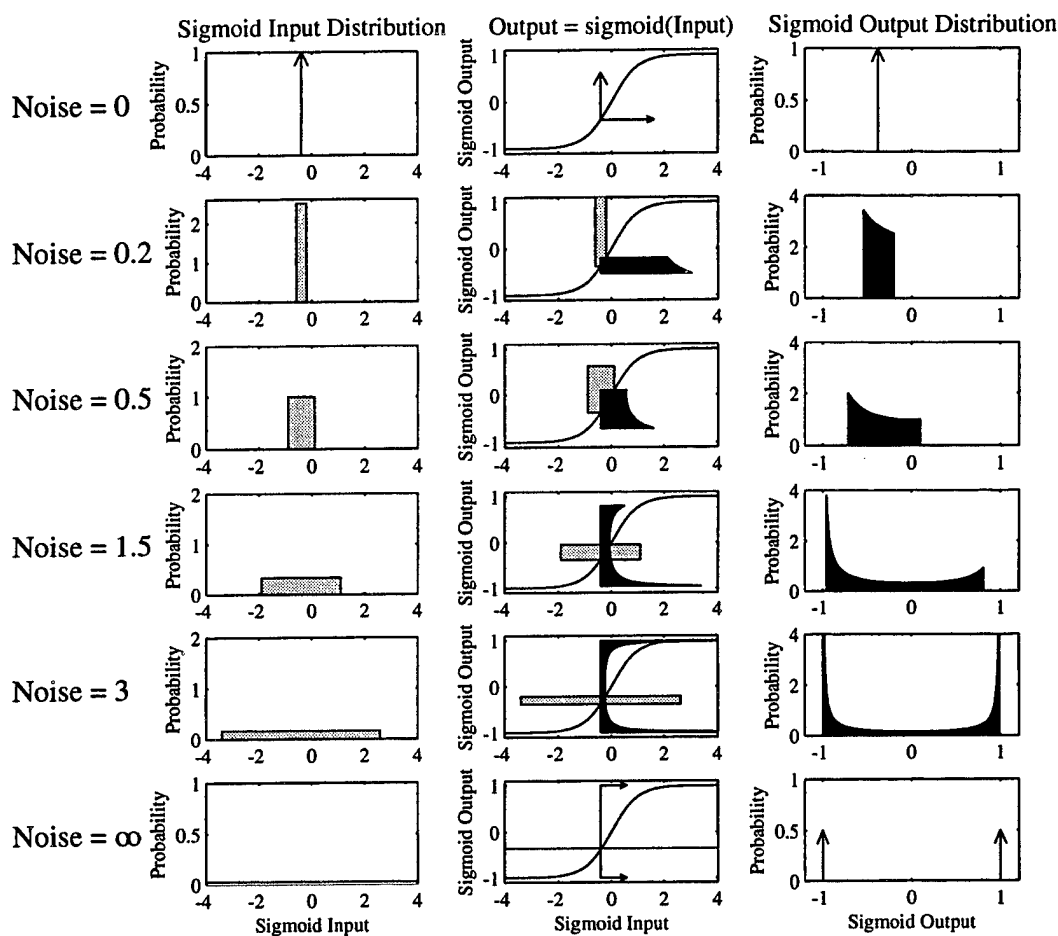


Figure 5.2: Effect of Noise Level on Sigmoid Output Distribution

Lightly-shaded region in column 1 represents the sigmoid input probability distribution (in this case, $-0.3 +$ uniformly distributed noise). Darkly-shaded region in column 3 is the sigmoid output distribution (from -1 to 1). Each distribution has an area of 1. Input and Output are plotted together in column 2 to show how the sigmoid produces this input-output relationship. As noise level increases, and the input distribution spreads out, the sigmoid output approaches that of a hard-limiter, while remaining differentiable.

the decision boundaries would be with no noise (i.e. if it's a sharp sigmoid to begin with, not much noise is needed to force it off the transition region).

When multi-level sigmoids are used, as seen in Figure 5.8, there is an upper limit to the noise level: too much noise may cause the individual sigmoids to overlap, which in this example would blur out the middle level. The specific level of noise at which this effect

begins depends upon the sharpness of the sigmoids and the discrete values approximated. In Figure 5.8, with a sharpness factor of 4 (slope at midpoint = 4) and one unit between discrete levels $(-1,0,1)$, this effect begins around $N = 0.2$ and is significant at around $N = 0.3$. These values could have been predicted by sketching the limits of the noise-altered function (the shaded region in Figure 5.8) and determining at what point the middle region ($input = 0 \Rightarrow output = 0$) becomes affected by the noise.

The key idea in this algorithm is that the network performance error is linked to round-off error due to use of the sigmoid transition region. The goal of the noise injection is to discourage use of this transition region. Therefore, whether use is discouraged using Gaussian noise, uniform noise (used here), fixed-level noise, or additive penalty functions, the effect is qualitatively the same.

5.5.2 Discrete-Valued Functions Other Than Bi-Level Sigmums

If adapting a system that contains discrete-valued functions that are not simple Heaviside step functions, the method may work if a continuously differentiable approximating function is used. For example, a function whose output can take on multiple discrete values may be approximated by combining multiple sigmoid functions. For the thruster mapping problem described in Section 4, the thruster can take on three states: forward, off, or backward. Two bi-level $(-1,1)$ sigmoids were summed to produce a tri-level $(-1,0,1)$ sigmoid.

In fact, the sigmoid-based approximation may be developed through a supervised training technique using standard backpropagation. The limitation introduced by the attenuation of error signals is again a factor, and must be considered when developing the smooth approximating function. This can be done by limiting the sharpness of the sigmoids if programming by hand. If training the approximating function, adding a complexity cost [59] [71] will keep the weights small, and will systematically limit the sharpness.

5.5.3 Batch Learning

The randomness introduced with the addition of noise could make learning slow because of the reduction in signal-to-noise ratio in the weight gradient estimation. Batch-learning, using the exact same training set from one epoch to the next worked well (considering the “training set” to include the “input set” and “noise set”). Freezing the training set and noise set defines a fixed deterministic cost hyper-surface. With a fixed cost function,

on-line tuning of momentum and learning rate can be applied to improve dramatically the convergence rate.

5.5.4 Optimization of Discrete-Valued Parameters

Another area where this method has potential is for optimization problems that have discrete valued parameters. For example, a design optimization problem where the task is to select the right DC motor, pipe diameter, or gear ratio from a finite set of discrete-valued options. It is expected that this method will extend well to this family of problems [31].

5.6 Application to Training Multi-Layer Signum Networks

In this section, this method is shown to extend to multiple layers of hard-limiting units with no modification. Figure 5.3 summarizes the method: during training, replace each hard-limiter with a sigmoid and zero-mean independent noise source. Note that the sharpness of the sigmoids does not matter *at all* here (except for numerical considerations), since the sharpness factor simply multiplies the weights, and the weights are adapted.

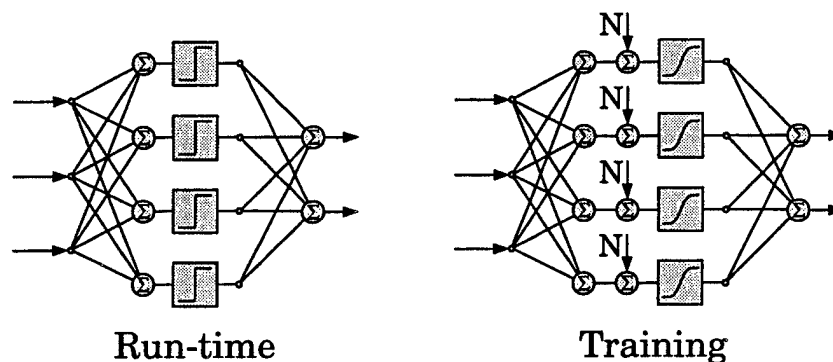


Figure 5.3: A Multi-Layer Signum Network, Seen at Run-Time and During Training

In the first test, an adaptive 3 – 5 – 4 signum network is trained to emulate the input-output mapping defined by an independent, fixed, 3 – 10 – 4 sigmoidal network. Fewer hidden neurons are used in the adaptive network to ensure that overfitting will not introduce unnecessary complications. The 3 – 10 – 4 network's fixed weights were randomly chosen between -2 and 2.

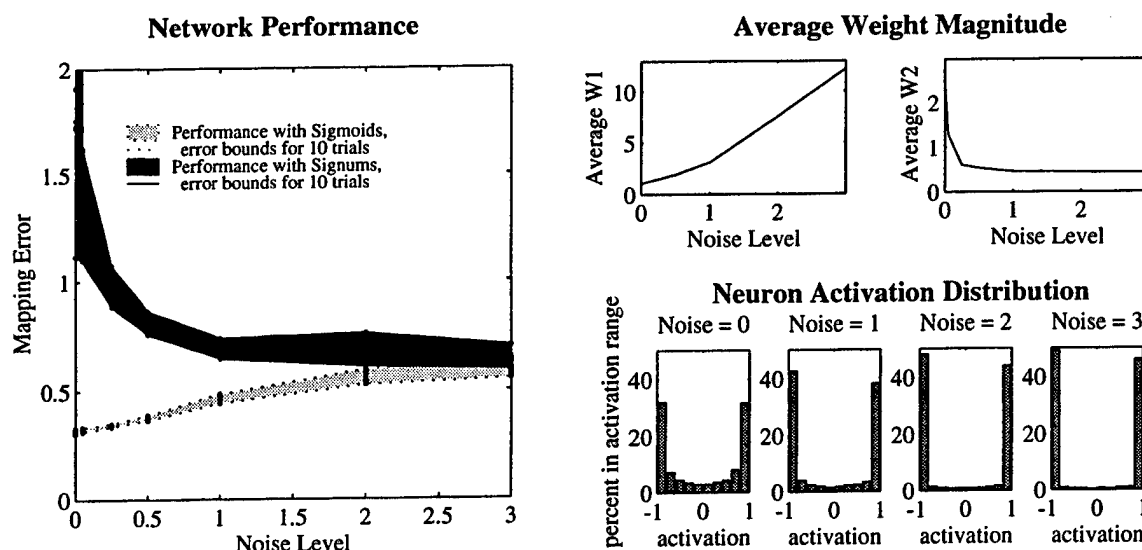


Figure 5.4: Training with Noisy Sigmoids of a Multi-Layer Signum Network, Artificial Training Set

Left: with higher noise levels, performance on the noisy sigmoidal network approaches that of the signum network, indicating that the noisy sigmoid is a valid (and differentiable!) approximation for the signum. Right: As noise increases, the network adapts to sharpen its sigmoids, causing the first layer weights to increase, and the sigmoid output distributions to approach hard-limiters. Activation distributions were collected over the whole training set, with no noise added.

Performance is shown in Figure 5.4. Each dot on the graph represents the final performance after a full training run (10,000 epochs or until a local minimum was reached). Seven values for noise level were chosen, and ten different network initial conditions were used at each noise value. With no noise, performance is good for the sigmoidal network, but when the signums are reintroduced at run-time, the error increases dramatically. One point is off the graph at an error of over 6 units. As noise increases, performance on the sigmoid network decreases, as expected, but the signum-network-performance improves, and approaches the sigmoid-network-performance. The weight magnitude and neuron activation distribution plots confirm that as noise increases, the noisy sigmoids behave like hard-limiters. Note that these activation distributions could not have been achieved by manually increasing the sharpness of the sigmoids: this would have had *zero* net effect, since the network would adapt the first layer weights to counteract *exactly* the sharpness increase.

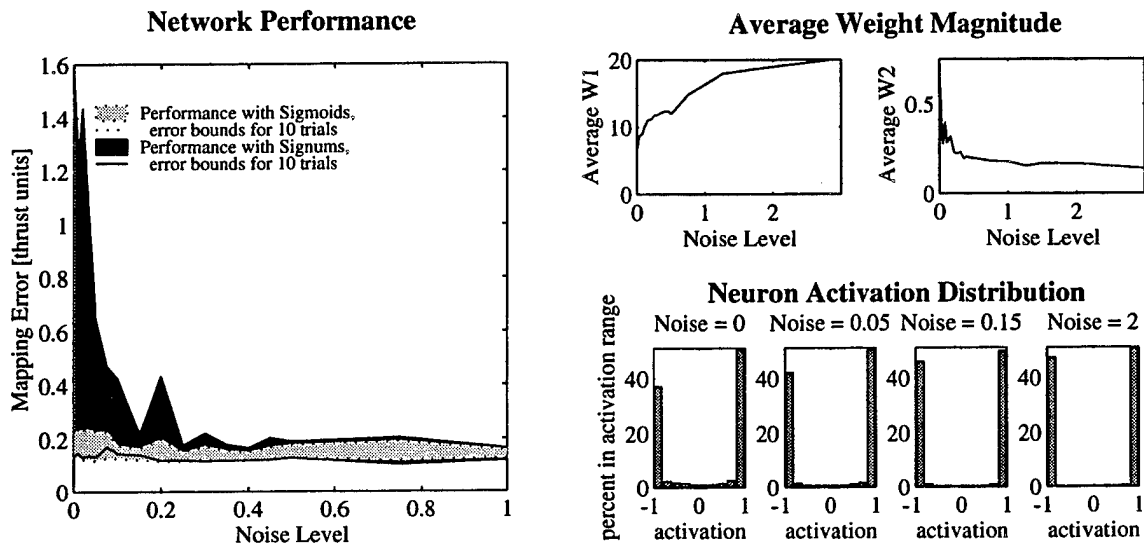


Figure 5.5: Training a Multi-Layer Signum Network, Thruster Mapping

In the second application, shown in Figure 5.5, the hard-limiting network is trained to emulate the optimal thruster mapping, which will be described in detail in the next section. For now, this mapping is used as an independent second test of the method. A similar dramatic improvement in hard-limiting performance occurs as noise increases past about 0.5. It is not shown on the plot, but good performance is obtained at least up to a noise level of three. The training set for this mapping represents continuous values being mapped to discrete values, so the first-layer weights are high (indicating sharp decision hyper-surfaces), even for *noise* = 0.

5.7 Application to Thruster-Mapping Problem

In order to demonstrate this new training procedure, it was applied to the thruster mapping with indirect training, as shown in Figure 5.6 or the top section of Figure 2.13. In this case, the optimal mapping is not used, and the neural network must learn the mapping through experimentation with the plant model. This requires back-propagation of error through the discontinuous thrusters, which motivated development of the noise injection method presented in this chapter.

