NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ARTIFICIAL NEURAL NETWORK MODELING
OF DAMAGED AIRCRAFT

by

Clifford A. Brunger

March, 1994

Thesis Advisor: Daniel J. Collins

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
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): AA/PL	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): Artificial Neural Network Modeling of Damaged Aircraft

12. PERSONAL AUTHOR(S): Clifford A. Brunger

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 01/92 TO 03/94	March 1994	109

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 Network, Parallel Processors

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

Aircraft design and control techniques rely on the proper modeling of the aircraft's equations of motion. Many of the variables used in these equations are aerodynamic coefficients which are obtained from scale models in wind tunnel tests. In order to model *damaged* aircraft, every aerodynamic coefficient must be determined for every possible damage mechanism in every flight condition. Designing a controller for a damaged aircraft is particularly burdensome because knowledge of the effect of each damage mechanism on the model is required before the controller can be designed. Also, a monitoring system must be employed to decide when and how much damage has occurred in order to re configure the controller. Recent advances in artificial intelligence have made parallel distributed processors (artificial neural networks) feasible. Modeled on the human brain, the artificial neural network's strength lies in its ability to generalize from a given model. This thesis examines the robustness of the artificial neural network as a model for damaged aircraft.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Daniel J. Collins	22b. TELEPHONE (Include Area Code): (408) 656 - 2311; 22c. OFFICE SYMBOL: AA/CO

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 Unclassified

Approved for public release; distribution is unlimited.

Artificial Neural Network Modeling of Damaged Aircraft

by

Clifford A. Brunger
Lieutenant, United States Navy
B.S.E.E., United States Naval Academy, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

MARCH 1994

Author: _____
Clifford A. Brunger

Approved by: _____
Daniel J. Collins, Thesis Advisor

_____
Richard M. Howard, Second Reader

_____
Daniel J. Collins, Chairman
Department of Aeronautics and Astronautics

ABSTRACT

Aircraft design and control techniques rely on the proper modeling of the aircraft's equations of motion. Many of the variables used in these equations are aerodynamic coefficients which are obtained from scale models in wind tunnel tests. In order to model *damaged* aircraft, every aerodynamic coefficient must be determined for every possible damage mechanism in every flight condition. Designing a controller for a damaged aircraft is particularly burdensome because knowledge of the effect of each damage mechanism on the model is required before the controller can be designed. Also, a monitoring system must be employed to decide when and how much damage has occurred in order to re configure the controller. Recent advances in artificial intelligence have made parallel distributed processors (artificial neural networks) feasible. Modeled on the human brain, the artificial neural network's strength lies in its ability to generalize from a given model. This thesis examines the robustness of the artificial neural network as a model for damaged aircraft.

TABLE OF CONTENTS

To my beautiful wife Joie, who makes life enjoyable.

I. INTRODUCTION

Designing modern aircraft controllers requires advance knowledge of aircraft dynamics. In order to determine these dynamics, the aircraft is usually modeled. First, the equations of motion (based on Newton's second law) are derived. Neglecting higher order derivatives and making small angle approximations, these equations are linearized around a trim position. The aircraft's equations of motion can then be written in the following state space format:

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{Y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

where,

$\mathbf{x}(t)$ = state vector
$\mathbf{u}(t)$ = input vector
$\mathbf{Y}(t)$ = output vector

Although the previous steps can be carried out to a high degree of accuracy, the aerodynamicist's job is still daunting. Using wind tunnel tests on scale models, the numerous partial derivatives must be obtained in order to compute the state space matrices (A, B, C and D). Since a simple model of an aircraft is at least eighth order, and some models reach more than fiftieth order, this makes determining the partial derivatives an arduous task. Once the derivatives are

known, the transfer functions can be computed and, therefore, the static and dynamic response of the aircraft can be determined.

These responses, however, are valid only around the trimmed position. In order for the controller to function throughout the aircraft's entire flight regime, several other trim positions, or flight conditions, must be defined, the equations of motion must be linearized around these new trim positions, and the new aerodynamic derivatives must be determined or estimated. This results in additional A, B, C, and D matrices. The more trim positions selected, the more schedules or plants the controller has to pick from and, therefore, the more robust the controller.

A problem arises, however, when the aircraft is damaged. The controller is hard-wired with the modeled data, but that data is no longer valid. The aerodynamic derivatives of the aircraft have changed, but those used by controller have not. A solution to this problem is to make the controller more robust, i.e., include more trim positions and plant variations in its development. However, identifying every possible damage mechanism and its effect on every specific aerodynamic derivative is prohibitive. Also, the complexity of the controller and the amount of effort required to obtain all of the additional derivatives makes this method impractical.

Recently, a new method of modeling has become feasible: artificial neural networks. Artificial neural networks excel in generalizing and extrapolating. This could make them robust models, i.e., reducing the number of trim positions and damage mechanisms

required to be identified and properly modeled to define an aircraft's motion throughout its flight regime. Earlier work in this area has proved the validity of artificial neural network controllers [SCOT 89] [DROR 92]. This thesis will investigate the use of artificial neural networks to model damaged aircraft.

II. NEURAL NETWORK FUNDAMENTALS

The computer, with its ability to carry out millions of instructions per second, does many things amazingly well. Its strength lies in the ability of its microprocessor to carry out simple arithmetic at an extremely fast rate while maintaining a high degree of accuracy. The human brain is not as proficient at performing simple calculations. It excels in pattern recognition and the extrapolation of data from patterns, far surpassing the ability of computers. The idea behind the artificial neural network is to use the speed and accuracy of the computer to imitate some of the brain's functions. It is, therefore, not coincidental that artificial neural networks are patterned on a crude approximation of the human brain.

A. BIOLOGICAL NEURONS

The fundamental cellular unit of the human nervous system is the neuron. The brain is made up of trillions of interconnected neurons forming a neural network. As shown in Figure 2-1, a neuron receives signals from several other neurons via its dendrites or input tentacles. The neuron processes these inputs in the cell body and, if the combined sum of these signals is strong enough, it "fires," sending an output signal to other neurons' dendrites via its axon or output tentacle. This process is not as simple as it seems because the axon and dendrites do not connect; they intermingle at synaptic

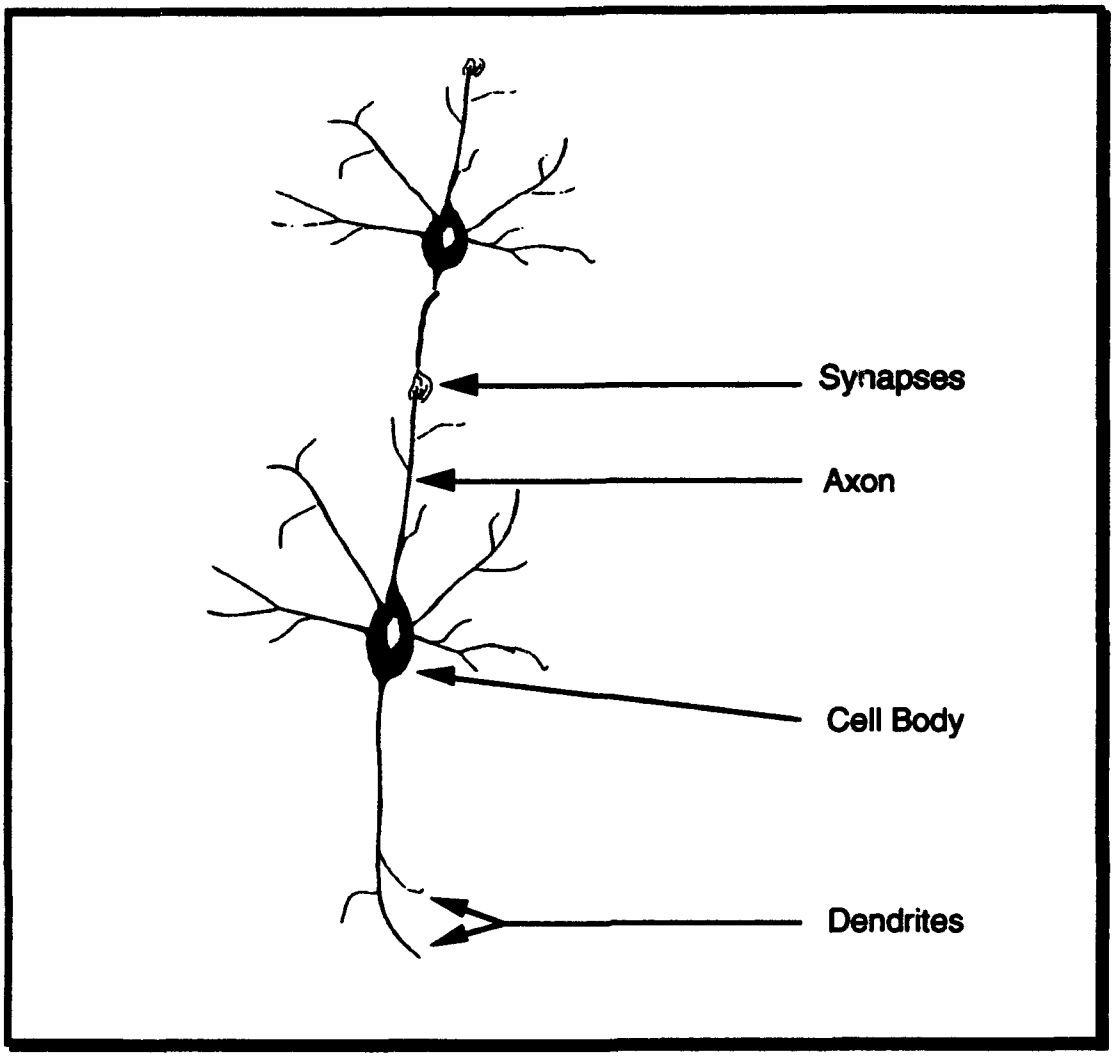


Figure 2.1 Human Neuron

junctions which have various strengths (or gains) depending on the gap size and the properties of the synaptic cleft material. Also, the signals sent are actually chemicals, of which more than thirty have been identified [WASS 89 p. 12]. Some chemicals excite the cell body to fire and some inhibit firing. As the chemicals (inputs from the dendrites) mix in the cell body, a certain level of activation is reached which causes the cell to fire. When a cell fires, it picks a

specific chemical and concentration to send out the axon, which is released it into the synaptic cleft for the next group of dendrites to pick up.

As a signal processing unit, the neuron is slow. A neuron must wait approximately one millisecond between firings. For comparison, a standard 33 MHz computer processes one byte of information every 30 nano seconds, thirty thousand times faster. The human nervous system gets its overall speed from the extreme parallelism of the neurons. It is estimated that in one meter of neural pathway there are 100 billion neurons, and more than 1000 trillion interconnections [WASS 89 p. 194]. Because of this parallel processing capability, the brain's neural network can recognize patterns, learn from experience, see through noise and distortion, generalize from previous examples, self-adjust, and abstract essential characteristics much faster and much better than a computer. In an effort to get computers to be able to do these things, artificial neural networks have been developed.

