NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ARTIFICIAL NEURAL NETWORK CONTROL SYSTEM
FOR
SPACECRAFT ATTITUDE STABILIZATION

by

Clement M. Segura

June 1990

Thesis Advisor: Jeff. B Burl

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLAS	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey,Ca 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, Ca 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

AN ARTIFICIAL NEURAL NETWORK CONTROL SYSTEM FOR SPACECRAFT ATTITUDE STABILIZATION (UNCLAS)

12. PERSONAL AUTHOR(S) SEGURA, CLEMENT M.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MASTER"S THESIS	FROM _____ TO _____	1990 JUNE	78

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Neural Networks, Attitude Stabilization, Pattern Matching Training, Time-Optimal Control Law, Performance Index Training

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document reports the results of research into the application of artificial neural networks to controlling dynamic systems. The network used is a feed-forward, fully-connected, 3-layer perceptron. Two methods of training neural networks via error back-propagation were used. Pattern matching training is a direct method that teaches the basic response. Performance index training is a new technique that refines the response. Performance index training is based on the concept of enforced performance. A neural network will learn to meet a specific performance goal if the performance standard is the only solution to a problem. Performance index training is devised to teach the neural network the time-optimal control law for the system. Real-time adaptation of a neural network in closed loop control of the Crew/Equipment Retriever was demonstrated in computer simulations.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLAS
22a. NAME OF RESPONSIBLE INDIVIDUAL Jeff B. Burl	22b. TELEPHONE (Include Area Code) 646-2390 22c. OFFICE SYMBOL 62Bi

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

☆ U.S. Government Printing Office: 1986—606-243

Approved for public release; distribution is unlimited.

AN ARTIFICIAL NEURAL NETWORK CONTROL SYSTEM FOR SPACECRAFT ATTITUDE STABILIZATION

by

Clement M. Segura
Lieutenant, United States Navy
B.S.S.E., United States Naval Academy

Submitte ' in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1990

Author: Clement M. Segura

Approved by:

Jeff B. Burl, Thesis Advisor

Roberto Cristi, Second Reader

John P. Powers, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This document reports the results of research into the application of artificial neural networks to controlling dynamic systems. The network used is a feed-forward, fully-connected, 3-layer perceptron. Two methods of training neural networks via error back-propagation were used. Pattern matching training is a direct method that teaches the basic response. Performance index training is a new technique that refines the response. Performance index training is based on the concept of enforced performance. A neural network will learn to meet a specific performance goal if the performance standard is the only solution to a problem. Performance index training is devised to teach the neural network the time-optimal control law for the system. Real-time adaptation of a neural network, employed in the closed loop control of the Crew/Equipment Retriever, was demonstrated by computer simulation.

TABLE OF CONTENTS

I. INTRODUCTION

A. GOAL OF THE RESEARCH

The goal of this research was to develop a neural network that can be used as a real-time, intelligent control system. Adaptive control has been used in situations where the physical system to be controlled is time-varying and uncertain. Adaptive control schemes usually rely on multiple feedback loops that track critical system parameters (e.g., mass, thrust magnitude, or time constants). The range of variation in any system parameter must be small or the controller often cannot adapt. Neural networks can be used as nonlinear, time-varying controllers if some method for measuring the performance of the system and adapting the network can be devised. For this research, system states were used as inputs to a neural net which was trained by two methods to mimic the nonlinear minimum-time control law for the Crew/Equipment Retriever (CER).

Neural net training was conducted by direct and indirect means. Direct training, or pattern matching, required the neural network to reproduce the desired control signal when fed corresponding system states. This method of training requires a priori knowledge of the desired control function. Performance index training is a new concept devised to teach a neural network to meet a specified performance goal while the neural network is actively controlling the CER. Performance index training requires no advance knowledge of the desired control function.

B. CHARACTERISTICS DESIRED FOR REAL-TIME CONTROL

Real-time control of systems by a neural network depends strongly on the implementation of the network. High throughput is desired in order that small sample intervals may be used. The high order of parallelism in a neural network that gives processing speed advantages over sequential systems is lost when the network is implemented in software on a single processor computer. The long-term solution to throughput will be custom VLSI implementations of neural networks; however, software

implementations must be used at present. Small neural networks can be implemented on modestly sized computers and give the throughput desired.

A second requirement for real-time control by a neural network is on-line adaptability. The significant costs of implementing a neural network in software and providing a computer to run the program is justified only if superior performance can be gained by such an effort. Real-time adaptation depends on a measure of error in the system state which is being controlled, a means by which the error can be related to the network weights, and a fast implementation of the training algorithm.

C. THESIS ORGANIZATION

This report is organized into five chapters and one appendix. Chapter I is this introduction and motivates the research and describes the contents of this thesis. Chapter II develops the control problem and the state equations of the CER. Artificial neural networks, error back-propagation, and closed loop control by neural networks are introduced in Chapter III. The results of this research are presented in Chapter IV including three-dimensional graphic output from the neural networks and time simulation examples of neural network learning. Conclusions and recommendations for further study are covered in Chapter V. The FORTRAN computer programs used to implement, train, and test the neural networks investigated in this thesis are listed in Appendix A.

II. CONTROL PROBLEM

A. CREW/EQUIPMENT RETRIEVER

The Crew/Equipment Retriever (CER) was designed by McDonnell Douglas Astronautics Company in response to a NASA request for proposal in Reference 1. The CER was designed to autonomously intercept, capture and retrieve objects or astronauts that have become detached from Space Station FREEDOM. Figure 2.1 shows the layout of the CER and Table I summarizes its physical characteristics.

Hansen [Ref. 2] investigated time optimal and fuel-time optimal control laws for the CER. Time optimal control trades fuel usage for high accuracy. Fuel-time optimal control strikes a balance

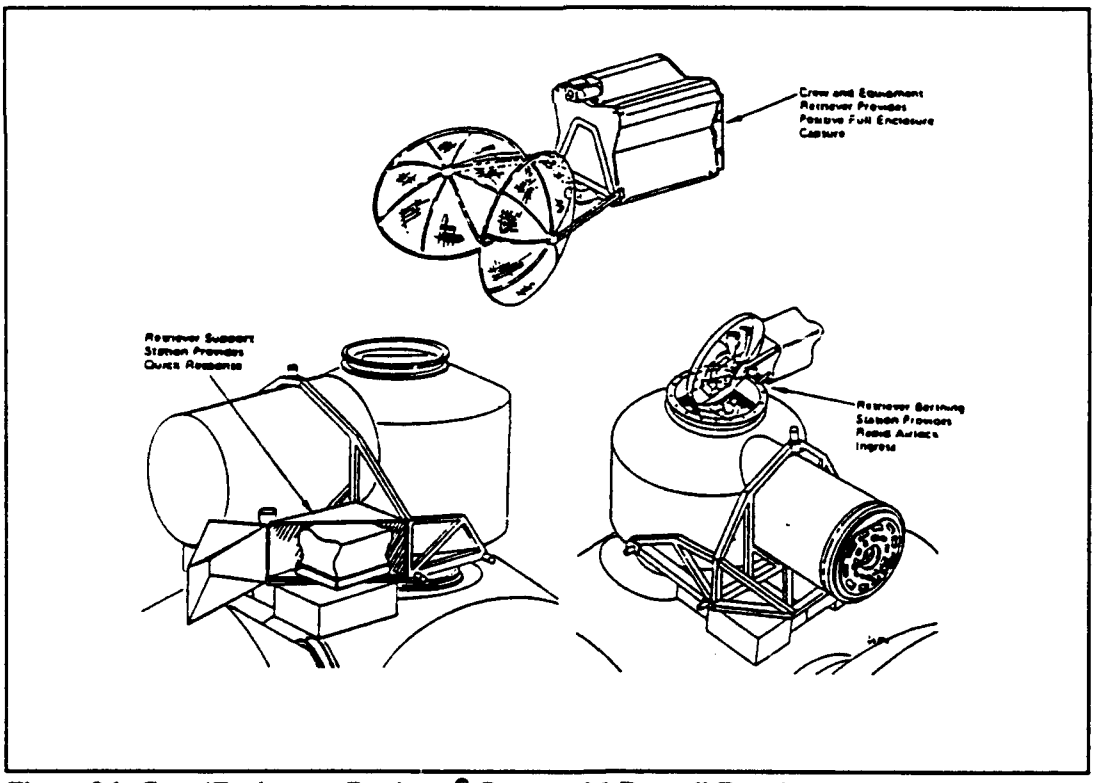

Figure 2.1. Crew/Equipment Retriever.© Courtesy McDonnell Douglas Astronautics Co.

Table I. CER CHARACTERISTICS

MASS	402.3kg
DIMENSIONS	
LENGTH	1.27m
WIDTH	1.02m
HEIGHT	1.02m
THRUSTERS	4.448N
PAYLOAD	302.9kg

between conserving fuel and accurate pointing. Both control schemes can be used for different CER mission phases.

Synthesis of both of the above control laws is highly dependent on good knowledge of the size and location of the object being retrieved. The CER is modeled as a rigid body, with no viscous or spring damping, acted upon by thruster torques. The mathematical model used in control law synthesis depends on the size and location of the object retrieved. Hansen demonstrated the sensitivity of the optimal control law with respect to uncertainty in the location and size of the recovered object. Inaccuracies in estimates of the size and location of the recovered object results in a control law that is not optimal and may cause instability.

B. CER SYSTEM STATE EQUATIONS

The equation of motion for a rigid body acted upon by a torque is given in Equation (2-1).

$$\Sigma T = \frac{dI\omega}{dt} \tag{2-1}$$

where T is a torque vector, I is the moment of inertia tensor and $\vec{\omega}$ is the vector of rotation rates, about three orthogonal axes, in radians per second. For simplicity, this project will investigate single axis control of the CER. The torque equation for rotation about the x-axis is simplified to:

The moment of inertia of the CER about the x axis (I_{xx}) is given by:

$$\Sigma T = I_{xx} \frac{d\omega}{dt} \tag{2-2}$$

$$I_{xx} = \int (z^2 + y^2)\, dm \tag{2-3}$$

where y and z are spatial coordinates and dm is the differential of the mass. The CER is assumed to be a uniformly distributed mass inside a parallelpiped structure (Figure 2.2). The differential of the mass can then be replaced with the density multiplied by the differentials of each of the spatial dimensions:

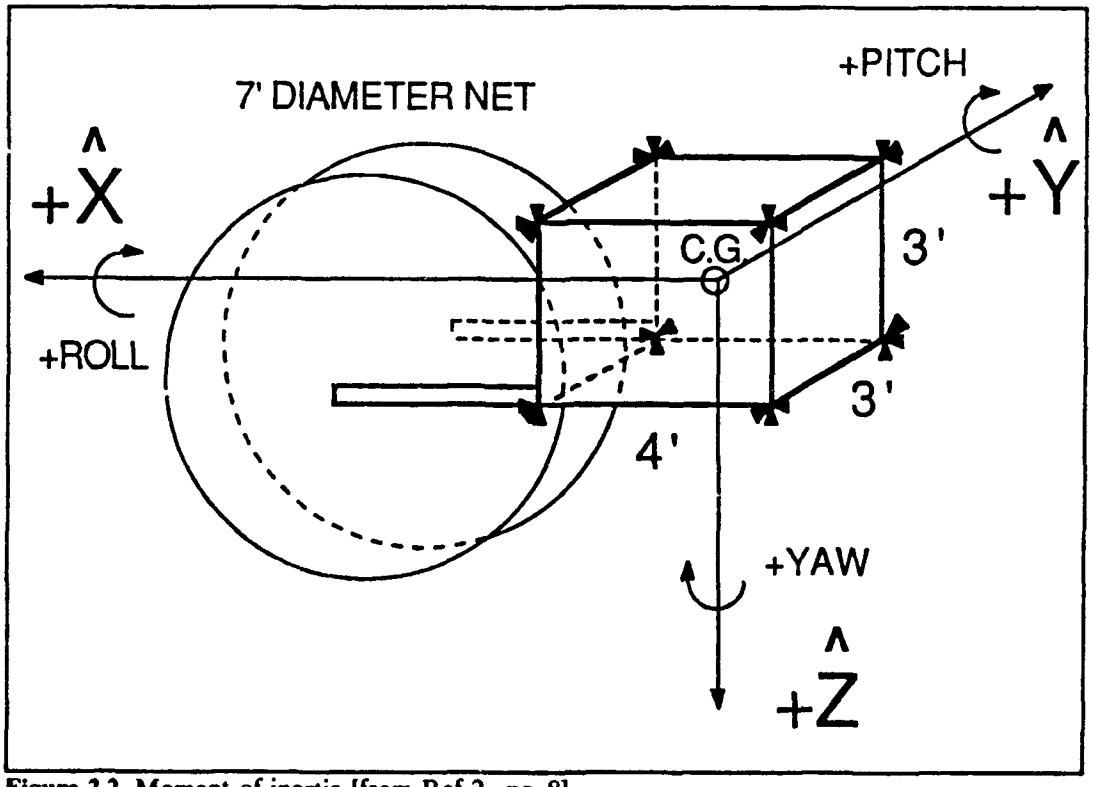

Figure 2.2. Moment of inertia [from Ref 2., pg. 8].

$$I_{xx} = \frac{M}{V}\iiint (z^2 + y^2)\, dx\, dy\, dz \tag{2-4}$$

where M is the mass of the CER and V is the total volume. The volume occupied is 1.311 m^3 and the mass is 402.3 kg. Integrating equation (2-5) with respect to each linear dimension gives the moment of inertia about the x-axis.

$$I_{xx} = \frac{402.3\ kg}{1.311\ m} \int_{-0.51}^{0.51}\int_{-0.51}^{0.51}\int_{-0.635}^{0.635} (z^2 + y^2)\, dx\, dy\, dz \tag{2-5}$$

$$I_{xx} = 54.82\ kg\ m^2 \tag{2-6}$$

Roll maneuvers are accomplished by firing two pair of thrusters simultaneously. Torque from the control thrusters around the x-axis (T_x) is the product of the force supplied (F_x) and the distance between the thrusters (d_x):

$$T_x = d_x F_x \tag{2-7}$$

$$T_x = (0.914\ m)(4.448\ N)(2) \tag{2-8}$$

$$T_x = 9.0388\ N\ m \tag{2-9}$$

Assigning states (2-10) and solving the torque equation gives the state equations of the CER (2-11):

$$\frac{d\theta}{dt} = \omega \tag{2-10}$$

$$\begin{vmatrix} \dot{\theta} \\ \dot{\omega} \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} \theta \\ \omega \end{vmatrix} + \begin{vmatrix} 0 \\ 0.16488 \end{vmatrix} u \tag{2-11}$$

where u is defined as the control signal.

The control signal has been normalized to:

$$u = \begin{Bmatrix} +1.0 & \textit{Positive thruster torque} \\ 0 & \textit{Zero thruster torque} \\ -1.0 & \textit{Negative thruster torque} \end{Bmatrix}$$

The state equations can be converted to a system of linear discrete state equations using a time step size of 0.01 seconds. The resulting discrete state equations are:

$$\begin{vmatrix} \theta(n+1) \\ \omega(n+1) \end{vmatrix} = \begin{vmatrix} 1 & 0.01 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} \theta(n) \\ \omega(n) \end{vmatrix} + \begin{vmatrix} 8.244*10^{-6} \\ 1.6488*10^{-3} \end{vmatrix} u(n) \qquad (2\text{-}12)$$

C. MINIMUM TIME CONTROL LAW

The cost function that leads to minimum time control laws is:

$$J = \int_{t_0}^{t_f} d\tau \qquad (2\text{-}13)$$

By applying Pontryagin's Minimum Principle, the optimal control law may be found:

$$u = \begin{Bmatrix} -1 & ,S>0 \\ 0 & ,S=0 \\ +1 & ,S<0 \end{Bmatrix} \qquad (2\text{-}14)$$

where $S = x_1 + \dfrac{x_2|x_2|}{2\left(\dfrac{T}{I}\right)}$.

The optimal switching curve represents the zero-trace of the switching function S and is displayed in Figure 2.3. The control signal u is equal to -1.0 for state space coordinates above and to the right of the curved line. For locations below and to the left of the line the control signal is +1.0. The optimal control law can also be interpreted as a control surface whose height corresponds to the optimal control signal for each theta-omega location in the state space.

Figure 2.4 illustrates a typical minimum time trajectory for initial conditions in the first quadrant of the state space. The CER accelerates in the negative omega direction until it intersects the switching

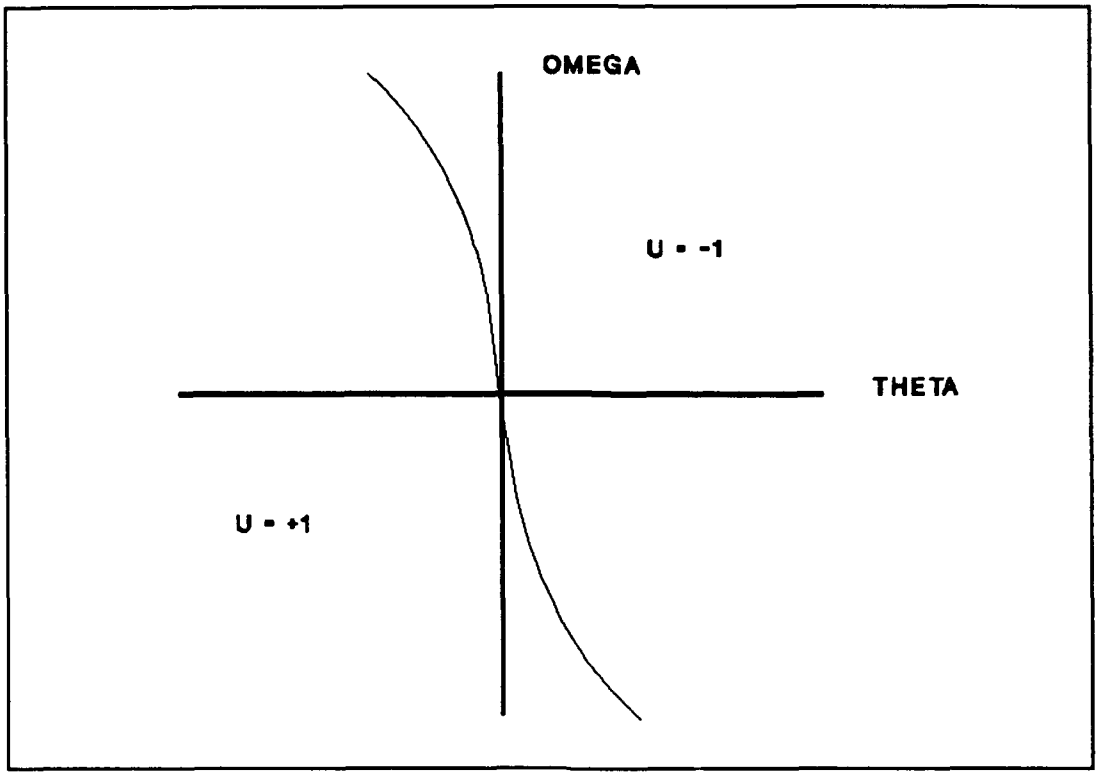

Figure 2.3. Minimum time switching curve.

curve. The direction of acceleration then reverses until the CER reaches the center. Minimum time control provides accurate pointing but uses large amounts of fuel. This control scheme should be used only when highly precise maneuvers are required.

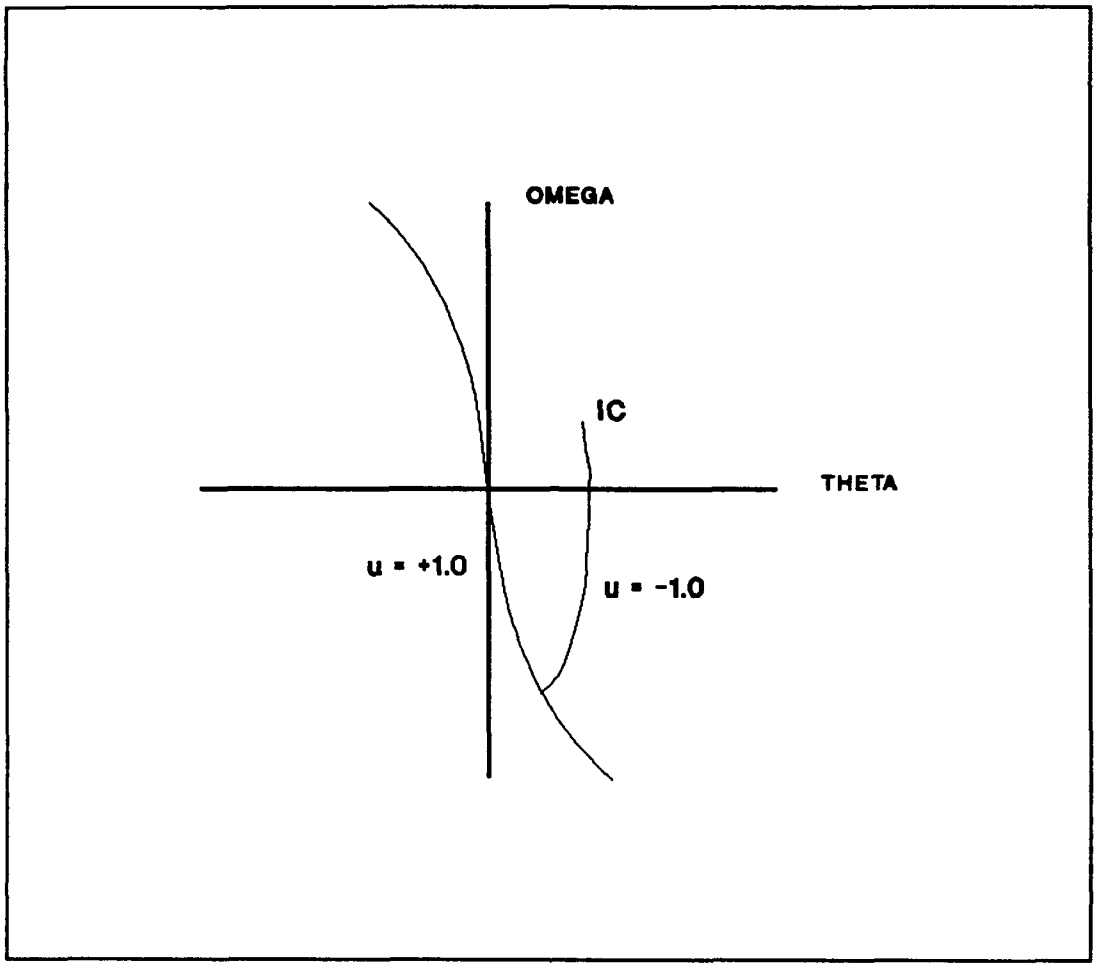

Figure 2.4. Minimum time trajectory.