Training without this noise-injection technique produces large errors, because the discrete-valued nature of the thrusters is not enforced during network training, and large

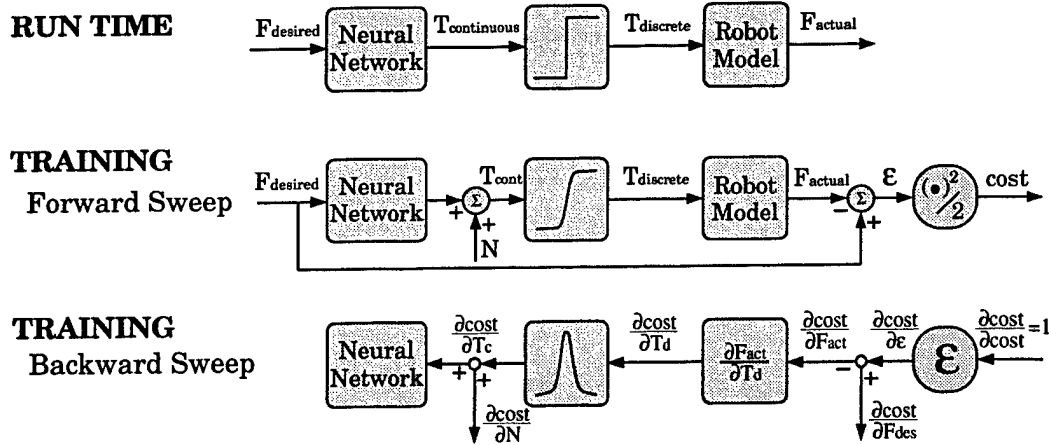


Figure 5.6: Thruster Mapping, Indirect Training Method

roundoff errors result at run-time. For example if one unit of thrust is requested in the $+x$ direction, during training, the network will set T_4 and T_5 to $+0.5$; but at run time, for requested forces near 1.0, T_4 and T_5 are likely to both be 0 or both be 1, resulting in a large error.

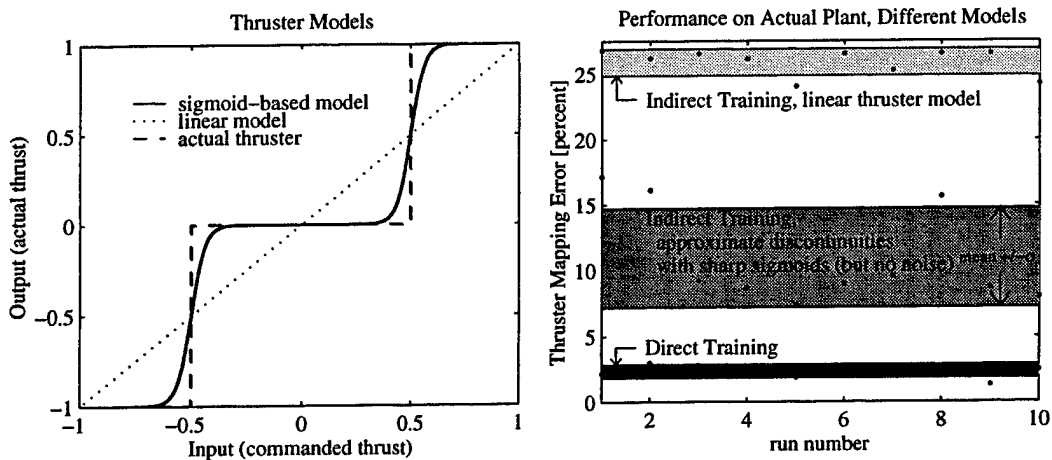


Figure 5.7: Results of Indirect Training, Two Differentiable Thruster Models

The sigmoid-based approximation (without noise) is better than the linear model, but has limited performance. The results from direct training represent a lower limit for comparison. Mapping error is average percent error above the optimal mapping (which results from an exhaustive search of all possible thruster combinations). The shaded areas represent the mean $\pm \sigma$ for ten different runs. 3 – 10 – 4 layered networks were used.

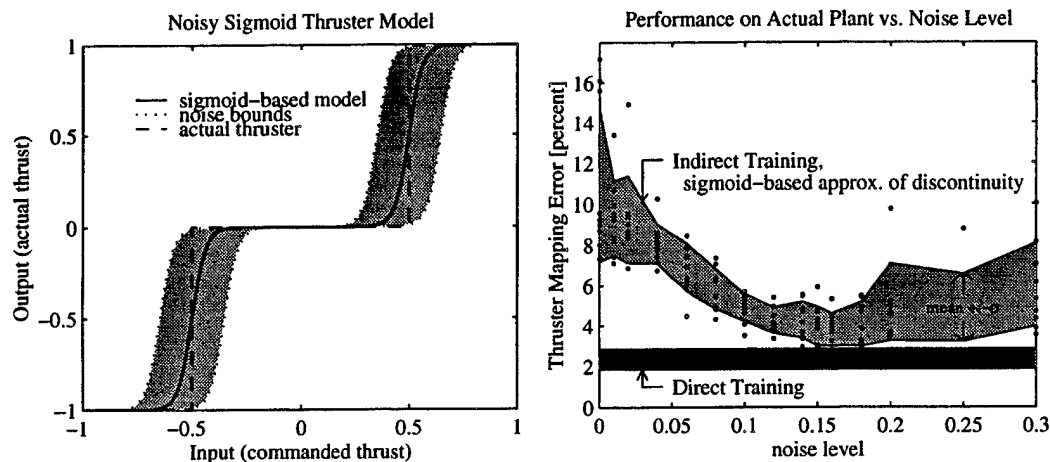


Figure 5.8: Results of Indirect Training, Noisy Tri-Level Sigmoid Thruster Model

Left: the sigmoid sharpness factor (slope at the midpoint = 4) and noise level (0.15) for the noisy tri-level sigmoid appear to be intuitively correct. Right: as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping), with best performance at a noise level of about 0.15. 3–10–4 layered networks were used.

Figure 5.7 shows the result of indirect training with two differentiable thruster models. During training with the continuous thruster models, the neural network produces a mapping with a very low error, which is not plotted here. However, when the continuous thrusters are replaced by signum thrusters at run-time, the error is large, and is the “thruster mapping error” plotted in the right half of Figures 5.7 and 5.8. The errors are high because the network learned to optimize the solution using outputs that would be unavailable at run-time. The resulting roundoff error is unknown to the neural network during training.

In Figures 5.7 and 5.8, each dot represents the final performance after a 10,000 epoch training run. The shaded regions represent $mean \pm \sigma$ performance for ten runs.

Figure 5.8 shows the results when the thrusters are modelled by *noisy* tri-level sigmoids. With $noise = 0$, error is high, corresponding to the data in Figure 5.7, but as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping).

The direct-training performance represents a lower bound set by the functional complexity of the 3 – 10 – 4 layered network. The best noise value in this application seems to be around 0.15, and the resulting noisy sigmoid is shown in the left half of Figure 5.8. Examining this figure, the sigmoid sharpness and noise levels seem to be set correctly according to intuition. As noise increases beyond 0.2, error increases as expected (the “off” region of the sigmoid becomes blurred). The method is fairly robust to the noise value selected, and the effect of noise level on performance makes intuitive sense.

A good solution results when noise is added, because it prevents the network from using a solution that uses non-saturated portions of the tri-level sigmoid. Such a solution would give a nearly random output and high error during training. The training algorithm must find a solution that works well *despite* the noise addition. This means the expected value of the output must be well into the saturated regions to work consistently well. The results approximate the optimal solution very well, and work when the tri-level sigmoids are replaced with tri-level signums.

5.8 Other Uses of Noise in Related Problems

Noise has been shown to be central to the success of this new algorithm. While this particular use of noise is new, artificially-injected noise has been used successfully in previous applications for control, neural networks, and optimization.

In control and signal processing, quantization error results when an analog signal is sampled digitally (with inevitably finite precision) by an A/D converter. This effect was first studied extensively in the Ph.D. work of Bernard Widrow, and published in [66]. In analyzing this phenomenon, the roundoff error may be treated as a source of noise. While this work has little direct bearing on the algorithm presented here, the presence of noise and roundoff error in the same problem is interesting.

In control applications, it is common to add an artificial dither signal to break the effects of stiction. This dither is usually chosen to cause a force just large enough to overcome the static friction, and is input at a frequency high enough that it does not affect the rest of the control system. Again, there is little direct connection with the noisy-sigmoid training algorithm, but it represents a previous application of artificial noise injection in control.

In the human vision system, the limitation of a finite number of receptors in the retina is overcome by the artificial addition of a dither signal. Very small, high-frequency motions

of the eye are used to allow people to see thin far-away objects that might otherwise go unseen due to the finite number of receptors.

When training a neural network with a limited set of training data, one approach to control the effect of overfitting is to duplicate the elements of the data set, and add different amounts of noise to each one, in an attempt to increase the effective size of the data set.

Adding noise to the weight updates has been tried, with some success, to improve the learning speed of neural-network training [34]. This is a similar concept to simulated annealing, the addition of a random element in the weight update rule whose magnitude decreases exponentially. The idea in simulated annealing is to prevent the common optimization problem of getting stuck in a local minimum. If the magnitude of the random element is decreased slowly enough (i.e., the time constant approaches infinity), convergence to the global optimum is guaranteed. This gradual reduction in temperature is similar to that in a metallurgical annealing process – hence the name. Simulated annealing is a common algorithm in optimization for systems other than neural networks.

In genetic algorithms, species are evolved using two primary methods to go from one generation to the next: (1) crossover, the combination of traits between competitively-selected parents; and (2) mutation, the addition of a random element in the next-generation chromosome.

While the above examples show that the concept of artificially-added noise for control and optimization problems is well-tested, the use of noise presented in this thesis – to produce an accurate differentiable approximation to a DVF for gradient-based optimization – is completely new.

5.9 Summary

This chapter has described a new technique that allows backpropagation learning to work with systems containing discrete-valued functions, despite the discontinuity that exists between discrete values. The modification to backpropagation is very small, simply requiring sigmoidal approximation of the discrete-valued functions, and the careful injection of noise into the smooth approximating function on the forward sweep. The noise injection is critical to ensuring that the noisy sigmoid behaves like a signum during training.

Multi-layered networks of hard-limiters require simpler processing hardware than do multi-layered sigmoid networks. Sigmoid networks are commonly used, however, due to

their increased functionality as well as the lack of a reliable training algorithm for signum networks. Multi-layered signum networks have now been successfully trained using this noise injection method in two different applications, clearly demonstrating its usefulness in this area.

Application to a complex thruster-control problem, with implementation on a laboratory model of a free-flying space robot, has demonstrated the method's realizability and usefulness for on-off control problems.

In each application, the training behavior in the presence of noise has been well understood, and the algorithm appears to be relatively robust to the amplitude of the injected noise.

Chapter 6

Experimental Demonstration of Reconfiguration

Experiments were performed on the mobile robot described in Chapter 2 to verify the applicability of these neural network results. Position and attitude of the robot base are controlled while subject to multiple, large, possibly-destabilizing changes in thruster characteristics. The plant is linear and well-modelled, except for the actuators, which are on-off thrusters that could have altered characteristics. An off-board vision system provides high-bandwidth position feedback, which is then digitally filtered and differentiated to provide velocity feedback. On-board accelerometers and an angular-rate sensor are used to provide base-acceleration measurements used by the failure-detection and control-reconfiguration capability. This chapter reviews the complete control system, and presents experimental results.

6.1 System Overview

Figure 6.1 shows the overall system block diagram which was discussed initially in Chapter 3. In this chapter, each block will be described in detail.

The **User** issues motion commands with a mouse-based graphical user interface (GUI) that runs on a Sun¹ workstation adjacent to the robot. The user views an image of the robot that is updated with real-time data from the **Position Sensor** described below. He or she can use the mouse to drag a ghost image of the robot to the desired final location,

¹Sun is a trademark of Sun Microsystems, Inc.

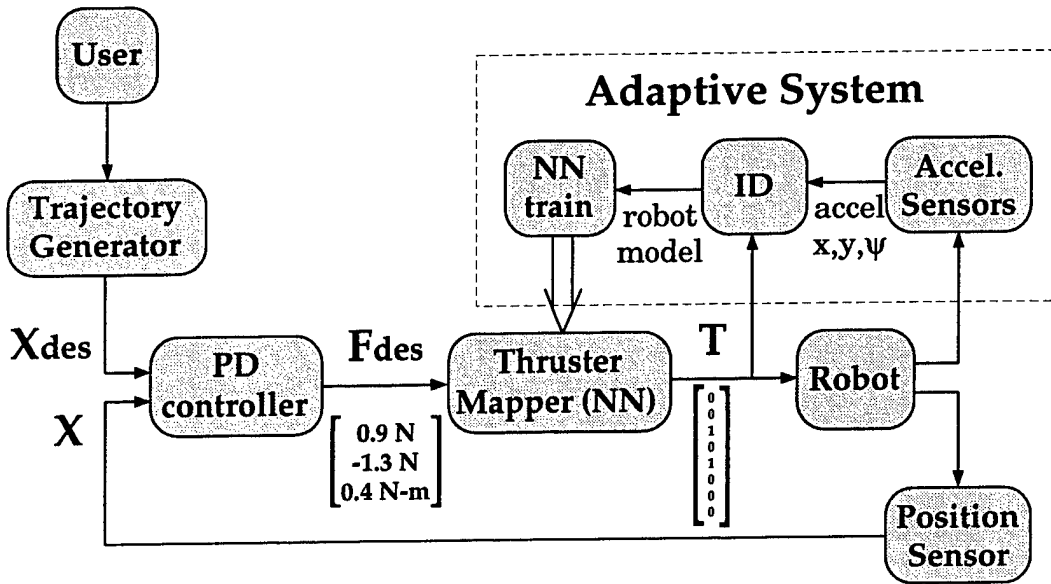


Figure 6.1: Reconfigurable Control System – Block Diagram

This control system is based upon a conventional indirect adaptive controller, such as a self-tuning regulator. Examples of the continuous-valued F_{des} vector and the corresponding discrete-valued T vector are shown. The ID block represents a recursive-least-squares identification of thruster strength and direction. This continually-updated model is passed to the neural network training block, shown in detail in Figure 5.6. The continually-updated neural thruster mapper is copied periodically into the active control loop.

adjusting its position and orientation. The motion is then initiated by clicking on a button that is part of the GUI.

The **Trajectory Generator** receives the current and desired position and velocity vectors and generates a quintic-polynomial trajectory between the two locations. A quintic-polynomial means there are six coefficients of a polynomial function of time. These parameters are chosen to match the initial and final position and velocity (four parameters) and set acceleration to zero at initial and final times (the two remaining parameters). The duration of the slew is minimized automatically while not exceeding the pre-defined acceleration limits (corresponding to the limits in actuation). The result is a time history of desired states, X_{des} , consisting of $[x_{des}, y_{des}, \psi_{des}, \dot{x}_{des}, \dot{y}_{des}, \dot{\psi}_{des}]$.

The **PD Controller** takes the desired state, X_{des} , from the **Trajectory Generator**, and the measured state, X , from the **Position Sensor**. The translational proportional and derivative gains are 32.5 N/m and 91 N/(m/s), resulting in closed loop poles

at $s = -0.65 \pm 0.2j$ (neglecting effects of the on-off actuators). The output of this component is a continuous-valued desired force vector, $F_{des} = [F_{x_{des}}, F_{y_{des}}, \tau_{\psi_{des}}]$, such as [0.9 N, -1.3 N, 0.4 N-m].

The **Thruster Mapper** takes the desired force vector, F_{des} , and produces the thruster vector, T , that causes the thrusters to open or close. An FCA network is used to implement the thruster mapper. Like the rest of the low-level control loop, it is written in the C programming language and executed on a Motorola[®] 68040 processor (MVME 167) on the robot. The real-time control system was developed with ControlShell² development software and the VxWorks³ operating system. Details of the network are described below. The signal flow of the thruster-mapper component is shown in Figure 6.2. The final output is the binary eight-element vector of thrusters to fire, $\mathbf{T} = [T_1 \ T_2 \ T_3 \ T_4 \ T_5 \ T_6 \ T_7 \ T_8]$.