B. PROCESSING ELEMENTS

Figure 2.2 shows the processing element, the fundamental component of an artificial neural network. Based on the design of the human neuron, the processing element has inputs (X), weights (W), a summer (Σ), an activation function (F) and an output (Y). Mathematically:

$$Y=F(W \cdot X)$$

where,

Y = output (scalar)
F = activation function
X = input (vector)
W = gain (vector)

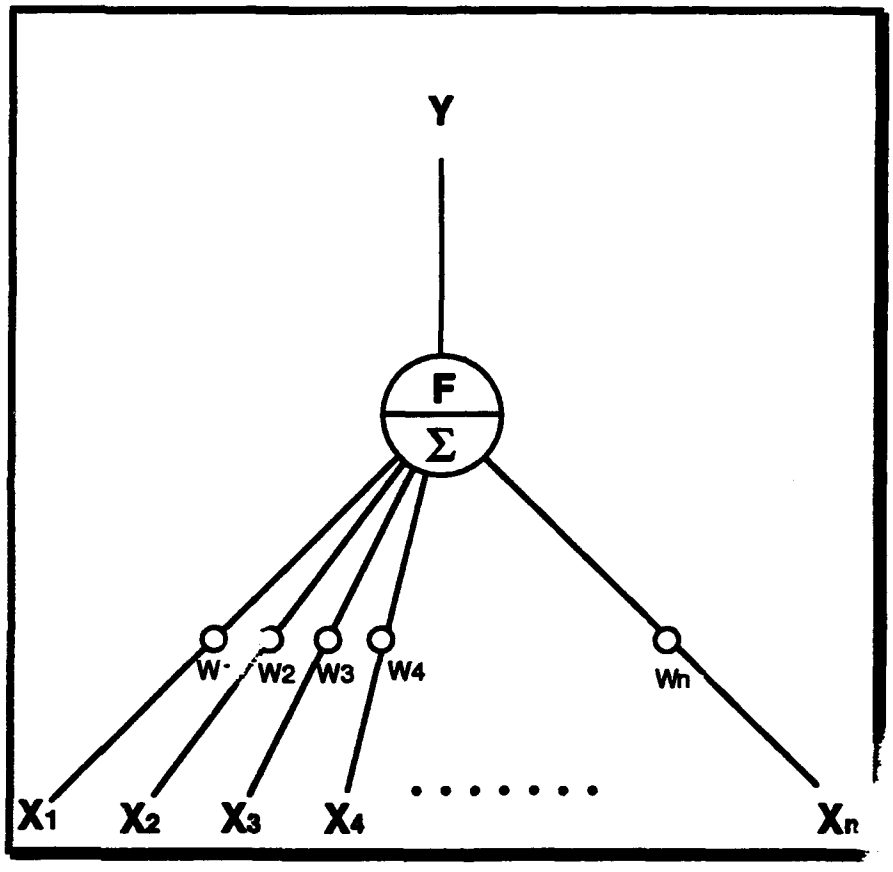


Figure 2.2 Processing Element

The input vector, **X**, is multiplied by a gain vector, **W**, to produce an activation level, **WX** (scalar). In other words, every input to the processing element has a unique gain associated with it and the sum of the products of the inputs with their respective gain is the

activation level. This level is applied to an activation function, F, to produce the output, Y. The activation function can be a threshold function, which fires at a certain level, or a continuous function. Although it can be linear, the threshold function is usually nonlinear. In fact, in order to achieve any advantage from using multiple layer networks, the threshold function must be non-linear, otherwise, multi-layer networks reduce to just two layers, defeating the purpose of creating multi-layer networks. [WASS 89 p. 19] Often, the activation function is a type of squashing function, i.e., the output of the processing element is never allowed to exceed a certain value. The two most prevalent types of squashing functions are the sigmoid and hyperbolic tangent:

$$F(x) = \frac{1}{1+e^{-x}} \qquad F(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \qquad \text{respectively,}$$

with derivatives:

$$F'(x) = \frac{e^{-x}}{(1+e^{-x})^{2}} \qquad F'(x) = 1-\left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right)^{2}, \quad \text{respectively,}$$

where, x = excitation level (**WX**).

These plots are shown in Figures 2.3 and 2.4. The sigmoid function is normally used when the input to the processing element is non polar (i.e., negative or positive values only), while the

hyperbolic tangent function should be used whenever the input is bipolar (both negative and positive values).

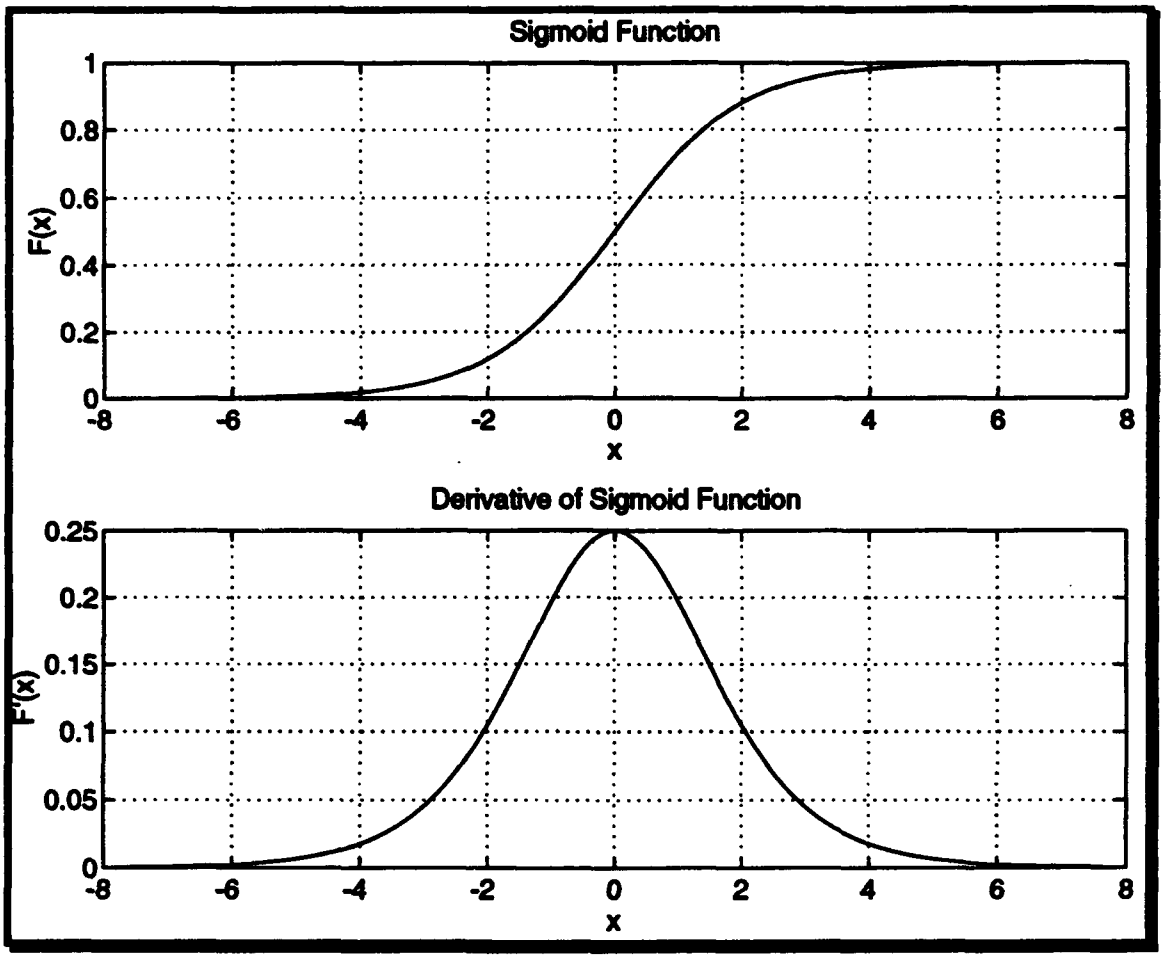


Figure 2.3 Sigmoid Function and Derivative

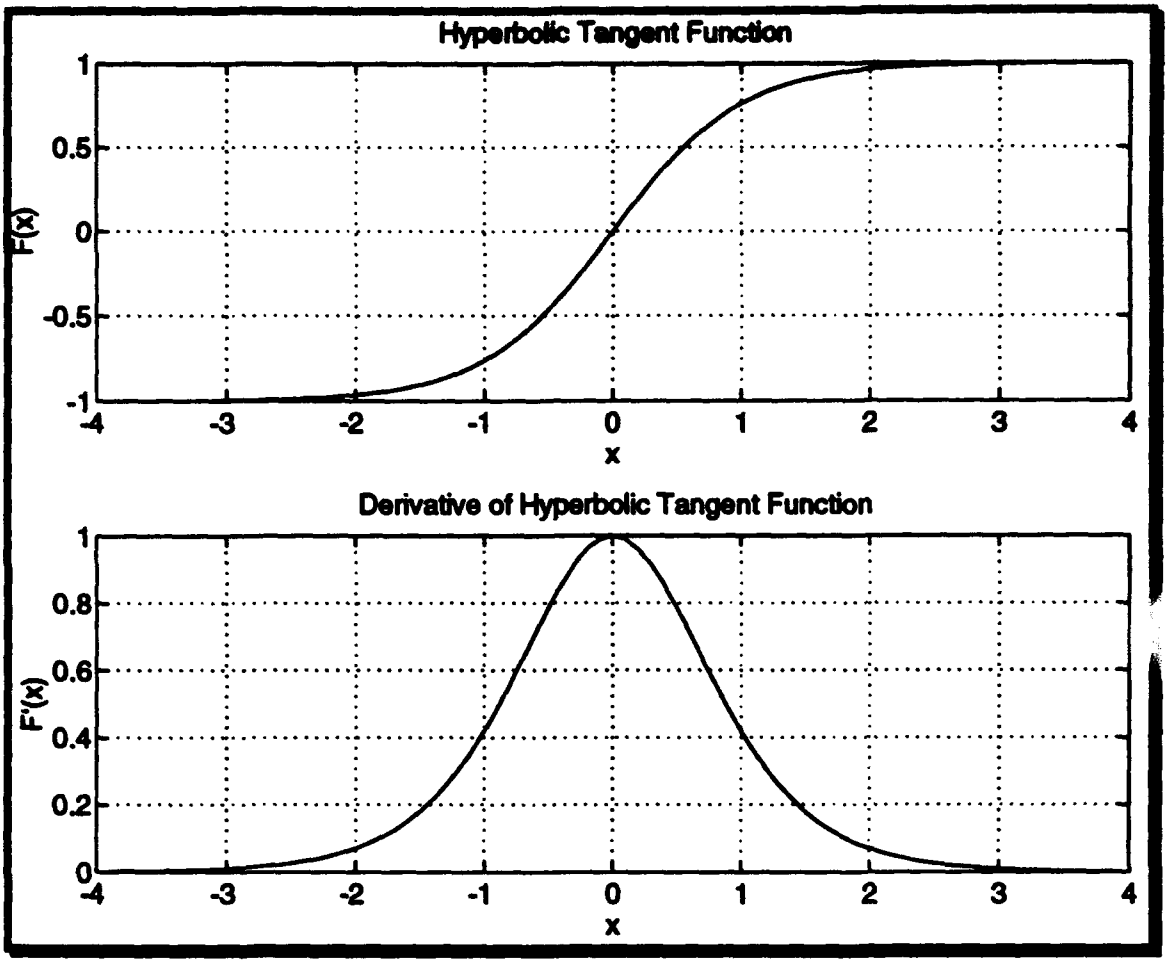


Figure 2.4 Hyperbolic Tangent Function and Derivative

Activation functions can be thought of as the nonlinear gain of the processing element, where the gain of the processing element is proportional to the slope, or derivative, of the activation function evaluated at the particular level of excitation. Since processing elements often deal with very small as well as very large inputs, using the sigmoid or the hyperbolic tangent as the squashing function, gives large gains for small signals and small gains for large signals. This was Grossberg's solution to his own dilemma. A

small signal required a large gain, but since processing elements were connected in series, if a fixed gain was chosen to properly amplify the small signals, then when the larger signals were applied, the downstream processing elements would saturate. In order for the network to respond equally well to both large and small input signals, a varying gain was needed. Grossberg found that these squashing functions provided a proper gain over a wide range of input levels, thus helping to prevent saturation of the artificial neural network. [WASS 89 p.19]

The sigmoid and hyperbolic squashing functions also possess an important quality; their derivatives can be expressed in terms of their original functions:

$$F'(x) = F(x) - F(x)^2 \quad \text{and} \quad F'(x) = 1 - F(x)^2, \quad \text{respectively.}$$

During back-propagation (discussed later), the rate of change, or derivative, of the output of every processing element in the artificial neural network has to be calculated and evaluated at the current excitation level of that processing element. Since the sigmoid and hyperbolic tangent functions have derivatives that can be expressed in terms of their original functions, the *derivative* of the output of the individual processing elements can be directly determined from their *current* outputs i.e., $Y'=Y-Y^2$ (sigmoid) and $Y'=1-Y^2$ (hyperbolic tangent) where Y is the output of the processing element.