III. ARTIFICIAL NEURAL NETWORK

A. NEURAL NETWORK ARCHITECTURE

A neural network is a collection of simple processing elements (neurons) joined by weighted connections. The weights may be positive, negative or zero and are assigned values during the network training process. The neural network performs a mapping from the inputs via the neuron transfer characteristic and the weights to the output(s).

Each neuron possesses a transfer characteristic that describes its input-output relationship. A key requirement for this function is that it should be nonlinear. The neurons used in this project are identical and possess a sigmoid transfer characteristic given by Equation (3-1) and diagrammed in Figure 3.1:

$$f(net) = \frac{2.0}{(1 + e^{-net})} - 1.0 \qquad (3\text{-}1)$$

The sigmoid function in Equation (3-1) was selected for this application because it saturates at 1.0 and passes through the origin of the input-output space. Symmetry is not necessarily required, but is suggested by the range of inputs to the neural network. This function is also differentiable, which is a requirement for the training method employed.

The neural network used in this project can be described as a feed-forward, fully-connected, 3-layer perceptron. Figure 3.2 is a diagram of a neural network used for single axis control showing the inputs, weight matrices, neurons, and outputs. Each neuron input, net_j, is the dot product of the previous layer neurons' outputs, inp_i and the next layer's corresponding weights, w_{ij}:

$$net_j = \sum_i w_{ij} inp_i \qquad (3\text{-}2)$$

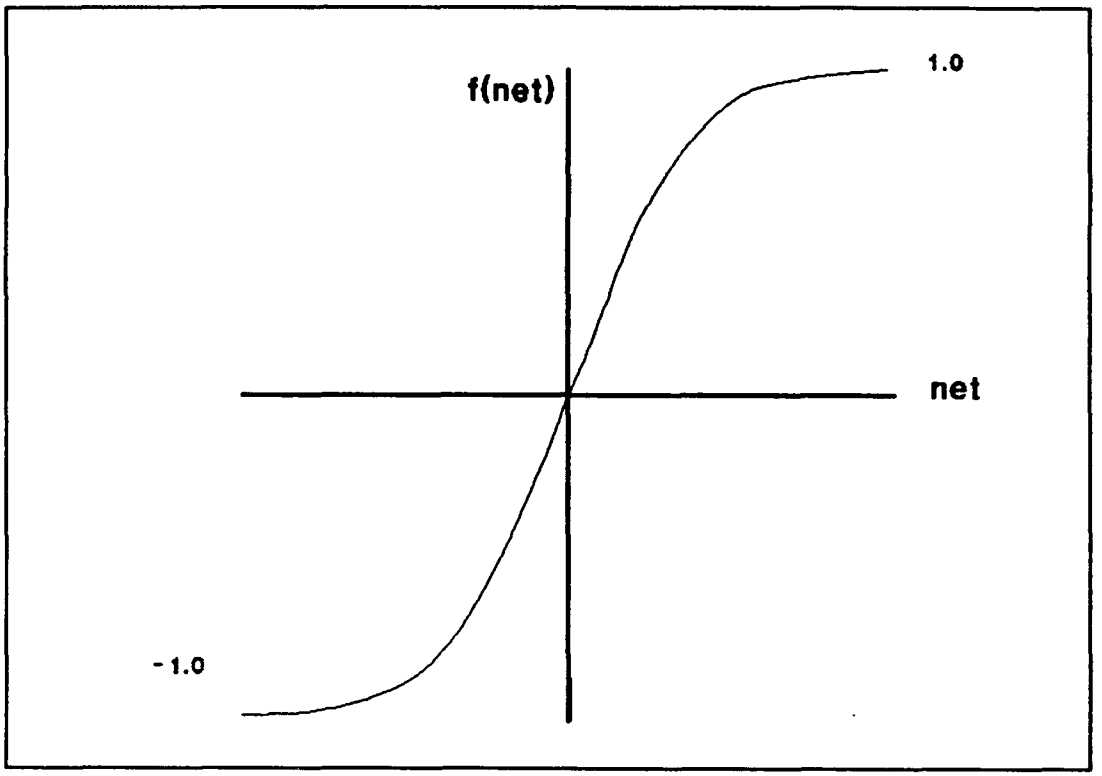

Figure 3.1. Neuron Transfer Characteristic

The constant neurons at +1.0 act as a bias in the input of the neuron and aid in learning. Lapedes and Farber [Ref. 3] reported that 3 layer neural networks of this type are capable of learning any arbitrary input-output mapping. Unfortunately, there is no rule for selecting the transfer characteristic of the neurons or number and arrangement of the neurons.

Variable names used throughout this thesis are the same as those in the FORTRAN computer programs listed in Appendix A. Inputs to the neural net enter from the left, as illustrated in Figure 3.2 and are distributed by input nodes (linear neurons) to the first hidden layer neurons via the W matrix by Equation (3-3).

$$hnet_j = \sum_i w_{ij} inp_i \qquad (3\text{-}3)$$

The output of the first hidden layer Equation (3-4), propagates to the second hidden layer via the V weight matrix as shown by Equation (3-5).

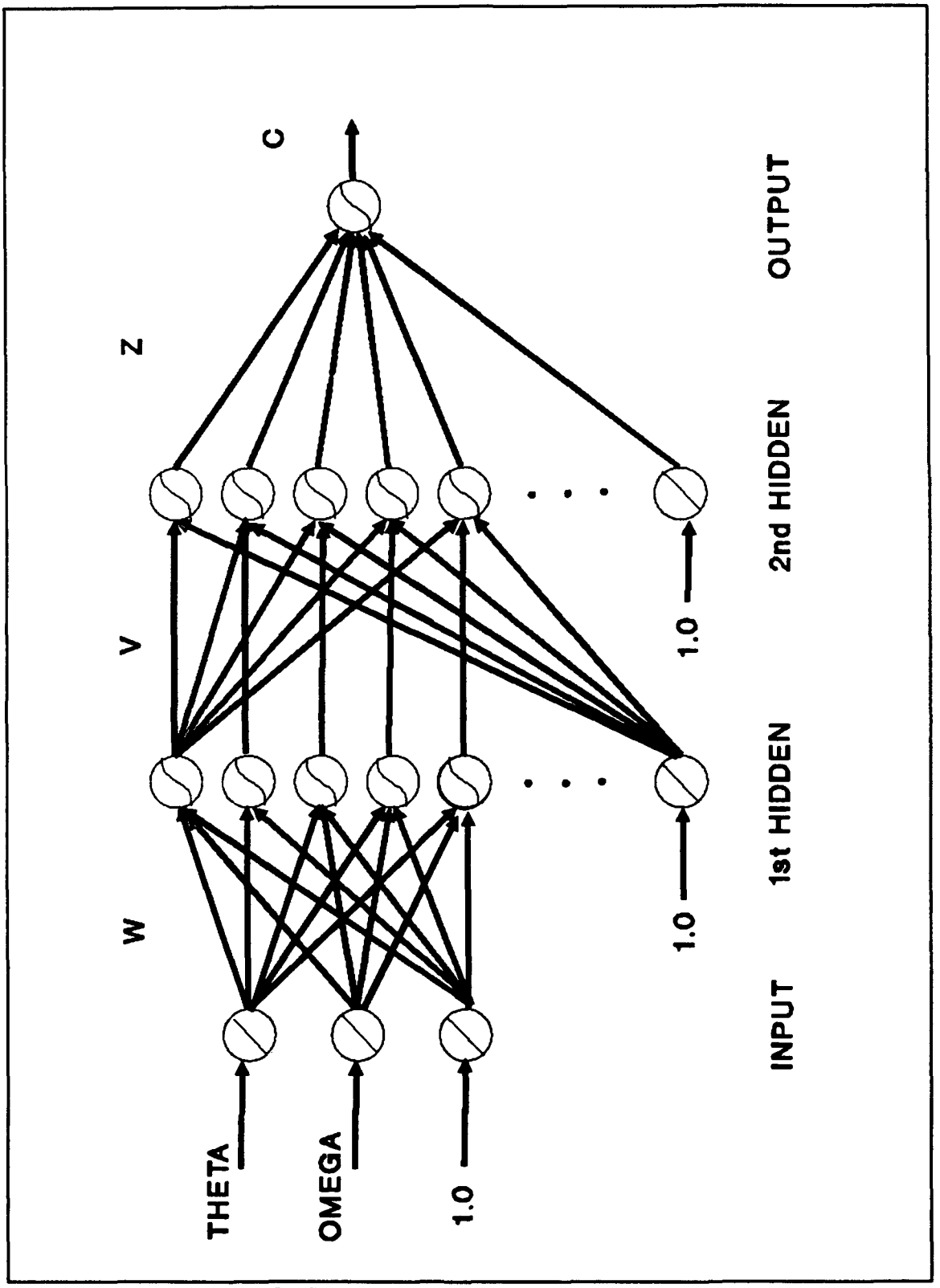

Figure 3.2. Neural Network Configuration.

$$hf_j = f(hnet_j) \tag{3-4}$$

$$onet_k = \sum_j v_{jk} hf_j \tag{3-5}$$

The outputs of the second hidden layer, given by Equation (3-6) are sent, in turn, to the output neuron through the Z weight vector, shown by Equation (3-7).

$$of_k = f(onet_k) \tag{3-6}$$

$$fnet = \sum_k z_k of_k \tag{3-7}$$

The output of the neural network is the thruster control signal "C" as shown in Equation (3-8).

$$C = f(fnet) \tag{3-8}$$

Only the input values and network output are accessible to the outside world. The center two layers are thus "hidden" from view. The error back-propagation algorithm was designed to adjust the weights of the hidden neurons based on observable inputs and outputs, and knowledge of the network configuration.

B. ERROR BACK-PROPAGATION

Error back-propagation is a technique by which neural network weights are adjusted (trained) in a recursive manner to minimize the sum of squared error of the neural network output [Ref. 4]. This technique is an application of the generalized delta rule which is a gradient optimization procedure. This procedure allows a neural network to learn an input-output mapping by successive applications of the training algorithm over a wide range of inputs. This procedure also solves the problem of adjusting the weights of the hidden layer neurons by calculating an error component for each hidden neuron. The

error back-propagation algorithm will be derived below. This derivation will closely follow the development in [Ref. 4:pp 324-327] except it will be applied to the network investigated in this thesis.

Equations (3-1) through (3-8) may be combined into a single relation between the input signals and the neural network output

$$C(inp) = f\left(\sum_k f_k\left(\sum_j f_j\left(\sum_i inp_i w_{ij}\right)v_{jk}\right)z_k\right) \tag{3-9}$$

The error value to be minimized via error back-propagation is:

$$E = \frac{1}{2}\sum (DESIRED - C)^2 \tag{3-10}$$

where DESIRED is the function that the network must learn. The error is measured at the output node of the neural network.

Error back-propagation adjusts each weight in the network by an amount proportional to the gradient of the error taken with respect to the variable weights. The learning rate (LR) is a fixed constant of proportionality used to adjust the speed of learning and to avoid instability of the network during training. The rule for adjusting the weights is:

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial w_{ij}}(LR) \tag{3-11}$$

A few general terms will be defined now and used in the following derivation. The derivative of the error with respect to the output of the neural net is:

$$DELEC = \frac{\partial E}{\partial C} = -1.0\,(DESIRED - C) \tag{3-12}$$

The derivative across a neuron is given by:

$$SIGDER = \frac{df(net)}{dnet} = \frac{2.0}{(1 + e^{-net})^2} \tag{3-13}$$

Successive application of the chain rule gives formulas for adjusting each of the weights.

1. Z-WEIGHTS

The adjustment to each of the weights in the Z vector is defined as the gradient of the output error taken with respect to the weight being examined. Equation (3-14) represents the application of the chain rule to the output error.

$$\Delta z_k = LR\left(\frac{\partial E}{\partial z_k}\right) = LR\left(\frac{\partial E}{\partial C}\right)\left(\frac{\partial C}{\partial fnet}\right)\left(\frac{\partial fnet}{\partial z_k}\right) \tag{3-14}$$

Substituting the previously defined values of SIGDER and DELEC into Equation (3-14) gives:

$$\Delta z_k = LR\left(\frac{\partial E}{\partial C}\right)(SIGDER(fnet))\frac{\partial fnet}{\partial z_k}\left(\sum_k of_k z_k\right) \tag{3-15}$$

and

$$\Delta z_k = LR\,(DELEC)\,(SIGDER(fnet))\,of_k \tag{3-16}$$

Equation (3-16) gives the change of weight z_k and is applied for each training presentation.

2. V-WEIGHTS

The gradient of the error with respect to the V weights is given by:

$$\Delta v_{jk} = LR\left(\frac{\partial E}{\partial v_{jk}}\right) = LR\left(\frac{\partial E}{\partial fnet}\right)\left(\frac{\partial fnet}{\partial of_k}\right)\left(\frac{\partial of_k}{\partial onet_k}\right)\left(\frac{\partial onet_k}{\partial v_{jk}}\right) \tag{3-17}$$

Substituting as before gives:

$$\Delta v_{jk} = LR\left(\frac{\partial E}{\partial fnet}\right)SIGDER(fnet)\left(\frac{\partial}{\partial of_k}\sum of_k z_k\right)SIGDER(onet_k)\left(\frac{\partial}{\partial V_{jk}}\sum v_{jk} hf_j\right) \tag{3-18}$$

Defining:

$$DELF = \left(\frac{\partial E}{\partial fnet}\right) = SIGDER(fnet)\,(DELEC) \tag{3-19}$$

and

$$DELTAO_k = SIGDER(onet_k) \tag{3-20}$$

and substituting Equations (3-19) and (3-20) into Equation (3-18) completes the derivation of the change to weight V_{jk}:

$$\Delta v_{jk} = LR(DELF)\,DELTAO_k(z_k)\,hf_j \tag{3-21}$$

3. W-WEIGHTS

The changes in the W weights are calculated last in the same fashion as before. The gradient of the error taken with respect to the W weights is:

$$\Delta w_{ij} = LR\left[\sum_k \left(\frac{\partial E}{\partial fnet}\right)\left(\frac{\partial fnet}{\partial of_k}\right)\left(\frac{\partial of_k}{\partial onet_k}\right)\left(\frac{\partial onet_k}{\partial hf_j}\right)\right]\left(\frac{\partial hf_j}{\partial hnet_j}\right)\left(\frac{\partial hnet_j}{\partial w_{ij}}\right) \tag{3-22}$$

Substituting the values defined in Equations (3-12), (3-13), (3-19), and (3-20) simplifies the relation to:

$$\Delta w_{ij} = LR\left[\sum_k (DELF)(z_k)(DELTAO_k)v_{jk}\right] SIGDER(hnet_j)\left(\frac{\partial}{\partial w_{ij}}\sum_i w_{ij}\,inp_i\right) \tag{3-23}$$

Accumulating the first term (square brackets) in Equation (3-23) in a new variable:

$$DELCH_j = \sum_k v_{jk}\,z_k\,DELTAO_k \tag{3-24}$$

and defining:

$$DELTAH_j = SIGDER(hnet_j) \tag{3-25}$$

gives the final result for the change to the w_{ij} weights:

$$\Delta w_{ij} = LR(DELF)(DELCH_j)(DELTAH_j)(inp_i) \tag{3-26}$$

The above relations are carried out in the order presented during a training cycle. After all adjustments are calculated, they are subtracted from the respective weights and a new training cycle begins. Cumulative back-propagation, in which several training cycles are computed before the weights are adjusted, may also be implemented with the above scheme. The above derivation can be used when the desired output of the neural net is known. The procedure then implements pattern matching training by causing the neural network to learn the desired input-output mapping.

When the neural network is part of a control system and the error being measured is a function of the end state of the system, an extension to the original error back-propagation rule must be used [Ref. 3]. Extending the error back-propagation algorithm to include a cascaded system requires that a derivative of the output of the system with respect to the input be computed. Psaltis, et al. [Ref. 5] proposed this extension as a way to link the neural network output, used as a control input to a system, to the system states. Nguyen and Widrow [Ref. 6] used a neural network trained to emulate the system being controlled during the training process instead of state equations. While the application made by Nguyen and Widrow simplifies the computation of the error signals (the error is back-propagated through a static neural network), it requires the neural net emulator to be trained to simulate the system dynamics before training of the controller can begin. The increased development time created by training the emulator and the increased computation time to back propagate the system state error (over the system state equations) can be avoided as shown below.

C. PARTIAL PLANT DERIVATIVES

The development outlined below was proposed by Burl [Ref. 7]. A linear, time-invariant dynamic system may be described as a set of first-order linear differential equations of the form:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3-27}$$

where:

$\dot{x}(t)$ = *time derivative of state vector,*
$x(t)$ = *state vector,*
A = *system matrix,*
B = *input matrix, and*
$u(t)$ = *control input.*

The future state of the system can be calculated by integrating Equation (3-17) with respect to time:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\lambda)}Bu(\lambda)\,d\lambda \tag{3-28}$$

Equations (3-27) and (3-28) may be discretized over a small interval to form the discrete state equations:

$$\dot{x}(n+1) = \Phi x(n) + \Gamma u(n) \tag{3-29}$$

Matrices A and B have been replaced by their discrete counterparts Φ and Γ. The discrete state equations can be solved for an arbitrary number of time steps into the future:

$$x(n) = \Phi^n x(0) + \sum_{k=0}^{n-1} \Phi^{n-k-1}\Gamma u(k) \tag{3-30}$$

where n = time index.

The partial derivative of the state vector x(n) with respect to the network weights is:

$$\frac{\partial x(n)}{\partial w} = \sum_{k=0}^{n-1}\left[\frac{\partial x(n)}{\partial u(k)}\frac{\partial u(k)}{\partial w}\right] \tag{3-31}$$

Since the partial derivative of the control input with respect to the neural network weights is constant for all time indices, k, the derivative can be written as:

$$\frac{\partial x(n)}{\partial w} = \left[\sum_{k=0}^{n}\frac{\partial x(n)}{\partial u(k)}\right]\frac{\partial u}{\partial w} \tag{3-32}$$

The partial derivative of the control input, u, with respect to the weights was given in section B above.

From Equation (3-30):

$$\sum_{k=0}^{n-1} \frac{\partial x(n)}{\partial u(k)} = \sum_{k=0}^{n-1} \left[\Phi^{n-k-1}\Gamma\right] \tag{3-33}$$

Equation (3-33) represents a set of vectors, indexed by the time index n, that relate the output state at time step, n, to the input signal. These values were calculated in MATLAB and placed in a lookup table for use by the performance index training program. For a given time step, n, the value of propagated error is:

$$PLANTDER = \frac{\partial x_1(n)}{\partial u} x_1 + \frac{\partial x_2(n)}{\partial u} x_2 \tag{3-34}$$

The procedure described above incorporates the history of the system over the time step interval (0,n) for a given n. The calculations ignore the effect of closing the control loop around the plant. Previous work by Nguyen and Widrow [Ref. 6] utilized only a single step derivative emulated by a neural network. The results of Nguyen and Widrow can be achieved by setting n = 1 in Equation (3-34).