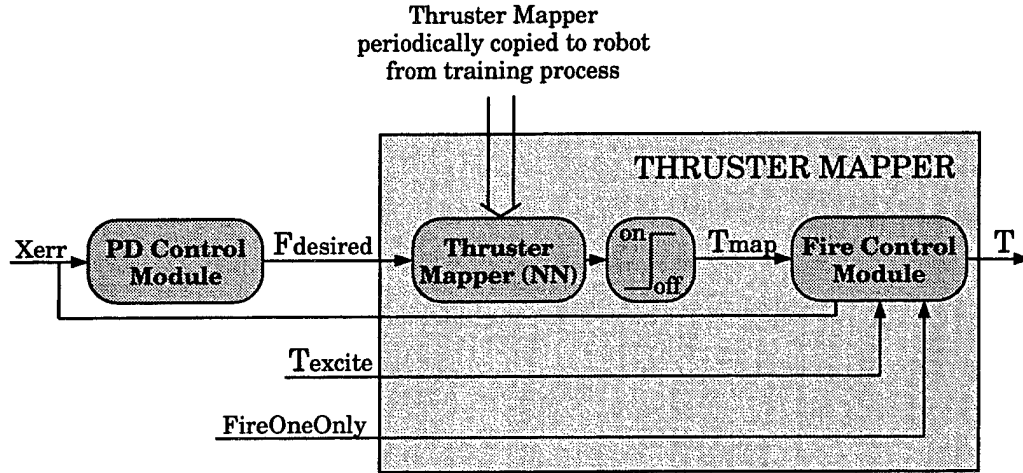


Figure 6.2: Thruster Mapper – Signal Flow

The signal flow of the “thruster-mapper” component shown in Figure 6.1 is presented. The mapper produces a T_{map} vector based upon the desired force, but this signal may be changed by the “Fire Control Module” during the identification process. A list of thrusters to excite, T_{excite} , is provided by the “ID” component. A *FireOneOnly* signal is also used to simplify the identification by limiting firing to one thruster at a time. Both of these ID-related functions may be over-ridden if the tracking error, X_{err} , is too high. The parameters (neural-network weights) that define the function implemented by the thruster mapper are periodically copied from the neural-network training process.

²ControlShell is a trademark of Real-Time Innovations, Inc.

³VxWorks is a trademark of Wind River Systems, Inc.

The **Robot** has a mass of 70 kg, floats nearly frictionlessly on the granite table, has eight thrusters, each nominally producing 1 Newton of thrust. Since control of the robot manipulators was not relevant to this research, the arms are commanded to maintain a fixed position at all times. This involves RVDT sensors, an analog pre-filtering and differentiating circuit, A/D converters, a PD controller for each of the four joints, D/A converters, motor driver boards, and finally the brushless DC motors and cable drive system that actuate the arms. The arm endpoints are equipped with pneumatic plungers, allowing the robot to capture a free-floating target object.

The **Position Sensor** is a pair of CCD cameras mounted to the ceiling above the robot. Two cameras are required to cover the total surface area of the 2.74×3.65 meter (9×12 foot) granite table. The cameras detect a pattern of LEDs mounted to the top of the robot. A custom vision processing board processes the camera output, and produces position information at a 60 Hz update rate that is accurate to better than 1 mm. This $[x, y, \psi]$ vector is digitally filtered and differenced to produce a velocity vector. The processing is performed off-board and then communicated back to the robot via a wireless Ethernet data/communications link.

The **Sample Rate** for the low-level control loop was chosen to be 10 Hz. This is more than an order of magnitude faster than the PD controller bandwidth, and is slow enough to allow transient acceleration effects to die out, leading to the accurate acceleration information needed for reconfiguration. If reconfiguration is not required, the sample rate can be increased to 60 Hz. Sampling faster than that produces no benefit, since the vision system operates at 60 Hz, and the thruster bandwidth is approximately 30 Hz.

Summary of the signal flow in the low-level control loop: LEDs on top of the robot emit infrared light. CCD cameras on the ceiling receive the light, and send the signal via a coaxial cable to the custom vision processing board mounted on a fixed rack adjacent to the granite table. The "pointgrabber" vision board scans the image for bright pixels. When the known pattern of LEDs is located, the vision board calculates the orientation and geometric center of the robot. Velocity is also calculated on the vision board by digitally filtering the position information. The 6-element robot state vector is broadcast to the robot at a 60 Hz update rate (and less than 30 ms total time delay) over the Motorola Altair wireless Ethernet system. The robot then sends this information back to the user interface running on a Sun workstation. The on-board microprocessor takes the state vector and uses the **PD controller** to calculate the desired force, convert to robot coordinates,

and uses the **Thruster Mapper** to calculate the thruster vector (e.g. $[1\ 0\ 1\ 0\ 0\ 0\ 0\ 1]$). This vector is sent over the VME backplane to the digital I/O board, which then controls the opening and closing of the eight solenoid valves. This releases air from the 100 psi reservoir out through the converging-diverging nozzles to produce one Newton of thrust per thruster.

The **Acceleration Sensors** are described in detail in Chapter 2. Two accelerometers are mounted on the base orthogonal to one another, along with an angular-rate sensor. The acceleration signals and angular-rate signal are pre-filtered to remove the effects of extraneous vibrations. The filtered signals pass through an A/D converter, and then through the VME backplane to the microprocessor. The base translational acceleration vector is derived by subtracting centrifugal accelerations (calculated using angular-rate information) and converting to the robot frame. The angular-acceleration signal is obtained by digitally filtering and differencing the angular-rate signal. The $[\ddot{x}, \ddot{y}, \ddot{\psi}]$ vector of the robot base is the output of this component, as shown in Figure 6.3.

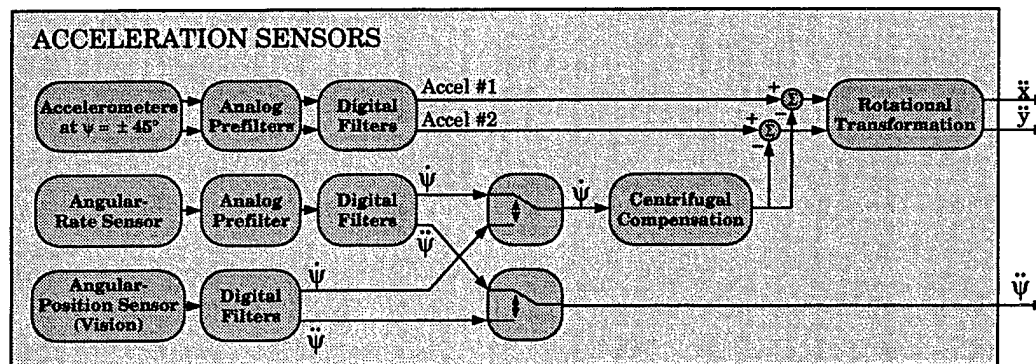


Figure 6.3: Acceleration Sensors – Signal Flow

The signal flow of the “acceleration sensors” component shown in Figure 6.1 is presented. The accelerometers are filtered with analog and digital filters to produce the Accel #1 and #2 signals. The angular-rate sensor signal is similarly filtered, with the additional step of a digital difference, which produces $\ddot{\psi}$ as well as $\dot{\psi}$. $\ddot{\psi}$ is output directly, while $\dot{\psi}$ is used to compensate for centrifugal accelerations measured by the accelerometers. The acceleration signals are then rotationally transformed to align with the x and y coordinates of the robot. When the angular-rate sensor saturates, angular rate and acceleration derived from the overhead vision system are used, as indicated by the logical switches.

The **ID** component identifies the characteristics for each of the eight thrusters. This is described in detail below. At a simple level, it takes in the acceleration vector and thruster

vector, and performs a recursive linear regression to identify the thruster parameters. The more-complicated factors, such as failure detection and thruster excitation, are described below, and summarized in Figure 6.9. A linear regression may be used here, since the forward model of the thrusters is linear, e.g. firing thruster ① may produce -1.03 N in the x direction, 0.07 N in y , and 0.137 N-m in ψ . The result is a 24-element matrix, containing the thrust produced by each of the eight thrusters in each of the three degrees of freedom. This is the “robot model” indicated in Figure 6.1.

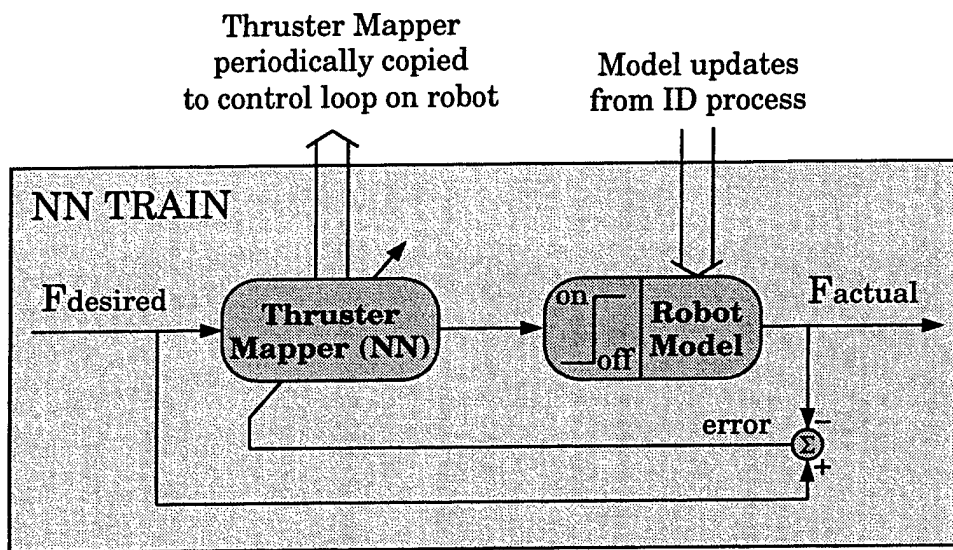


Figure 6.4: Neural-Network Training – Signal Flow

The signal flow of the “NN train” component shown in Figure 6.1 is presented. The model used in training is updated by the ID process, and the neural-network thruster mapper developed here is copied periodically to the thruster mapper running on the robot. The algorithm used to adapt the neural network based on the error signal is shown in Figure 5.6.

The NN **train** component is responsible for redesigning the thruster mapper to account for changes in the robot model. It waits until a major change is detected, calculates a linear mapper, and implements it on the robot using the FCA, described in Chapter 4. When smaller changes occur (as the ID process converges), the model used for training is updated. If further major changes are detected, the network is reinitialized to a newly-calculated linear mapper. Indirect training is performed using the modified backpropagation algorithm described in Chapter 5. The thruster mapper being trained is copied periodically to the thruster mapper running on the robot. The network is grown gradually, resulting in

a fast initial learning rate. The details of the training are presented below, and summarized graphically in Figure 6.4.

Summary of the signal flow in the adaptive system: Accelerometers and an angular-rate sensor measure motion of the robot base. The raw signals are prefiltered on-board, pass through an A/D converter to the microprocessor, where the dynamics are accounted for, and the base acceleration vector is computed. This signal is transmitted using the wireless Ethernet to a Sun workstation that is running the **ID** process. The **ID** process forms the robot model and transmits updates to the **NN training** process running on another Sun workstation. The updated neural-network **thruster mapper** is copied periodically to the robot via the wireless Ethernet, where it is substituted for the **thruster mapper** running in the control loop.

6.2 Trajectory-Following Performance

Before the reconfiguration capabilities are presented, trajectory-following performance with all thrusters working is discussed. This serves two purposes. First, it demonstrates that the base-control strategy of separating the thruster-control system into a control component and a thruster-mapping component is valid. Second, it demonstrates that a neural-network emulation of the search-based thruster mapper (which is optimal) can provide near-optimal performance.

When evaluating performance, the effects of the on-off actuators should be considered. Due to the control structure, PD-control gains, and thruster-mapping cost function selected, a deadband exists within which the thrusters will not fire, even with an optimal thruster mapper. While the size of this deadband is difficult to characterize due to the thruster-coupling effects, the maximum static deadband (assuming zero velocity error and error in one degree of freedom only) is approximately 2.9 cm in translation and 10.6° in yaw angle (with the nominal thruster configuration).

6.2.1 Trajectory-Following Performance: One Degree of Freedom

Figure 6.5 shows the trajectory-following performance for a single-degree-of-freedom maneuver. The robot base position is commanded to follow a quintic-polynomial trajectory in the $+x$ direction. The trajectory parameters are chosen to achieve the desired final position while setting initial and final velocity and acceleration to zero. Because of this, a couple

of seconds pass before the thrusters fire, even though the trajectory begins at $t=0$. The duration of the maneuver is set automatically, by keeping the peak acceleration within the actuation limits of the robot. In this case, the 1-meter slew took 20 seconds.

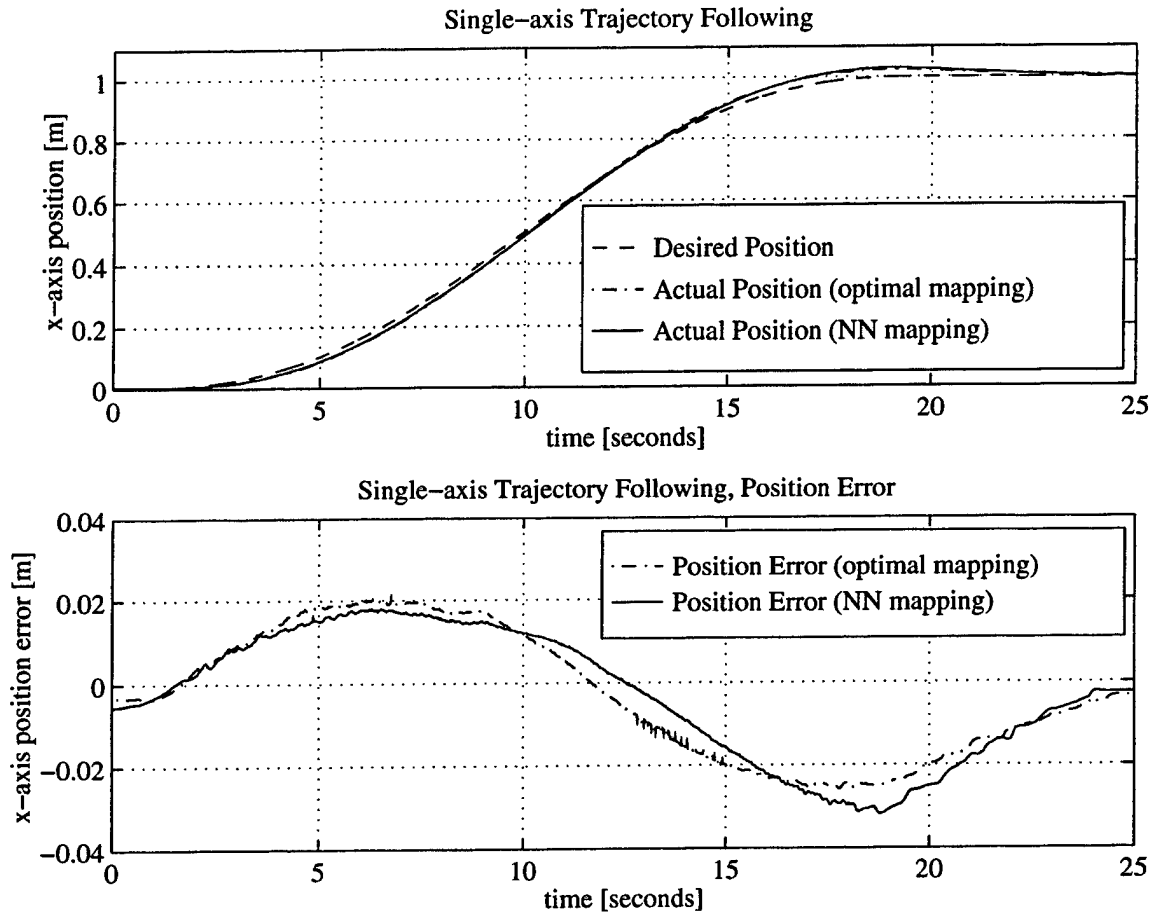


Figure 6.5: Single-Degree-of-Freedom Trajectory Following, Experimental Results

Trajectory-following performance is plotted for a quintic-polynomial trajectory of length 1 meter and duration 20 seconds, in the $+x$ direction. The nominal thruster configuration is present. An FCA network with 5 hidden neurons provides trajectory-following performance comparable to that of the optimal thruster mapper, which is implemented via exhaustive search.