C. ARTIFICIAL NEURAL NETWORKS

An artificial neural network consists of layers of processing elements joined together in series. In a fully connected system, the output of every processing element in one layer is connected to the input of every processing element in the next layer via an amplifying weight. Typically there is an input or buffer layer, one or two hidden layers, and an output layer. The hidden layer is so called because there is no direct access to it, i.e., it is not directly connected to the input or output. Figure 2.5 shows a typical artificial neural network with four processing elements in the input layer, eight processing elements in the hidden layer and two processing elements in the output layer. The output of every processing element in one layer is connected to the input of every processing element in the next layer, fully connected, and a weight, or gain, is associated with each connection. Feedback loops and bypass loops are possible as well. Note that a bias is shown. The purpose of the bias is to provide a trainable offset to the origin of the squashing function. This results in faster learning. [HECH 89 p. 53]

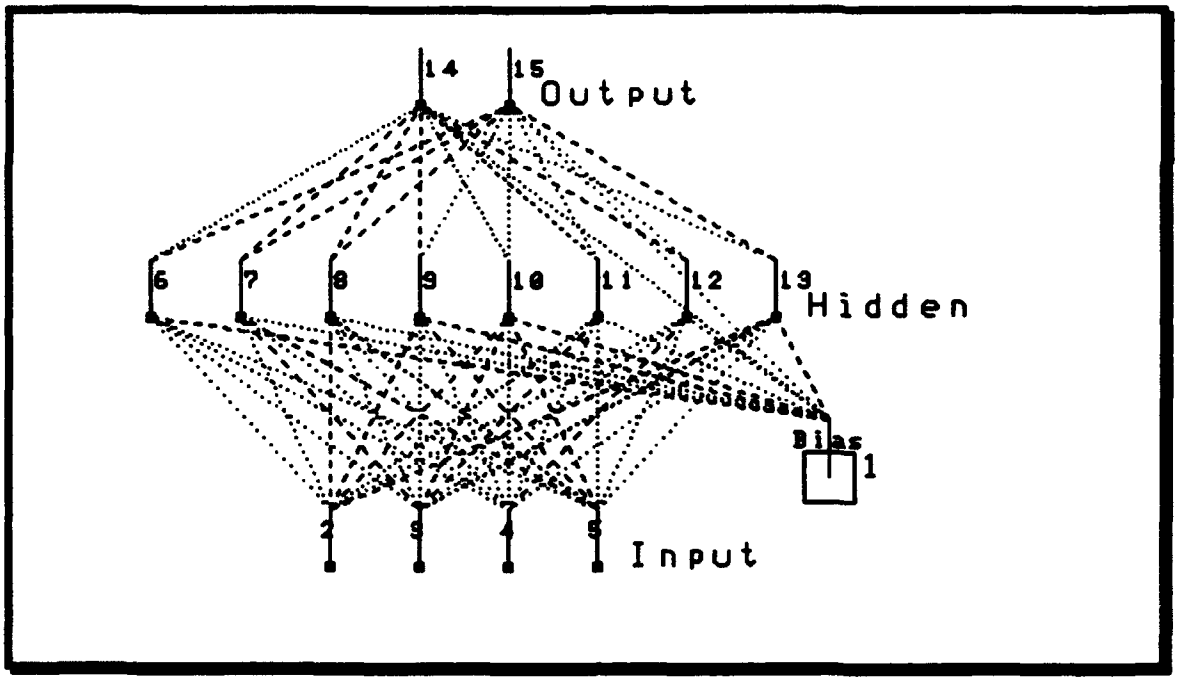
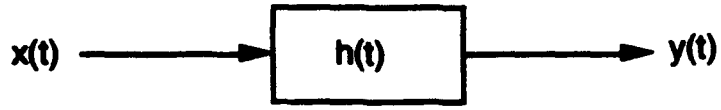


Figure 2.5 Artificial Neural Network

D. ARTIFICIAL NEURAL NETWORK OPERATION

An artificial neural network possesses no inherent knowledge. After the structure of the network is determined, the weights are initialized to random values as a precursor to the learning process. (If they were not randomly initialized, then all of the weights would be altered by the same amount and no learning would occur.) After learning, the network must be tested for proper operation.

In a linear time invariant system, an output is obtained by convolving the input with the impulse response of the system. In the frequency domain, the transfer function, H(s), is the ratio of the output to the input signal.



$$y(t) = x(t) \otimes h(t) \qquad \text{and} \qquad H(s) = \frac{Y(s)}{X(s)}$$

In comparison, a neural network is given an input and output sequence and attempts to learn the mathematical relationship or transfer function of the two sequences.

1. Learning

Learning is the process of modifying the weights of the connections between the processing elements such that the given input results in the desired output. There are three types of learning: supervised, unsupervised and reinforcement. Supervised learning, discussed below, was the type employed in this research

a. Supervised Learning

During supervised learning, an artificial neural network is provided with an input sequence along with the desired output sequence. If the input sequence is identical to the desired output sequence, then the network is auto-associative. If the input sequence is different from the desired output sequence, as it was in this research, then the network is hetero-associative.

When the first record of the input sequence is applied to the network's buffer layer, it computes an output based on the values of the randomized weight values and the network's configuration. This output is compared to the desired output and an

error signal is generated. Using this error signal, an algorithm alters the values of the weights to reduce the network's global error and then the cycle repeats. If the error between the network's output and the desired output goes to zero or an acceptable low level, as more training pairs are applied to the network, then the network has learned the mathematical function connecting the input data with the output data.

Usually the network is trained over a large number of desired input/output pairs in order to generate a range or frequency response to the data. For example, if the elevator of an aircraft is randomly excited and the airspeed is recorded simultaneously with elevator position, then the two measurements make up an input and desired output pair. If the data is recorded fast enough, inherent in these two sequences is the transfer function from elevator to airspeed. If the sequence is long enough, and the proper excitation was applied, then the transfer function of the aircraft's airspeed to elevator input can be determined. If,

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\delta_e(t)$$
$$\mathbf{Y}(t) = \mathbf{C}\mathbf{x}(t)$$

then, $\mathbf{Y}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\delta_e(s)$

where, $\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ is the transfer function from δ_e(s) to Y(s).

2. Testing

Testing is required to validate a neural network's learning. During testing the weights are fixed to those values obtained during the training phase, and another input sequence is applied to the network. The output of the neural network is compared to the desired output and the error is measured. If the error is sufficiently small, then the network has indeed learned the transfer function of the data.

Normally, the testing sequence is different from the learning sequence. This is done to test the network's ability to generalize to data upon which it has not actually trained. There have been many occasions that networks have adequately learned from the training sequences, but could not extrapolate to the test data. Since the training sequence is not a continuous function, and, since it is not infinitely long, it does not contain all of the possible combinations of input and desired output pairs. If the neural network has learned the mathematical relationship expessed in the training data, and the test data has expressed that same relationship, then the network should be able to determine the correct output associated with the test data. Since artificial neural networks are designed to generalize, if a network learns too well during training, it might not be able to generalize when tested on data it has not specifically learned. As a result, there is a trade off between learning and generalizing.

3. Learning Algorithm

The learning algorithm (or learning rule) determines the manner by which the weights are adjusted during learning. The best method for learning, i.e., reducing the network's output error to the smallest value, may not necessarily be the fastest. And with data sets hundreds of thousands of lines long, processing time is an important consideration. There is also the problem of global verses local minimum. The algorithm may find a weight matrix that minimizes the error, but the weight could be for a local minimum and not a global minimum. Presently, there is no known learning rule which can determine a weight matrix which guarantees a global error minimum for the a network based on the back-propagation technique.

E. BACK-PROPAGATION

There are many different artificial network types, each with several learning algorithms and each with specific advantages and disadvantages. Only the back-propagation technique and the delta rule algorithm, used in this research, will be discussed.

The back-propagation technique was discovered by David Rumelhart [RUME 86] and David Parker [PARK 85], separately, as a way of solving the credit assignment problem posed by Minsky and Papert in *Perceptrons* [MINS 69]. The perceptron was the first type of artificial neural network to gain wide spread importance. It was flawed because it was unable to solve problems that were not linearly separable. Complex multi-layer perceptrons could be built,

but, in order to solve non-linearly separable problems, a method of determining which processing element caused the error in the output needed to be determined. The back-propagation algorithm solves this problem by assuming that all of the processing elements are responsible for the error and, therefore, requires modification of all of the weights in the network. The error between the desired output and the network's computed output is fed back through the network layers, starting at the output layer, adjusting the weights, layer by layer, until the input layer is reached, hence the error is "back-propagated" through the network to adjust the weights.

The typical back-propagation network has an input layer, an output layer and one or more hidden layers. Although there is no limit on the number of hidden layers required for a network to properly learn a transfer function, there is some analysis which shows that only three hidden layers are required in order to solve complex classification problems [NEUR A 93 p. 63]

1. Generalized Delta Rule

Although back propagation is the method by which the weights in the network are modified, the specific values by which the weights are changed are determined by the learning rule. The learning rule employed in this research was the generalized delta rule. This rule requires that the learning be supervised.

Learning using the generalized delta rule can be broken down into four steps. The first is the forward pass. During this step, one record of the training sequence is applied to the network and, using

the networks configuration and initial weights, an output is generated. Next, the error between the network's output and the desired output is computed. This can be expressed in vector form as:

$$\mathbf{E} = \mathbf{Y}_d - \mathbf{Y}_a$$

where, $\mathbf{E}$ = error (column vector)
$\mathbf{Y}_d$ = desired output (column vector)
$\mathbf{Y}_a$ = actual output (column vector)

Multiplying this error by the derivative of the squashing function evaluated at the excitation level of the output layer gives a modified error or delta, for the output layer:

$$\delta_o = (\mathbf{I} * \mathbf{E}) * F'(\mathbf{W}_o \mathbf{X}_o)$$

where, δ_o = modified error (column vector)
$\mathbf{I}$ = identity matrix
$\mathbf{E}$ = error (column vector)
F' = derivative of the activation function (hyperbolic tangent)
$\mathbf{W}_o$ = output layer's weight matrix
$\mathbf{X}_o$ = input vector to output layer during forward pass

but, $F'(\mathbf{X}_o \mathbf{W}_o) = 1 - F(\mathbf{X}_o \mathbf{W}_o)^2$, for the hyperbolic tangent function

therefore, $\delta_o = (\mathbf{I} * \mathbf{E}) * [1 - F(\mathbf{X}_o \mathbf{W}_o)^2]$

The final step is the back propagation of this modified delta to generate the change in the weight matrix.

$$\Delta W_o = \eta_o * F(W_o X_o) * \delta_o^T$$

where, ΔW_o = change in output layer's weight matrix
η_o = learning rate coefficient of output layer (scalar)

To determine the weight adjustments for the hidden layer, the modified error, δ_o, is fed backwards through the weight matrix, in order to calculate a desired output for the hidden layer, $W_o^T \delta_o$. A modified delta for the hidden layer can then be computed,

$$\delta_h = (I * W_o^T \delta_o) * F'(W_h X_h)$$

where, δ_h = modified error of hidden layer
W_h = hidden layer's weight matrix

but, $F'(X_h W_h) = 1 - F(X_h W_h)^2$, for the hyperbolic tangent function

therefore, $\delta_h = (I * W_o^T \delta_o) * [1 - F(X_h W_h)^2]$

and the change in the weight matrix can then be determined as before:

$$\Delta W_h = \eta_h * F(W_h X_h) * \delta_h^T$$

where, ΔW_h = change in hidden layer's weight matrix

η_h = learning rate coefficient of the hidden layer (scalar)

This process is continued until the input layer is reached.

The learning rate coefficient, η, is a scaling factor to cause different layers to learn at different rates.

> ...using different learning coefficients for each layer in a multi-layer network can decrease learning time. In particular, having a larger learning [rate] coefficient at the hidden layer than for the output layer allows the hidden layer to form feature detectors during the early stages of training. ... With large learning rates, a network may go through large oscillations during training. In fact, if the rates are too large, the network may never settle or converge. Smaller rates tend to be more stable. [NEUR C 93 p. 54]

III. EXPERIMENTAL PROCEDURE

A. HARDWARE-SOFTWARE

This experiment was carried out on a Sun SPARC2 workstation (15 MIPS) with 64 megabytes of RAM and 2.2 gigabytes of hard disk space. The software used for the design, learning and testing of the neural network was NeuralWorks II (version 5.0). MATLAB (version 4.1) was used extensively to create data files and to correlate and plot output data.

Even though the Sun workstation is considered a fast computer, some data runs required over an hour to complete. Since the NeuralWorks program is attempting to create a parallel processor using a single micro-processor, it requires a significant amount of RAM. In order to reduce run time, data files were converted into binary and loaded into RAM by the NeuralWorks program. Also, since the data files for the neural network were ten megabytes in size, a large external storage capacity was needed. This validates the use of a UNIX machine with such a large RAM and disk storage capacity.

NeuralWorks is a software program which simulates a parallel processor (hardware). It is a design tool which can be used to determine the size and configuration of an artificial neural network and can be used to extensively test and modify the network design before the network is constructed. Since the program is using a single processor to model multiple parallel processors, the software

version of the network runs much slower than an actual parallel processor. The data from NeuralWorks can be downloaded in order to physically build the designed artificial neural network.

B. OVERVIEW

The artificial neural network's training and testing files were generated using MATLAB. This was accomplished by creating a random binary signal and using it to excite a digitized state space model of an A-4D. The output values of the state space model of the A-4D [u, alpha, q, and theta], were then organized and recorded in data files. Using NeuralWorks, the *training* data file was presented to the artificial neural network until satisfactory learning occurred. Next, a test file was generated in the same manner and was presented to verify the learning. The artificial neural network's output was recorded and MATLAB was used again to perform an analysis of the test data.