D. NEURAL NETWORK CONTROL OF THE CER

The neural network described above performs state variable feedback control by implementing a mapping between the system states and the control law it has been trained with. A diagram of the control scheme is given in Figure 3.3. The neural network output is passed through a level quantizer to ensure that thruster control signals are only 1.0. The CER states become the inputs to the neural network which produces thruster control signals. After training, the neural network produces the same effect as a nonlinear state feedback controller. The introduction of the quantizer presents a problem in implementing error back-propagation. The derivative of the output of the quantizer with respect to its input is zero everywhere except at the origin where it is infinite. This limitation can be avoided by letting

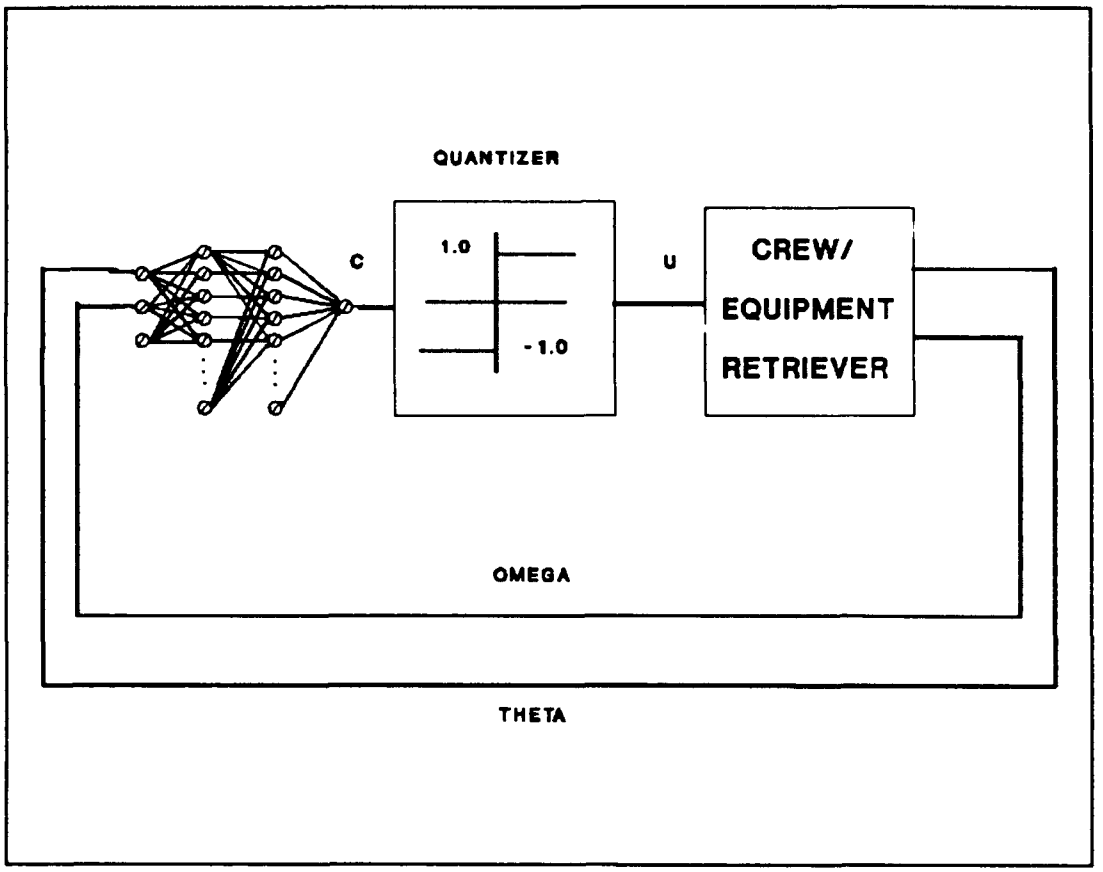

Figure 3.3 Neural Network Control.

the network error be defined by simply subtracting a quantized state error from the network output at the final time of the simulation. This additional extension is not needed for pattern matching training where the network output error is directly measured and back-propagated.

IV. NEURAL NETWORK TRAINING

A. TRAINING ENVIRONMENT

The set of programs, data files, and utility routines that support initialization, training, and evaluation of neural networks make up the training environment. The training environment for the neural networks investigated in this thesis was composed from several FORTRAN programs written by the author. Neural net weights and configuration could be stored on disk and recalled for additional training or analysis. The source code of the programs and supporting files can be found in Appendix A.

Two measures of the performance of the networks were used, a linear sum of the squared error over a fixed domain of inputs, Equation (4-1), and a sum of the quantized error, Equation (4-2):

$$LSSE = \frac{1}{2} \sum_{\theta=-0.1}^{0.1} \sum_{\omega=-0.1}^{0.1} [\, DESIRED(\theta,\omega) - C(\theta,\omega) \,]^2 \tag{4-1}$$

$$QSSE = \frac{1}{2} \sum_{\theta=-0.1}^{0.1} \sum_{\omega=-0.1}^{0.1} [\, DESIRED(\theta,\omega) - SIGN\{C(\theta,\omega)\} \,]^2 \tag{4-2}$$

The linear sum provides a measure of the actual fit of the neural network output to the desired control signal surface. The quantized error represents a measure of the correctness of the decisions made by the neural network. The integral of the squared error over the input space is a standard measure of the performance of a network. The data collected was scaled by a factor of 1000. Both sums were calculated at periodic intervals during training as a figure of merit for the current set of neural network weights. The weights of the network with the lowest linear error value were saved in holding cells during training until they were replaced by the weights of subsequent network with a better error measure.

The state space domain is a square region between -0.1 and +0.1 radians (θ) and -0.1 to +0.1 radians per second (ω). This region was chosen to approximate the domain of operation used by Hansen [Ref. 2].

The Frobenius norms of the weight matrices were calculated every 10 training presentations. These values give a measure of the size of each matrix and were used to monitor the amount and direction of change over time. Visual evaluation of the performance of the neural network was illustrated by plotting the output of the neural net over the state space domain. The decision effectiveness can be shown by plotting the quantized network error over the domain. Three-dimensional graphics allowed the user to view the data from any angle.

Finally, time simulations of the neural network controlling the CER compared with an optimally controlled trajectory were conducted to demonstrate on-line performance. The time simulations started at user-supplied initial conditions and proceeded until the optimal control law reached the origin.

B. INITIAL CONDITIONS

Most of the neural networks used in this research consisted of three input nodes, ten first hidden layer neurons, five second hidden layer neurons, and a single output neuron. One neuron in each of the first three layers was designated as a constant neuron at a value of +1.0. The networks were initialized using weights randomly picked between 1.0 using a uniform random number generator. It was noted that the initial weight values assigned cannot all be identical. If the values are the same, the partial derivatives of the output error is the same for all weights in a layer and no learning occurs [Ref 4]. Neural nets initialized with random weights on smaller intervals were found to be less successful and slower in learning the desired mapping than those initialized using the 1.0 interval. Learning rates were varied from 0.5 to 0.001 depending on the performance of the network. Initial learning rates near 0.5 caused rapid learning. Near the end of the useful learning phase small learning rates were used to avoid instability.

Figure 4.1 is the output of a randomly initialized neural network. The output is near zero for the entire domain of interest due to the random weights and constant neuron bias. When fully trained, the network output should be (ideally) an exact replica of the control signal surface. The z-direction of the picture is network output (linear) over the state space (theta-omega) coordinate plane.

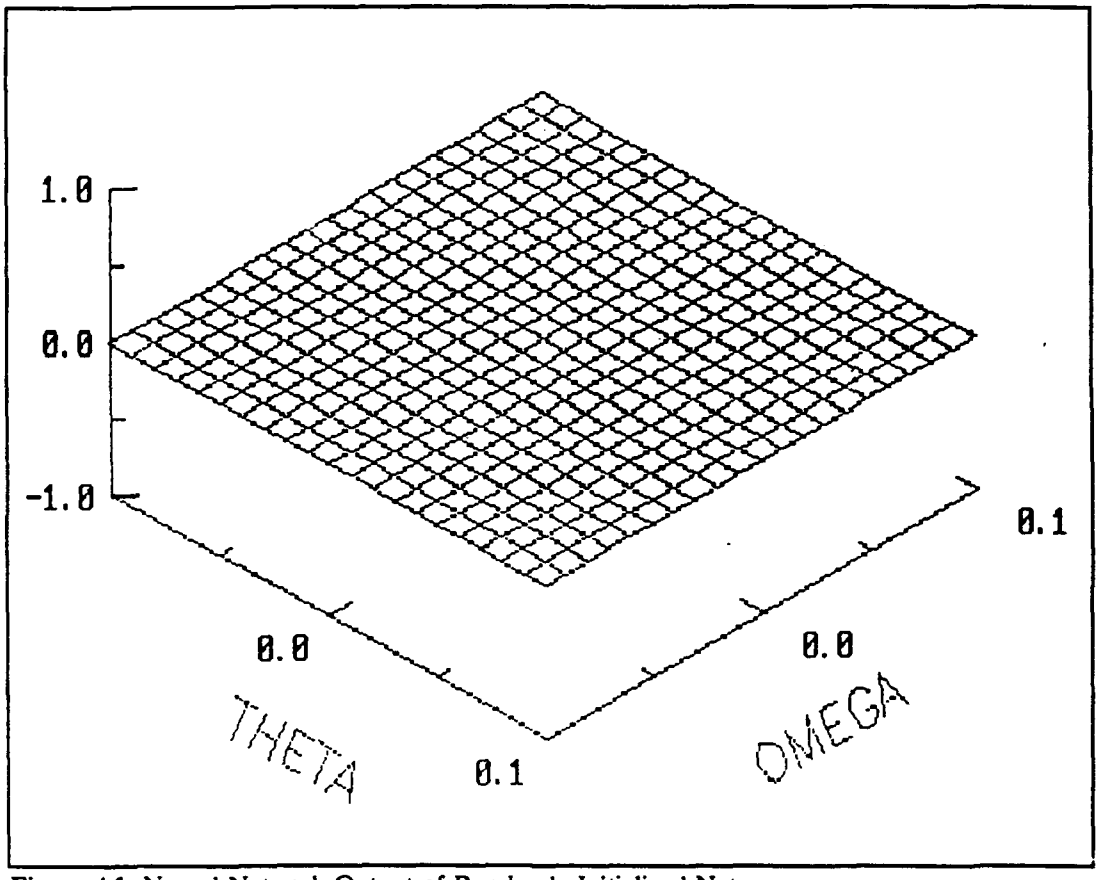

Figure 4.1. Neural Network Output of Randomly Initialized Net.

Figure 4.2 is the corresponding plot of the quantized error of the network. Ideally the error plot should be zero over the entire field after training. This neural net has not been trained and has significant error. The curved cliff-shaped feature corresponds to the trace of the optimal switching curve from Chapter II. The floor has a value of -2.0 which is the difference between the quantized neural net output and the optimal control law output at each position in the state space.

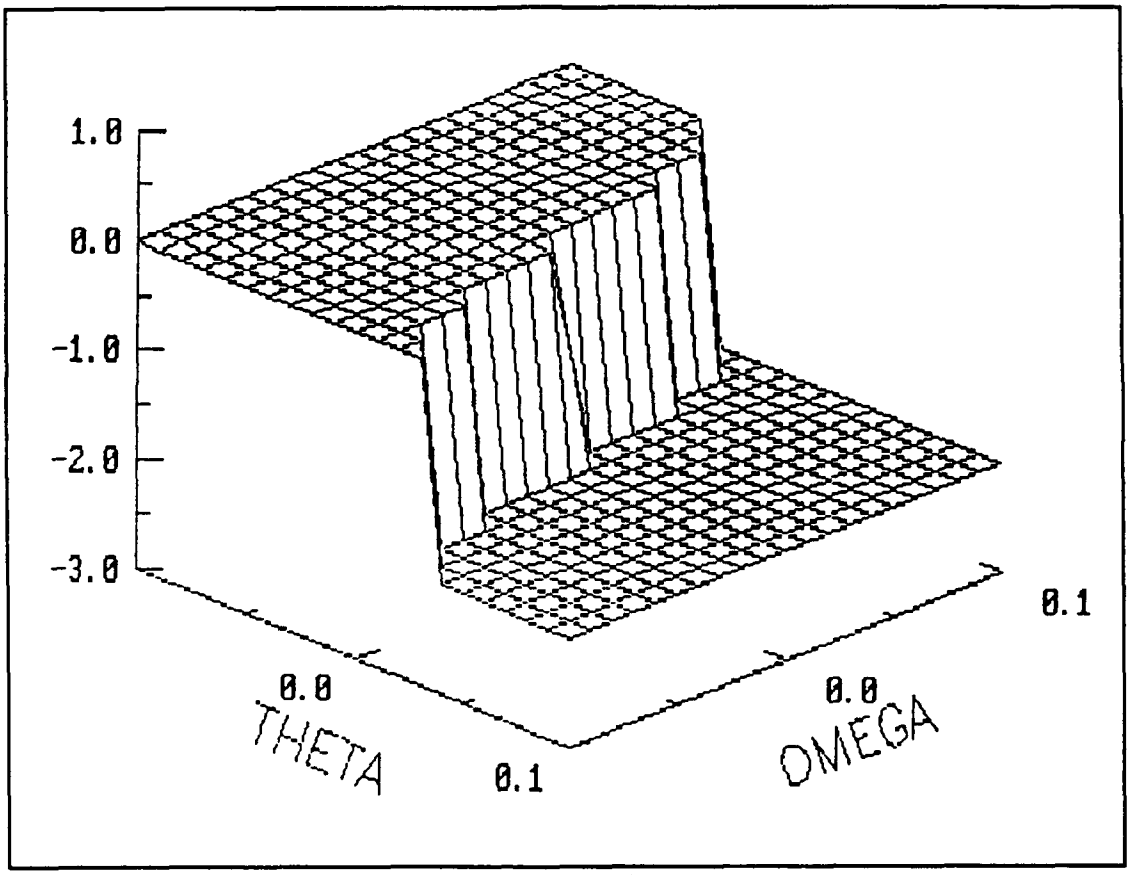

Figure 4.2. Neural Network Error for Randomly Initialized Net.

Figure 4.3 displays the paths of both the neural network controlled CER (solid) and a CER controlled by the minimum time control law (dashed). The control law drives the CER from its initial condition (0.05,0.01) to the center while the neural net drives the CER away from the origin. This behavior will be modified, by training, to yield substantially better results.

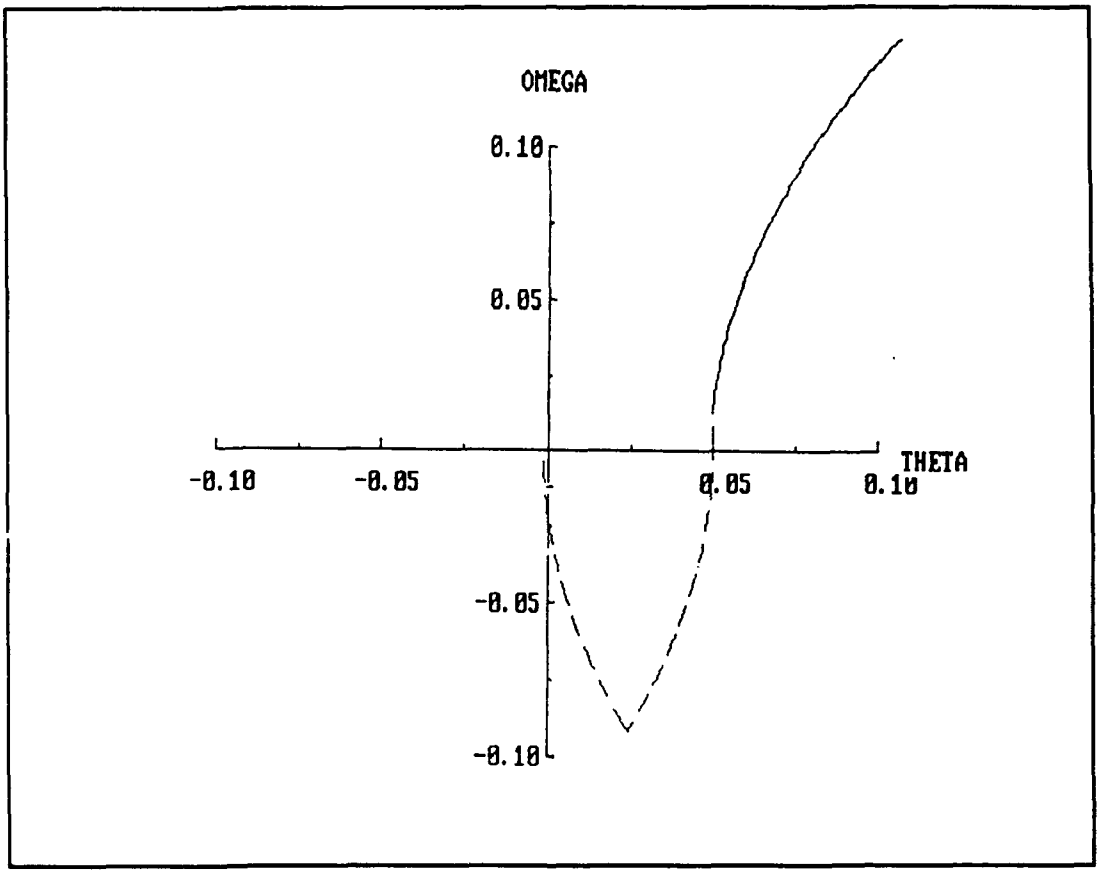

Figure 4.3. State Space Trajectories of Optimally Controlled (Dashed) and Neural Net Controlled (Solid) CERs

C. PATTERN MATCHING

Pattern matching training was conducted on a newly initialized neural network because it provides rapid learning. When the desired neural net response in known, pattern matching is a directly applicable technique. Random inputs in the range (-0.1 to +0.1) for both system states were applied to the net and the network output was compared to the desired optimal control law, equation (4-3).

$$\frac{\partial E}{\partial C} = -1.0 \left(DESIRED(\theta,\omega) - C(\theta,\omega) \right) \tag{4-3}$$

This error value was back-propagated into the network as described in Chapter III at each training presentation. The time advantage of pattern matching over other methods is chiefly due to the elimination of trial and error by the network. Training a neural net by experience is slower because the entire system must be simulated. Random inputs are necessary to avoid biasing the network weights to a specific input region. The error back-propagation rule would minimize the local error instead of finding a global minimization of the error.

As training progressed, both the linear and quantized error values began to decrease. The learning rate was initially 0.5 and was gradually reduced to 0.25. Figure 4.4 is the record of error values for this network. The linear error starts near 500 and decreases as training continues. The large fluctuations are caused by high learning rates. The error back-propagation algorithm used to train the network does not attempt to scale the weight changes to get optimum error decrease at each step. Random inputs and many training cycles combine to drive the system to a global minimum error. Occasional increases are largely unavoidable. The learning rule attempts to minimize the error at each presentation but may cause an increase in global error. The quantized error starts at 881 and fluctuates wildly due to the nonlinearity of the signum function applied to network output. Wide swings in the error values during training can be advantageous if the training program has the weight saving procedure described in Section A. The best performance of this network during pattern matching training occurred at trial 970 with a linear error of 93 and a quantized error of 97. The norms of the three weight

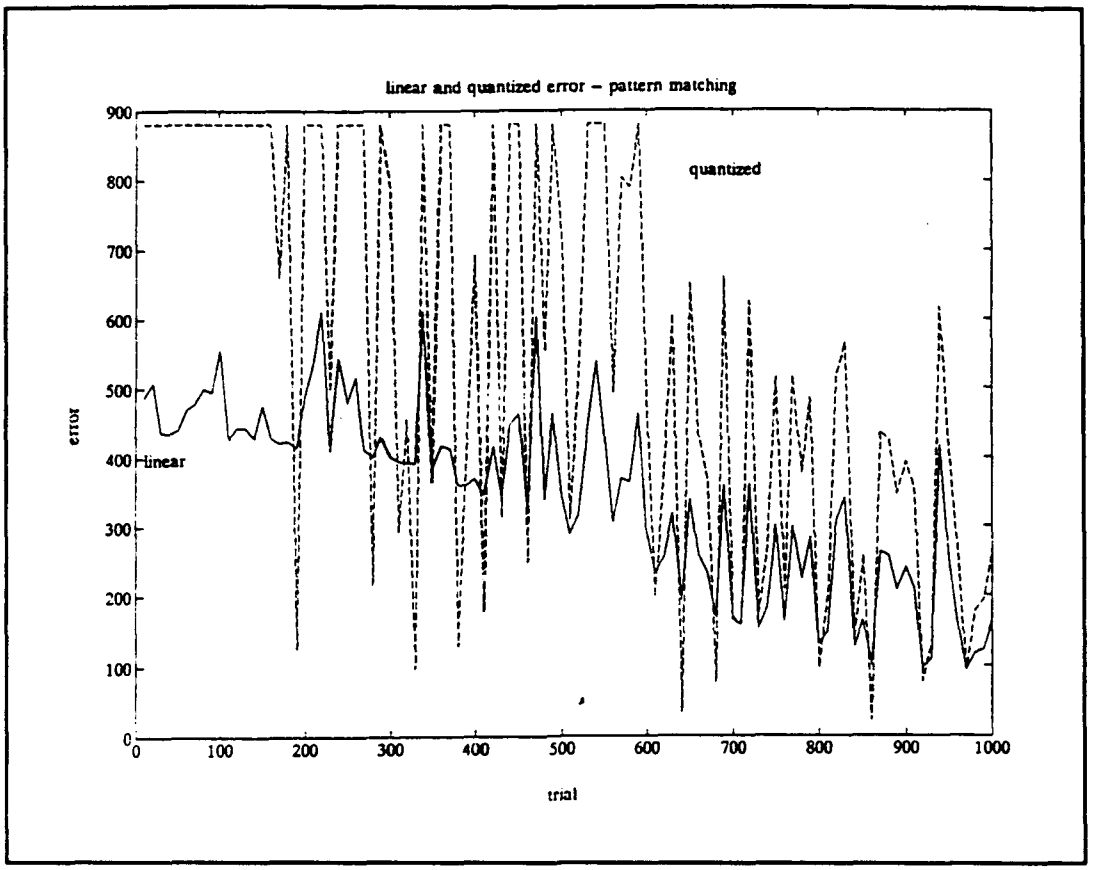

Figure 4.4. Linear and Quantized Error Record During Pattern Matching Training.

matrices, W, V, and Z, are shown in Figure 4.5. Matrix W is a 3 by 9 matrix that scales the state variable inputs and the bias input into the first hidden layer neurons. Since the neurons saturate at 1.0 outside of a relatively short span of their inputs, the W matrix normalizes the inputs to this useable range. If the product of the inputs and the W weights is too large, no training occurs because the derivative of the neuron would be near zero. Matrix V connects the first and second hidden layers. Although V is 10 by 4 its norm is smaller than W, probably because the range of values developed in the first hidden layer are between 1.0. The norm of matrix Z, a 1 by 5 vector, is the smallest. Growth of the norms is evident in all phases of training.