The control system used is the one described above, except that no adaptation is required. Two different thruster mappers are used: a neural-network mapper implemented with an FCA network with 5 hidden neurons; and an optimal thruster mapper, implemented

via exhaustive search. Both mappers are aided by symmetries (as described in Chapter 2). Although the neural mapper is sub-optimal (mapping performance on a set of test data resulted in average force errors 3.5% greater than the optimal mapper), the trajectory-tracking performance is comparable. Due to the presence of feedback, the 3.5% mapping error is not significant, considering the other disturbing factors, such as imperfect thruster characteristics (steady-state and transient), sensor noise, and deadband.

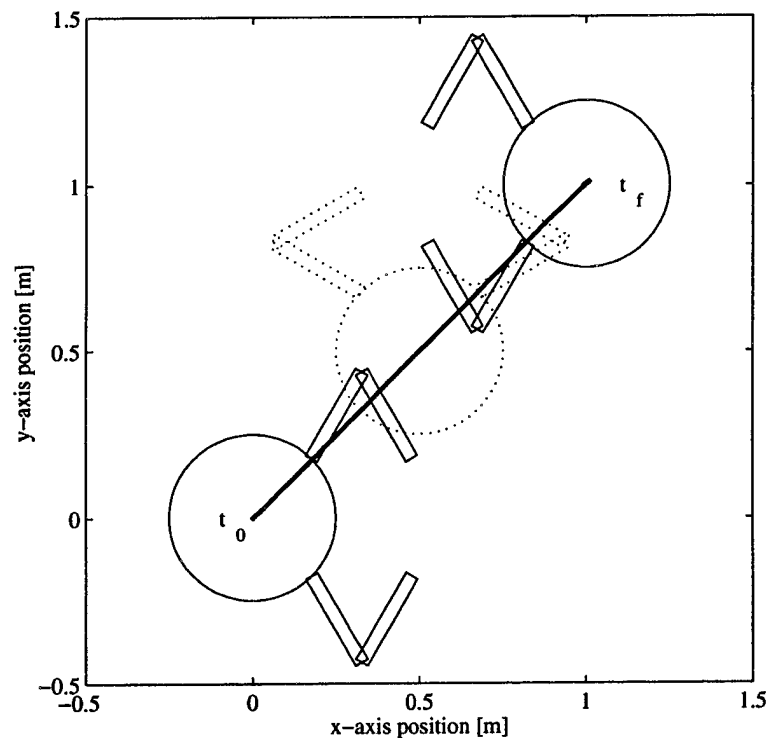


Figure 6.6: Multiple-Degree-of-Freedom Trajectory Following

The initial, middle, and final positions are illustrated for a multi-coordinate maneuver (x, y, ψ) . Quintic-polynomial trajectories are followed simultaneously in each of the three degrees of freedom. The position of the robot's geometric center obtained using the FCA mapper is also plotted (heavy black line).

6.2.2 Trajectory-Following Performance: Three Degrees of Freedom

For the multi-coordinate maneuver $([x, y, \psi])$ shown in Figure 6.6, good tracking is obtained again from both optimal and neural-network thruster-mapping components. In this 22-second-long trajectory, the robot simultaneously translates 1 meter in the $+x$ direction,

1 meter in the $+y$ direction, and 180° in the $+\psi$ direction. The position of the robot's geometric center is plotted in this figure. Quintic-polynomial trajectories are used for each degree of freedom. Each is executed simultaneously, with the peak acceleration for each degree of freedom limited to the physical actuation limits.

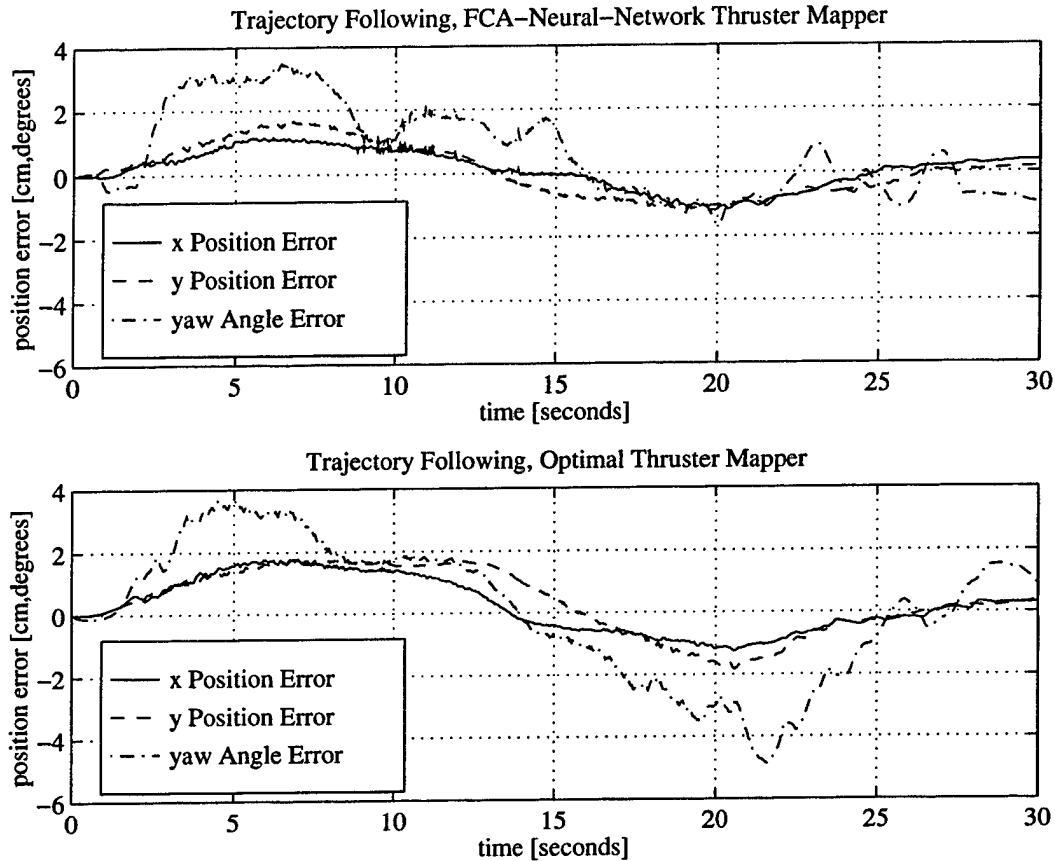


Figure 6.7: Multiple-Degree-of-Freedom Trajectory Following, Experimental Results

Trajectory-following error for the multi-coordinate maneuver illustrated in Figure 6.6 is plotted for each of the three coordinates, (x, y, ψ) . The performance of the FCA Mapper with 5 hidden neurons is excellent, and is comparable to that of the mapper implemented with exhaustive search ("Optimal Thruster Mapper").

Trajectory-following errors for this multi-coordinate maneuver are plotted in Figure 6.7, providing a comparison of the neural and optimal thruster mappers. This experiment used the same controllers used for the single-degree-of-freedom maneuver described above.

Again, the performance is excellent, and comparable results are obtained from the neural and optimal mappers.

6.2.3 Trajectory-Following Performance: Summary

This preliminary experiment has verified the applicability of the non-adaptive portions of the neural network control system. The neural mapper was shown to provide trajectory-following performance comparable to the optimal mapper, which was implemented by exhaustive search. As discussed earlier, the advantages of the neural-network approach do not apply strongly in this application until there is the requirement for reconfigurability.

6.3 Control Reconfiguration Problem Definition

Figure 6.8 shows the thruster layout in the nominal configuration as well as an example of a dramatically-failed configuration. The magnitude and direction of each thruster is shown. Nominally, each thruster produces 1 Newton of force, directed as shown. The failures were produced by mechanically changing the thrusters. Failures include: half-strength (②), plugged completely (⑧), angled at 45°(⑤ and ⑥), and angled at 90°(③ and ④). The 90° failure modes place high demands on the control-reconfiguration system, since they destabilize the robot (changing the direction of torque results in positive feedback!).

Requirements for the reconfigurable control system include:

1. The robot is not informed of the nature of these failures, or even that a failure has occurred. The adaptive system must first detect the failure(s), then identify the new thruster characteristics, and finally train and implement a new neural-network thruster mapper that accounts for these changes.
2. Control must be maintained at all times, but artificial excitation is allowed when position errors are small. This requirement keeps the robot within the bounds of the workspace (i.e. on the table), and allows it to carry on with its mission during the reconfiguration. For example, in this case, the robot can be commanded to move throughout the workspace during reconfiguration.
3. The entire adaptive system, including ID and re-training, is to be autonomous, requiring no user intervention at all.

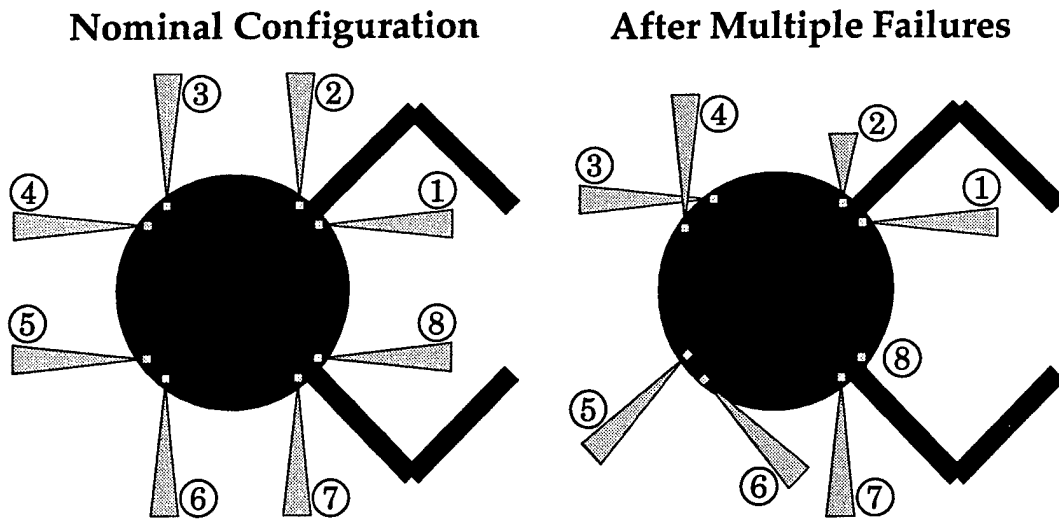


Figure 6.8: Example Failure Mode

Magnitude and direction of each of the eight thrusters is shown. Thruster failures were simulated mechanically with weaker thrusters and 90° and 45° elbows. Some of the elbows destabilize the robot by changing the sign of thrust in the ψ direction.

Six out of eight thrusters have failed in the case presented here. There is no theoretical limit to the number or type of failure that can be identified and be accounted for by the reconfigurable control system. However, there is a limitation if the controllability of the robot is impacted. For example, if both thrusters on the front of the robot (① and ⑧, as labelled in Figure 6.8) had failed completely, and no other thrusters contributed force in that direction ($-x$), there would be no actuation authority in the $-x$ direction. If it were necessary to accommodate failures like this, a higher level process (perhaps part of the trajectory generator) could command the robot to rotate, bringing working thrusters in line to provide the required thrust. In the example failure mode shown in Figure 6.8, there is sufficient actuation authority in plus and minus directions for all three degrees of freedom, so this issue is not yet addressed here.

To summarize, the basic reconfiguration strategy is to first detect the failure(s), then identify the new thruster characteristics, and finally train and implement a new neural-network thruster mapper. The structure of the control system is summarized in Figure 6.1. Running the adaptive process (neural-network training) in parallel with the identification process leads to stabilization within seconds, and causes the robot to be well-controlled during the identification.

6.4 Identification of Failures

Before reconfiguration can occur, the failures must be identified. The control system is not informed of the number or type of failures beforehand. It must detect, and subsequently identify, each of the failed thrusters. Failure detection and system identification are closely related in this implementation, so they are presented together here. The signal flow of the identification process is shown in Figure 6.9.

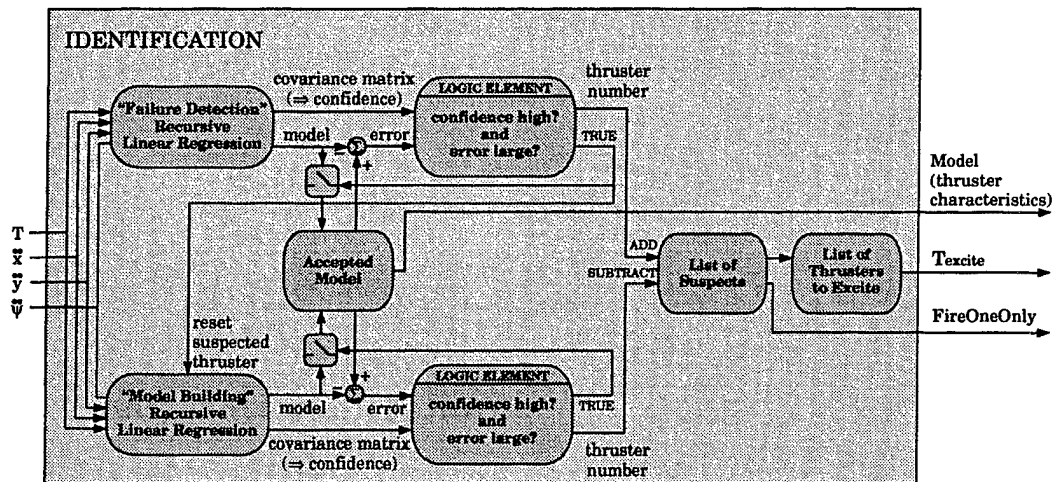


Figure 6.9: Identification Process – Signal Flow

Inputs are the thruster commands and acceleration signals, sampled from the real-time system at 10 Hz. The primary output is the model of thruster characteristics, a 3×8 matrix containing the forward mapping from thrusters to accelerations. Additional outputs, T_{excite} and $FireOneOnly$, are used in the control loop as part of the artificial excitation process.