C. SETUP SPECIFICS

1. Modeling of A-4D Longitudinal Motion

The A-4D was chosen as the investigation platform for artificial neural network modeling. This aircraft is a US Navy attack aircraft weighting approximately 17,600 lbs. The aircraft can be modeled in state space as an eighth order system. This model can be linearly separated into two fourth order modes: longitudinal and lateral. After considering the complexity of the problem as well as the memory limitations of the available computer system and processing time, it was determined that an analysis of just the

longitudinal mode would be adequate. Using the data from *Aircraft Dynamics and Automatic Control*, [MCRU 93] flight condition 8 (400 kts, 35,000 ft) the state space representation of the A-4D longitudinal mode were created using MATLAB.

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{Y}(t) = \mathbf{C}\mathbf{x}(t)$$

where,

$$\mathbf{A} = \begin{bmatrix} X_u & \frac{X_\alpha}{u_o} & 0 & \frac{-g*\cos(\theta)}{u_o} \\ \frac{u_o * Z_u}{u_o - Z_{\dot{\alpha}}} & \frac{Z_\alpha}{u_o - Z_{\dot{\alpha}}} & \frac{u_o + Z_q}{u_o - Z_{\dot{\alpha}}} & \frac{-g*\cos(\theta)}{u_o - Z_{\dot{\alpha}}} \\ u_o * M_u + \frac{M_{\dot{\alpha}} * u_o * Zu}{u_o - Z_{\dot{\alpha}}} & M_\alpha + \frac{M_{\dot{\alpha}} * Z_\alpha}{u_o - Z_{\dot{\alpha}}} & Mq + \frac{M_{\dot{\alpha}}(u_o + Z_q)}{u_o - Z_{\dot{\alpha}}} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \frac{X_{\delta e}}{u_o} \\ \frac{Z_{\delta e}}{u_o - Z_{\dot{\alpha}}} \\ M_{\delta e} + \frac{M_{\dot{\alpha}} * Z_{\delta e}}{u_o - Z_{\dot{\alpha}}} \\ 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} \frac{u}{u_e} \\ \alpha \\ q \\ \theta \end{bmatrix}$$

$\frac{u}{u_e}$ airspeed pertubation

α angle of attack pertubation

q pitch rate pertubation

θ pitch angle pertubation

$\mathbf{U} = \delta e$ elevator deflection

The bode diagrams in Figure 3.1 show the dominance of the two primary modes of the A-4D longitudinal dynamics.

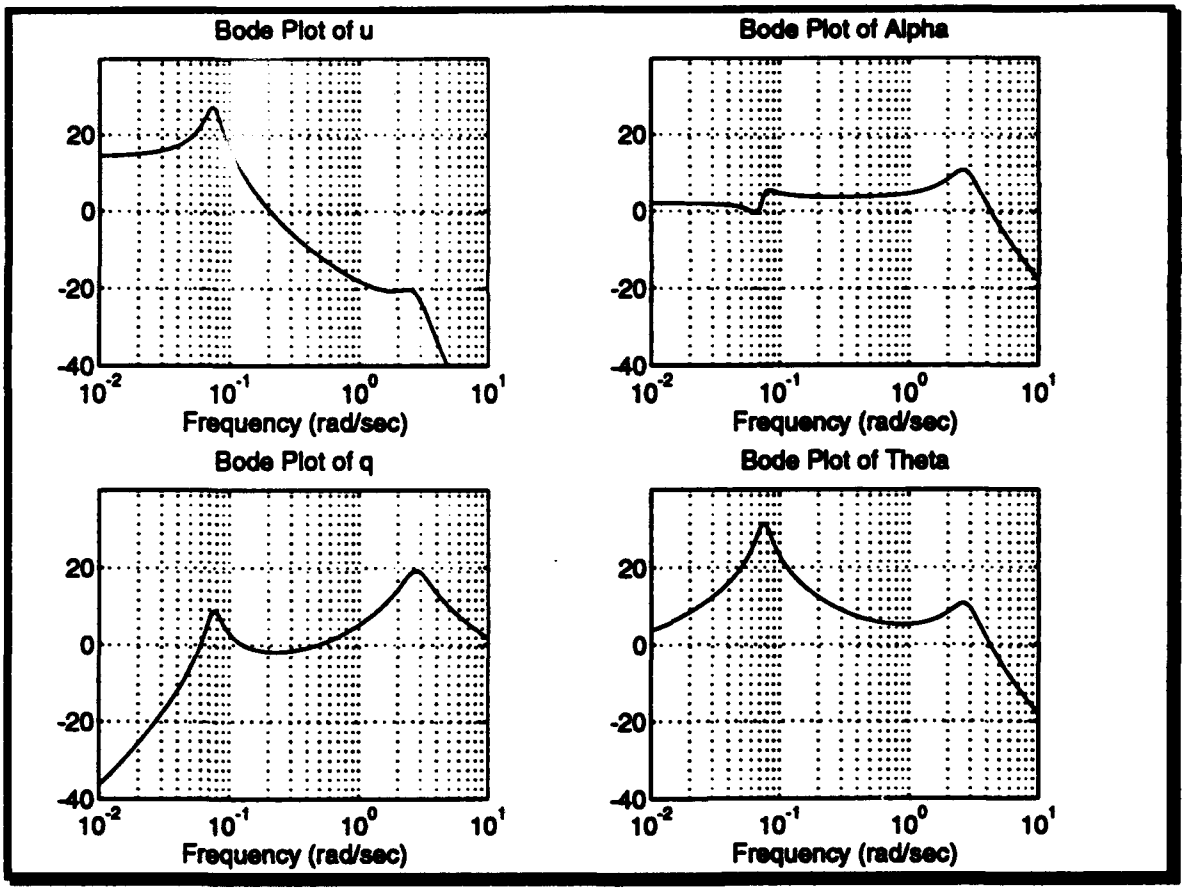


Figure 3.1 Bode Plot of A-4D

The short period mode occurs at a frequency of 2.77 rad/sec and the long period or phugoid mode occurs at 0.075 rad/sec. Low frequencies dominate the response in airspeed and pitch angle while high frequencies dominate the response in angle of attack and pitch rate. The eigenvalues of the plant are summarized in Table 3.1. Notice that all of the eigenvalues are stable. Although it is possible to model an unstable plant with an artificial neural network, doing so would only add another level of complexity to the artificial neural network model.

TABLE 3.1
EIGENVALUES OF A-4D

Eigenvalues	Damping	Freq. (rad/sec)	τ (sec)
-0.6245+2.6988i	0.2254	2.7701	2
-0.6245-2.6988i	0.2254	2.7701	2
-0.0088+0.0747i	0.1177	0.0752	83.5
-0.0088+0.0747i	0.1177	0.0752	83.5

Using MATLAB, the state space matrices were digitized with a sampling frequency of 50 Hz, 10 times the highest frequency of interest, two decades above the short period mode, thereby satisfying the Nyquist criteria. The digitized matrices were then inserted into the MATLAB Simulink diagram shown in Figure 3.2 in order to compute the output state values. The correlation between the analog and digital A-4 plant is shown in Figure 3.2. Notice that

the bode plots of the two plants are indistinguishable (they appear as a single line) and are, for all practical purposes, identical.

MATLAB files *A4_cond3.m*, *A4_matrix.m* and *A4_bode.m* in Appendix A were used to model the A-4D.

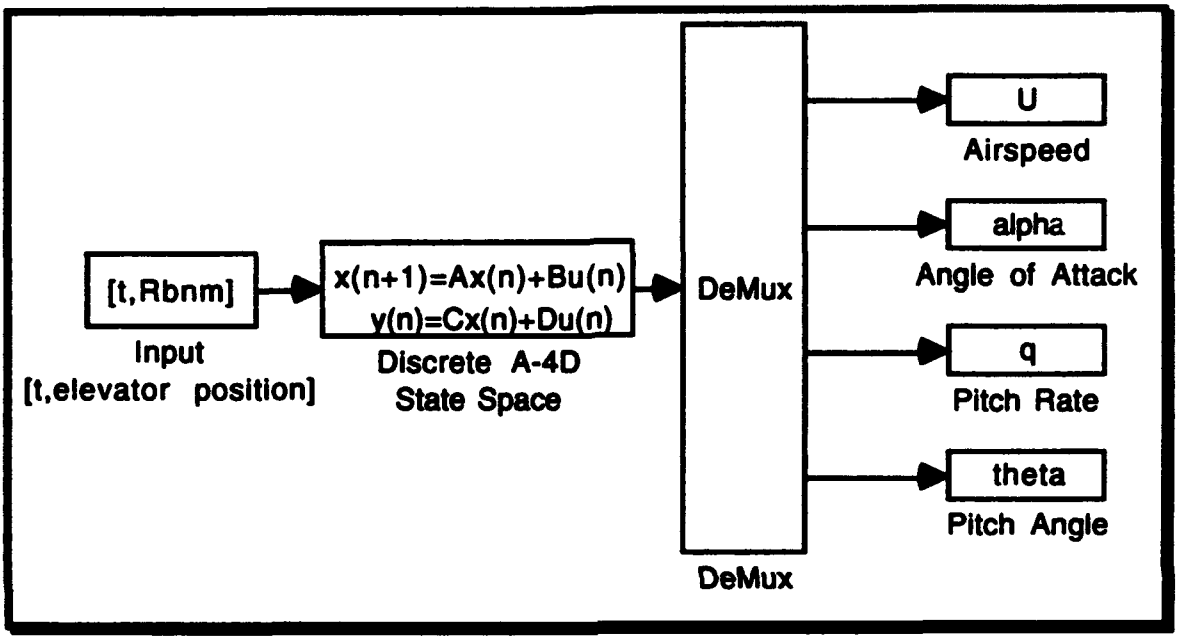


Figure 3.2 Simulink Representation of A-4D.

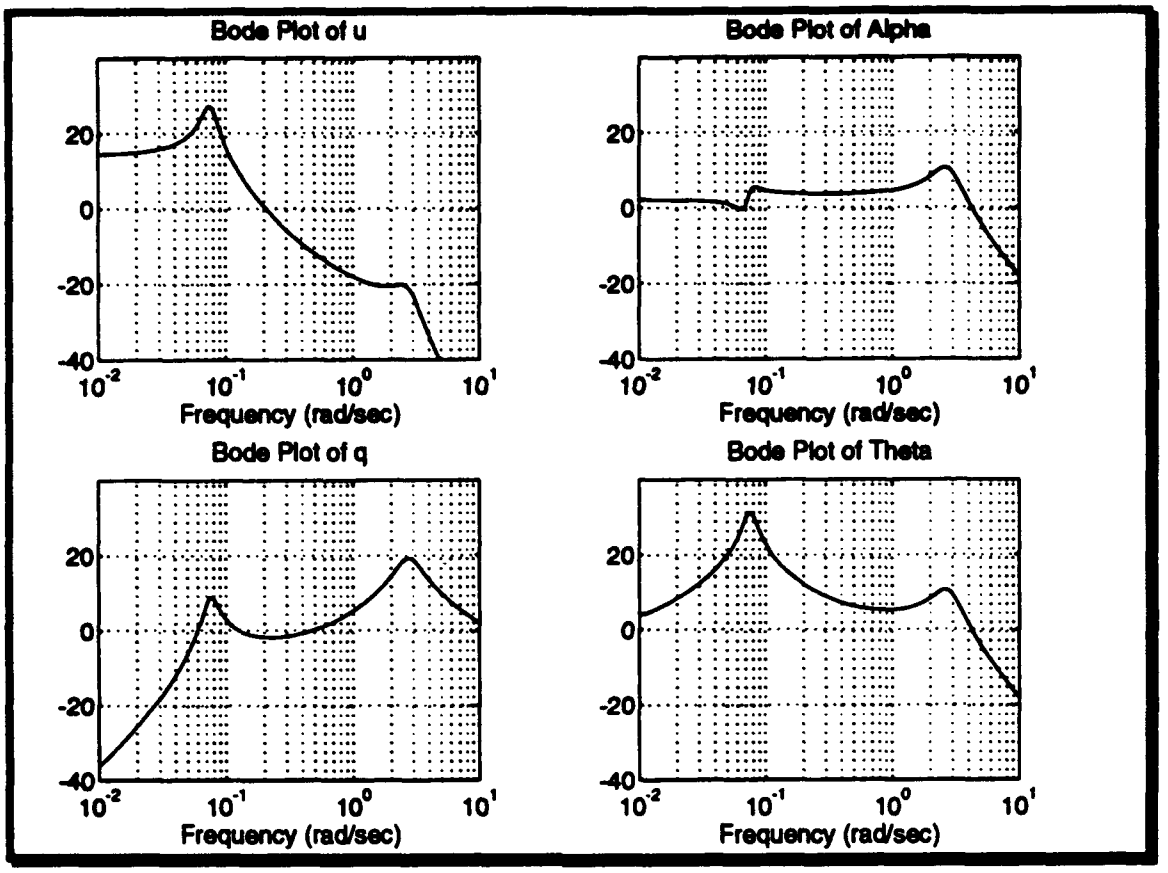


Figure 3.3 Comparison of Continuous and Digital Plant

2. Generation of Learn and Test Files

a. Random Binary Sequence

A random binary sequence, [+1,-1], was constructed to excite the digitized A-4D model. The primary consideration in its design was its power spectral density or frequency content. Since there is a frequency band of interest for the A-4D, the random binary's power spectral density was designed to match it. This keeps the energy of the random binary signal in the frequency band of interest and suppresses the aliasing from higher frequencies.