Figure 4.6 is the neural network output after 970 training cycles. The network has begun to adapt its output to approximate the desired optimal control surface. The initial flat (Figure 4.1) profile has been replaced by a sloping plane that descends in the positive theta direction.

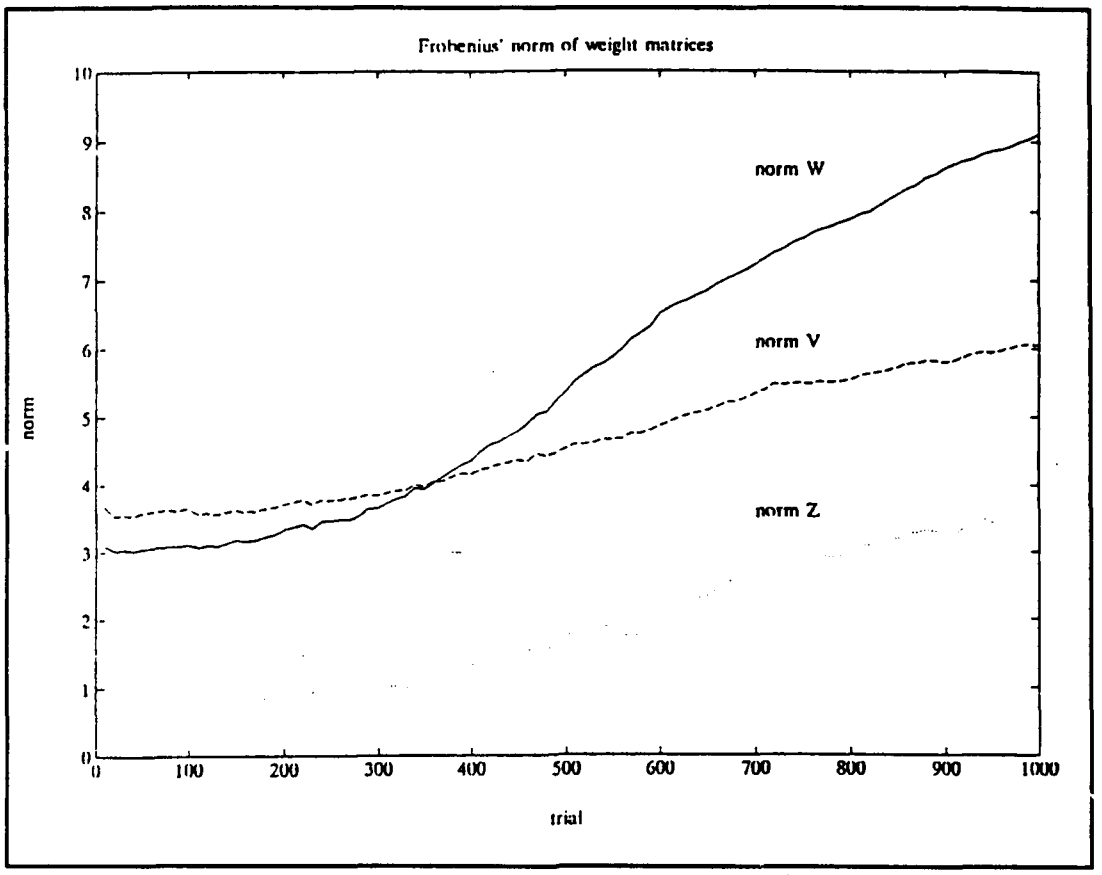

Figure 4.5. Norms of Weight Matrices During Pattern Matching Training.

The network error surface is illustrated in Figure 4.7. The central ridge shows the region of the state space where the sign of the network output is wrong. This plot indicates that the zero-trace of the neural network function is not coincident with the desired optimal control zero-trace. Network response in the regions on either side of the ridge is correct.

The error shown in Figure 4.7 causes inaccuracy in controlling the CER. Figure 4.8 displays the state space trajectory (solid line) that partly overlays the optimal trajectory (dashed line). The incorrect decision region corresponding to the ridge in Figure 4.7 causes the CER to continue to accelerate past the optimal switching curve instead of reversing thrust and driving the states to the origin. However, the improvement in control from that shown in Figure 4.3 is significant. Neural networks are not like negative feedback loops where the feedback acts to reduce the stimulus to the system. The desired response must be trained.

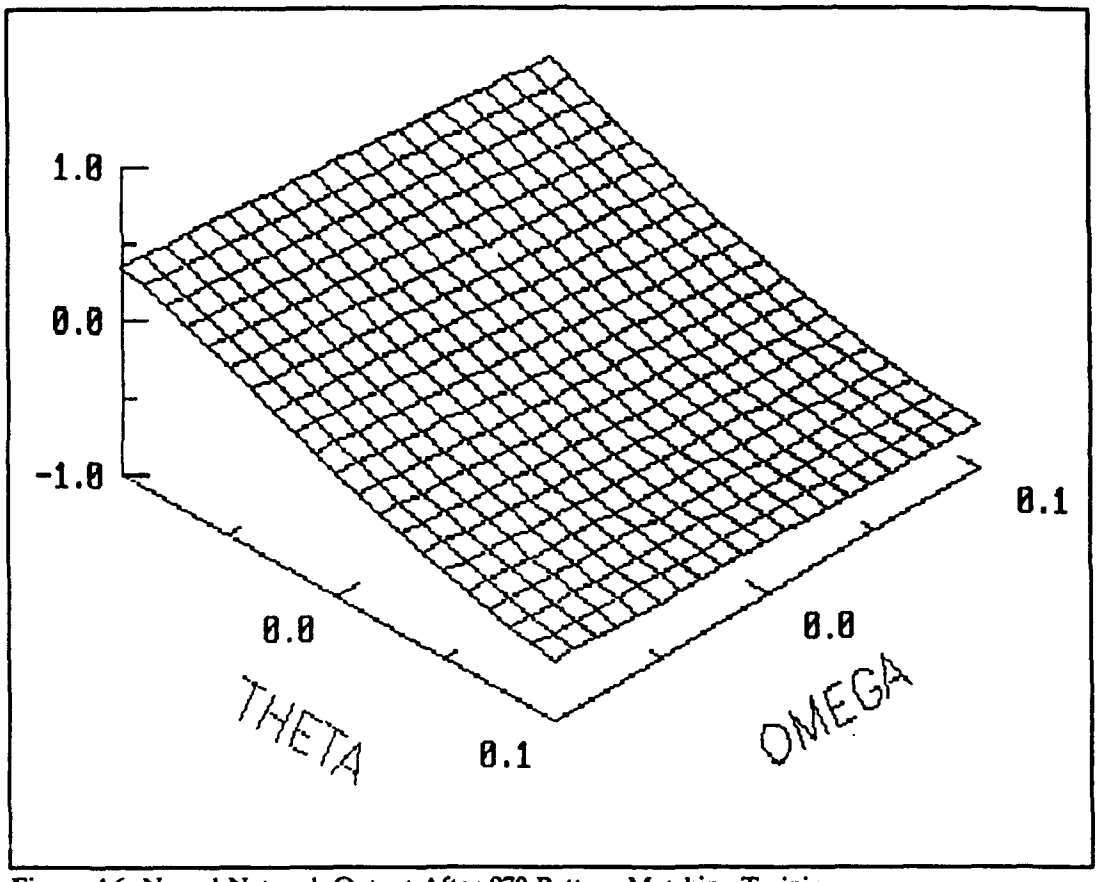

Figure 4.6. Neural Network Output After 970 Pattern Matching Training.

D. PERFORMANCE INDEX TRAINING

Performance index training is a new idea devised to enforce a performance measure on the combined neural network and physical system. The technique is based on the concept that a neural network will learn to meet a specified performance goal if it is the only solution to the problem. Nguyen and Widrow [Ref. 3] have demonstrated that a neural network can be trained to control a dynamic system to achieve a desired state. However, the control signal mapping learned by their neural network is largely dependent upon the early training presentations and can vary between different networks. Performance index training develops a neural network that can be compared with classically designed control laws in terms of minimizing a specified cost function.

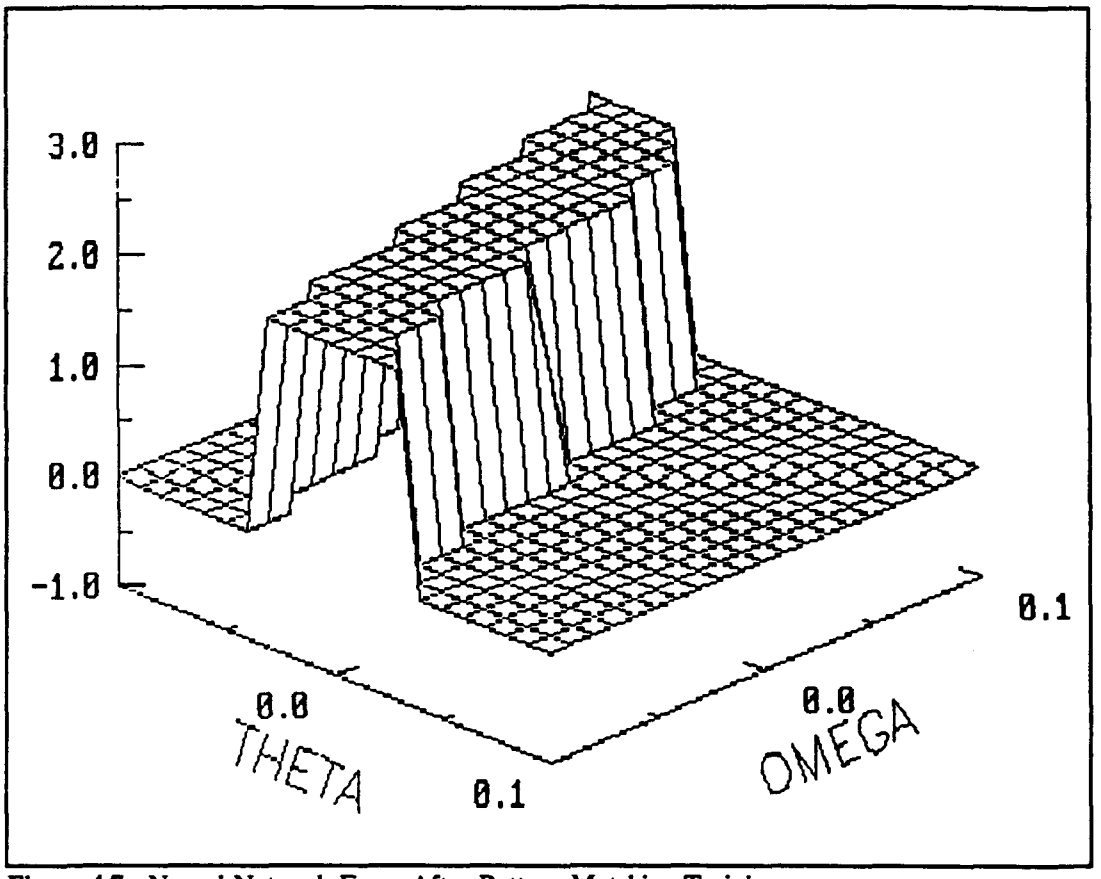

Figure 4.7. Neural Network Error After Pattern Matching Training.

For this research the cost function to be minimized was the time to reach the origin. Random initial conditions were applied to a time simulation of the physical system. An estimate of the minimum time required to reach the origin of the state space was computed based on the initial conditions. For the minimum time trajectory the estimate can be computed by solving the equations of motion based on a two-legged trajectory with constant acceleration (in opposite directions) starting at the initial conditions and terminating at the origin (Figure 4.9).

The time estimate sets the end time of the simulation. The neural network is allowed to control the CER from time zero until the maximum time estimate is reached. When the maximum time was reached the simulation was stopped and the final states forms the state error. Optimal performance by the neural network would drive the system to the origin in exactly the amount of time predicted. Sub-optimal performance by the neural net results in a non-zero final state. The final state was compared

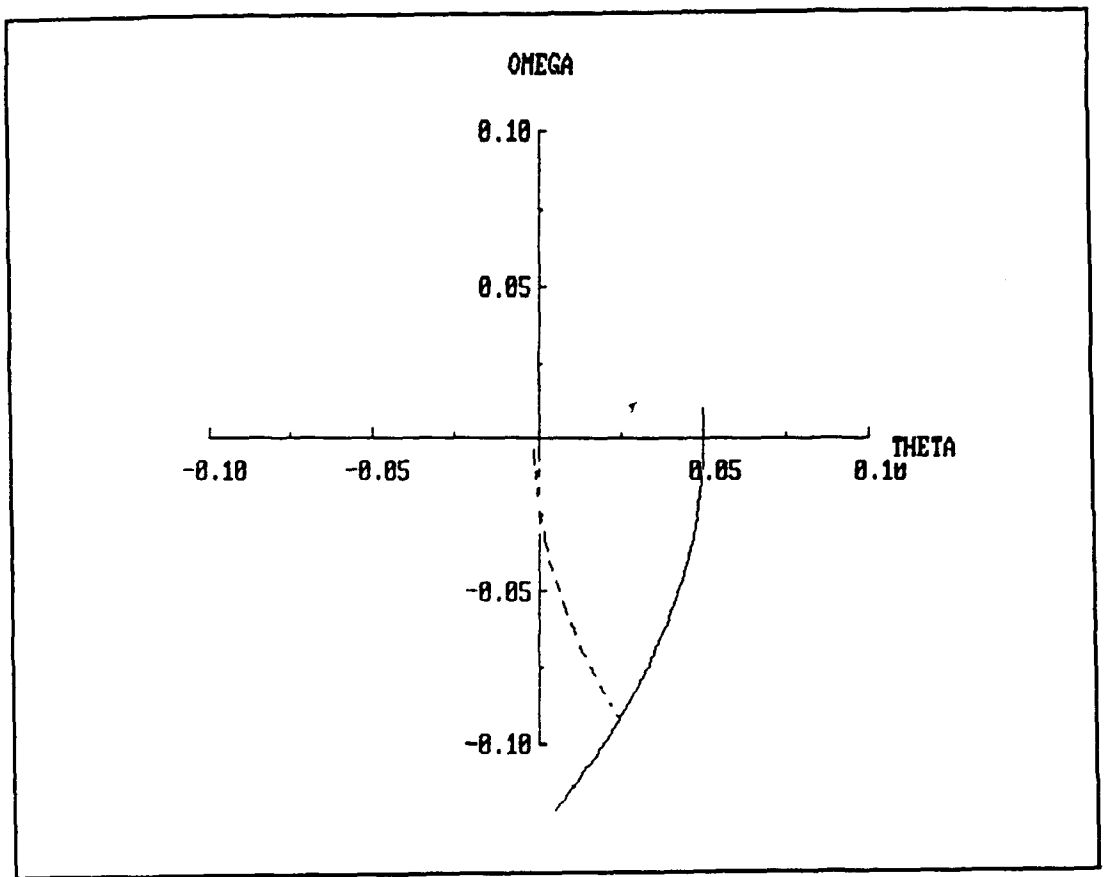

Figure 4.8. State Space Trajectories After Pattern Matching Training.

to the desired final state (Equations. 4-4 and 4-5) and the error was back-propagated through the plant partial derivatives (Eqn. 3-34) here repeated as equation (4-6).

$$E_s = \frac{1}{2} \sum \left[\vec{0} - \vec{x}(final)\right]^2 \tag{4-4}$$

$$\frac{\partial E}{\partial x} = -1.0\ (-x_1 - x_2)\ |_{final} = -1.0\ (-\theta - \omega)|_{final} \tag{4-5}$$

$$DC = SIGN\left(\frac{\partial E}{\partial x} \frac{\partial x}{\partial u} \right) \tag{4-6}$$

The sign of equation (4-6) can be interpreted as the desired control signal (DC). The desired control signal was compared to the actual neural network output at the final time (Eqn. 4-6) and this value (NE) was back-propagated into the network to train the weights.

$$NE = -1.0\left[SIGN(DC) - C(\theta,\omega)_{final} \right] \tag{4-7}$$

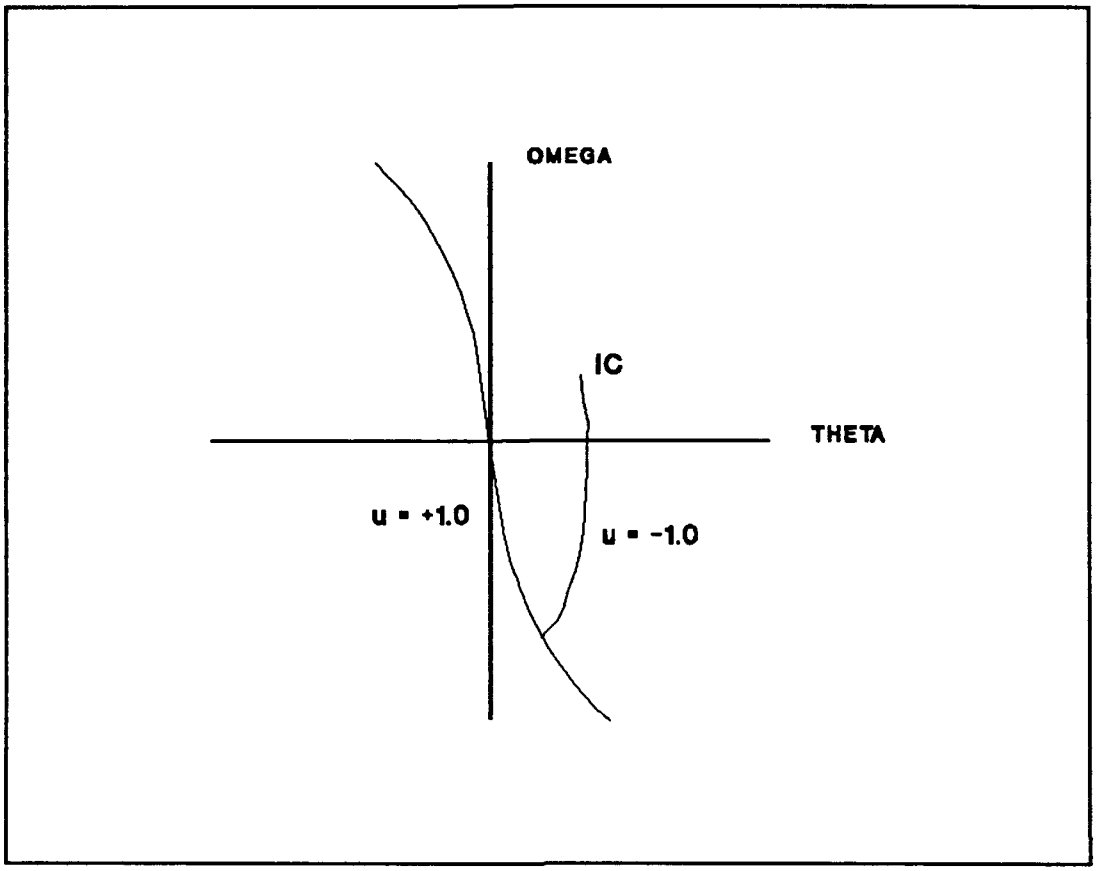

Figure 4.9. Optimal Trajectory Time Estimation.

Equation (4-7) is the modification to error back-propagation that takes into account the quantizer between the neural network output and the thruster input.

Linear and quantized error calculations are carried out as before. Figure 4.10 shows the error values for the neural network described in Section B. The network was trained by pattern matching first, then subjected to performance index training. The neural network was trained 710 times using learning rates from 0.05 to 0.02. The network with the best performance yielded a linear error of 67.7 and a quantized error equal to 25. Several networks with a quantized error of 17 were not retained because their linear error was greater than 67.7. Although numerical improvement of the linear error is modest, the reduction in quantized error is dramatic. The actual optimal control law was not used

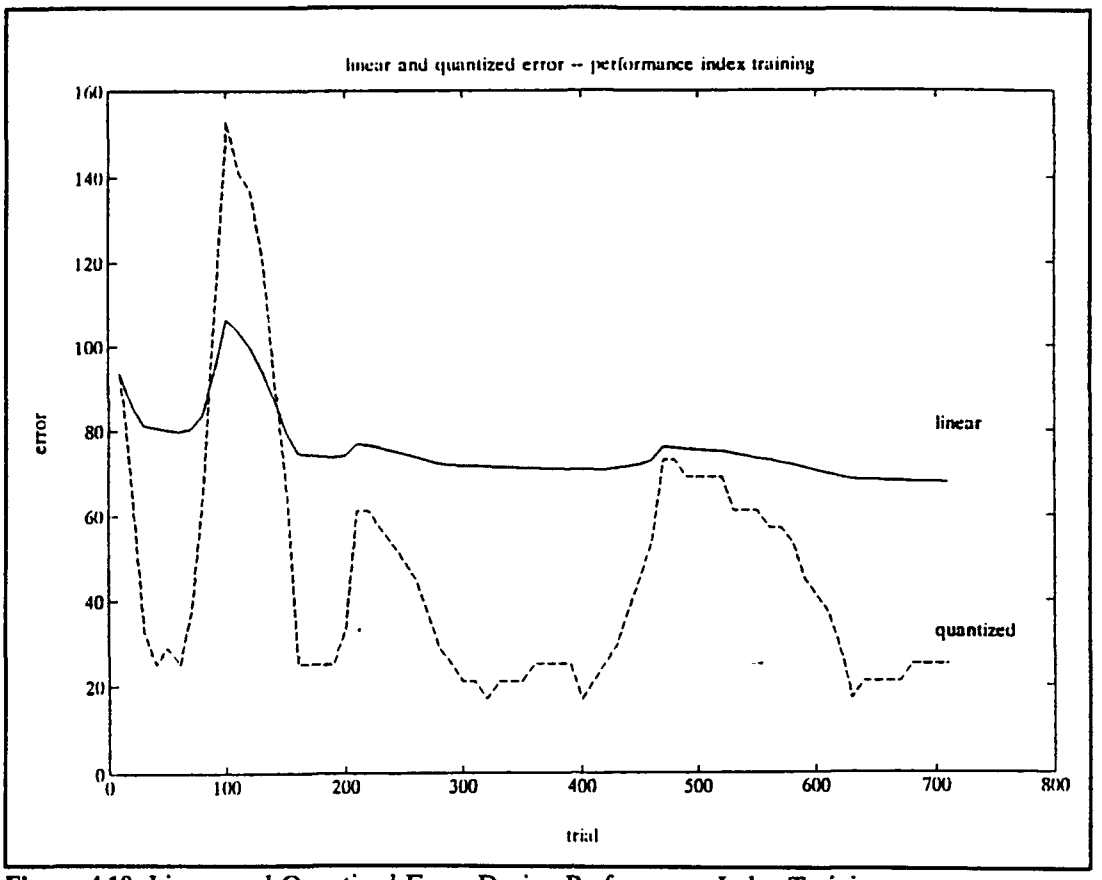

Figure 4.10. Linear and Quantized Error During Performance Index Training.

to train the neural net in this case. Improvements in the network response are due to the errors in thruster control signals, integrated over time, and manifested as non-zero final states.