6.4.1 Identification Summary

The task is to take in acceleration signals, $(\ddot{x}, \ddot{y}, \ddot{\psi})$, and thruster commands, and form a model of the strength and direction of each thruster. Since this is a purely linear relationship, there is no need for a neural network, and a linear-systems approach works well. When the thruster model is found to deviate from the nominal, a thruster failure is “suspected.” The thruster in question will be excited artificially to obtain more information about it, speeding up the identification process. When a certain level of confidence is reached and the new characteristics of the suspected thruster are confirmed, the artificial excitation is turned

off. Throughout the identification, model updates are sent to the neural network training component. This procedure, explained here for one thruster, runs in parallel for each of the eight thrusters.

There are a number of complicating factors for the system identification process: multiple thrusters may be fired simultaneously; the acceleration signals are corrupted by extraneous mechanical vibrations of the robot in the frequency range of interest; the response time of the thrusters is on the order of the sample period; and variations in the reservoir pressure during the firing of multiple thrusters affects the thrust output. These problems are addressed by filtering, reduction of the sample rate to 10 Hz, and design of the system ID process (e.g. waiting for a certain confidence level to be reached – i.e. collecting enough data – before declaring a thruster failure)

At the heart of the identification process are two recursive linear-regression processes running in parallel, incorporating acceleration and thruster signals as they become available. Each linear regression yields a 24-parameter model containing the x , y , and ψ acceleration associated with each of the eight thrusters.

6.4.2 Failure Detection

The first recursive linear-regression process is used primarily to detect when a failure has occurred for each thruster. This “Failure-Detection” process has a weighting factor that causes it to focus on the most recent few seconds of data (the weighting parameter decays exponentially in time – a “forgetting factor”). The time constant of the exponential decay was chosen to allow quick response to a failure, but still allowing enough data collection to prevent premature failure declaration.

This process, shown in Figure 6.9, is initialized with a model of the nominal thruster configuration; however, due to the forgetting factor, the model can change quickly based upon new data. The recursive process propagates a model (3×8 matrix representing the best estimate of the acceleration resulting from each thruster) and covariance matrix (8×8 matrix representing the amount of information collected for each of the eight thrusters).

Every time a thruster is fired, the ID process collects more information about that thruster, leading to a higher level of confidence in the estimate of the model parameters for that thruster. A “confidence factor” is calculated by taking the diagonal terms from the inverse of the covariance matrix. Due to the forgetting factor, the confidence factor does not rise monotonically – it will fall if the thruster is not fired for some time.

The model generated is compared with an "Accepted Model." The "Accepted Model" is the overall best estimate of thruster characteristics, and is the one sent to the neural-network-training process. It is set initially to correspond to the nominal thruster configuration, but may be updated by either of the two recursive linear-regression processes.

If an error in the model is detected (i.e. difference between identified and accepted models exceeds a certain threshold) for one or more thrusters, and the confidence level for the thruster(s) is high enough, a suspected thruster failure is declared. This decision process is shown as the LOGIC ELEMENT in Figure 6.9. When this condition is met, three things happen:

1. The suspected thruster is added to the "List of Suspects."
2. A reset signal is sent to the "Model-Building" Recursive Linear-Regression process. For the newly-suspected thruster(s) only, all prior information is to be erased. This is achieved by inverting the covariance matrix, zeroing the row and column corresponding to each newly-suspected thruster, inverting this matrix (setting the diagonal element to a small number so the inversion is possible), and setting the covariance matrix equal to this quantity. This has the effect of eliminating any prior information concerning the newly-suspected thruster, while leaving the rest of the model intact.
3. The identified model (for the newly-suspected thruster(s) only) is copied to the "Accepted Model." This is shown by the closing of the switch in Figure 6.9. This new "Accepted Model" is then sent immediately to the neural-network training process. There, a linear approximate solution is calculated immediately⁴, infused into an FCA network and copied to the robot. The result is a near-instantaneous stabilization of the robot, once the thruster failure has been detected.

Once a thruster is suspected, it will not be labelled as a suspect again until the adaptation process is reset. It will remain on the list of suspects until it is removed by the "Model-Building" Recursive Linear-Regression process.

⁴The *a priori* linear solution used here was found by assuming that the thrusters are capable of continuous-valued thrust output (a linearized version of this problem). The solution is an 8×3 pseudo-inverse of the 3×8 matrix which maps thrusters to base forces, F . Some simple adjustments are then made to account for the one-sided aspect of the thrusters (i.e. they can not produce negative thrust).

6.4.3 System Identification

The second recursive linear-regression process is used to build the model of the thrusters that is used for neural-network training. This "Model-Building" process does not have a forgetting factor – it incorporates all of the information equally, so the result is exactly the same as if a single batch-least-squares identification were run using all of the data.

This model is meant to be stable, basically changing only after a suspected thruster has been flagged. For this reason, it is initialized to the nominal model with a high level of confidence, and therefore does not vary significantly with random fluctuations in the data. However, when a thruster is flagged as being suspected by the "Failure-Detection" process, all information about that thruster is eliminated, as described above. Information about the other thrusters remains unchanged. New information about the suspected thruster is then incorporated into the ID process, and it reacts quickly to the new situation due to the elimination of old information.

The model and covariance matrices are updated recursively as new data comes in, as with the "Failure-Detection" Linear Regression. Since there is no forgetting factor, the confidence factor rises monotonically. When certain levels of confidence are reached and error criteria are met, the "Accepted Model" is updated. When confidence reaches a high level, the thruster(s) in question will be removed from the "List of Suspects."

During the time between first suspicion and final confirmation, the thruster in question is excited artificially, as described below.

6.4.4 Artificial Excitation

When a thruster is suspected of having failed, an artificial-excitation method will cause that thruster to fire more than it normally would, allowing for more information to be collected, and ultimately, expediting identification. The excitation is achieved with two basic methods: (1) when position-control errors are "small," a thruster may be fired open-loop for a brief period of time (until the thruster characteristics are identified or the errors are no longer "small;" (2) when position-control errors are "medium," the thrusters that are targeted for excitation are used exclusively for closed-loop control. When when position-control errors become "large," artificial excitation is suspended until the errors are reduced.

The excitation is controlled by two signals sent from the identification process to the robot:

1. A list containing which, if any, of the eight thrusters should be subjected to artificial excitation: T_{excite} .
2. A TRUE/FALSE command indicating whether the robot should limit itself to firing one thruster at a time: *FireOneOnly*.

List of Suspects

The "List of Suspects" component in Figure 6.9 keeps track of the suspected thrusters. Thrusters are added to the list by the "Failure-Detection" component, and then removed by the "Model-Building" component once their new characteristics have been confirmed (and possibly a confirmation of no change, if the initial failure-detection signal was erroneous).

FireOneOnly

If the "List of Suspects" contains any thrusters, *FireOneOnly* is set to be TRUE. Firing of multiple thrusters complicates the identification process, and identification accuracy will be improved if firing is limited to one thruster at a time. However, keeping the tracking error low is a priority, and may override this limitation. The flow of signals is summarized in Figure 6.2.

List of Thrusters to Excite

When suspected thrusters exist, they are copied directly to the List of Thrusters to Excite, and are sent to the robot as T_{excite} , shown in Figures 6.2 and 6.9. When all suspected thrusters have been cleared by the "Model-Building" process, any thrusters that have not yet been identified to a high level of confidence are added to T_{excite} . The logic behind this is that if some failures have been detected already, then whatever caused them (such as a plumbing failure, micro-meteorite impact, or intentional damage imparted by a graduate student) may have caused other as-yet-undetected failures, and identifying them quickly is important.

Thruster excitation will be attempted as long as at least one thruster remains in T_{excite} . If the robot position error is "large," no excitation will be used – the robot is most concerned with maintaining control. If the position error is "medium," and *FireOneOnly* is set to be TRUE, the robot will fire exactly one thruster. The thruster is chosen by finding the thruster from those in T_{excite} whose currently-estimated characteristics best matches the desired force

vector. In this middle region, artificial excitation takes place, but also serves to control the robot. If the position error is “small,” the robot will fire exactly one thruster. The thruster is chosen by finding the thruster from T_{excite} whose currently-estimated characteristics most differ from the nominal. Some hysteresis is added to prevent chatter across small/medium and medium/large boundaries.

This artificial excitation method leads to quick identification. For the case presented here, with 6 of 8 thrusters failed, the ID process consistently takes less than 60 seconds from when the first thruster is fired until the last thruster is identified to a high level of confidence.

A reconfiguration example, including error and thruster-firing plots, is presented at the end of this chapter. The effects of the ID process and neural-network training will be presented there.

6.5 Neural-Network Training

The system identification process can be completed less than 60 seconds, due to the artificial excitation. However, the control system requirements do not allow the system to remain unstable for that length of time (the size of the granite table is the limiting factor). Use of linear approximate solutions implemented via the FCA provide stability, but with a low level of performance. Running the neural-network training process in parallel with the ID process results in higher-performance control, as the nonlinear capabilities of the neural network optimize beyond the starting point of the linear approximation. The neural-network training process is shown in Figures 5.6 and 6.4.

The neural-network training is not activated until the first thruster failure is detected. From this point on, it is running continuously, using the most-recent thruster model provided by the ID process. When a significant change is detected, such as the total loss of a thruster, a linear solution is calculated, and the network starts from scratch, with the linear solution input via the FCA. When small changes are detected, such as the convergence of an identification on the new final value, learning is continued with the updated model.

The performance of the neural-network thruster mapper is evaluated periodically on a “test set” of thruster-mapping input-output patterns. If the performance (a weighted combination of force matching and gas conservation) is better than the test-set performance of the thruster mapper currently on the robot, it will be copied to the robot. The copying is

performed by sending the FCA weight matrix (as in Figure 4.2) over the wireless Ethernet. The neural-network function running on the robot swaps in these new parameters, resulting in an instantaneous change in the functionality of the thruster mapper.

6.6 Rapid Reconfiguration

One of the major issues in neural control is speed of learning. This is important in the robot application due to the goal of stabilizing an unstable system within a limited workspace. Rapid reconfiguration has been achieved here, and it is due to a combination of two aspects of the learning process: first due to the FCA, and second due to the growing of the network.

1. The FCA helps before training begins by immediately giving the network a good linear solution that stabilizes the robot.
2. The neural network is grown during training. This refers to starting with a few hidden neurons and gradually adding new ones as training progresses. With few hidden neurons, very-quick learning takes place, since fewer computations are required, and fewer training patterns are required (to avoid overfitting). As more hidden neurons are added, the learning rate slows down, but the greater functionality can be used to further optimize performance.

The network begins with 3 inputs, 8 hidden neurons, and 8 outputs, and gradually grows to 30 or more hidden neurons as training progresses. New hidden neurons are added when performance begins to plateau. To prevent overfitting, the training-set size is grown proportionally with the number of hidden neurons. With this arrangement, a mapping with about 30% error above optimal results in 30 seconds, 20% above optimal within 60 seconds, and 10% above optimal⁵ within 300 seconds, running on a Sun Sparc 10 workstation. As more hidden neurons are added, the network performance approaches optimality, but at the expense of slower training.

⁵Due to the use of discrete-valued actuators, there is almost always a force error vector. The error value reported here indicates that the average magnitude of the force error vector is 1.10 times the magnitude achievable with an exhaustive search.

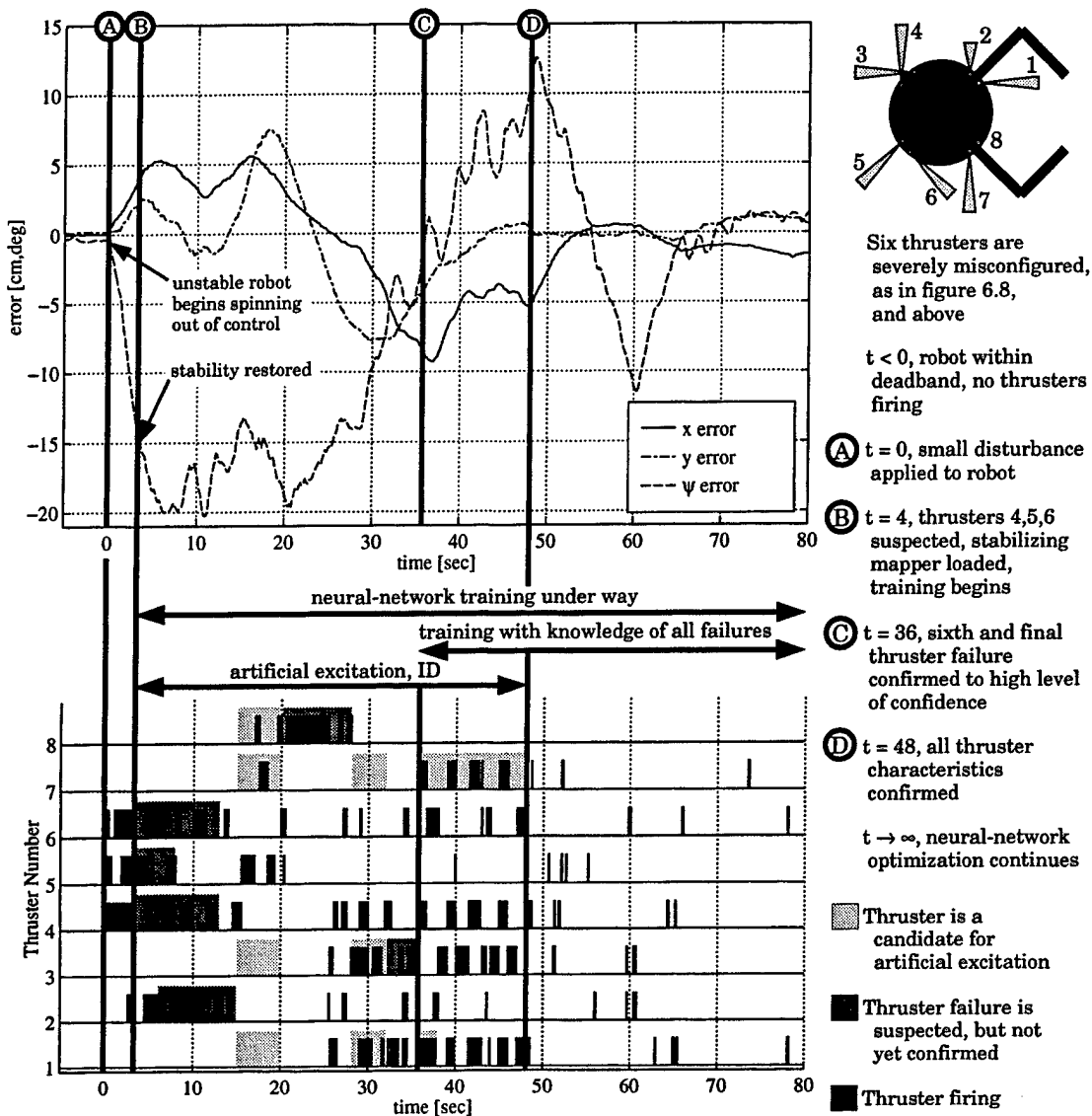


Figure 6.10: Experimental Results of Reconfiguration

$[x, y, \psi]$ position errors (desired - actual) are plotted during autonomous reconfiguration of the control system in response to the six severe thruster failures shown in Figure 6.8. Static control deadband is approximately ± 3 cm in translation and $\pm 11^\circ$ in rotation. The robot begins at rest within the deadband, is disturbed at $t = 0$, stabilizes itself within 4 seconds, and completes identification (aided by artificial excitation) after 48 seconds. The neural-network thruster mapper continues to optimize after the identification is complete. Thruster signals are shown in lower plot. Black rectangular regions indicate periods of thruster firing. Darkly-shaded regions indicate the time during which the thruster was suspected. In addition to artificial excitation of the suspected thrusters, excitation of un-suspected thrusters is used to expedite the identification process. These periods are indicated by the lightly-shaded regions.