Since the short period of the A-4 occurs at 0.45 Hz and the phugoid mode occurs at 0.012 Hz, the random binary sequence was designed to have a frequency content from 0.001Hz to 5 Hz; i.e. from one decade above, to one decade below the two primary modes. The period of the lowest frequency was used to determine the time length of the sequence, i.e., 1000 seconds, one complete cycle of 0.001 Hz. Since a random binary sequence in the time domain (triangular), transforms to a sinc function, sin(x)/x, in the frequency domain, the frequency of the random binary sequence corresponds to the first cross-over of its sinc function. In order to get a bandwidth of 5 Hz, the cross-over was chosen to be twice that or 10 Hz.

Since the digitized model of the A-4D was created using a sampling rate of 50 Hz, an adjustment to the random binary signal was needed, i.e., instead of one random number every 0.1 seconds, the same number was repeated 5 times every 0.02 seconds and then a new one was selected. This maintained the same designed random binary frequency and power spectrum. The first three seconds of the 1000 second long random binary signal are shown in Figure 3.4, and the power spectral density plot of the entire sequence is shown in Figure 3.5.

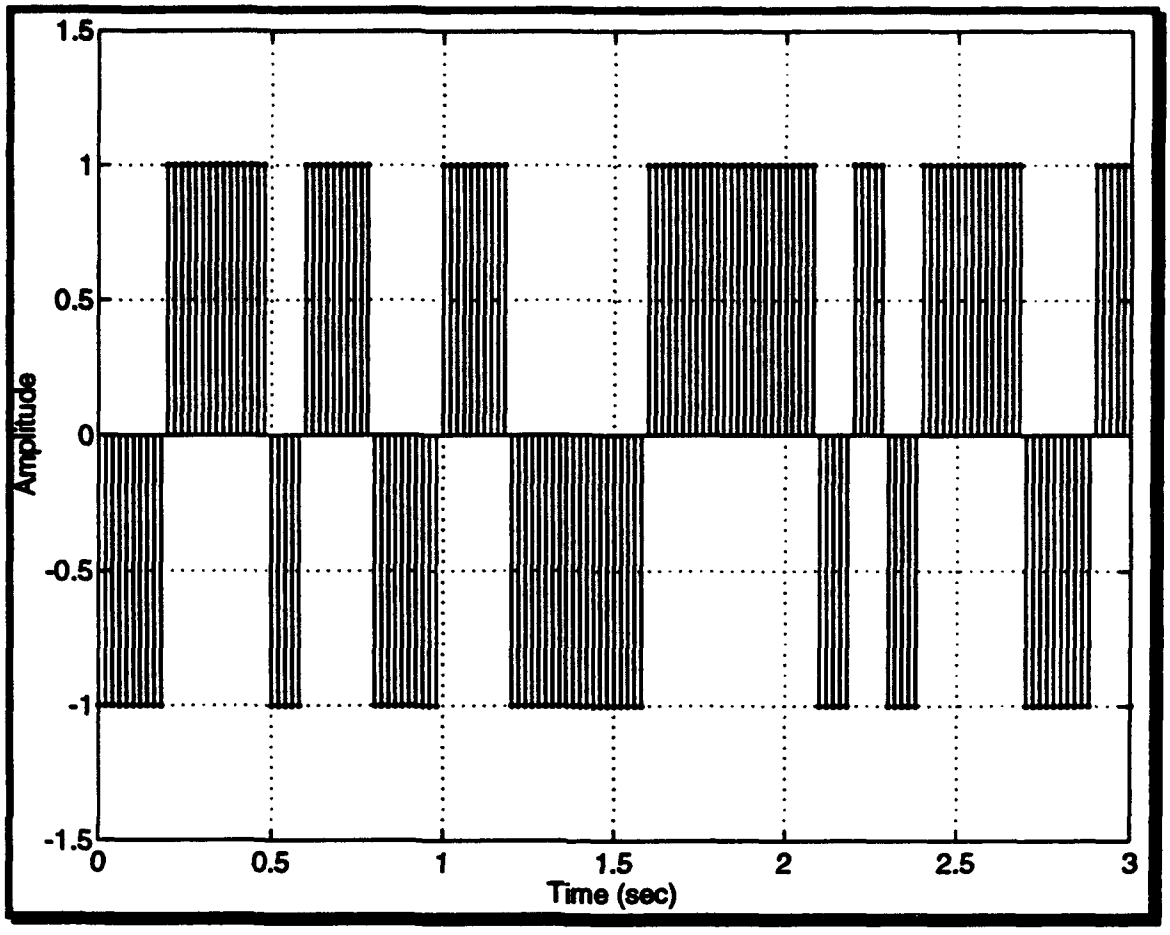


Figure 3.4 Random Binary Sequence

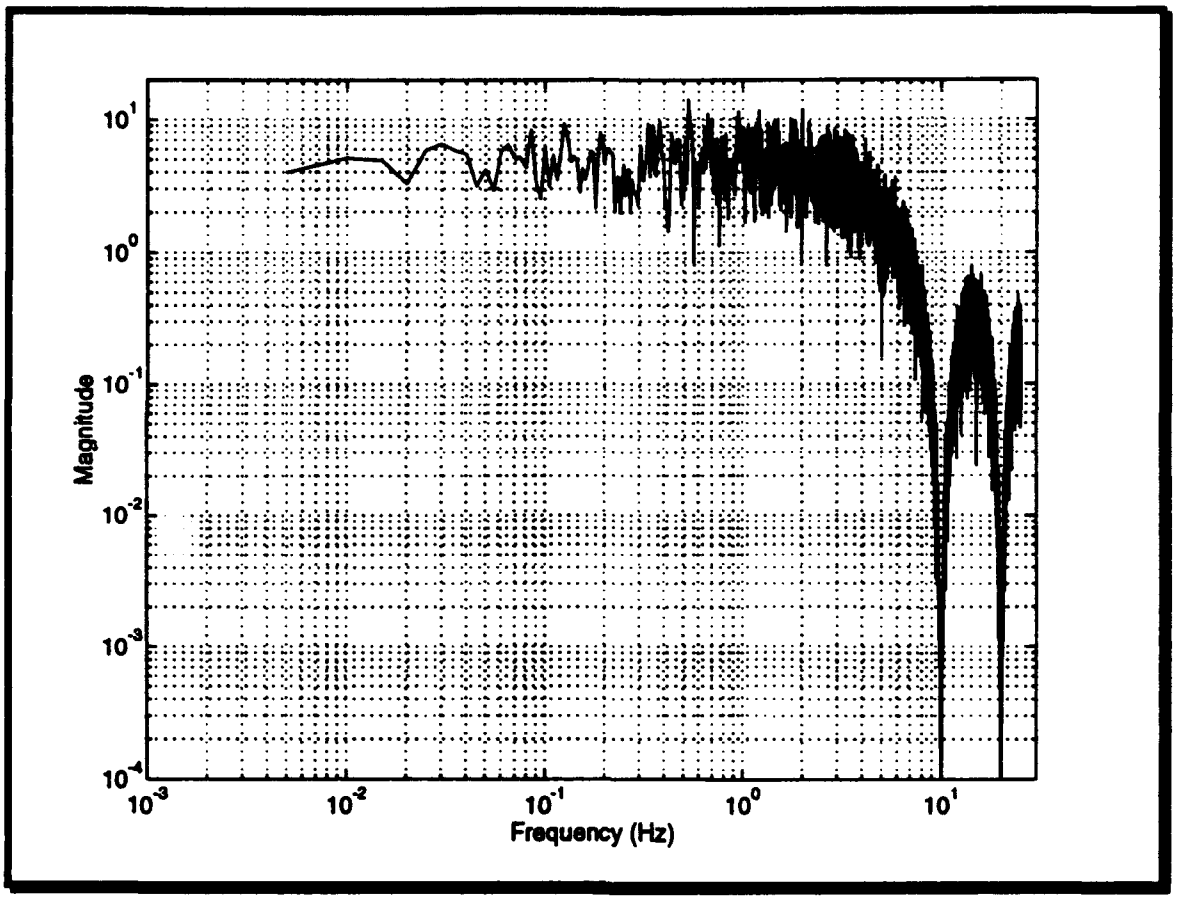


Figure 3.5 Power Spectral Density of Random Binary Sequence

Notice that the energy in the signal rolls off (-3dB) at 5 Hz and the first cross-over frequency is at 10 Hz. There are two other notable observations to be gained from the power spectral density plot. First, the plot is not smooth like a sinc function, but rather noisy. This is primarily due to the signal not being a continuous function but also due to the randomness of the signal and the size of the Hamming window (25000) selected for the power spectral analysis. Secondly, there in no defined DC (zero frequency) gain. This is because the sequence is finite in length (time).

Since this signal will be eventually applied to a neural network, it is important to realize that the 1000 second (16.67 minute) time frame is 50,000 points long. Also, during the 1000 seconds, one would expect the 0.001 Hz signal to be represented only once, while the 5 Hz signal would appear 5000 times. This fact becomes important when comparing the results of the neural network.

MATLAB files *Rand_seq.m*, *Rand_plot.m* and *Rand_short_seq.m* in Appendix A were used to generate the random binary signal.

b. A-4D Learn and Test Data

Previous research has been done using a user input/output program for the generation of data for an artificial neural network [SCOT 89] [DROR 92]. The user input/output program was capable of generating a time varying model of an aircraft, providing the neural network an infinite amount of learning if necessary. In order to provide a finite data set upon which to train and test the neural network, data files were used. As mentioned earlier, these files represent 1000 seconds of test data which came from a MATLAB A-4D model. Flight test data or wind tunnel data could just as easily have been used.

The learning and testing data used by the neural network for modeling the *non-damaged* A-4D was created by running the MATLAB program *Gen_A4_data.m* (Appendix A). This program loads the A-4D data, creates the digitized state space matrix, calls up the

random binary sequence, runs a simulation to generate the measured (sensor) outputs, organizes the data for the neural network, and stores the data in an ASCII file (Learn.nna). It then changes the seed value of the random binary sequence, and repeats the steps above to create the test file (Test.nna). These files consist of 50,000 rows and 24 columns and require approximately 10 megabytes of disk storage space each. The program also compares the two files, determines the maximum and minimum value of each file and saves these values in a third file. This file is considerably smaller than the other two. Even though NeuralWorks can generate its own minimum and maximum values from the data, the learn and test files created were too large for the NeuralWorks software to do automatically.

c. Damaged A-4D Learn and Test Data

The MATLAB program *Gen_damaged_data.m* (Appendix A) computes the damaged A-4D data. This program is the same as the one used to generate the non-damaged A-4D data, except, to simulate a damaged A-4D, the aerodynamic derivative $M_{\delta e}$, the change in pitching moment due to a change in elevator deflection, was arbitrarily reduced by 70 percent. Figure 3.6 depicts the bode plot of the damaged A-4D model.