The flattening of the linear error curve may be due to a fundamental limitation of the learning capacity of a network with only 14 neurons. The size of a network appears to be analogous to the number of terms in a Taylor or Fourier series. The accuracy of a series representation of a function increases with an increase in the number of terms being summed. Neural networks also follow this pattern [Ref. 3]. Nets with more neurons are able to approximate arbitrary functions better than smaller networks. The norms of the weight matrices (Figure 4.11) show little change due to the small learning rate applied. As the network output approaches the desired function the growth of the weight matrices will also slow.

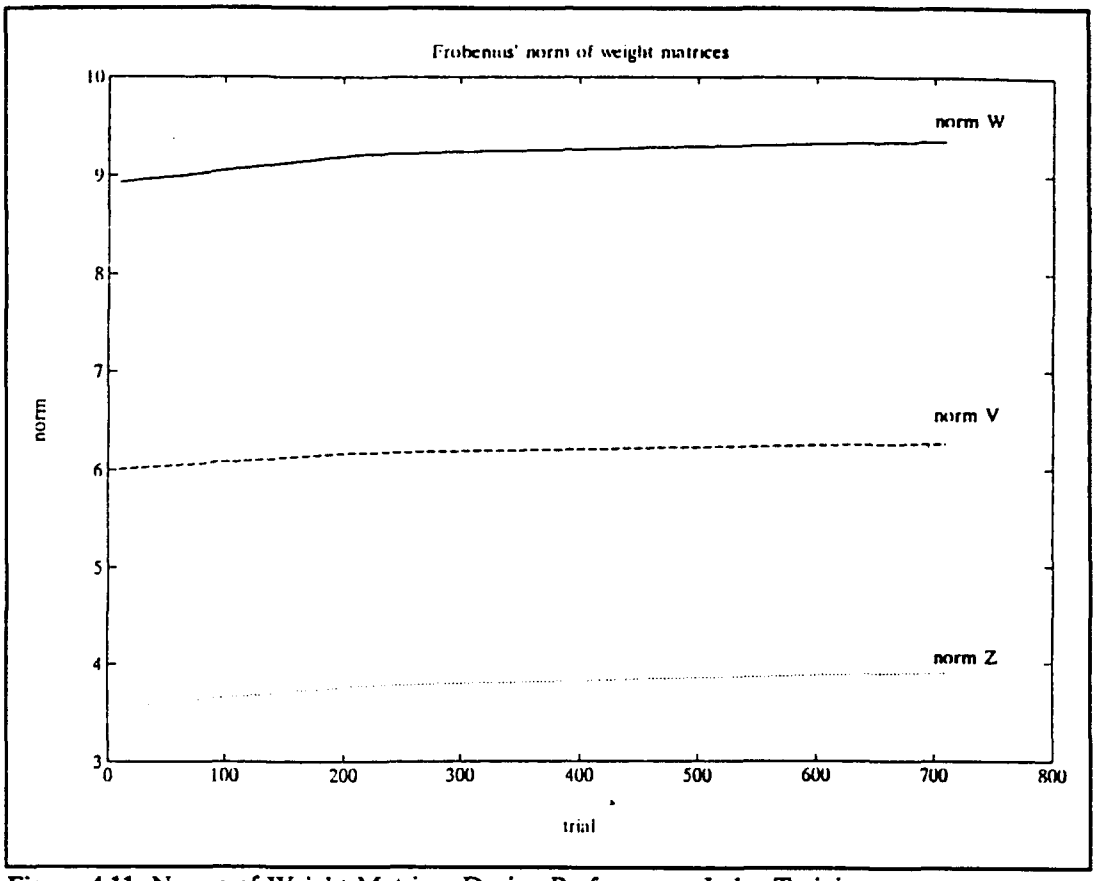

Figure 4.11. Norms of Weight Matrices During Performance Index Training

The neural net output after performance index training (Figure 4.12) shows a more complicated shape. The maximum and minimum values more closely approximate the desired thruster control signal values. The curved slope has evolved from the straight slope (observed in Figure 4.6) and is beginning to be a continuous approximation to the finite discontinuity of the optimal control law. The steepness of the slope is a function of the number of neurons in the second hidden layer. As more neurons are added or as the weights increase, the weighted sum appearing at the output neuron will change value more quickly causing the output to change accordingly.

The output error plot in Figure 4.13 illustrates the accuracy of the learned response. The spikes depicted are symmetric about the origin of the state space and control signal plane. The zero-trace of the neural network output closely approximates that of the optimal control law. The neural network

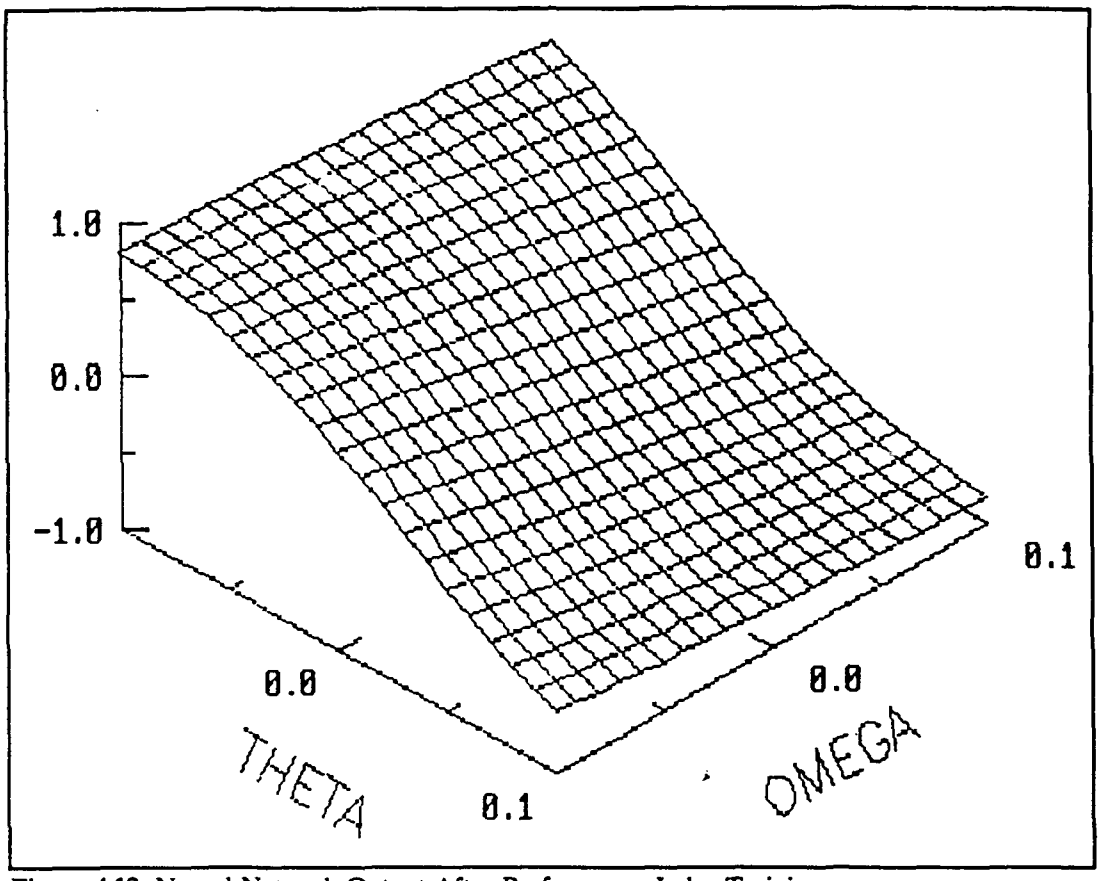

Figure 4.12. Neural Network Output After Performance Index Training.

has learned the optimal control mapping over most of the state space domain. The zero-trace of the network output is still approximately a straight line. A network with more neurons may be able to more closely fit the parabolic shape of the optimal control switching curve.

The state space trajectory shown in Figure 4.14 indicates that the network has learned a near-optimal control law. The neural net (solid line) turns before the optimal control (dashed). Since any path is less optimal than the optimal control, the neural net fails to reach the origin in the allotted time. The improvement in system control from the untrained network shown in Figure 4.3 is significant. The uncontrolled system has now been replaced with a system that performs with near-optimal precision. While some of the initial training has taken place with the benefit of full knowledge of the correct answer, the performance index training has improved the network by virtue of experience and without the knowledge of the correct answer.

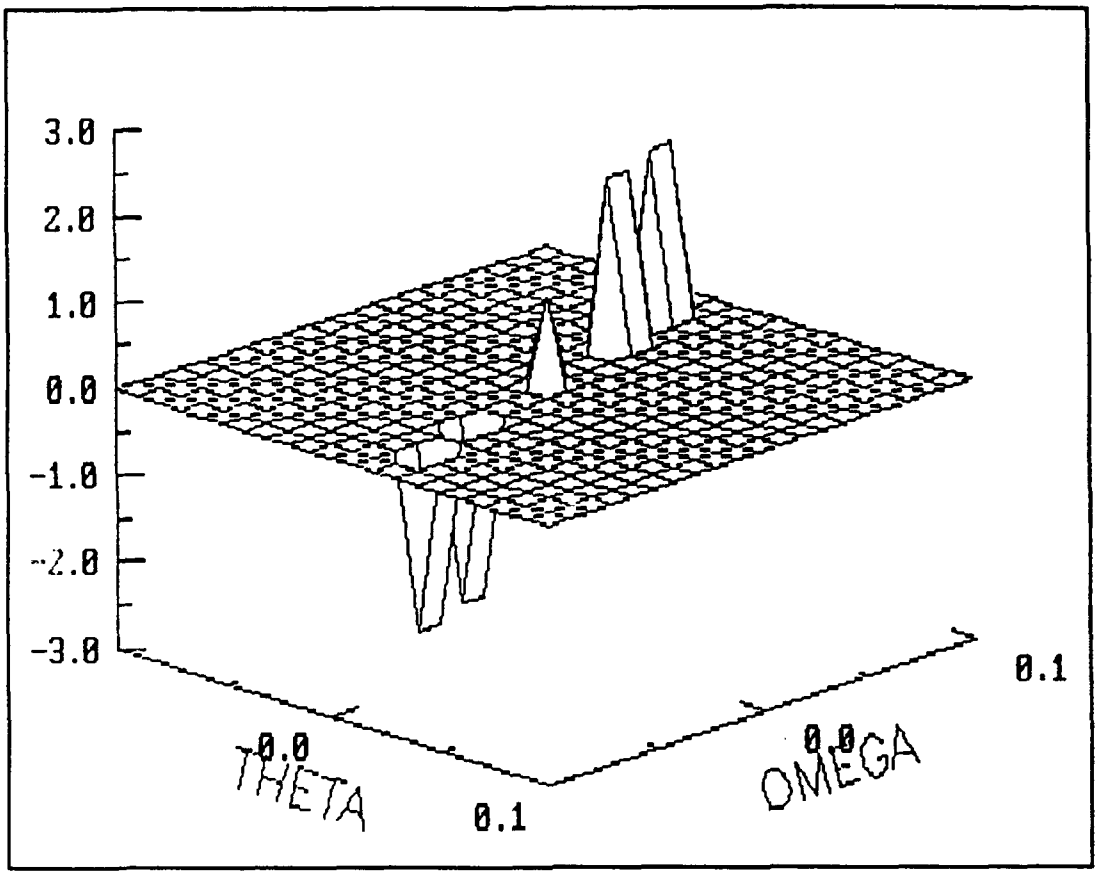

Figure 4.13. Neural Network Error After Performance Index Training.

E. REAL-TIME ADAPTATION

Performance index training represents a method by which neural networks in real-time control systems can be adapted to improve their performance while performing their tasks. In the case of the CER with a payload, the neural net controls the CER until a specified time (based on the initial error) is exceeded. Non-zero states are then back-propagated and the neural network weights are adjusted. This process can be repeated until a given performance measure is achieved. If the neural network is performing in a near-optimal manner initially, it should converge to the correct control law. An experiment to demonstrate this was conducted on several networks. The networks were initially trained

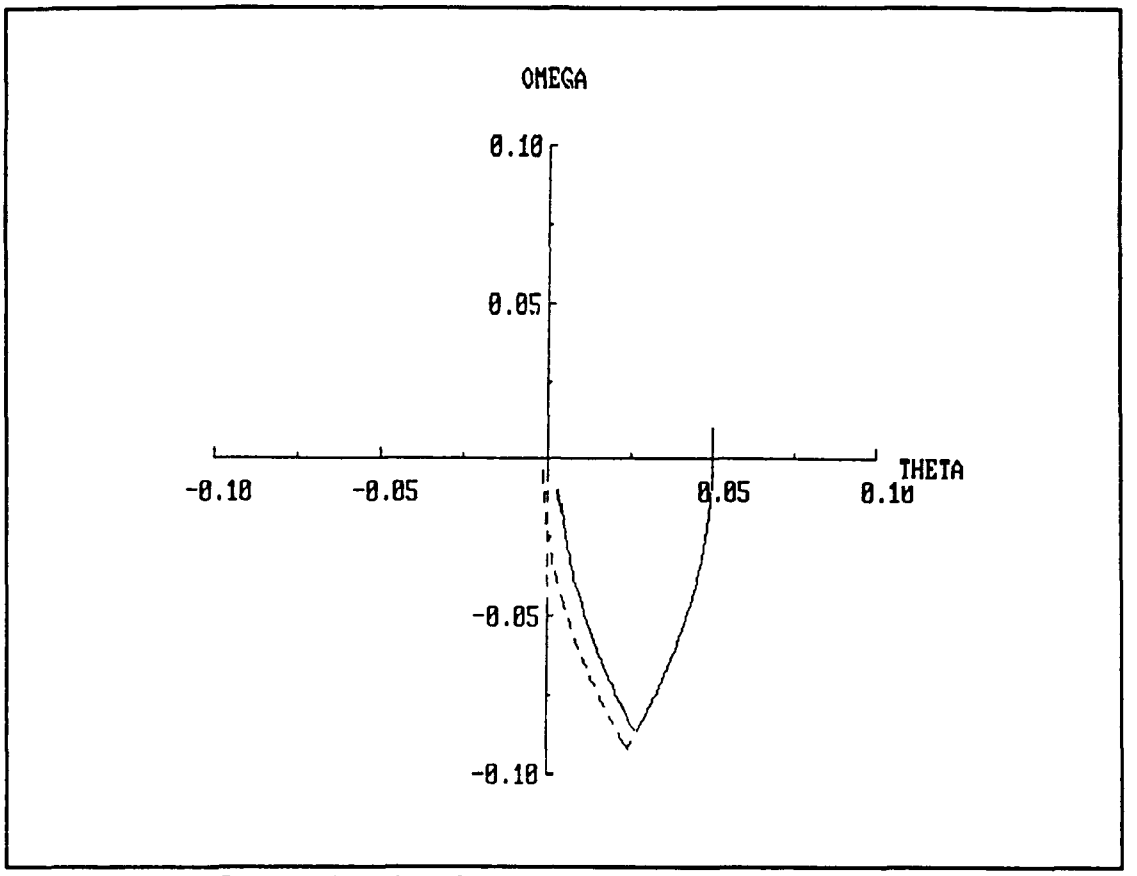

Figure 4.14. State Space Trajectories After Performance Index Training.

to control a baseline CER, then allowed to control a CER with twice the original moment of inertia. Time estimates for this experiment were based on the doubled mass, but the plant partial derivatives came from the baseline CER values.

Figure 4.15 shows the error surface of the network trained on a baseline CER which produced Figure 4.13. The optimal control law used to evaluate the error corresponds to the increased mass CER. Error values for this network were computed to be 160 and 55 for linear and quantized measures, respectively.

The state space trajectories of a double mass CER controlled by the optimal control law and the baseline trained network are diagrammed in Figure 4.16. The neural network (solid line) now turns late when compared to the correct control law (dashed line).

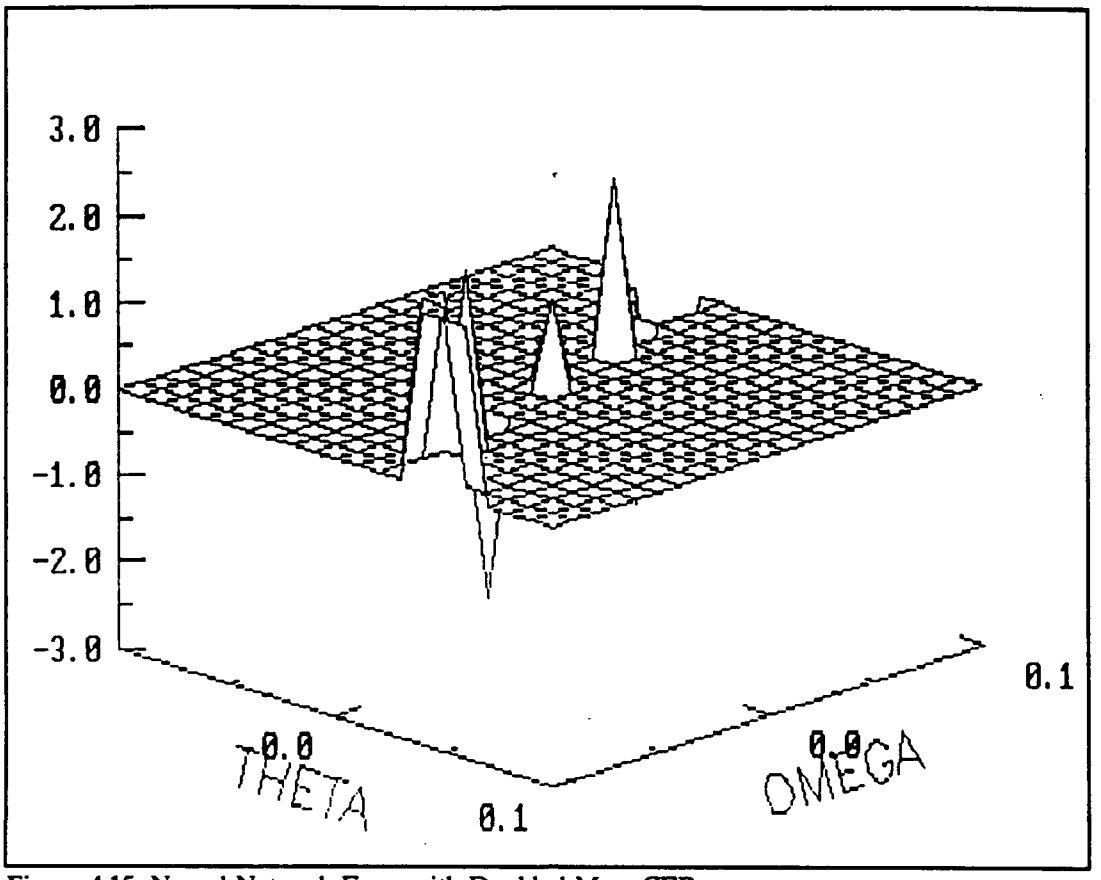

Figure 4.15. Neural Network Error with Doubled Mass CER.

Real-time adaptation of the neural network by performance index training was conducted for 1200 trials. Average length of each trial, in clock time, is 3 seconds. The final linear error has been reduced to 89.9 with quantized error of 61. Although the quantized error has not been greatly reduced, the linear error has been halved and the quantized error would also follow in time.

Figure 4.17 shows the neural network output after the training. The slope of the output surface is noticeably steeper indicating a better fit to the desired response.

Neural network error (Fig. 4.18) again displays symmetric errors. The optimal control law switching curve for the doubled mass CER does not coincide with the baseline CER. The zero-trace of the network output is still a straight line; however, the slope of the line has adjusted to best fit the parabolic shape of the new optimal control law.