6.7 Experimental Results of Reconfiguration

The previous sections have provided a description of the structure of the control system used for reconfiguration, as well as detailed descriptions of several of the key components. In this section, experimental data from a typical reconfiguration to recover from multiple destabilizing thruster failures is presented. Figure 6.10 plots the position errors (desired - actual) and thruster-firing histories during the reconfiguration.

The thrusters have been misconfigured severely, as in Figure 6.8. Before $t = 0$, the complete control system is active, but no thrusters fire, since the robot is drifting within the control deadband. With no thrusters firing, the thruster failures do not cause problems, but they also cannot be detected.

At $t = 0$ seconds, a small disturbance is applied to the robot. One of the first thrusters to fire is thruster ④ (shown in Figure 6.8 and the upper right corner of Figure 6.10), which is destabilizing in yaw, causing the robot to begin spinning out of control. The error signals in all degrees of freedom grow significantly following the disturbance, as seen in Figure 6.10. The robot spins to its left, causing ψ_{actual} to increase and ψ_{error} to decrease ($\psi_{error} = \psi_{desired} - \psi_{actual}$). The lower half of Figure 6.10 shows how thrusters ④ ⑤ and ⑥ stay on almost continuously (indicated by the black regions) due to the instability.

During this time ($t = 0 - 4$ seconds), the "Failure-Detection" process, shown in Figure 6.9, has been collecting data. At $t = 4$ seconds it declares failures in thrusters ④ ⑤ and ⑥. This triggers a series of events, all occurring at $t = 4$ seconds:

1. The "Accepted Model" is updated with the new parameter estimates for thrusters ④ ⑤ and ⑥, as identified by the "Failure-Detection" process. The exponentially-forgetting linear regression weights recent data more heavily than old data, so the model built between $t = 0$ and $t = 4$ may be used effectively as a crude first approximation of the characteristics of thrusters ④ ⑤ and ⑥.
2. The new "Accepted Model" is sent to the neural-network-training process, where a linear solution is calculated immediately and implemented on the robot in the form of an FCA network. The model at this point is just a rough estimate, and the linear controller is far from optimal, yet these methods combine to result in the immediate stabilization of the robot.

3. The neural-network-training process begins now, at $t = 4$ seconds, using the above-mentioned linear solution as a starting point for training a new thruster mapper to accommodate the updated model. This process continues indefinitely: model updates are received from the identification process and incorporated into the training; if the change in model is small (such as the change in force estimate from 1.03 N to 0.95 N), training continues; if the change is significant (such as the initial detection of a major failure), the training is re-started with the linear solution as a starting point.
4. Thrusters ④ ⑤ and ⑥ are added to the “List of Suspects” (shown in Figure 6.9), as indicated by the darkly-shaded areas for thrusters ④ ⑤ and ⑥ in Figure 6.10.
5. The T_{excite} vector is set to [4 5 6] and sent to the robot, along with a TRUE *FireOneOnly* signal. As discussed earlier in this chapter, a TRUE *FireOneOnly* signal means that the controller will fire only one thruster at a time (to obtain a more-direct identification), unless the regulation error becomes excessive. Furthermore, it will select thrusters to fire from only those that are listed in the T_{excite} vector, again, unless the regulation error becomes excessive. This will expedite the identification of these newly-suspected thrusters. The effect of these actions is immediately apparent in Figure 6.10: after $t = 4$ seconds, only one thruster is fired at a time, and the firing of thrusters ④ ⑤ and ⑥ is favored.
6. The “Model-Building” process, shown in Figure 6.9, is reset for thrusters ④ ⑤ and ⑥. That is, all information about thrusters ④ ⑤ and ⑥ in this model is immediately and completely eliminated, while the information about thrusters ① ② ③ ⑦ and ⑧ remain unaltered. Since a dramatic failure has been detected for thrusters ④ ⑤ and ⑥, these models are built from scratch, beginning at $t = 4$ seconds.

Each of the 6 items mentioned above occurred at $t = 4$ seconds.

The cumulative effect of these events at $t = 4$ is immediate and dramatic. The robot is stabilized immediately, as seen by the leveling off of position errors. This rapid stabilization is made possible by the quick estimation of what thrusters ④ ⑤ and ⑥ are doing, and the subsequent linear control design and implementation as an FCA-neural-network thruster mapper on the robot. The errors can be seen to increase initially due to the momentum of the robot, and it takes a few seconds for them to turn around, due to the limitation to firing of one thruster at a time, but the restoration of stability is clear. The initial identification

is so fast in this case that errors never grew to be "large." If they had, the restriction to firing one thruster at a time would have been lifted until the errors were reduced to lower levels.

At $t = 6$ seconds, thruster ② is suspected and the entire process described above is repeated: a linear mapper is calculated and implemented via FCA; T_{excite} becomes [2 4 5 6]; and the "Model-Building" process is reset for thruster ②. The position-error results are less dramatic, as the failure of thruster ② is not destabilizing.

At $t = 8$ seconds, the "Model-Building" process reaches a sufficient confidence value for its estimate of thruster ⑤. It updates the "Accepted Model" and removes thruster ⑤ from the "List of Suspects" and then from T_{excite} . This is indicated on the plot by the termination of the darkly-shaded region for thruster ⑤. It stops firing at that point, as it is no longer subject to artificial excitation.

At $t = 13$ seconds, thrusters ④ and ⑥ are confirmed similarly, as is thruster ② at $t = 15$ seconds. Observation of the thruster-firing histories and error plots shows what is happening during this time: the suspected thrusters are excited artificially, one at a time, resulting in a more-accurate identification. The regulation error is kept roughly constant during this period – i.e. within the bounds acceptable by the artificial-excitation process.

When thruster ② is confirmed at $t = 15$ seconds, no thrusters remain on the "List of Suspects." The remaining as-yet-unsuspected thrusters, [1 3 7 8], are added to the T_{excite} vector. They do not fire immediately, as the error is too high, but once it is within acceptable range (after thruster ⑤ is used to reduce the error), they fire.

Thruster ⑧ fires at about $t = 17$ and $t = 19$ seconds. Since no other thrusters are firing at these times, it does not take long to identify it as a suspect, which occurs at $t = 20$ seconds. Thruster ⑧ simulates a complete thruster failure, producing only about 1/40th of the thrust from a nominal thruster. It stays on for several seconds, yet the error plots are fairly straight lines during this period, indicating constant momentum and very little thrust. The "Model-Building" process confirms this at $t = 28$ seconds, removing it from the list of suspects.

With an empty list of suspects, thrusters ① ③ and ⑦ are labeled for artificial excitation. Thruster ③, the other 90° elbow (the second strongly-destabilizing failure) is fired for the first time, causing a second loss of stability. Thruster ③ had not been excited up until this point for two reasons:

1. The artificial excitation algorithm restricts thruster use to those thrusters that have already been tagged as suspects, unless regulation errors exceed a certain limit.
2. The new characteristics for thruster ④ match the nominal characteristics of ③ except that ④ is more efficient in producing torque. This makes ④ more likely to fire than ③ for most (but not all) force-vector requests.

These two effects conspire to prevent the firing before the first short pulse occurs at $t = 26$ seconds (when both of the above conditions allow firing for the first time), and then for a sustained firing at $t = 28$ seconds (when thruster ③ is targeted for artificial excitation to expedite the identification).

The instability is caught quickly, since the rest of the plant is well-characterized at this point in the identification. When its new characteristics are confirmed at $t = 36$ seconds, this represents the identification of the sixth thruster failure and the final reset of the neural-network training process (the final major change detected in thruster characteristics).

Thrusters ① and ⑦, the only un-altered thrusters, are confirmed to have nominal characteristics at $t = 48$ seconds, marking the end of the artificially-excited identification phase. From this point on, model updates are small, and made only when they exceed a certain threshold, so as not to disrupt the neural-network-training process. In this case, one final minor adjustment was made at $t = 95$ seconds.

With the completion of identification (all thrusters identified to a high level of confidence) at $t = 48$ seconds, artificial excitation ends, and the sole objective of the controller is to regulate to the desired position. Position errors in all degrees of freedom are reduced immediately, as seen in the top half of Figure 6.10 between $t = 48$ and $t = 55$ seconds. There is some overshoot in ψ , peaking at $t = 60$ seconds. This is due primarily to the deadband associated with the on-off thrusters⁶. Following this single major overshoot, the regulation error is reduced to be well-within the static deadband, and results in an occasional single thruster pulse, as shown in the thruster-firing plot from $t = 66 - 80$ seconds.

⁶Due to the control structure, PD-control gains, and thruster-mapping cost function selected, a deadband exists within which the thrusters will not fire, even with an optimal thruster mapper. While the size of this deadband is difficult to characterize due to the thruster-coupling effects, the maximum static deadband (assuming zero velocity error and error in one degree of freedom only) is approximately 2.9 cm in translation and 10.6° in yaw angle (with the nominal thruster configuration).

6.8 Summary of Experimental Results

The performance of the neural-network-based reconfigurable control system, displayed in Figure 6.10, was excellent, providing stabilization of the robot within four seconds, despite the presence of six major thruster failures.

The system-level design of the control system, discussed in Chapter 3, resulted in the selection of a linear-systems approach for identification, but a neural-network approach for the thruster mapping. The decision concerning identification was critical to achieving the quick failure detection and identification that resulted (initial recovery occurred after only four seconds). The neural-network mapper provided flexibility in adapting to the changes in thruster characteristics.

The new Fully-Connected Architecture, discussed in Chapter 4, allowed the neural network to make immediate use of the model provided by the identification component. A linear approximate thruster mapper was calculated immediately following the initial failure detection at $t = 4$ seconds. Implementation of this linear solution with the FCA provided immediate stabilization. This was followed by optimization of the nonlinear portion of the neural network, resulting in near-optimal performance within 2 minutes. This performance was obtained despite the implementation on a serial microprocessor; implementation on parallel-processing hardware would provide dramatically-faster performance.

The new learning algorithm described in Chapter 5 was used to allow gradient-based optimization, in spite of the presence of the non-differentiable thrusters. The use of gradient information to direct the optimization resulted in a dramatic improvement in learning rate over what could have been obtained with a method that was not gradient-based.

2

Chapter 7

Conclusions

This final chapter consists of two sections. The first section summarizes the findings of this research. The second gives suggestions for future research.

7.1 Summary

This thesis has described four new developments in neural-network control that grew out of a research program using a laboratory-based experimental prototype of a free-flying space robot. The advances were motivated by, and developed for, a complex reconfigurable thruster control problem applicable to real spacecraft. Focussing on a specific complex control task was useful in identifying some of the real-world issues in neural-network control. The work has led to the conclusions that follow.

7.1.1 System-Level Design Approach: The Superiority of Hybrid Control

One basic conclusion from this research is that a *combination* of the nonlinear processing capabilities of neural networks with existing conventional control theory can be very powerful. A careful system-level analysis and design that considers the costs and benefits of all available tools from the fields of neural networks and control is likely to be more successful than an approach that has already decided up-front what tools will be used. An objective evaluation of the costs and benefits of each available approach, followed by an efficient integration of these approaches, and development of extensions to existing theory where they are needed, constitutes a powerful strategy for solving complex nonlinear control problems in the real world.

In this work, the costs and benefits of neural network approaches have been outlined, and objectively compared with alternative conventional approaches. The overall success of the reconfigurable control system resulting from application of this strategy provides additional support to this conclusion.

To summarize the criteria for valuable applications of neural networks, it was shown that applications should involve systems with inscrutable (if the exact form can be derived, that will probably be more effective than a neural approach) nonlinearities (if the system is purely linear, linear methods tend to have better convergence and provability characteristics than do neural networks) that may require some form of adaptation (neural networks excel here, since they are already designed for iterative training). Additionally, neural networks are well suited to applications that require the processing speed of a parallel computer, since their architecture is inherently parallel.

7.1.2 Quick Adaptation – FCA

A major issue in neural network control, and particularly in reconfigurable or adaptive control, is the requirement for speed of adaptation. The control application addressed here highlights this need, since the robot suffers a destabilizing change in its actuators. The instability required a recovery within seconds, not minutes or hours. And this was achieved.

A new fully-connected neural network architecture (FCA) was developed to address this speed issue. It is a feedforward network that brings together for the first time many useful architecture features developed by other researchers. It has connections beyond those provided by a layered network, yet is trainable with backpropagation. Aided by a systematic complexity control scheme, this network was shown to have certain advantages over layered networks, particularly for control problems.

The most significant advantage in this application is the ability to incorporate seamlessly a linear solution before training begins. In control, as with other fields, linear approximate solutions are often easily calculated based upon prior knowledge of the system properties. Quickly calculating the linear approximation, and directly inputting that solution into the neural network facilitates rapid adaptation without the need for time-consuming iteration. This feature may be especially useful if it allows immediate stabilization, as it does here.

Another feature of the FCA that contributes to its rapid rate of adaptation is the growing of the network. A smaller (fewer hidden neurons) network converges more rapidly, since there are fewer parameters to adapt, fewer calculations need to be made, and a smaller

training set is required to prevent over-fitting. The network begins with a small number of hidden neurons, and gradually adds more, as greater functional capacity is called for.

7.1.3 Gradient-Based Optimization for DVFs – Noisy Sigmoids

A new technique was developed that extends gradient-based optimization (e.g. backpropagation learning) for the first time to systems involving discrete-valued functions (DVFs) (which are not continuously differentiable). This approach was motivated by the need for adapting to the changing properties of the on-off thrusters used to control the robot. The solution to this difficult, but specific problem is to approximate the DVFs with noisy sigmoids. This simple solution has been demonstrated to extend to other applications involving optimization with DVFs. One important example is for neural networks built with hard-limiting nonlinearities rather than sigmoid functions. These are attractive because they are cheaper and easier to implement in hardware. Another example is design optimization for systems with DVFs (e.g. a structural design optimization that chooses between 1/4 inch and 3/8 inch wall thickness, 3, 4, 5, or 6 screws, and 2 or 3 beams). This has not yet been demonstrated, but it is expected to work well.

The modification to backpropagation is very small, simply requiring continuous-approximation of the DVFs, and injection of noise on the forward sweep; yet the improvement in network performance is dramatic.