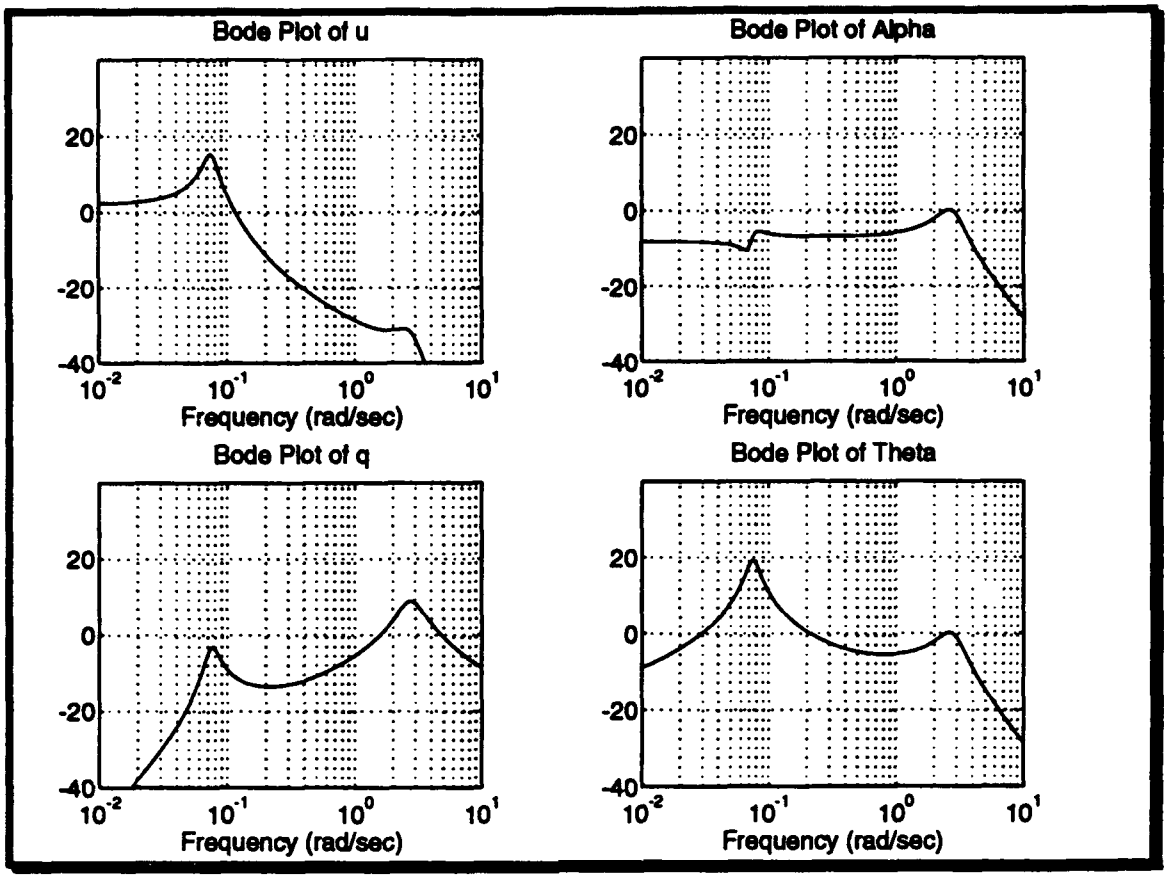


Figure 3.6 Bode Plot of Damaged A-4D

Comparing this response to the one in Figure 3.3, certain commonalties are apparent. First, the two primary modes, occur at the same frequency. This is because $M_{\delta e}$ only appears in the **B** matrix, which does not affect the eigenvalues of the plant. Secondly, the plots have the same general shape. The only difference is the amplitude of the peaks. Although the amplitude shift appears to be linear, it is not. From state space:

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{Y}(t) = \mathbf{C}\mathbf{x}(t)$$

the transfer function is

$$C(sI - A)^{-1}B.$$

A linear change in the **B** matrix results in a linear shift in the amplitude of the bode plot but changing $M_{\delta e}$ does not change the **B** matrix linearly:

$$\mathbf{B} = \begin{bmatrix} \frac{X_{\delta e}}{u_o} \\ \frac{Z_{\delta e}}{u_o - Z_{\dot{\alpha}}} \\ M_{\delta e} + \frac{M_{\dot{\alpha}} * Z_{\delta e}}{u_o - Z_{\dot{\alpha}}} \\ 0 \end{bmatrix}$$

$M_{\delta e}$ was chosen as the damage mechanism to model because of its connection to a realworld damage mechanism; a reduction in $M_{\delta e}$ is akin to a partial loss of elevator surface area due to battle damage. Other models using different values of $M_{\delta e}$ would represent different amounts of damage to the elevator. As one might expect, a reduction in $M_{\delta e}$ is prevalent in all four of the longitudinal states (See Figure 3.7: The solid line represents the undamaged aircraft, the dotted line represents $M_{\delta e}$ reduced 30 percent and the dash-dot line represents $M_{\delta e}$ reduced 70 percent.)

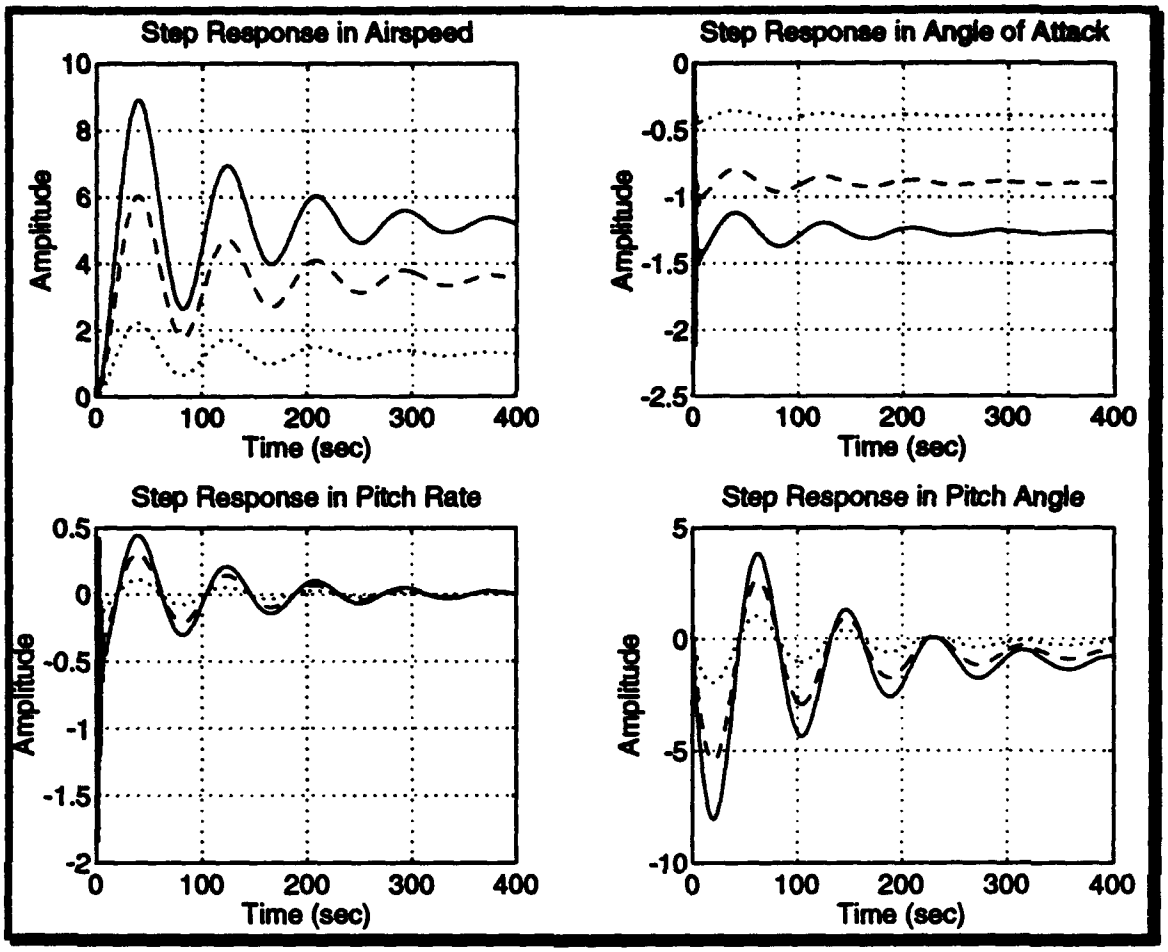


Figure 3.7 Step Response with Original $M_{\delta e}$ And with $M_{\delta e}$ Reduced 30 and 70 Percent

3. Neural Network Configuration

The neural network model for the A-4D is depicted in Figure 3.8. It was built using the "InstaNet" menu in NeuralWorks. The model has 20 processing elements in the input layer, one hidden layer with 20 processing elements and an output layer with 4 processing elements. The specifics of the NeuralWorks setup is provided in Appendix B.

Since the longitudinal A-4D digital plant is of fourth order, four consecutive values of all of the inputs and outputs were used for learning. These inputs and outputs are listed in Table 3.2. Although, a fourth order artificial neural network was used to model a fourth order plant, this is not a requirement. Dror obtained very good results modeling a 25th order F/A-18 with a third order neural network [DROR 92].

TABLE 3.2
INPUTS AND OUTPUTS FOR THE NEURAL NETWORK

Inputs					Output
$\delta e(n)$	$u(n)$	$q(n)$	$\theta(n)$	$\alpha(n)$	$u(n)$
$\delta e(n-1)$	$u(n-1)$	$q(n-1)$	$\theta(n-1)$	$\alpha(n-1)$	$q(n)$
$\delta e(n-2)$	$u(n-2)$	$q(n-2)$	$\theta(n-2)$	$\alpha(n-2)$	$\theta(n)$
$\delta e(n-3)$	$u(n-3)$	$q(n-3)$	$\theta(n-3)$	$\alpha(n-3)$	$\alpha(n)$

Optimizing the minimum number of processing elements in the hidden layer was not performed. Pruning the network may have led to faster learn times, but this was not the focus of the thesis.

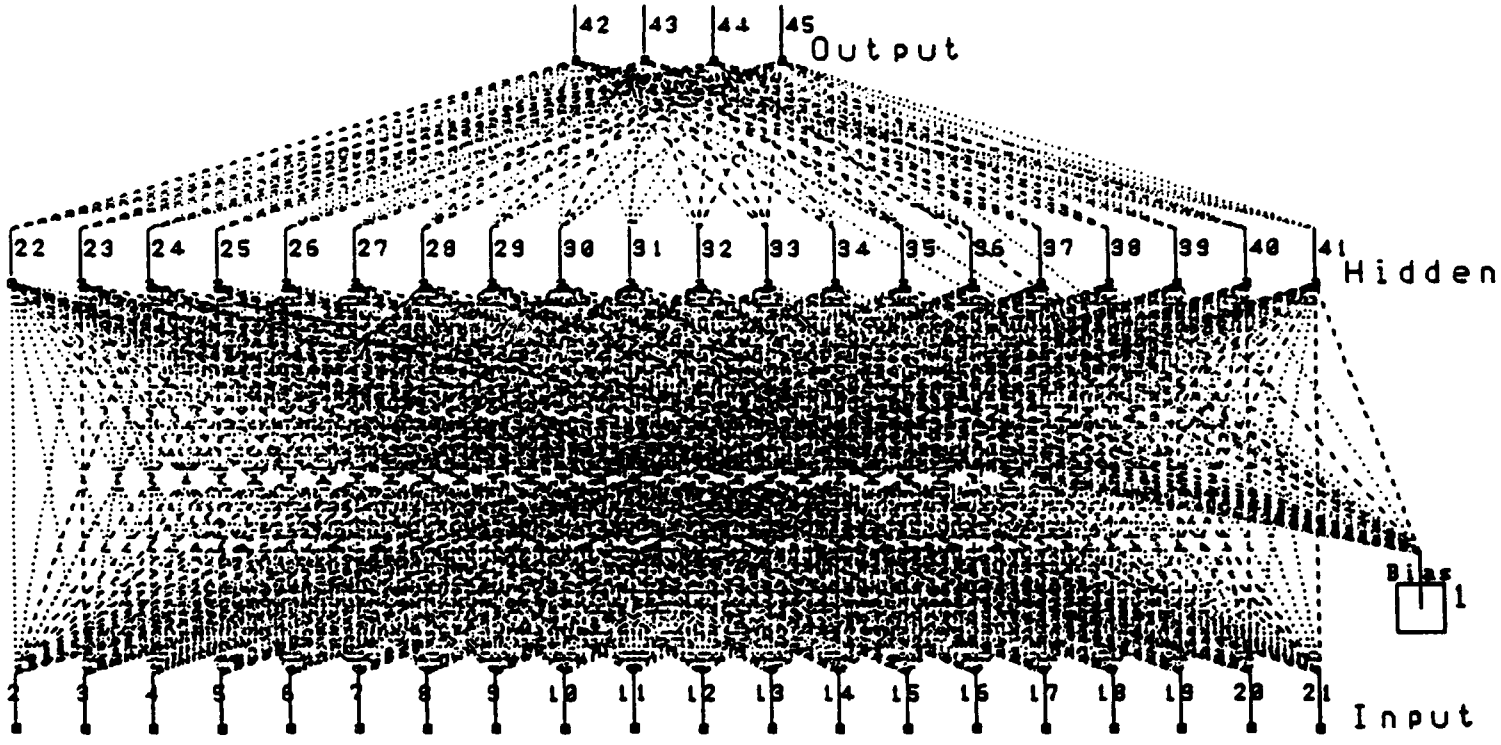


Figure 3.8 Artificial Neural Network of A-4D

4. Network Validation

After the artificial neural network has been fabricated, it must be trained. The first question to be answered is how much training is necessary. Under-training a network results in an incorrect output while over-training a network reduces the network's ability to generalize.