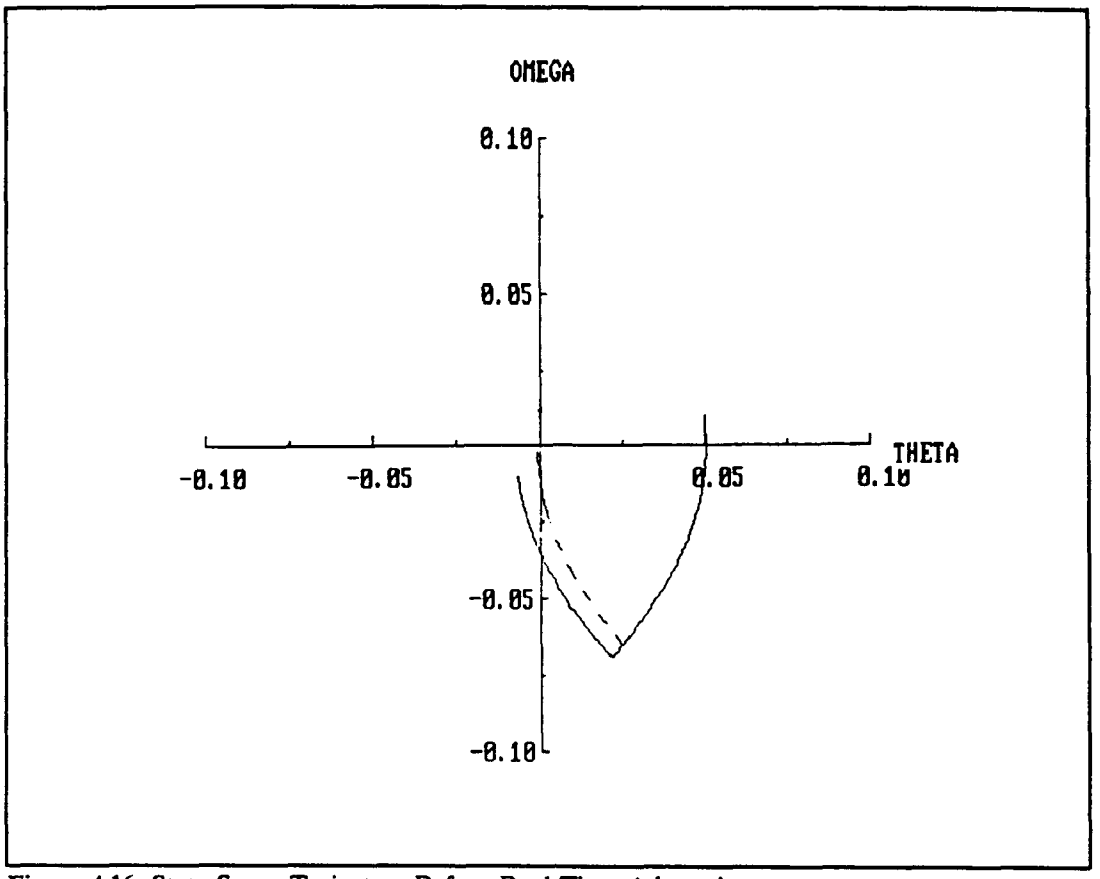

Figure 4.16. State Space Trajectory Before Real-Time Adaptation.

Figure 4.19 shows the state space trajectories for the network after real-time adaptation by performance index training. The turning point for the neural net-controlled CER (solid line) has moved to the right of the optimally controlled CER (dashed line). The end states are slightly closer to the origin than those of Figure 4.16. The trajectory produced by the neural net is better than before in that overshoot has been reduced. The end velocity of the previous trajectory (Figure 4.16) was increasing the position error while the end velocity in Figure 4.19 tends to reduce the error. The neural network has partially adapted to the change in mass.

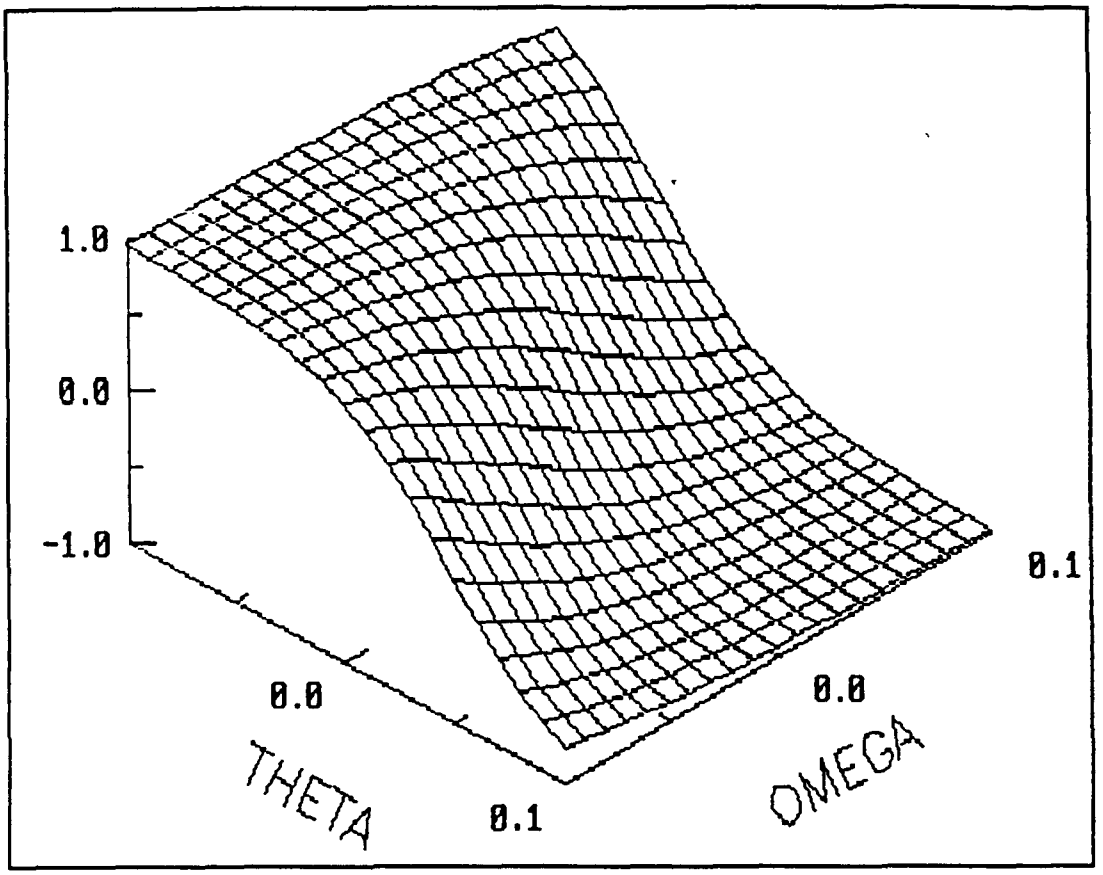

Figure 4.17. Neural Network Output After Real-Time Adaptation.

Real-time adaptation via performance index training may yield intelligent control systems that constantly adapt themselves to changing system characteristics. The doubled mass experiment could have also been interpreted as a reduction in thrust. Off-axis payloads in the CER's nets can change the moment of inertia characteristics of the system. All of the situations described above are circumstances in which performance index training may be able to yield an adaptive controller that can overcome unknown system parameter changes. Pattern matching training may be conducted off-line to give a neural network control system the response characteristic to enable it to immediately control a system. On-line adaptation can then be implemented to adjust for changing conditions.

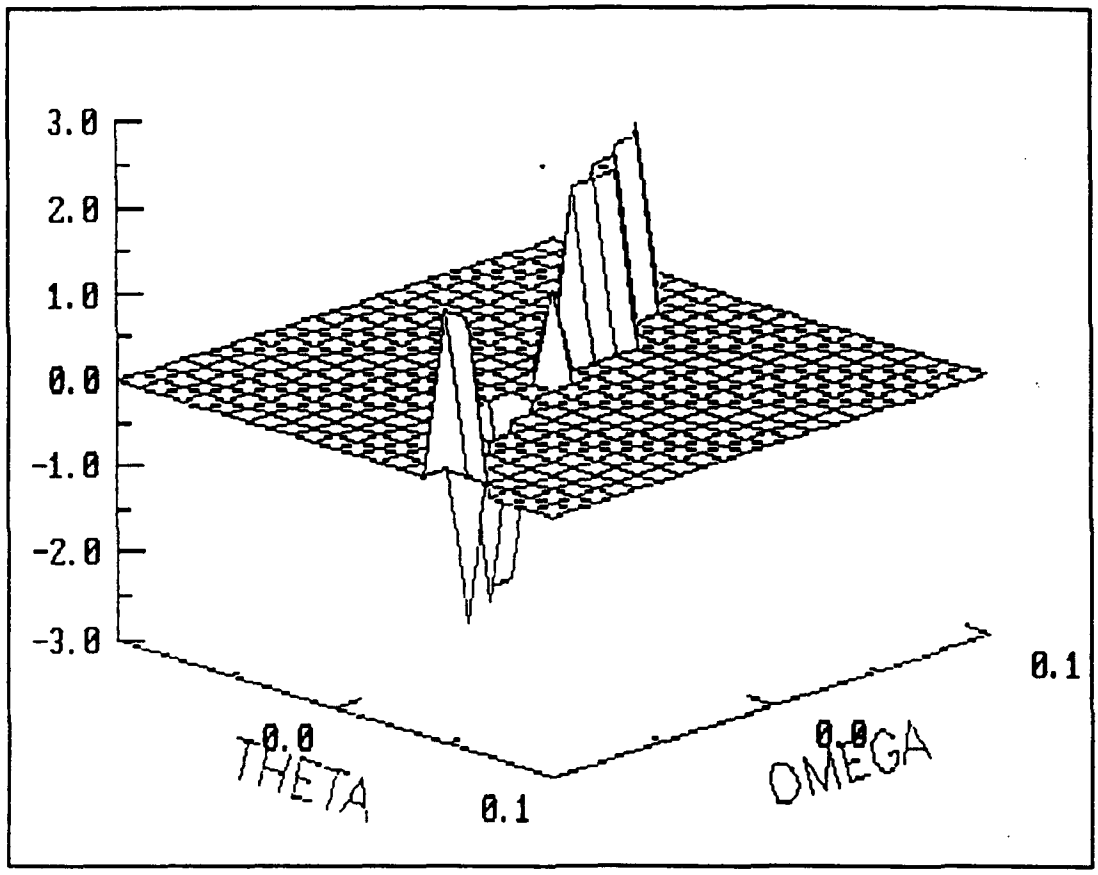

Figure 4.18. Neural Network Error After Real-Time Adaptation.

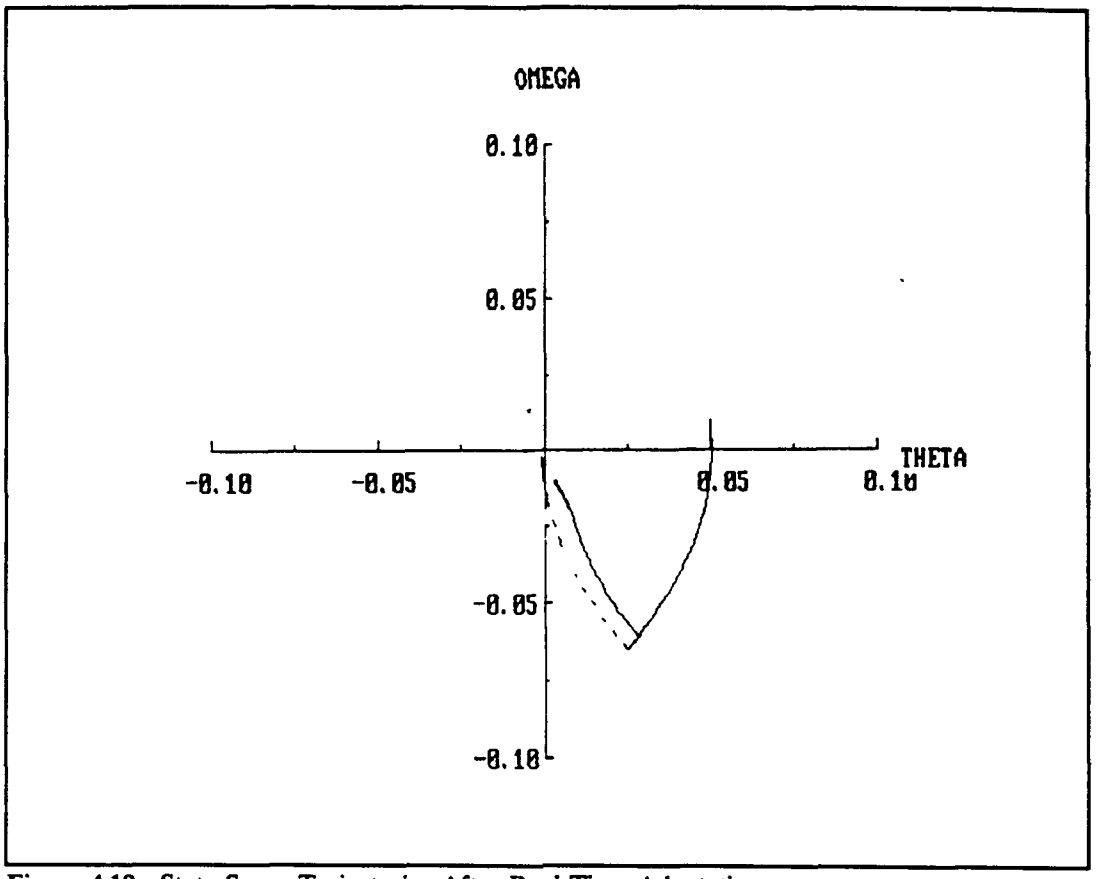

Figure 4.19. State Space Trajectories After Real-Time Adaptation.

V. CONCLUSIONS AND RECOMMENDATIONS

Artificial neural networks can be trained to control dynamic systems in a near-optimal manner. Pattern matching training delivers rapid learning when the desired response is known. Performance index training causes the neural network to adapt to meet the performance goal specified by the designer. Real-time intelligent control with on-line learning can be implemented via performance index training.

Future research in the area of applications of neural networks to control systems should focus on several aspects of real-time intelligent control. A more sophisticated back-propagation algorithm may reduce the variance of the error calculations and yield a more graceful error reduction curve.

Training a neural network to learn the optimal control law without the quantizer could allow extended error back-propagation to be used.

Performance index training should be investigated to determine if the concept is general enough to be applied to multiple performance indices. The multiple index training scheme could be applied to weighted fuel-time optimal control. Estimates of both the fuel and time required to reach the origin from arbitrary initial conditions can be used to control the time simulation. Excessive thrusting would exhaust the fuel allotment and cause the CER to coast. The states at the final time can be back-propagated to train the network. Arbitration between the time optimal (baseline CER) derivatives and fuel optimal derivatives could be used to decide on the desired control signal.