It works by solving the problem unaddressed by earlier methods: roundoff error. For gradient-based optimization to work, during training a gradient must exist and be non-zero; so the obvious first step is to approximate the DVF with a continuous approximation which is continuously differentiable (e.g. sigmoid-based functions). This method provides some success, but errors result when extensive use of the transition regions occurs during training, and round off to the nearest discrete level is required at run time.

Identifying this roundoff error as the problem was probably as important a step as the solution. Identification was aided by the ability to compare the results to a known optimal solution, as is known for the thruster-mapping problem. Without knowing the level of performance that was possible, the problem of roundoff error might never have been identified.

Once the problem was identified, several attempts were made to address it. The successful method involves the simple modification of injecting noise into the sigmoid during training. Noise creates random outputs if the transition regions are used, but has little effect

if saturated regions close to the allowed discrete levels are used. Therefore, the transition regions are avoided during training, and roundoff error is minimal at run time.

7.1.4 Experimental Demonstration of Reconfigurable Control System

The task of rapid reconfiguration in response to destabilizing thruster failures first motivated these developments and then drew upon them heavily in the experimental demonstration.

- The system-level control design approach resulted in a system related to a conventional indirect adaptive control system, and used a neural network as an efficient, adaptive method to implement the nonlinear thruster mapping component.
- The FCA resulted in near-immediate stabilization and rapid learning, due to the feedthrough connections, and growing of the network.
- The gradient-based optimization for discrete-valued functions resulted in a more accurate mapping due to a good approximation to the on-off thrusters, while allowing the rapid optimization made possible with use of gradient information.

When trained off-line and tested experimentally on the real robot, the neural-network thruster mapper provided near-optimal performance during multiple-degree-of-freedom trajectories. Arbitrary accuracy could be obtained depending upon the size of the network used. With no thruster failures (so symmetries may be used) and 5 hidden neurons, a thruster-mapping force error of 3.5%¹ was achieved. This small error is barely perceptible due to the use of feedback in the control system.

When reconfiguring the control system in response to previously-unknown, major, destabilizing thruster failures, rapid stabilization and optimization were achieved, as seen in Figure 6.10. Detection of a destabilizing failure took from 2-5 seconds (the problem is complicated due to noisy accelerometers, and to firing multiple thrusters simultaneously). After the initial detection, calculation of a stabilizing linear approximate solution, and implementation via the FCA took less than one second. As thrusters are suspected to have changed characteristics (e.g. to be angled at 45° or 90°, have degraded thrust output, or be plugged completely), they are artificially excited to speed up the identification. Stability and closed-loop control are maintained during this time. With six out of the eight

¹Due to the use of discrete-valued actuators, there is almost always a force error vector. The error value reported here indicates that the average magnitude of the force error vector is 1.035 times the magnitude achievable with the optimal thruster mapper.

thrusters failed (two were strongly destabilizing), the identification converges within about 60 seconds. The neural network thruster mapper is trained concurrently with the identification, and the model used for training is continuously updated. Near-optimal performance is achieved by the end of the identification phase (e.g. 20% error above optimal), and it improves to arbitrary accuracy with further training and growing of the network.

7.2 Recommendations for Future Work

Performing this research generated a number of ideas for possible future research. The following is a list of possible future project ideas. As this research has encompassed a broad range of issues, from the details of an experimental implementation to the derivation of a new optimization algorithm, the following suggestions have been grouped into specific areas.

7.2.1 Integration of Neural-Network and Conventional Control

- One of the conclusions of this research has been that the merging of neural network technology with control systems engineering can lead to the development of highly-capable control systems. Much neural network theory and much control theory already exist that could produce significant advances in control capability simply through astute *integration* of them. With this in mind, some possible research areas that are related to the robot application are suggested. Control systems for physical plants that are difficult to model, and have inscrutable nonlinearities are good targets. These may include high-angle-of-attack aerodynamics, or underwater robot control.
- This research has presented a reconfigurable control system implemented in real-time. Reconfigurable control is an important area of research in the military aircraft industry, as it is desirable to have a control system that can recover from partial system failure – e.g. battle damage, where portion of a wing is shot off, or some control surfaces become inoperable. Neural networks are attractive for this application due to their ability to deal with the nonlinear aerodynamics, adaptive capability, and real-time processing speed if implemented in hardware. The reconfiguration time requirement for an unstable aircraft is likely to be measured in hundredths of a second, instead of seconds, as for the space robot application. Hardware implementation of the concepts

developed here, combined with further developments tailored to the aircraft application, could make this goal feasible. A memory-based approach may be required due to the high speed requirement and limited data availability.

- Feedforward neural networks built with sigmoidal activation functions were used exclusively in this research, primarily because they appear to hold much promise for neural network control applications in general. Other neural network architectures exist, and may prove to offer advantages depending upon the application:

1. Radial Basis Function (RBF) networks may be viable, as described in Chapter 1.
2. Sigmoidal (and RBF) networks work by attempting to form a *function* that “fits” the data (training cases) they are presented. The hope is that this function forms a generalization of the training data, and the network will perform well on new data. However, other neurally-motivated approaches are memory-based, rather than function-based. Rather than learn a generalizing function of the data, these methods remember the training inputs directly, and interpolate/extrapolate as needed when new points are input. CMAC [2] [3] is one example of a memory-based neural network that has been used successfully in control applications [23]. Briefly, the tradeoff is that memory-based approaches learn very quickly, since they simply remember each training input; but the recall can be much slower, since the nearest neighbors must be found and then interpolated to produce an output. Function-based approaches train more slowly, as they must compress the data into the functional format created by the network topology, but have very fast recall.

7.2.2 Optimal (Hybrid) Combination of Neural Networks with Conventional Control: FCA

- The ability to incorporate prior knowledge has proven to be the most useful aspect of the FCA for this application. It is currently limited to linear solutions. Extensions to other types of solutions, perhaps closely tailored to conventional control methods may be useful.
- Fuzzy logic has been an area of recent interest in the control community recently. The appeal of fuzzy logic is its ability to incorporate knowledge from a human expert into

a logic system. A number of rules are programmed by the expert (e.g. “if you’re close to your destination, and the brakes aren’t too hot, and you’re going medium speed, and there are no immediate obstacles, then apply the brakes gently”), and the fuzzy logic is used to blend the effects of these rules together, in a more-graceful manner than is possible with crisp logic. This ability to interface with human expertise has proven to be useful for tasks for which such human expertise can be encoded into a logic system. Research in incorporating this type of knowledge may be useful. Again, an astute “hybrid” combination of fuzzy logic and conventional control may well be superior to either alone.

- The major drawback is the possibility for overfitting, and a complexity control method was applied to address this issue here. Several other neural network pruning methods exist that may deserve investigation.
- The general problem of determining the optimal topology of neural networks (including the number and connections of nonlinear elements) remains an important research issue. The technique presented here (use of the FCA to allow the implementation of any possible set of layers or connections, and the gradual addition of hidden neurons until acceptable performance is reached) provides a workable solution, but there is room for other advances that may improve the efficiency of the topology selection.

7.2.3 Gradient-Based Optimization for DVFs

- A significant feature of the algorithm presented here is that the specific type of noise used (e.g. Gaussian, uniform, etc.) is not important. Furthermore, for bi-level DVFs, the algorithm is robust to wide variations in the magnitude of the noise distribution. Although tuning the noise level is a relatively simple operation, elimination of this requirement is a clear advantage. Unfortunately, for DVFs with more than two levels, tuning of the noise level is required (although selection is fairly robust and intuitive). It has been suggested that this tuning may be avoided if a different form of the continuous approximation function is chosen [45].
- The algorithm has been applied to two very different applications so far – optimization of a neural network built with hard-limiters, and optimization of a neural-network controller for a system with on-off actuators. The simplicity of the algorithm, combined

with the success on two unrelated problems, causes optimism for the applicability of the algorithm to other fields. One clear application is design optimization, as mentioned in the research summary above.

- The application to neural networks built with hard-limiters has provided an efficient training algorithm for a new class of neural network hardware. Study of the details of such an implementation, or modification of the algorithm to allow for on-chip learning would be beneficial.

7.2.4 Thruster Mapping

These suggestions reflect further advancements toward a better thruster control system. This project was chosen as a challenge problem to highlight some of the current issues in neural network control. However, if the goal were to make the best thruster control system possible, these are some issues that have not been fully addressed in this research.

- A more complex mathematical model of the robot could be used:
 1. Include thruster transients: due to the response time of the solenoid valve, and additionally, the finite size of the chamber between the valve and the nozzle, the thrust output is time-dependent. These effects were ignored.
 2. Include low-gas-reservoir effects: the amount of gas remaining in the high and low pressure reservoirs affects the thrust output. In these experiments, reservoir levels were kept close to nominal so these effects were minimized (and ignored).
 3. Account accurately for multiple thruster firings: due to limited flow in the plumbing, the thrust from each thruster is reduced when multiple thrusters are fired simultaneously (on the order of 10% loss per extra thruster). A simple linear approximation was used for these experiments.
- Identification of thruster characteristics is performed by analyzing the direct relationship between thruster firings and the resulting acceleration. While this is a robust, self-contained ID scheme that meets the requirements of this application, incorporation of position information (available from a vision system or Global Positioning System) with a Kalman filter should improve the identification.

- The initial decision to separate the controller into control (PD controller that ignores discrete-actuator effects) and thruster mapping components, was made to simplify the problem. It simplified the problem at the expense of optimality. A subsequent step, made possible by the developments in this work, is to merge the robot-base controller and thruster mapper design into a single component. This should result in improved total system performance, as the neural network provides a fast method for calculating an approximation to the optimal control solution that can be calculated in real time. One approach could be to use the network for trajectory optimization, accounting for the on-off actuators.

2

Appendix A

Thruster-Mapping Cost Function

This Appendix presents a variation on the cost function used to define the optimal thruster mapping. This function places weighted costs on the force-mapping error and the amount of gas used.

The cost functions that were used for the neural-network developments in Chapters 4 and 5 and the experiments in Chapter 6 were both presented in Chapter 2. The complexity-control term used to augment those cost functions was presented in Chapter 4. This Appendix presents an alternative cost function that has merit, but was not used extensively in this research.

In minimizing the force error (and possibly also gas usage) only, the thruster mapper does not consider the dynamics of the plant. It assumes that the F_{des} vector output by the controller feedback law is chosen carefully enough that it needs only concern itself with producing the closest matching F_{act} . In fact, in this application, the controller component is a simple proportional-derivative controller (shown in Figure 2.12) that does not take into account the thruster limitations.

The decision to separate control and mapping components was made largely for simplicity in design. Ideally, the controller component would be aware of thruster limitations – for example, a bang-bang controller instead of a PD controller. Implementing a bang-bang controller in real time would probably result in the same decision that was made for the thruster mapper here: use a neural network to implement a nonlinear approximation to the optimal controller – one that can be computed in real time. In this case, it may be beneficial to merge the neural-network control component with the neural-network mapping component. The single neural network would then map the six-element state error vector,

$X_{err} = [x_{err}, y_{err}, \psi_{err}, \dot{x}_{err}, \dot{y}_{err}, \dot{\psi}_{err}]$, directly to the binary eight-element vector of thrusters to fire, $\mathbf{T} = [T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8]$. This even-more-complex nonlinear control problem is not addressed here, but a much-simpler first step is proposed.

A first step to address the presence of plant dynamics in the thruster mapper is to use an alternative error-weighting scheme. This plan does not address the on-off nature of the thrusters directly, but does incorporate the effect of variations in the mass properties of the robot. For example, if the moment of inertia were relatively small compared to the mass, it might be more important to match the desired torque than the desired translational forces.

In this plan, instead of minimizing normalized force error, the force error is considered to be a disturbance, and the resulting normalized acceleration error vector is minimized. The normalization factor chosen is the acceleration vector resulting at the perimeter of the base, radius r , when a single thruster is fired. In this instance, the error becomes:

$$\min_{\mathbf{T}} J = \left[\left(\frac{F_{x_{err}}(\mathbf{T})}{mass} \right)^2 + \left(\frac{F_{y_{err}}(\mathbf{T})}{mass} \right)^2 + \left(\frac{\tau_{\psi_{err}}(\mathbf{T})}{(I_{\psi}/r)} \right)^2 + \alpha_{gas} \sum_{i=1}^8 T_i \right] \quad (\text{A.1})$$

where,

- J = thruster-mapping performance cost
- \mathbf{T} = binary thruster values, $[T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8]$
- i = thruster number
- $F_{x_{err}}(\mathbf{T})$ = net force error in x-direction, $(F_{x_{des}} - F_{x_{act}})$, resulting from \mathbf{T}
- $F_{y_{err}}(\mathbf{T})$ = net force error in y-direction, $(F_{y_{des}} - F_{y_{act}})$, resulting from \mathbf{T}
- $\tau_{\psi_{err}}(\mathbf{T})$ = net torque error about ψ -axis, $(\tau_{\psi_{des}} - \tau_{\psi_{act}})$, resulting from \mathbf{T}
- $mass$ = robot total mass
- I_{ψ} = robot moment of inertia about ψ axis
- r = robot base radius

Due to the dimensions and mass properties of the robot used in these experiments, this ends up being close to the original cost function, and the actual performance improvement in this case is minimal.

Appendix B

Accelerometer Specifications

Two Systron Donner 4310A Linear Servo Accelerometers were used on the robot, as described in Sections 2 and 6. This Appendix contains specifications and dimensions for these accelerometers [10].