First six identical artificial networks were built. The first was trained on one pass of the learning file (50,000 records), the second on two passes (100,000 records), the third on three passes (150,000 records), the fourth on four passes (200,000 records), the fifth on five passes (250,000 records) and the sixth on ten passes (500,000 records). These were all tested with the same test file, and a power spectral density analysis was performed on the results. Figure 3.9 through Figure 3.14 show these results plotted along with the bode plot of the correct response. Although no over-training can be observed, it is evident from these plots that the network has learned the transfer function and that there is only slight improvement between training on 4 passes (200,000 presentations) and training on 10 passes (500,000 presentations). Since the goal was to find the number of passes which adequately trained the network without over training, 4 passes (200,000 presentations) was considered adequate. The slight improvement in performance using 10 passes did not outweigh the increase in computer run time required.

Also notice that the low frequency response of the artificial neural network is not as smooth as the high frequency response. This is due to the fact that the test data applied to the network corresponds to 1000 seconds of time. In 1000 seconds, a .001 Hz signal is represented only once while a 10 Hz signal is represented 10,000 times; therefore, the network is trained ten thousand times more on a 10 Hz signal than a .001 Hz signal. Low frequency components are presented to the network fewer times than high frequency components.

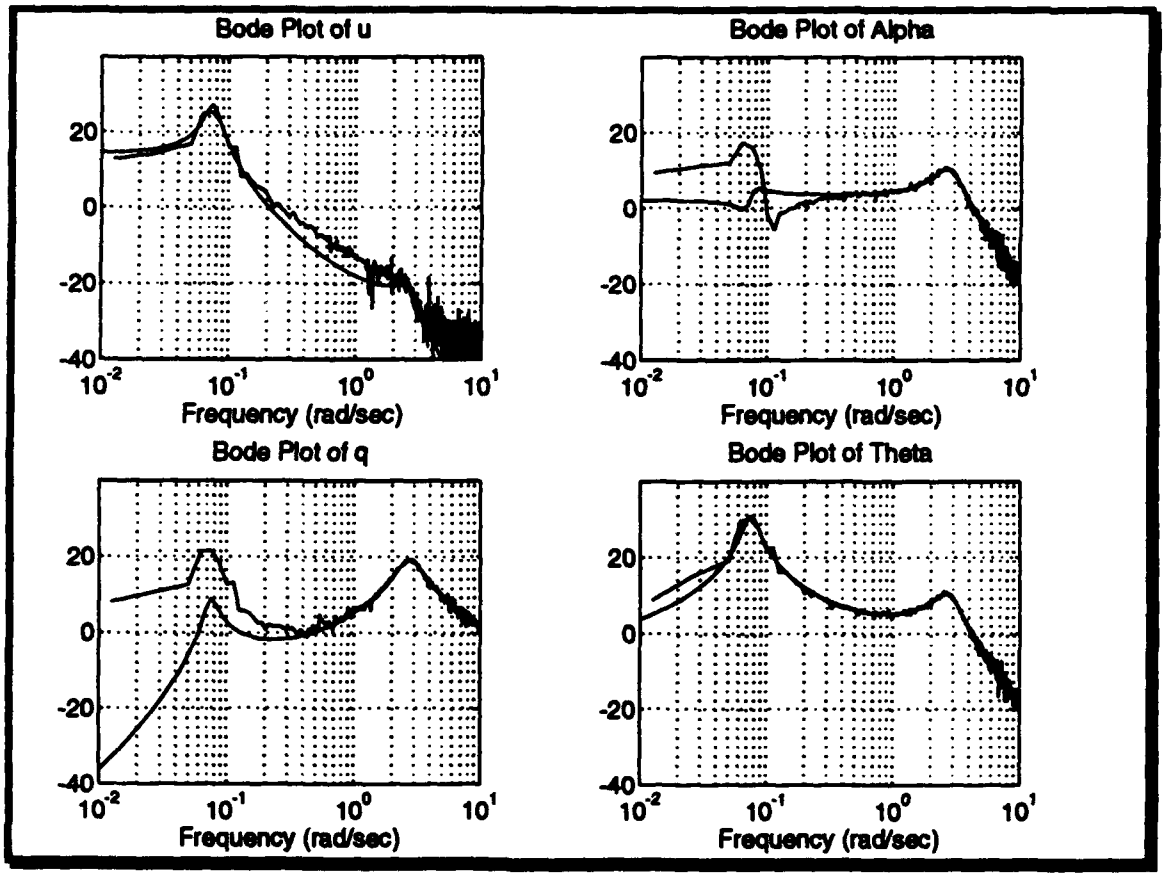


Figure 3.9 Network Training (one pass)

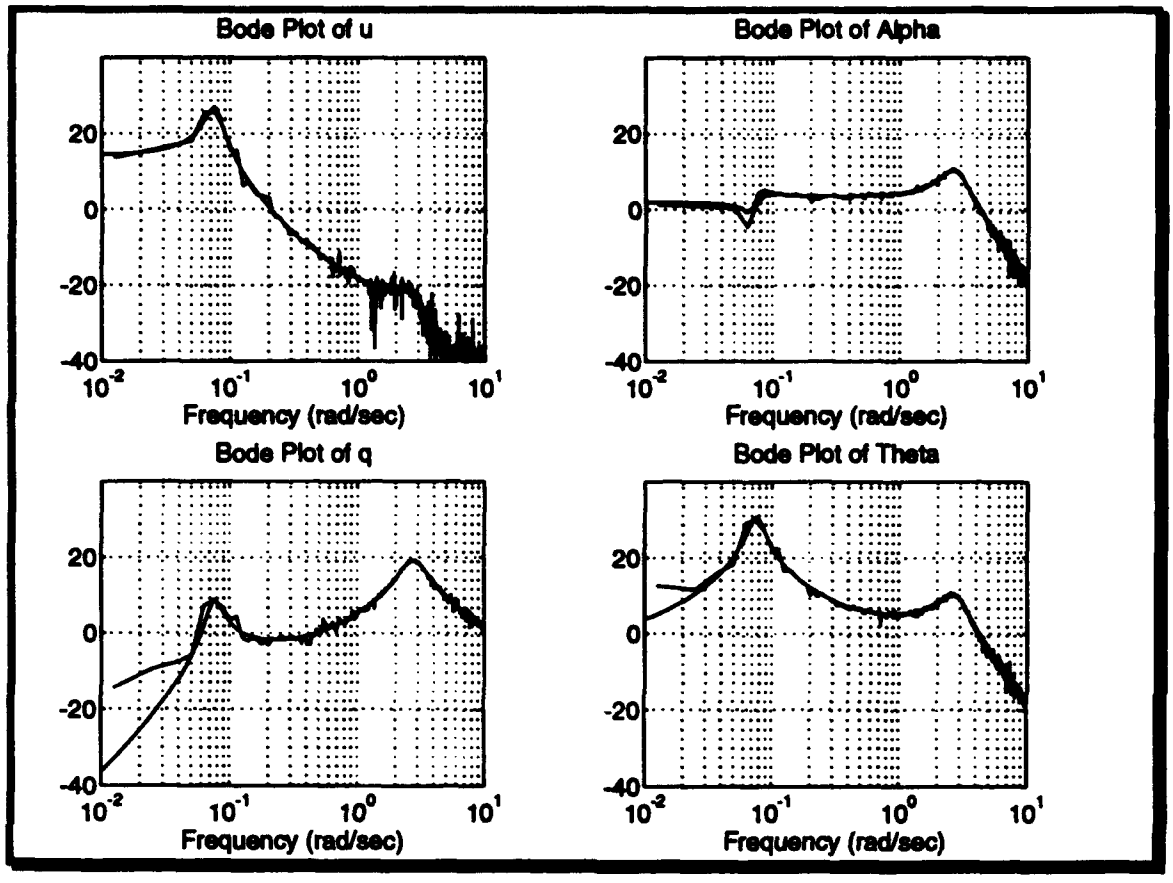


Figure 3.10 Network Training (two passes)

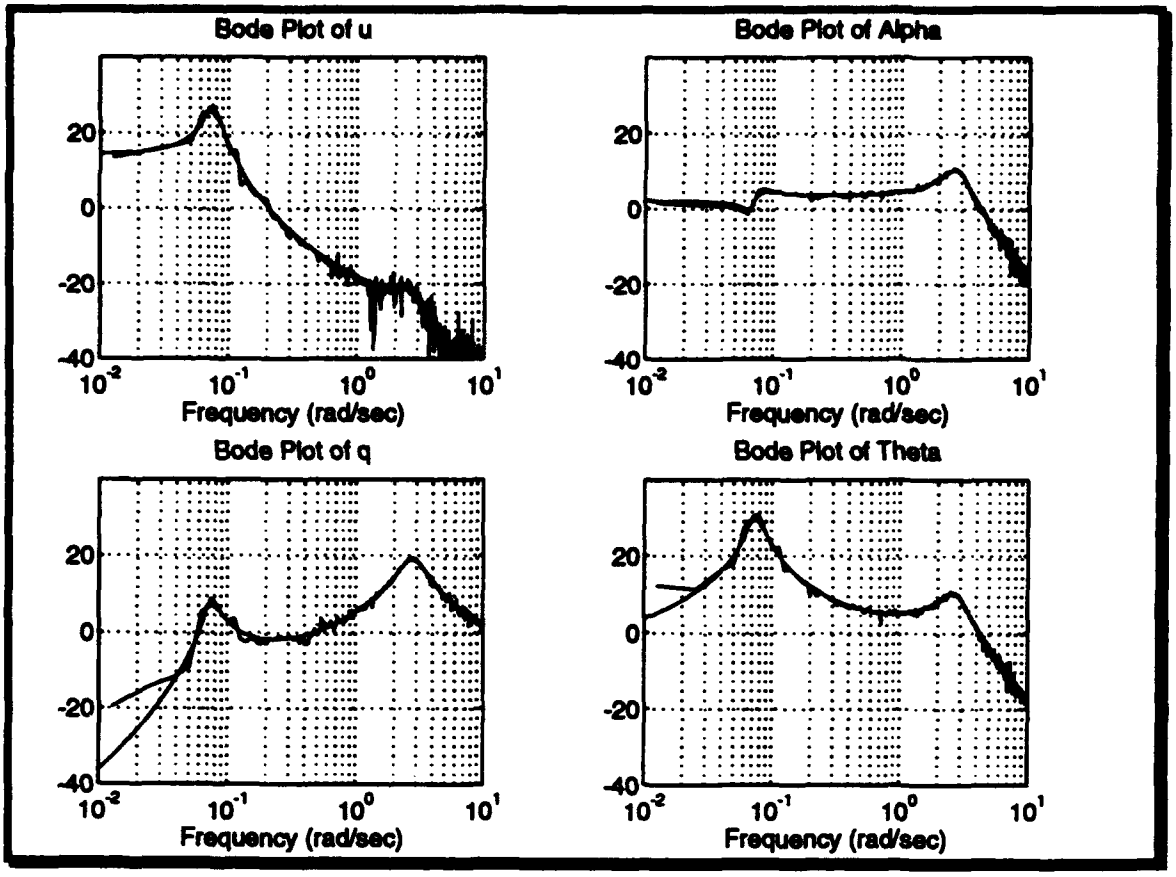


Figure 3.11 Network Training (three passes)

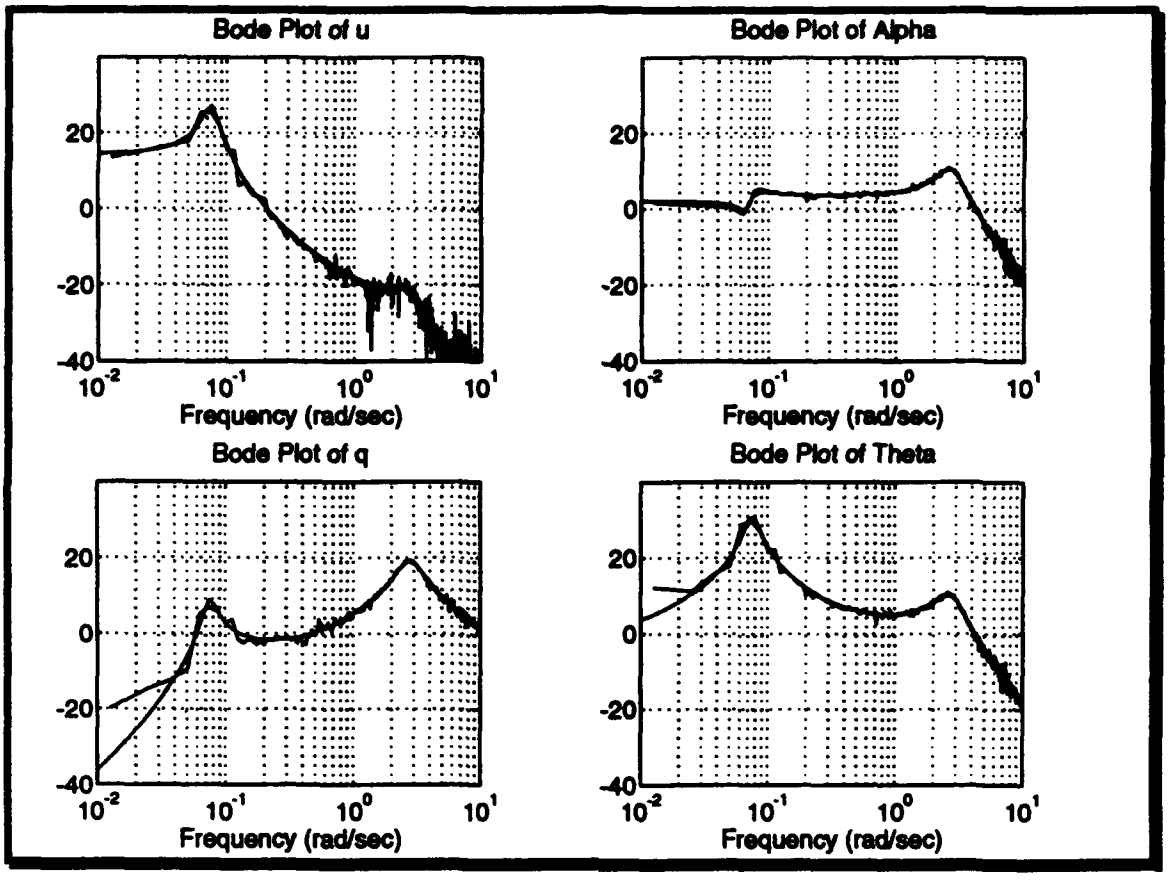


Figure 3.12 Network Training (four passes)