APPENDIX A. COMPUTER PROGRAMS

```
C     PROGRAM PATDATA.FOR
C     AUTHOR: C. M. SEGURA
C     DATE:  1 OCTOBER 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOFT FORTRAN 4.0
C     REVISED: 13 OCTOBER 1989
C
C     THIS PROGRAM IMPLEMENTS THE TIME OPTIMAL CONTROL SWITCHING
C     CURVE TRAINING OF A NEURAL NETWORK WITH GRAPHIC OUTPUT.
C     3RD NETWORK CONFIGURATION
C     DATA EXTRACTION ADDED
$NOTRUNCATE
      COMMON /WEIGHT/W,V,ZW,HF,HNET,ONET,OF,LR,FNET
      COMMON /SIZE/NINPUT,NHIDE,NOUT
      REAL X(21,21),Y(21,21),Z(21,21),INP(10,1),NEURALNET,TIMEOPT,LR,
     1W(50,10),V(50,50),HF(50,1),HNET(50,1),ONET(50),ERR(21,21),RNG,
     2ZW(50),OF(50,1),DWN,DVN,DZN,NORMM,NORMV,LSSE,QSSE,BW(50,10),
     3BV(50,50),BZ(50)
      INTEGER NINPUT,NHIDE,NOUT,M,N,COUNT,CHOICE,BTRIAL
      INTEGER*4 IX,TRIAL
      CHARACTER*14 FNAME,NNAME
      CALL GETTIM(IHR,IMIN,ISEC,IHUN)
      IX = 1000*IHR + 100*IMIN + 10*ISEC +IHUN
C     SET NEURAL NET SIZE
      NINPUT = 3
      NHIDE = 9
      NOUT = 2
      LR = .5
C     SET DATA SPACE SIZE
      M = 21
      N = 21
C     PROGRAM STATUS AND CONTROL SECTION
100   CALL QCLEAR(0,7)
      PRINT *,'PROGRAM NAME: PATDATA.FOR'
      PRINT *,'PROGRAM TASK: CER TIME OPTIMAL CONTROL LAW W/ DATA'
PRINT *,'NETWORK FILE: ',NNAME,'   DATA FILE: ',FNAME
      PRINT *,'CONFIGURATION: ',NINPUT,NHIDE,NOUT
      PRINT *,'LEARNING RATE: ',LR,'  ITERATIONS: ',TRIAL
      PRINT *,'SUM OF SQUARED ERROR: ',SSE1,SSE2
      PRINT *,' '
      PRINT *,'ENTER FUNCTION CODE'
      PRINT *,'      1: INITIALIZE NETWORK'
      PRINT *,'      2: SET LEARNING RATE'
      PRINT *,'      3: SET OUTPUT FILE'
```

```
      PRINT *,'      4: CONDUCT TRAINING'
      PRINT *,'      5: VIEW SWITCHING SURFACE'
      PRINT *,'      6: VIEW ERROR SURFACE'
      PRINT *,'      7: VIEW SWITCHING CONTOURS'
      PRINT *,'      8: PLOT TRAJECTORIES'
      PRINT *,'      9: SAVE NETWORK TO A FILE'
      PRINT *,'     10: QUIT'
      READ (*,*)CHOICE
      GOTO (200,250,275,300,400,500,600,700,800,900),CHOICE
C     INITIALIZE THE NEURAL NETWORK
200   CALL NETINIT(NNAME)
      INP(NINPUT,1) = 1.0
      TRIAL = 0
      SSE1 = 0.0
      SSE2 = 0.0
      LSSE = 1.0E5
      GOTO 100
c       SET LEARNING RATE
250   PRINT *,'ENTER LEARNING RATE'
      READ(5,*)LR
      GOTO 100
C       SELECT OUTPUT DATA FILE NAME
275   PRINT *,'ENTER DATA FILE NAME'
      READ(*,*)FNAME
      OPEN(10,FILE=FNAME)
      GOTO 100
C     TRAINING PHASE OF NETWORK
300     PRINT *,'ENTER NUMBER OF ITERATIONS (0 TO STOP)'
        READ(*,*)COUNT
C       COMMENCE ITERATIONS
C       SELECT RANDOM INITIAL CONDITIONS WITHIN BOUNDARIES
C       OF -.1<OMEGA<.1, AND -.1 < THETA < .1.
20      INP(1,1) = (UNIF(IX)-.5)*.20
        INP(2,1) = (UNIF(IX)-.5)*.20
        C = NEURALNET(INP)
        DES = TIMEOPT(INP(1,1),INP(2,1))
        COUNT = COUNT-1
        TRIAL = TRIAL + 1
C         COMPUTE NETWORK ERROR
        E1 = -1.0*(DES - C)
C       PRINT *,'INPUT,OUTPUT,DESIRED',INP(1,1),INP(2,1),C,DES
C       PRINT *,'ERROR',E1
C         TRAIN NETWORK WEIGHTS
        CALL TRAINER(E1,INP)
      IF(MOD(TRIAL,10).NE. 0) GOTO 20
C     COMPUTE THE ERROR SURFACE BY COMPARING NEURAL NET OUTPUT
C     WITH THE ACUTAL SWITCHING CURVE
30    SSE1 = 0.0
      SSE2 = 0.0
      DO 40 I = 1,M
        DO 45 J = 1,N
```

```
        X(I,J) = FLOAT(I-11)/100.
        Y(I,J) = FLOAT(J-11)/100.
        INP(1,1) = X(I,J)
        INP(2,1) = Y(I,J)
        Z(I,J) = NEURALNET(INP)
        DES = TIMEOPT(X(I,J),Y(I,J))
        SSE1 = SSE1 + (DES - Z(I,J))**2
        ERR(I,J) = DES - SIGN(1.0,Z(I,J))
        SSE2 = SSE2 + ERR(I,J)**2.0
45     CONTINUE
40    CONTINUE
C       COMPUTE NORMS OF MATRICES
      DWN = NORMM(W,50,10,NHIDE-1,NINPUT)
      DVN = NORMM(V,50,50,NOUT,NHIDE)
      DZN = NORMV(ZW,50,NOUT)
C       WRITE DATA TO SCREEN AND OUTPUT FILE
50    PRINT 103,TRIAL,DWN,DVN,DZN,SSE1,SSE2
      WRITE(10,103)TRIAL,DWN,DVN,DZN,SSE1,SSE2
103   FORMAT(2X,I5,5(2X,G12.7))
C       CHECK PERFORMANCE AND SAVE BEST WEIGHTS
      IF(SSE1 .LT. LSSE) THEN
       CALL BESTNET(BW,BV,BZ)
       LSSE = SSE1
       QSSE = SSE2
       BTRIAL = TRIAL
      ENDIF
      IF (COUNT) 100,100,20
C     PLOT NEURAL NET OUTPUT
400   CALL ROTATE(X,Y,Z,M,N)
      GOTO 100
C     PLOT ERROR SURFACE
500   CALL ROTATE(X,Y,ERR,M,N)
      GOTO 100
C       PLOT NETWORK OUTPUT CONTOURS
600   CALL CONTOUR(X,Y,Z,M,N,5)
      GOTO 100
C       SIMULATE NETWORK CONTROL AND OPTIMAL CONTROL
700   CALL SIMUL
      GOTO 100
C       SAVE WEIGHTS WITH BEST PERFORMANCE VALUES
800   CALL NETSAVE(BW,BV,BZ)
      PRINT *,'BEST PERFORMANCE AT TRIAL ',BTRIAL
      PRINT *,'WITH ERRORS ',LSSE,QSSE
      PAUSE ' '
      GOTO 100
900   CLOSE(10)
      STOP
      END
C       INCLUDE THESE FILES DURING COMPILATION
$INCLUDE:'UTIL.FOR'
$INCLUDE:'NETWORK3.FOR'
```

```
$INCLUDE:'TRAINER3.FOR'
$INCLUDE:'PLOT.FOR'
$INCLUDE:'SIMUL.FOR'
$INCLUDE:'PLANT.FOR'
```

```
C     PROGRAM TIMDATA.FOR
C     AUTHOR: C. M. SEGURA
C     DATE:   10 OCTOBER 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOFT FORTRAN 4.0
C     REVISED: 22 OCTOBER 1989
C
C     THIS PROGRAM IMPLEMENTS THE TIME OPTIMAL CONTROL SWITCHING
C     CURVE TRAINING OF A NEURAL NETWORK WITH GRAPHIC OUTPUT.
C     3RD NETWORK CONFIGURATION, PLANT DERIVATIVE COEFFICIENTS
C     DATA EXTRACTION ADDED
$NOTRUNCATE
      COMMON /WEIGHT/W,V,ZW,HF,HNET,ONET,OF,LR,FNET
      COMMON /SIZE/NINPUT,NHIDE,NOUT
      REAL X(21,21),Y(21,21),Z(21,21),INP(10,1),NEURALNET,TIMEOPT,LR,
     1W(50,10),V(50,50),HF(50,1),HNET(50,1),ONET(50),ERR(21,21),RNG,
     2ZW(50),OF(50,1),DWN,DVN,DZN,NORMM,NORMV,LSSE,BW(50,10),
     3BV(50,50),BZ(50),FNET,CE,BX,BY,DX(270),DXD(270)
      INTEGER NINPUT,NHIDE,NOUT,M,N,COUNT,CHOICE,BTRIAL,T,TMAX,SSE2,
1QSSE
      INTEGER*4 IX,TRIAL
      CHARACTER*14 FNAME,NNAME
      CALL GETTIM(IHR,IMIN,ISEC,IHUN)
      IX = 1000*IHR + 100*IMIN + 10*ISEC +IHUN
      BX = .0218
      BY = 8.734E-4
C     SET NEURAL NET SIZE
      NINPUT = 3
      NHIDE = 9
      NOUT = 2
      LR = .5
C     SET DATA SPACE SIZE
      M = 21
      N = 21
C     READ STEPWISE PLANT DERIVATIVE DATA FROM FILE
      OPEN(11,FILE = 'PART1.DAT')
      DO 5 I = 1,45
       LN = 6*(I-1) + 1
       HN = LN+5
       READ(11,*)(DX(J),J = LN,HN)
       READ(11,*)(DXD(K),K= LN,HN)
5     CONTINUE
      CLOSE(11)
C     PROGRAM STATUS AND CONTROL SECTION
100   CALL QCLEAR(0,7)
      PRINT *,'PROGRAM NAME: TIMDATA.FOR'
      PRINT *,'PROGRAM TASK: CER TIME OPTIMAL CONTROL LAW W/ DATA'
PRINT *,'NETWORK FILE: ',NNAME,'  DATA FILE: ',FNAME
      PRINT *,'CONFIGURATION: ',NINPUT,NHIDE,NOUT
      PRINT *,'LEARNING RATE: ',LR,'  ITERATIONS: ',TRIAL
      PRINT *,'SUM OF SQUARED ERROR: ',SSE1,SSE2
```

```
      PRINT *,' '
      PRINT *,'ENTER FUNCTION CODE'
      PRINT *,'      1: INITIALIZE NETWORK'
      PRINT *,'      2: SET LEARNING RATE'
      PRINT *,'      3: SET OUTPUT FILE'
      PRINT *,'      4: CONDUCT TRAINING'
      PRINT *,'      5: VIEW SWITCHING SURFACE'
      PRINT *,'      6: VIEW ERROR SURFACE'
      PRINT *,'      7: VIEW SWITCHING CONTOURS'
      PRINT *,'      8: PLOT TRAJECTORIES'
      PRINT *,'      9: SAVE NETWORK TO A FILE'
      PRINT *,'     10: QUIT'
      READ (*,*)CHOICE
      GOTO (200,250,275,300,400,500,600,700,800,900),CHOICE
C     INITIALIZE THE NEURAL NETWORK
200   CALL NETINIT(NNAME)
      INP(NINPUT,1) = 1.0
      TRIAL = 0
      SSE1 = 0.0
      SSE2  0
      LSSE = 1.0E5
      GOTO 100
C       SET LEARNING RATE
250   PRINT *,'ENTER LEARNING RATE'
      READ(5,*)LR
      GOTO 100
C       SELECT OUTPUT DATA FILE NAME
275   PRINT *,'ENTER DATA FILE NAME'
      READ(*,*)FNAME
      OPEN(10,FILE=FNAME)
      GOTO 100
C     TRAINING PHASE OF NETWORK
300   PRINT *,'ENTER NUMBER OF ITERATIONS (0 TO STOP)'
      READ(*,*)COUNT
C     COMMENCE ITERATIONS
C     SELECT RANDOM INITIAL CONDITIONS WITHIN BOUNDARIES
C     OF -.1<OMEGA<.1, AND -.1 < THETA < .1.
20    INP(1,1) = (UNIF(IX)-.5)*.2
      INP(2,1) = (UNIF(IX)-.5)*.2
C       ESTIMATE TIME FROM INITIAL CONDITIONS
      TMAX = TIMEST(INP(1,1),INP(2,1))/.01
C       PERFORM SIMULATION FROM TIME ZERO TO TIME MAXIMUM
      DO 25 T = 0,TMAX
       C = SIGN(1.0,NEURALNET(INP))
       CALL CER(INP(1,1),INP(2,1),C)
25    CONTINUE
      COUNT = COUNT-1
      TRIAL = TRIAL + 1
C       COMPUTE DESIRED CONTROL SIGNAL
      DC = -1.0*(DX(TMAX)*INP(1,1) + DXD(TMAX)*INP(2,1))
C     PRINT *,'INPUT,OUTPUT,DESIRED',INP(1,1),INP(2,1),C,DES
```

```
C     PRINT *,'ERROR',E1
      E = -1.0*(SIGN(1.0,DC) - NEURALNET(INP))
      CALL TRAINER(E,INP)
      IF(MOD(TRIAL,10).NE. 0) GOTO 20
C     COMPUTE THE ERROR SURFACE BY COMPARING NEURAL NET OUTPUT
C     WITH THE ACUTAL SWITCHING CURVE
30    SSE1 = 0.0
      SSE2 = 0
      DO 40 I = 1,M
        DO 45 J = 1,N
          X(I,J) = FLOAT(I-11)/100.
          Y(I,J) = FLOAT(J-11)/100.
          INP(1,1) = X(I,J)
          INP(2,1) = Y(I,J)
          Z(I,J) = NEURALNET(INP)
          DES = TIMEOPT(X(I,J),Y(I,J))
          SSE1 = SSE1 + (DES - Z(I,J))**2
          ERR(I,J) = DES - SIGN(1.0,Z(I,J))
          SSE2 = SSE2 + ERR(I,J)**2.0
45      CONTINUE
40    CONTINUE
C       COMPUTE NORMS OF MATRICES
      DWN = NORMM(W,50,10,NHIDE-1,NINPUT)
      DVN = NORMM(V,50,50,NOUT,NHIDE)
      DZN = NORMV(ZW,50,NOUT)
C       WRITE DATA TO SCREEN AND OUTPUT FILE
50    PRINT 103,TRIAL,DWN,DVN,DZN,SSE1,SSE2
      WRITE(10,103)TRIAL,DWN,DVN,DZN,SSE1,SSE2
103   FORMAT(2X,I5,4(2X,G10.5),2X,I5)
C       TEST FOR BEST PERFORMING WEIGHTS
      IF(SSE1 .LT. LSSE) THEN
        CALL BESTNET(BW,BV,BZ)
        LSSE = SSE1
        QSSE = SSE2
        BTRIAL = TRIAL
      ENDIF
      IF (COUNT) 100,100,20
C     PLOT NEURAL NET OUTPUT
400   CALL ROTATE(X,Y,Z,M,N)
      GOTO 100
C     PLOT ERROR SURFACE
500   CALL ROTATE(X,Y,ERR,M,N)
      GOTO 100
C       PLOT NEURAL NETWORK OUTPUT CONTOURS
600   CALL CONTOUR(X,Y,Z,M,N,5)
      GOTO 100
C       TIME SIMULATION OF CER WITH NETWORK AND OPTIMAL CONTROL
700   CALL SIMUL
      GOTO 100
C       SAVE BEST PERFORMING WEIGHTS
800   CALL NETSAVE(BW,BV,BZ)
```

```
      PRINT *,'BEST PERFORMANCE AT TRIAL ',BTRIAL
      PRINT *,'WITH ERRORS ',LSSE,QSSE
      PAUSE ' '
      GOTO 100
900   CLOSE(10)
      STOP
      END
C       INCLUDE THESE FILES AT COMPILATION
$INCLUDE:'UTIL.FOR'
$INCLUDE:'NETWORK3.FOR'
$INCLUDE:'TRAINER3.FOR'
$INCLUDE:'PLOT.FOR'
$INCLUDE:'SIMUL.FOR'
$INCLUDE:'PLANT.FOR'
```

```
C    FILE NETWORK3.FOR
C    AUTHOR: C. M. SEGURA
C    DATE:  9 OCTOBER 1989
C    SYSTEM: IBM PC AT
C    COMPILER: MICROSOFT FORTRAN 4.0
C    REVISED: 4 DECEMBER 1989
C
C    THIS FILE CONTAINS NEURAL NETWORK ROUTINES USED TO CREATE
C    AND OPERATE NEURAL NETWORKS.
C    3RD CONFIGURATION VERSION
C
C    ***********************************************
C    VARIABLE DEFINITIONS
C
C    NINPUT     NUMBER OF INPUT NODES
C    NHIDE      NUMBER OF 1ST HIDDEN LAYER NEURONS
C    NOUT       NUMBER OF 2ND HIDDEN LAYER NEURONS
C    INP()      INPUT VECTOR
C    W( , )     INPUT TO 1ST HIDDEN LAYER WEIGHTS
C    HNET()     ACTIVATION VALUE OF 1ST HIDDEN LAYER NEURONS
C    HF( , )    OUTPUT VALUES OF 1ST HIDDEN LAYER NEURONS
C    V( , )     1ST HIDDEN TO 2ND HIDDEN LAYER WEIGHTS
C    ONET()     2ND HIDDEN LAYER ACTIVATION VALUES
C    OF( , )    OUTPUT VALUES OF 2ND HIDDEN LAYER NEURONS
C    Z()        2ND HIDDEN LAYER TO OUTPUT NEURON WEIGHTS
C    FNET       OUTPUT NEURON ACTIVATION VALUE
C    DOT        DOT PRODUCT FUNCTION
C    LR         LEARNING RATE
C
C    ***********************************************
     REAL FUNCTION SIGMOID(NET)
C    THIS FUNCTION EVALUATES THE SIGMOID TRANSFER CHARACTERISTIC
C
C    SIGMOID    VALUE OF NEURON OUTPUT
C    NET        INPUT SIGNAL TO NEURON
C    NEURALNET  OUTPUT OF NEURAL NETWORK
C
     REAL NET
     SIGMOID = 2.0/(1.0 + EXP(-NET)) - 1.0
     RETURN
     END
C    *********
C    THIS FUNCTION EVALUATES THE OUTPUT OF THE NEURAL
C    NETWORK.
C
     REAL FUNCTION NEURALNET(INP)
     COMMON/WEIGHT/W,V,Z,HF,HNET,ONET,OF,LR,FNET
     COMMON/SIZE/NINPUT,NHIDE,NOUT
     REAL W(50,10),V(50,50),HNET(50,1),HF(50,1),LR,ONET(50),DOT,
    1INP(10,1),Z(1,50),OF(50,1),FNET
     INTEGER NHIDE,NOUT,NINPUT
```

```
C    COMPUTE HIDDEN LAYER INPUTS AND OUTPUTS
     DO 10 I = 1,NHIDE-1
      HNET(I,1) = DOT(W,50,10,INP,10,1,I,1)
      HF(I,1) = SIGMOID(HNET(I,1))
10   CONTINUE
     HF(NHIDE,1) = 1.0
     NEURALNET = 0.0
C    COMPUTE 2ND HIDDEN LAYER INPUTS AND OUTPUTS
     DO 30 I = 1,NOUT-1
       ONET(I) =DOT(V,50,50,HF,50,1,I,1)
       OF(I,1) = SIGMOID(ONET(I))
30   CONTINUE
     OF(NOUT,1) = 1.0
C    COMPUTE OUTPUT VALUE
     FNET = DOT(Z,1,50,OF,50,1,1,1)
     NEURALNET = SIGMOID(FNET)
     RETURN
     END
C    *********
C    THIS SUBROUTINE INITIALIZES THE NEURAL NET WEIGHTS TO RANDOM
C    VALUES OR READS THE WEIGHTS FROM A SPECIFIED FILE.
C
C    FNAME              NEURAL NETWORK FILE NAME VARIABLE
C    RNG               RANGE OF RANDOM WEIGHT ASSIGNMENTS
C    IHR,IMIM,ISEC,IHUN   REAL TIME CLOCK FOR RANDOM SEED
C    UNIF              UNIFORM RANDOM NUMBER GENERATOR
C    IX                RANDOM NUMBER SEED
C    CHOICE              USER DECISION VARIABLE
C
     SUBROUTINE NETINIT(FNAME)
     COMMON /WEIGHT/W,V,Z,HF,HNET,ONET,OF,LR,FNET
     COMMON /SIZE/NINPUT,NHIDE,NOUT
     REAL W(50,10),V(50,50),HF(50,1),HNET(50,1),ONET(50),LR,RNG,
    1Z(1,50),OF(50,1)
     INTEGER NINPUT,NHIDE,NOUT,IHR,IMIN,ISEC,IHUN,CHOICE
     INTEGER*4 IX
     CHARACTER*14 FNAME
     CALL GETTIM(IHR,IMIN,ISEC,IHUN)
     IX = 1000*IHR + 100*IMIN + 10*ISEC + IHUN
     PRINT *,'ENTER 1 FOR SAVED WEIGHTS OR 2 FOR RANDOM WEIGHTS'
     READ(5,*)CHOICE
     IF (CHOICE .EQ. 1) THEN
C    READ SAVED WEIGHTS FROM A FILE
      PRINT *,'ENTER WEIGHT FILE NAME'
      READ(5,*)FNAME
      OPEN(2,FILE = FNAME)
      READ(2,*)NINPUT,NHIDE,NOUT
      DO 10 I = 1,NHIDE-1
          READ(2,*)(W(I,J),J=1,NINPUT)
C          PRINT *,'I,W(I,J)',I,(W(I,J),J=1,NINPUT)
10    CONTINUE
```

```
      DO 15 I = 1,NOUT-1
         READ(2,*)(V(I,J),J=1,NHIDE)
C         PRINT *,'I,V(I,J)',I,(V(I,J),J=1,NHIDE)
15     CONTINUE
      READ(2,*)(Z(1,I),I=1,NOUT)
      CLOSE(2)
     ELSE
C     INITIALIZE WITH RANDOM WEIGHTS
      FNAME = 'RANDOM'
      PRINT *,'ENTER THE NETWORK CONFIGURATION # INPUT, # HIDDEN,'
      PRINT *,'AND # OUTPUT NEURONS'
      READ(5,*)NINPUT,NHIDE,NOUT
      PRINT *,'ENTER THE RANGE OF WEIGHTS (-R,+R)'
      READ(5,*)RNG
      DO 20 I = 1,NHIDE-1
        DO 25 J = 1,NINPUT
          W(I,J) = (UNIF(IX)-.5)*RNG*2.0
25      CONTINUE
C     PRINT *,'W(I,J)',I,(W(I,J),J=1,NINPUT)
20     CONTINUE
      DO 30 I=1,NOUT-1
        DO 35 J=1,NHIDE
          V(I,J) = (UNIF(IX)-.5)*RNG*2.0
35      CONTINUE
C       PRINT *,'V(I,J)',I,(V(I,J),J=1,NHIDE)
30     CONTINUE
     DO 40 I = 1,NOUT
      Z(1,I) = (UNIF(IX)-.5)*RNG*2.0
40   CONTINUE
     ENDIF
     RETURN
     END
C     *********
C     THIS SUBROUTINE SAVES THE WEIGHTS FROM A NEURAL NETWORK IN A C
USER SUPPLIED FILENAME
C
     SUBROUTINE NETSAVE(W,V,Z)
     COMMON /SIZE/NINPUT,NHIDE,NOUT
     REAL W(50,10),V(50,50),Z(1,50)
     CHARACTER*14 FNAME
       PRINT*,'ENTER DATA FILE NAME'
       READ(*,*)FNAME
       OPEN(3,FILE = FNAME)
       WRITE(3,*)NINPUT,NHIDE,NOUT
       DO 10 I=1,NHIDE-1
         WRITE(3,*)(W(I,J),J=1,NINPUT)
10      CONTINUE
       DO 20 I = 1,NOUT-1
         WRITE(3,*)(V(I,J),J=1,NHIDE)
20      CONTINUE
       WRITE(3,*)(Z(1,I),I=1,NOUT)
```

```
      CLOSE(3)
    RETURN
    END
C   *********
C   THIS SUBROUTINE SAVES THE OPTIMUM WEIGHTS
C
C   BW      BEST W WEIGHTS
C   BV      BEST V WEIGHTS
C   BZ      BEST Z WEIGHTS
C
    SUBROUTINE BESTNET(BW,BV,BZ)
    COMMON /SIZE/NINPUT,NHIDE,NOUT
    COMMON /WEIGHT/W,V,Z,HF,HNET,ONET,OF,LR,FNET
    REAL W(50,10),V(50,50),Z(1,50),HF(50,1),HNET(50,1),ONET(50),LR,
   1OF(50,1),BW(50,10),BV(50,50),BZ(50),FNET
    CHARACTER*14 FNAME
      DO 10 I=1,NHIDE-1
        DO 15 J = 1,NINPUT
          BW(I,J) = W(I,J)
15       CONTINUE
10     CONTINUE
      DO 20 I = 1,NOUT-1
        DO 25 J = 1,NHIDE
          BV(I,J) = V(I,J)
25       CONTINUE
20     CONTINUE
      DO 30 I = 1,NOUT
        BZ(I) = Z(1,I)
30     CONTINUE
    RETURN
    END
```

```
C     FILE TRAINER3.FOR
C     AUTHOR: C. M. SEGURA
C     DATE: 9 OCTOBER 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOFT FORTRAN 4.0
C     REVISED: 9 OCTOBER 1989
C
C     THIS FILE HOLDS THE ROUTINES THAT IMPLEMENT ERROR BACK
C     PROPAGATION FOR A NEURAL NETWORK.
C     3RD CONFIGURATION VERSION
C
C     *********
C     THIS SUBROUTINE IMPLEMENTS THE ERROR BACK-PROPAGATION
C     METHOD OF TRAINING.
C
C       VARIABLE DEFINITIONS
C
C       DELEC       DERIVATIVE OF ERROR WRT OUTPUT (C)
C       DELF        BACK-PROPAGATED OUTPUT ERROR (THROUGH
C                  OUTPUT NEURON)
C       DZ          CHANGE IN Z WEIGHTS
C       DELTAO()    DERIVATIVE ACROSS 2ND HIDDEN LAYER NEURONS
C       DV          CHANGE IN V WEIGHTS
C       DELCH       DERIVATIVE OF ERROR BACKED TO 1ST HIDDEN LAYER
C     DELTAH        DERIVATIVE ACROSS 1ST HIDDEN LAYER NEURONS
C       DW          CHANGE IN W WEIGHTS
C       LR          LEARNING RATE
C
      SUBROUTINE TRAINER(DELEC,INP)
      COMMON/WEIGHT/W,V,Z,HF,HNET,ONET,OF,LR,FNET
      COMMON/SIZE/NINPUT,NHIDE,NOUT
      REAL DELEC,DELTAO(50),DELTAH(50),LR,W(50,10),V(50,50),HF(50,1),
     1DV(50,50),INP(10,1),ONET(50),HNET(50,1),Z(1,50),OF(50,1),
     2DZ(50),DW(50,10),DELCH(50),FNET,DELF
      INTEGER I
C     COMPUTE Z COEFFICIENT CHANGES AND PROPAGATED ERROR
      DELF = DELEC*SIGDER(FNET)
      DO 10 I = 1,NOUT-1
        DZ(I) = DELF*OF(I,1)*LR
        DELTAO(I)= SIGDER(ONET(I))
10    CONTINUE
      DZ(NOUT) = DELF*OF(NOUT,1)*LR
C       PRINT *,'DZ',DZ(I)
C     COMPUTE V COEFFICIENT CHANGES
      DO 20 I = 1,NOUT-1
        DO 25 J = 1,NHIDE
          DV(I,J) = DELF*DELTAO(I)*Z(1,I)*LR*HF(J,1)
C         PRINT *,'DV',DV(I,J)
25      CONTINUE
20    CONTINUE
C     COMPUTE W COEFFICIENT CHANGES
```

```
      DO 30 I=1,NHIDE-1
        DELCH(I) = 0.0
        DO 35 K = 1,NOUT-1
         DELCH(I) = DELCH(I)+V(K,I)*Z(1,K)*DELTAO(K)
35      CONTINUE
        DELTAH(I) = SIGDER(HNET(I,1))
C       PRINT *,'DELTAH',DELTAH(I)
        DO 40 J = 1,NINPUT
          DW(I,J) = DELF*INP(J,1)*DELCH(I)*DELTAH(I)*LR
C         PRINT *,'W,I,J',W(I,J),I,J
40      CONTINUE
30    CONTINUE
C     UPDATE Z MATRIX WEIGHTS
      DO 50 I = 1,NOUT
        Z(1,I)= Z(1,I) - DZ(I)
50    CONTINUE
C     UPDATE V MATRIX WEIGHTS
      DO 60 I = 1,NOUT-1
        DO 65 J = 1,NHIDE
          V(I,J) = V(I,J) -DV(I,J)
C         PRINT *,'V(I,J)',V(I,J)
65      CONTINUE
60    CONTINUE
C     UPDATE W MATRIX WEIGHTS
      DO 70 I = 1,NHIDE-1
        DO 75 J = 1,NINPUT
         W(I,J) = W(I,J) - DW(I,J)
75      CONTINUE
70    CONTINUE
      RETURN
      END
C     *********
      REAL FUNCTION SIGDER(NET)
C     THIS FUNCTION EVALUATES THE DERIVATIVE OF THE SIGMOID
      REAL NET
C     ALPHA = 1.0
      SIGDER = 2.0*EXP(-NET)/(1.0 + EXP(-NET))**2
      RETURN
      END
```

```
C    FILE PLANT.FOR
C    AUTHOR: C. M. SEGURA
C    DATE: 28 AUGUST 1989
C    SYSTEM: IBM PC AT
C    COMPILER: MICROSOFT FORTRAN 4.0
C    REVISED: 4 OCTOBER 1989
C
C    THIS FILE CONTAINS THE PLANT DYNAMICS AND ASSORTED ROUTINES
C    REQUIRED TO SIMULATE THE CREW/EQUIPMENT RETRIEVER.
C
C    ***********************************************
C    VARIABLE DEFINITIONS
C
C    X          INPUT VALUE OF STATE THETA
C    XDOT       INPUT VALUE OF STATE OMEGA
C    ***********************************************
C
C    THIS FUNCTION IMPLEMENTS THE TIME OPTIMAL CONTROL LAW
C
C    S          VALUE OF SWITCHING FUNCTION
C    TIMEOPT    THRUSTER CONTROL SIGNAL
C
     REAL FUNCTION TIMEOPT(X,XDOT)
     S = X + XDOT*ABS(XDOT)/.32976
     IF(S .LT. 0.0) TIMEOPT = 1.0
     IF(S .EQ. 0.0) TIMEOPT = 0.0
     IF(S .GT. 0.0) TIMEOPT = -1.0
     RETURN
     END
C    *********
C    THIS SUBROUTINE CALCULATES THE NEXT STATE OF THE SYSTEM
C    GIVEN THE PRESENT STATE AND THE CONTROL INPUT
C
C    U       CONTROL SIGNAL
C
     SUBROUTINE CER(X,XDOT,U)
     REAL X,XDOT,U
     X = X + .01*XDOT + 8.245E-6*U
     XDOT = XDOT + 1.6488E-3*U
     RETURN
     END
C    *********
C    THIS FUNCTION ESTIMATES THE TIME REQUIRED TO TRAVERSE THE
C    MINIMUM TIME TRAJECTORY
C
C    TIMEST     VALUE OF TIME
C    A          TORQUE DIVIDED BY MOMENT OF INERTIA
     REAL FUNCTION TIMEST(X,XDOT)
     REAL A,X,XDOT
     A = .16488
     TIMEST = SIGN(1.0,X)*XDOT/A +2.0*SQRT(ABS(X/A)+.5*(XDOT/A)**2)
```

```
RETURN
END
```

```
C     FILE PLANT2.FOR
C     AUTHOR: C. M. SEGURA
C     DATE: 10 OCTOBER 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOFT FORTRAN 4.0
C     REVISED: 10 OCTOBER 1989
C     THIS FILE CONTAINS THE PLANT DYNAMICS AND ASSORTED ROUTINES
C     REQUIRED TO SIMULATE THE CREW/EQUIPMENT RETRIEVER.
C       DOUBLE MASS CER
C
C       *********
C       TIME OPTIMAL CONTROL LAW
      REAL FUNCTION TIMEOPT(X,XDOT)
      S = X + XDOT*ABS(XDOT)/(.16488)
      IF(S .LT. 0.0) TIMEOPT = 1.0
      IF(S .EQ. 0.0) TIMEOPT = 0.0
      IF(S .GT. 0.0) TIMEOPT = -1.0
      RETURN
      END
C      *********
      SUBROUTINE CER(X,XDOT,U)
      REAL X,XDOT,U
      X = X + .01*XDOT + 4.122e-6*U
      XDOT = XDOT + 8.244E-4*U
      RETURN
      END
C       *********
      REAL FUNCTION PLANTDER(X,XDOT)
      PLANTDER = 4.122E-6*X + 8.244E-4*XDOT
      RETURN
      END
C       *********
      REAL FUNCTION TIMEST(X,XDOT)
      REAL A,X,XDOT
      A = .08244
      TIMEST = SIGN(1.0,X)*XDOT/A +2.0*SQRT(ABS(X/A)+.5*(XDOT/A)**2)
      RETURN
      END
```

```
c    FILE PLOT.FOR
C    AUTHOR: C. M. SEGURA
C    DATE:  17 AUGUST 1989
C    SYSTEM: IBM PC AT
C    COMPILER: MICROSOFT FORTRAN 4.0
C    REVISED: 5 DECEMBER 1989
C
C    THIS FILE CONTAINS FORTRAN SUBROUTINES USED TO AUTO-
C    SCALE AND PLOT DATA AS 3-D SURFACES, 2-D TRAJECTORIES AND
C    2-D CONTOUR PLOTS. ALL SUBROUTINES BEGINNING WITH 'Q' ARE
C    FROM THE GRAFMATIC PLOTTING LIBRARY
C
C    ************************************************
C    VARIABLE DEFINITIONS
C    X( , )    X DIRECTION VALUES
C    Y( , )    Y DIRECTION VALUES
C    Z( , )    SURFACE TO BE PLOTTED
C    THETA     PLOT ROTATION ANGLE
C    PHI       PLOT ROTATION ANGLE
C    XMIN,XMAX  RANGE OF X VALUES
C    YMIN,YMAX  RANGE OF Y VALUES
C    ZMIN,ZMAX  RANGE OF Z VALUES
C    PMIN,PMAX  ROTATED PLOT RANGE
C    QMIN,QMAX  ROTATED PLOT RANGE
C    **********
     SUBROUTINE ROTATE(X,Y,Z,M,N)
     REAL X(21,21),Y(21,21),Z(21,21),PMIN,PMAX,QMIN,QMAX,PHI,THETA
     INTEGER M,N
     XMIN = X(1,1)
     YMIN = Y(1,1)
     ZMIN = Z(1,1)
     XMAX = XMIN
     YMAX = YMIN
     ZMAX = ZMIN
     PHI =-45.0
     THETA = 5.0
     DO 10 I = 1,M
       DO 15 J = i,N
         IF (X(I,J) .LT. XMIN) XMIN = X(I,J)
         IF (Y(I,J) .LT. YMIN) YMIN = Y(I,J)
         IF (Z(I,J) .LT. ZMIN) ZMIN = Z(I,J)
         IF (X(I,J) .GT. XMAX) XMAX = X(I,J)
         IF (Y(I,J) .GT. YMAX) YMAX = Y(I,J)
         IF (Z(I,J) .GT. ZMAX) ZMAX = Z(I,J)
15     CONTINUE
10   CONTINUE
     ZSTEP = 1.0
     XSTEP = .1
     YSTEP = .1
     IF ((ZMAX-ZMIN).GT. 4.0) ZSTEP = 2.0
     IF ((XMAX-XMIN).GT. 1.0) XSTEP = .5
```

```
      IF ((YMAX-YMIN).GT. 1.0) YSTEP = .5
      ZMAX = FLOAT(INT(ZMAX)+1)
      ZMIN = FLOAT(INT(ZMIN)-1)
C     SET SCREEN MODE TO CGA,4-COLOR
20    CALL QSMODE(4)
C     ROTATE DATA AND ESTABLISH SCREEN WINDOW
      CALL Q3DROT(X,Y,Z,M,N,PHI,THETA)
      CALL Q3DWIN(XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,PMIN,PMAX,QMIN,QMAX)
      IOPT = 0
      YOVERX = (QMAX-QMIN)/(PMAX-PMIN)
      XORG = XMIN
      YORG = YMIN
C     SET PLOTTING PARAMETERS
      CALL QPLOT(50,300,30,170,PMIN,PMAX,QMIN,QMAX,
     1XORG,YORG,IOPT,YOVERX,1.5)
C     PLOT X, Y, AND Z AXES THEN PLOT ROTATED SURFACE
C     Q3DINV INVERTS ROTATION TO PREPARE FOR NEXT PLOT
40    CALL Q3DXAX(XMIN,XMAX,XSTEP,1,1,1,YMIN,YMAX,ZMIN,1.0)
      CALL Q3DYAX(YMIN,YMAX,YSTEP,1,1,1,XMIN,XMAX,ZMIN,1.0)
      CALL Q3DZAX(ZMIN,ZMAX,ZSTEP,1,-1,1,XMIN,YMIN,1.0)
      CALL Q3DFIL(X,Y,M,N,2,1)
      CALL Q3DINV(X,Y,Z,M,N)
      PAUSE ' '
      CALL QSMODE(2)
      PRINT *,'ENTER NEW ANGLES PHI & THETA (DEG) OR (-999,0) TO QUIT'
      READ(5,*)PHI,THETA
      IF (PHI .NE. -999.) GOTO 20
      RETURN
      END
C     *********
C     CONTOUR PLOTTING SUBROUTINE
C
C     NUMCON    NUMBER OF CONTOURS
C     IDEF      CONTOUR VALUE FLAG
C     XA,YA     2-D RANGE OF DATA
C
C
      SUBROUTINE CONTOUR(X,Y,Z,M,N,NUMCON)
      REAL X(M,N),Y(M,N),Z(M,N),XSCALE,VALUES(10),LBL(10)
      DIMENSION XA(100),YA(100)
      INTEGER NUMCON,IDEF
      XSCALE = 1.5
      IDEF = 0
      DO 10 I = 1,M
        XA(I) = X(I,1)
10    CONTINUE
      DO 20 J = 1,N
        YA(J) = Y(1,J)
20    CONTINUE
      ZMIN = Z(1,1)
      ZMAX = ZMIN
```

```
      DO 30 I = 1,M
        DO 35 J = 1,N
          IF(Z(I,J) .LT. ZMIN) ZMIN = Z(I,J)
          IF(Z(I,J) .GT. ZMAX) ZMAX = Z(I,J)
35      CONTINUE
30    CONTINUE
C     COMPUTE CONTOUR SPACING
      DELZ = (ZMAX-ZMIN)/(NUMCON+1)
      DO 40 I = 1,NUMCON
        VALUES(I) = ZMIN + I*DELZ
        LBL(I) = 1.0
40    CONTINUE
C     SET SCREEN MODE TO CGA HIGH RESOLUTION (B/W) AND
C     PLOT DATA WITH AXES
      CALL QSMODE(6)
      CALL QCNTOU(XSCALE,XA,YA,Z,VALUES,LBL,M,N,NUMCON,IDEF)
      CALL QXAXIS(XA(1),XA(M),.1,1,-1,1)
      CALL QYAXIS(YA(1),YA(N),.1,1,-1,1)
      PAUSE ' '
      CALL QSMODE(2)
      RETURN
      END
```

```
C     FILE SIMUL.FOR
C     THIS PROGRAM SIMULATES THE CREW/EQUIPMENT RETRIEVER WITH
C     TIME OPTIMAL CONTROL LAW. CER DYNAMICS ARE IN SUBROUTINE
C     CER. CONTROL LAW IS IN FUNCTION TIMEOPT(). ALSO PLOTTED IS
C     THE TRAJECTORY CONTROLLED BY A NEURAL NETWORK.
C     AUTHOR: C. M. SEGURA
C     DATE:  28 AUGUST 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOFT FORTRAN 4.0
C     REVISED: 1 OCTOBER 1989
C
      SUBROUTINE SIMUL
      REAL X(2),U,Y(3),THETA,OMEGA,DT,BX(5),BY(5),
     1XC(270),XCD(270),XN(270),XND(270),NEURALNET,TIMEOPT
      INTEGER TMAX
      DATA BX/-.0218,-.0218,.0218,.0218,-.0218/
      DATA BY/-.00087,.00087,.00087,-.00087,-.00087/
      DT = .01
C     INITIAL CONDITIONS
5     PRINT *,'ENTER INITIAL CONDITIONS, THETA AND OMEGA, -9 TO QUIT'
      READ(5,*)THETA,OMEGA
      IF(THETA .EQ. -9.0) RETURN
      X(1) = THETA
      X(2) = OMEGA
      Y(1) = THETA
      Y(2) = OMEGA
      Y(3) = 1.0
      XC(1) = X(1)
      XCD(1) = X(2)
      XN(1) = Y(1)
      XND(1) = Y(2)
      TMAX = TIMEST(THETA,OMEGA)/.01 + 1
      DO 10 I = 2,TMAX
        U = TIMEOPT(X(1),X(2))
         CALL CER(X(1),X(2),U)
        C = SIGN(1.0,NEURALNET(Y))
         CALL CER(Y(1),Y(2),C)
        XC(I) = X(1)
        XCD(I) = X(2)
        XN(I) = Y(1)
        XND(I) = Y(2)
10    CONTINUE
C     PLOTTING STATEMENTS
      XMIN = -.1
      XMAX = .1
      YMIN = -.1
      YMAX = .1
      IOPT = 0
      CALL QSMODE(4)
      JCOL1 = 50
      JCOL2 = 250
```

```
JROW1 = 10
JROW2 = 170
YOVERX = 1.0
ASPECT = 1.5
CALL QPLOT(JCOL1,JCOL2,JROW1,JROW2,XMIN,XMAX,YMIN,YMAX,
10.0,0.0,IOPT,YOVERX,ASPECT)
CALL QXAXIS(XMIN,XMAX,.05,1,0,0)
CALL QYAXIS(YMIN,YMAX,.05,1,0,0)
CALL QPTXTB(5,'THETA',3)
CALL QPTXTC(5,'OMEGA',3)
CALL QSETUP(0,1,-2,1)
CALL QPTXT(1,'C',1,0,24)
CALL QTABL(1,TMAX,XC,XCD)
CALL QSETUP(0,2,-2,2)
CALL QPTXT(1,'N',2,0,23)
CALL QTABL(1,TMAX,XN,XND)
CALL QSETUP(0,2,-2,3)
CALL QTABL(1,5,BX,BY)
PAUSE ' '
CALL QSMODE(2)
GOTO 5
RETURN
END
```

```
C     FILE UTIL.FOR
C     AUTHOR: C. M. SEGURA
C     DATE: 17 AUG 1989
C     SYSTEM: IBM PC AT
C     COMPILER: MICROSOSFT FORTRAN 4.0
C     REVISED: 1 OCTOBER 1989
C
C     THIS FILE CONTAINS UTILITY PROGRAMS NEEDED TO RUN NEURAL C NETS
C
C     *********
      REAL FUNCTION UNIF(IX)
C      PORTABLE RANDOM NUMBER GENERATOR USING THE RECURSION:
C      IX = 16807*IX MOD(2**31-1)
C      USING ONLY 32 BITS, INCLUDING SIGN.
C      INPUT IX = INTEGER GREATER THAN 0 LESS THAN 2147483647
C      OUTPUT IX NEW VALUE
C           UNIF = RANDOM REAL NUMBER BETWEEN 0 AND 1
C      Adapted from: A GUIDE TO SIMULATION by Bratley, Fox, and
C                 Schrage; Springer-Verlag, New York, 1983. pg 319
      INTEGER*4 IX,KI
      KI = IX/127773
      IX = 16807*(IX-KI*127773)-KI*2836
      IF(IX .LT. 0) IX = IX+2147483647
      UNIF = IX*4.656612875E-10
      RETURN
      END
C      *********
C        COMPUTES DOT PRODUCT OF TWO VECTORS
      REAL FUNCTION DOT(A,AR,AC,B,BR,BC,ROW,COL)
      INTEGER AR,AC,BR,BC,ROW,COL
      REAL A(AR,AC),B(BR,BC)
      DOT = 0.0
      DO 10 I = 1,AC
       DOT = DOT + A(ROW,I)*B(I,COL)
10     CONTINUE
      RETURN
      END
C     *********
C     THIS FUNCTION COMPUTES THE NORM OF A VECTOR
      REAL FUNCTION NORMV(A,AR,L)
      INTEGER AR,L,I
      REAL A(AR)
      NORMV = 0.0
      DO 10 I = 1,L
      NORMV = NORMV + A(I)**2
10    CONTINUE
      NORMV = SQRT(NORMV)
      RETURN
      END
C     *********
C     THIS FUNCTION COMPUTES THE NORM (COLUMN-WISE) OF A MATRIX
```

```
      REAL FUNCTION NORMM(A,AR,AC,L,W)
      INTEGER AR,AC,W,L
      REAL A(AR,AC)
      NORMM = 0.0
      DO 10 I = 1,W
        DO 20 J = 1,L
          NORMM = NORMM + A(J,I)**2
20      CONTINUE
10    CONTINUE
      NORMM = SQRT(NORMM)
      RETURN
      END
```