Model 4310A & F Configuration

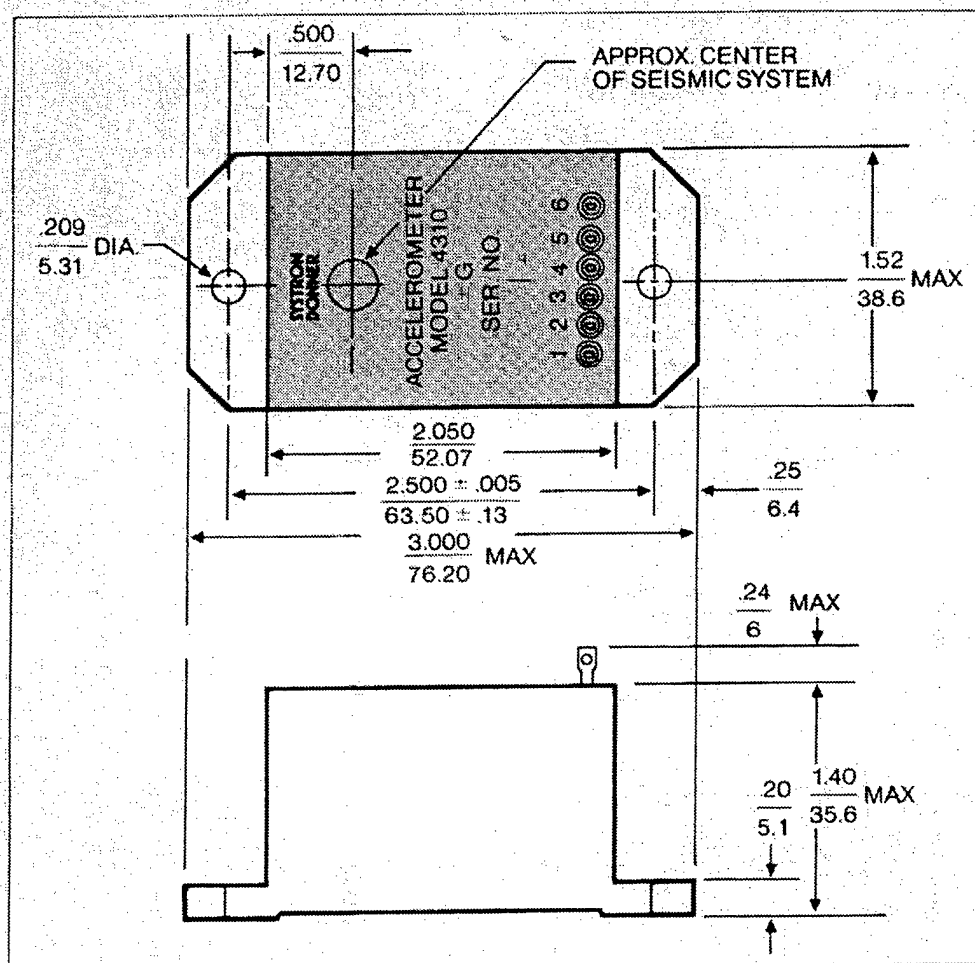


Figure B.1: Systron Donner 4310A Accelerometer - Dimensions

Accuracy

Parameters	Symmetrical Output (Option A)	Telemetry Output (Option B)
Non-Linearity	<0.05% of full range	
Hysteresis & Non-repeatability	<0.02% of full range	
Resolution	<0.001% of full range	
Zero Output (Null)	<0.05% of full range	2.5 v \pm 1%
Temp. Coeff. of Null (of full range)	<0.0018% per °C (<0.001% per °F)	<0.0108% per °C (<0.006% per °F)
Output Noise	<0.0075 vrms	<0.0025 vrms
Cross Axis Sensitivity (Referred to true sensitive axis)	<0.002 g/g of applied acceleration	
Temp. sensitivity of Scale Factor	<0.018% per °C (<0.01% per °F)	
Natural Frequency (90° Phase Shift) (Dependent on g range)	50-250 Hz	
Damping Ratio at 22°C (72°F)	0.7 \pm 0.1 Other ratios available	

Figure B.2: Systron Donner 4310A Accelerometer - Accuracy Specifications

Physical

Parameters	Symmetrical Output (Option A)	Telemetry Output (Option B)
Range*	$\pm 0.5 \text{ g}$ to $\pm 35 \text{ g}$	$\pm 0.5 \text{ g}$ to $\pm 30 \text{ g}$
Input Power	$\pm 15 \text{ vdc} \pm 10\%$ at 10 ma maximum	28 vdc $\pm 10\%$ at 20 ma maximum
Voltage Output— Nominal	$\pm 7.5 \text{ vdc}$ full scale set to $\pm 1\%$ at factory	0.2 to 4.8 vdc full range, nominal about 2.5 volt bias** (symmetrical range)
Output Impedance	5 k Ω nominal	7.4 k Ω nominal
Output Current	3 ma full range	2 ma full range
Electrical Connections	6 solder terminals	7 solder terminals
Case Alignment (each orthogonal axis)	$< 1^\circ$ to true sensitive axis	
Weight	128 grams (4.5 oz.) electrically damped 170 grams (6 oz.) fluid damped	

*Optional ranges of less than 10mg and greater than 100g are available.

**Signal reference common with power return.

Figure B.3: Systron Donner 4310A Accelerometer - Physical Specifications

Environmental Parameters

Temperature Range†	-54°C to $+93^\circ \text{C}$ storage (-65°F to $+200^\circ \text{F}$) -40°C to $+93^\circ \text{C}$ operating (-40°F to $+200^\circ \text{F}$)
Shock Survival	100g, 11 msec
Vibration Survival (Range Dependent)	15 g rms, bandwidth 20 to 2000 Hz (0.12 g ² per Hz)
Humidity, Salt, Spray Fungus, Sand & Dust	Hermetically sealed (meets MIL-E-5272C)
Ambient Pressure	0 to 5 atmosphere absolute

†Applies to electrically damped units. For fluid units consult factory.

Figure B.4: Systron Donner 4310A Accelerometer - Environmental Specifications

Bibliography

- [1] H.L. Akin and T. Tasoglu. Nuclear reactor control using backpropagation neural networks. In M. Baray and B. Ozguc, editors, *Computer and Information Sciences VI. Proceedings of the 1991 International Symposium*, volume 2, pages 889–98, Amsterdam, Netherlands, 30 Oct.-2 Nov. 1991. Elsevier.
- [2] J.S. Albus. Data storage in the Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control, Transactions of the ASME, Series G*, 97(3):228–233, September 1975.
- [3] J.S. Albus. A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control, Transactions of the ASME, Series G*, 97(3):220–227, September 1975.
- [4] P.L. Bartlett and T. Downs. Using random weights to train multilayer networks of hard-limiting units. *IEEE Transactions on Neural Networks*, 3(2):202–210, March 1992.
- [5] M. Bichsel. Image processing with optimum neural networks. In *First IEE International Conference on Artificial Neural Networks*, pages 374–7. IEE, October 1989.
- [6] Arthur Earl Bryson, Jr. and Yu-Chi Ho. *Applied Optimal Control*. Hemisphere Publishing Corporation, New York, NY, 1975.
- [7] A.B. Bulsari, B. Saxen, and H. Saxen. A chemical reactor selection expert system created by training an artificial neural network. In F. Dehne, F. Fiala, and W.W. Koczkodaj, editors, *Advances in Computing and Information - ICCI '91. International Conference Proceedings*, pages 645–56, Ottawa, Ontario, Canada, May 27-29 1991. Springer-Verlag.

- [8] Robert H. Cannon, Jr. *Dynamics of Physical Systems*. McGraw-Hill, New York, NY, 1967.
- [9] D.L. Chester. Why two hidden layers are better than one. In *International Joint Conference on Neural Networks*, volume 1, pages 265–268, Hillsdale, NJ, July 1990. Erlbaum.
- [10] Systron Donner. 4310 linear servo accelerometer, specifications. 2700 Systron Drive, Concord, California 94518, 800-227-1625.
- [11] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 190–196. Morgan Kaufmann Publishers, Los Altos, CA, 1990.
- [12] R.B. Ferguson. Chemical process optimization utilizing neural network systems. In *SICHEM 92 (Seoul, Korea)*, Hillsdale, NJ, July 1992. Erlbaum.
- [13] S. Geva and J. Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Transactions on Neural Networks*, 3(4):621–4, July 1992.
- [14] S. Grossberg. Adaptive pattern classification and universal recoding: Part I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134, 1976.
- [15] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. In *Proceedings of the 1987 IEEE Symposium on the Foundations of Computer Science*, pages 99–110, 1987.
- [16] C. Hall. Neural network technology: Ready for prime time? *IEEE Expert*, pages 2–4, December 1992.
- [17] D. Hammerstrom. Neural networks at work. *IEEE Spectrum*, pages 26–32, June 1993.
- [18] M.E. Hoff, Jr. *Learning Phenomena in Networks of Adaptive Switching Circuits*. PhD thesis, Stanford University, Stanford, CA 94305, July 1962. Tech. Rep. 1556-1, Stanford Electron. Labs.

- [19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359-366, 1989.
- [20] K. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthorp. Neural networks for control systems - a survey. *Automatica*, 28(6):1083-1112, 1992.
- [21] Don R. Hush. Classification with neural networks: a performance analysis. In *IEEE International Conference on Systems Engineering*, pages 277-80, Fairborn, Ohio, August 24-26 1989. IEEE.
- [22] Don R. Hush and Bill G. Horne. Progress in supervised neural networks: What's new since Lippmann? *IEEE Signal Processing Magazine*, pages 8-39, January 1993.
- [23] W. Thomas Miller III, F.H. Glanz, and L.G. Kraft. Application of a general learning algorithm to the control of robotic manipulators. *International Journal of Robotics Research*, 6(2):84-98, 1987.
- [24] W. Thomas Miller III, Richard S. Sutton, and Paul J. Werbos, editors. *Neural Networks for Control*. Neural Network Modeling and Connectionism. The MIT Press, Cambridge, MA 02142, 1990.
- [25] Haiping Jin. An optimal thruster configuration design and evaluation for Quick STEP. In *IFAC Symposium on Automatic Control in Aerospace*, September 12-16 1994.
- [26] M.I. Jordan. A neural network for improving terrain elevation measurement from stereo images. In *Applications of Digital Image Processing XIV*, pages 179-87, San Diego, CA, July 22-26 1991. SPIE. Proceedings of the SPIE, vol. 1567.
- [27] Tuevo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59-69, 1982.
- [28] A.N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk. USSR*, 114:953-956, 1957.
- [29] R.P. Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech and Signal Processing Magazine*, 4(2):4-22, April 1987.

- [30] Ho Chung Lui. Decision boundary formation from the backpropagation algorithm. In *International Symposium on Computer Architecture and Digital Signal Processing*, volume 1, pages 17–22. IEE, October 1989.
- [31] Timothy W. McLain. Personal communication, 1993.
- [32] Marvin Lee Minsky. *Perceptrons*. MIT Press, Cambridge, MA 02142, 1988. Expanded ed.
- [33] Marvin Lee Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA 02142, 1969.
- [34] A.F. Murray and P.J. Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks*, 5(5):792–802, September 1994.
- [35] M.R. Napolitano, C.I. Chen, and R. Nutter. Application of a neural observer as state estimator in active vibration control of a cantilevered beam. *Smart Materials and Structures*, 1(1):69–75, March 1992.
- [36] Kumpati S. Narendra and K. Parthasarathy. Identification and control of dynamic systems using neural networks. *IEEE Transactions on Neural Networks*, pages 4–27, March 1990.
- [37] Derrick H. Nguyen and Bernard Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *International Joint Conference on Neural Networks*, volume 3, pages 21–26. Erlbaum, July 1990.
- [38] Derrick H. Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, April 1990.
- [39] S.J. Nowlan and G.E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.
- [40] D.C. Plaut, S.J. Nowlan, and G.E. Hinton. Experiments on learning by backpropagation. Technical Report Tech. Rep. CMU-CS-86-126, Carnegie-Mellon University, 1986.

- [41] W. Raghupathi and B.S. Raju. A neural network application for bankruptcy prediction. In V. Milutinovic and B.D. Shriver, editors, *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, volume 4, pages 147-55, Kauai, Hawaii, January 8-11 1991. IEEE.
- [42] R. Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740-747, September 1993.
- [43] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, D.C., 1962.
- [44] David E. Rumelhart. Learning and generalization. In *Proceedings of the IEEE Int. Conf. Neural Networks*, San Diego, CA, 1988. (plenary address).
- [45] David E. Rumelhart. Personal communication regarding multi-level sigmoid, June 1994.
- [46] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, page 318. The MIT Press, Cambridge, MA 02142, 1986.
- [47] J. Shandle. Neural networks are ready for prime time. *Elect. Design*, 41(4):51-58, February 18 1993.
- [48] J. Sietsma and R.J.F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67-69, 1991.
- [49] A.E. Smith and C.H. Dagli. Relating binary and continuous problem entropy to back-propagation network architecture. In *Applications of Artificial Neural Networks II*, volume 2, pages 551-62, Orlando, Florida, April 2-5 1991. SPIE. Proceedings of the SPIE, vol. 1469.
- [50] E.D. Sontag. Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, 3(6):981-90, November 1992.
- [51] D.F. Specht. Probablistic neural networks. *Neural Networks*, 3:109-118, 1990.

- [52] D.F. Specht. A general regression neural network. *IEEE Transactions on Neural Networks*, 2(6):568-76, November 1991.
- [53] Alessandro Sperduti and David G. Stork. A rapid graph-based method for arbitrary transformation invariant pattern classification. In *Advances in Neural Information Processing Systems 7*. Morgan Kaufmann Publishers, 1995.
- [54] William E. Staib and Santosh K. Ananthraman. Neural networks in control: A practical perspective gained from Intelligent Arc Furnace (TM) controller operating experience. In *Proceedings of the World Congress on Neural Networks*, volume 2, pages 217-222, San Diego CA, June 1994. International Neural Network Society.
- [55] A.J. Surkan and J.C. Singleton. Neural networks for bond rating improved by multiple hidden layers. In *International Joint Conference on Neural Networks*, volume 2, pages 157-62, San Diego, CA, June 17-21 1990. IEEE, INNS, Erlbaum.
- [56] Marc A. Ullman. *Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-Flying Space Robots*. PhD thesis, Stanford University, Stanford, CA 94305, March 1993.
- [57] Eric Wan. Temporal backpropagation for FIR neural networks. In *International Joint Conference on Neural Networks*, pages 575-580, San Diego CA, June 1990. Erlbaum.
- [58] Zhenni Wang, M.T. Tham, and A.J. Morris. Multilayer feedforward neural networks: a canonical form approximation of nonlinearity. *International Journal of Control*, 56(3):655-72, September 1992.
- [59] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193-209, 1990.
- [60] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA 02142, August 1974.
- [61] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550-1560, October 1990.

- [62] Paul J. Werbos. Neural networks, system identification, and control in the chemical process industries. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, New York, 1992.
- [63] James R. Wertz, editor. *Spacecraft Attitude Determination and Control*. Kluwer Academic Publishers, Boston, MA, 1978.
- [64] David A. White and Donald A. Sofge, editors. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, New York, NY, 1992.
- [65] D. Whitley and C. Bogart. The evolution of connectivity: Pruning neural networks using genetic algorithms. In *International Joint Conference on Neural Networks*, volume 1, page 134, Washington, DC, 1990. IEEE, INNS, Erlbaum.
- [66] Bernard Widrow. A study of rough amplitude quantization by means of nyquist sampling theory. *IRE Transactions of the Professional Group on Circuit Theory*, CT-3(4):266-276, December 1956.
- [67] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, MADALINE, and backpropagation. *Proceedings of the IEEE*, 78(9):1415-42, September 1990.
- [68] Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing*. Signal Processing Series. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1985.
- [69] Edward Wilson. Experiments in neural network control of a free-flying space robot. In *Proceedings of the Fifth Workshop on Neural Networks: Academic / Industrial / NASA / Defense*, pages 204-209, San Francisco, CA, November 1993. SPIE. Proceedings of the SPIE, vol. 2204.
- [70] Edward Wilson. Backpropagation learning for systems with discrete-valued functions. In *Proceedings of the World Congress on Neural Networks*, volume 3, pages 332-339, San Diego CA, June 1994. International Neural Network Society.
- [71] Edward Wilson and Stephen M. Rock. Experiments in control of a free-flying space robot using fully-connected neural networks. In *Proceedings of the World Congress*

on Neural Networks, volume 3, pages 157–162, Portland, OR, July 1993. International Neural Network Society.

- [72] Edward Wilson and Stephen M. Rock. Neural network control of a free-flying space robot. In *Proceedings of the World Congress on Neural Networks*, volume 2, pages 15–22, San Diego, CA, June 1994. International Neural Network Society.
- [73] Edward Wilson and Stephen M. Rock. Neural network control of a free-flying space robot. *Simulation*, June 1995.
- [74] Ron Winter and Bernard Widrow. Madaline Rule II: a training algorithm for neural networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 401–408, San Diego CA, July 1988.