This is the format of a network weight file. The first line indicates the network configuration as three inputs, 10 first hidden layer neurons, and five second hidden layer neurons. The single output neuron is assumed.

Following the configuration data is a list of the weights for each of the three weight matrices. The subroutine NETINIT uses the configuration data to partition the remaining information in the file into the three weight matrices.

```
     3         10          5
  -13.906620      -2.940447    4.978316E-02
   -7.214338   -6.484410E-01    3.074306E-02
   -1.656613    5.949540E-01   -1.231645E-02
 4.014902E-01    2.472410E-01    1.685382E-02
    5.252939       1.217185   -1.283479E-02
    2.707932       1.246534    5.219876E-03
   -8.908709      -1.938318    3.536274E-02
   -5.286425      -1.283191    8.932211E-03
   -2.405496   -7.347796E-01    2.773683E-02
 2.907092E-02   -8.334237E-01    5.596985E-01       -1.078840
 2.512622E-01    1.562133E-01    3.469862E-02    8.748630E-01
-2.293951E-01    6.681408E-02
   -3.158125      -1.767610       -1.202942    5.020352E-01
    1.237375    9.950187E-01       -2.616258   -4.080161E-01
 6.180473E-02   -6.841267E-02
   -1.185558    4.164372E-01    5.117882E-01   -3.655882E-01
 3.128358E-01    7.593271E-01    6.716492E-01   -1.220128E-01
 4.027111E-01   -5.646229E-02
    3.999902       1.952387   -2.373382E-01    1.558341E-01
   -1.491720   -4.875921E-01       2.197183       2.230891
    1.262611    7.936431E-02
 9.830183E-01    9.827302E-01    9.823033E-01    9.816523E-01
 9.806274E-01
```

LIST OF REFERENCES

[1] McDonnell Douglas Astronautics Co., Best and Final Offer: Response to Questions and RFP Amendment 7, Book 2, September 1987.

[2] Daniel L. Hansen, "Modeling, Simulation, and Analysis of Attitude Control for the Crew/Equipment Retriever (CER) Proposed for Space Station," Masters Thesis, Naval Postgraduate School, April 1989.

[3] Alan Lapedes and Robert Farber, "How Neural Nets Work," Los Alamos National Laboratory report LA-UR-88-0418, February 1988.

[4] David E. Rumelhart and James L. McClelland, Editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations, Chapter 8, MIT Press, 1986.

[5] Demetri Psaltis, Athanasios Sideris, and Alan A. Yamamura, "A Multilayered Neural Network Controller," IEEE Control Systems, April 1988, pp. 17-21.

[6] Derrick Nguyen and Dr. Bernard Widrow, "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," IJCNN-89 Conference Record, July 1989.

[7] Private Communication from Professor Jeff B. Burl, Naval Postgraduate School, to the author, October 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Jeff B. Burl, Code 62Bl Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor Roberto Cristi, Code 62Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Rudolph Panholzer, Code 62Pz Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Professor Harold A. Titus, Code 62Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8.	Professor Ari Feuer, Code 62Fe Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
9.	LT Clement M. Segura, USN 5541 Quarterpath Gate Virginia Beach, Virginia 23455	3

10. Weapons Engineering Office, Code 33 — 1
Naval Postgraduate School
Monterey, California 93943

11. P. J. Duke — 1
MS 17-6
McDonnell Douglas Space Systems Co.
5301 Bolsa Avenue
Huntington Beach, CA 92647-2048