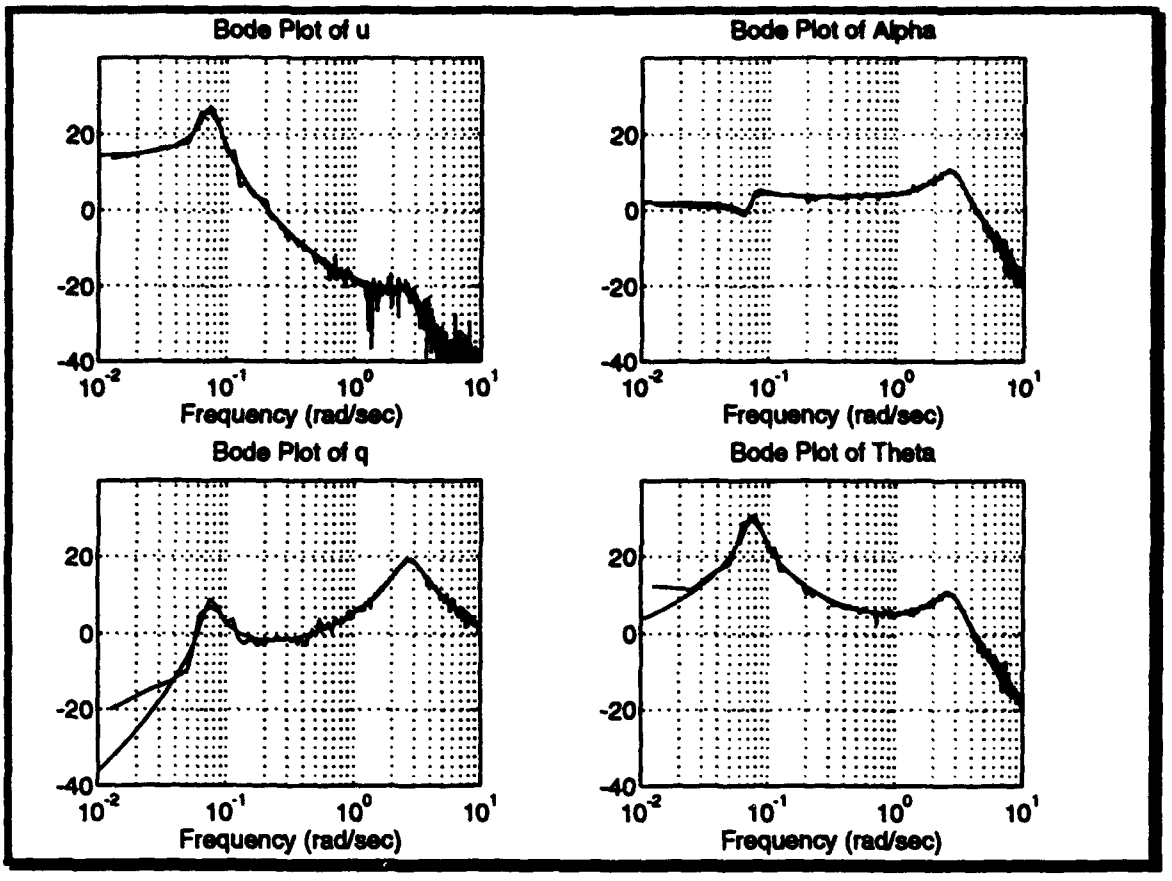


Figure 3.13 Network Training (five passes)

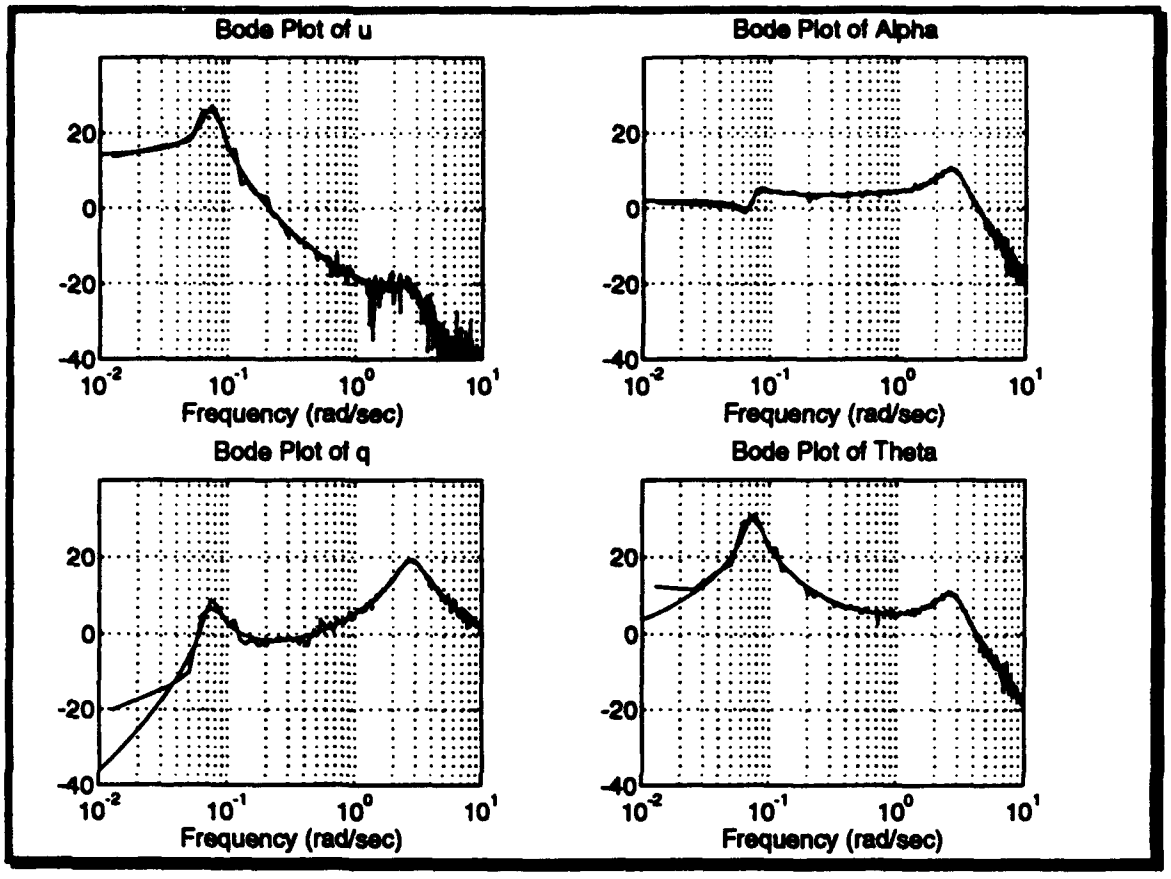


Figure 3.14 Network Training (ten passes)

After determining the number of training passes required, one of the artificial neural networks was reinitialized and the *damaged* A-4 data was presented. Again, the network was trained on 4 passes of the data (200,000 presentations). Figure 3.15 shows the result. Notice, the network has learned the damaged plant. As expected, the bode plots are similar in shape and shifted in amplitude.

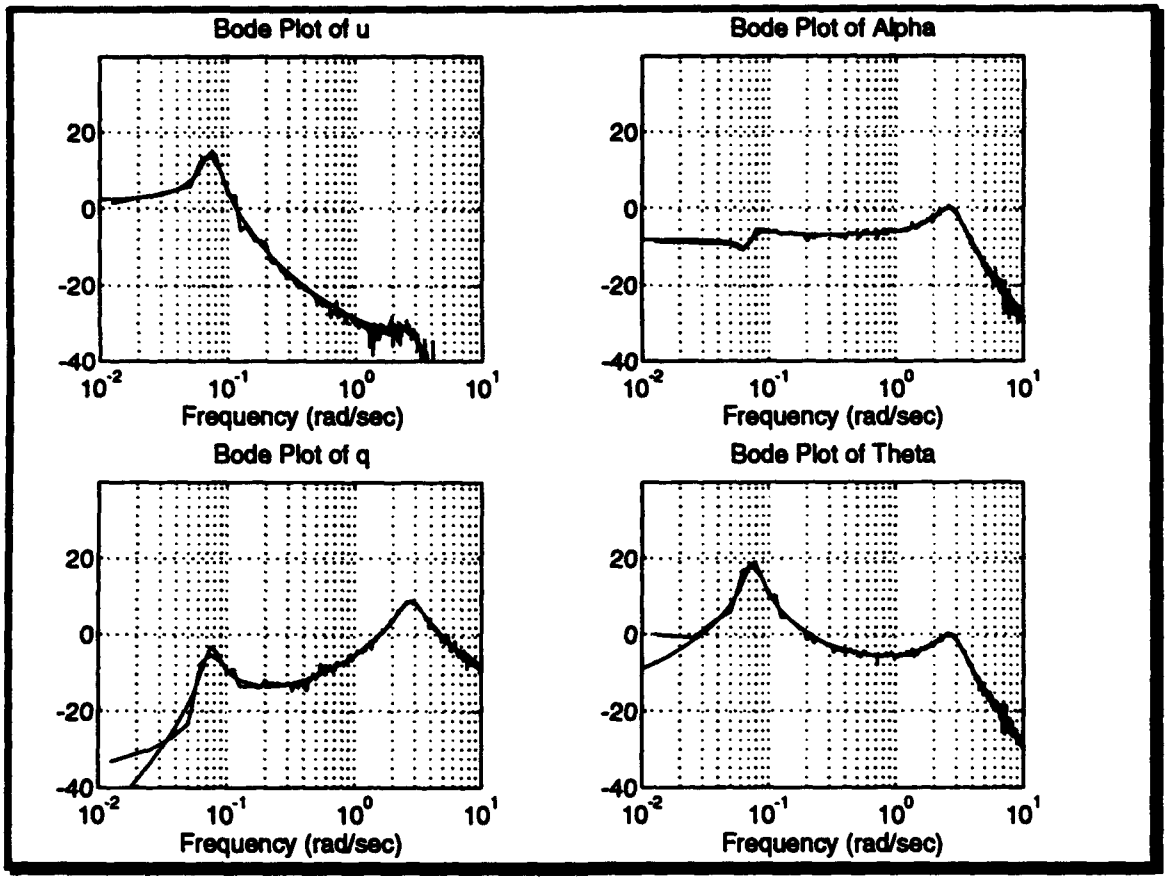


Figure 3.15 Frequency Response of Network Trained on Damaged A-4D

Next, both the damaged and undamaged training files were combined to see if the network could learn the transfer functions of the two plants simultaneously. It had been pointed out in earlier work [DROR 92] that to do this the files had to be intermingled. As a result, the two files were intermixed every 20 records, i.e. 20 damage, 20 undamaged, 20 damaged, etc. (see *Gen_both_data.m* in Appendix A). It was then tested on the test file for the undamaged aircraft and on the test file for the damaged aircraft. Figure 3.16 is the spectral density of the network's output using the test file for the undamaged A-4D and Figure 3.17 is the spectral density of the network's output using the test file for the undamaged A-4D.

Notice that they correspond very well to their respective bode plots. The network has simultaneously learned eight transfer functions: four for the damaged A-4D and four for the undamaged A-4D.

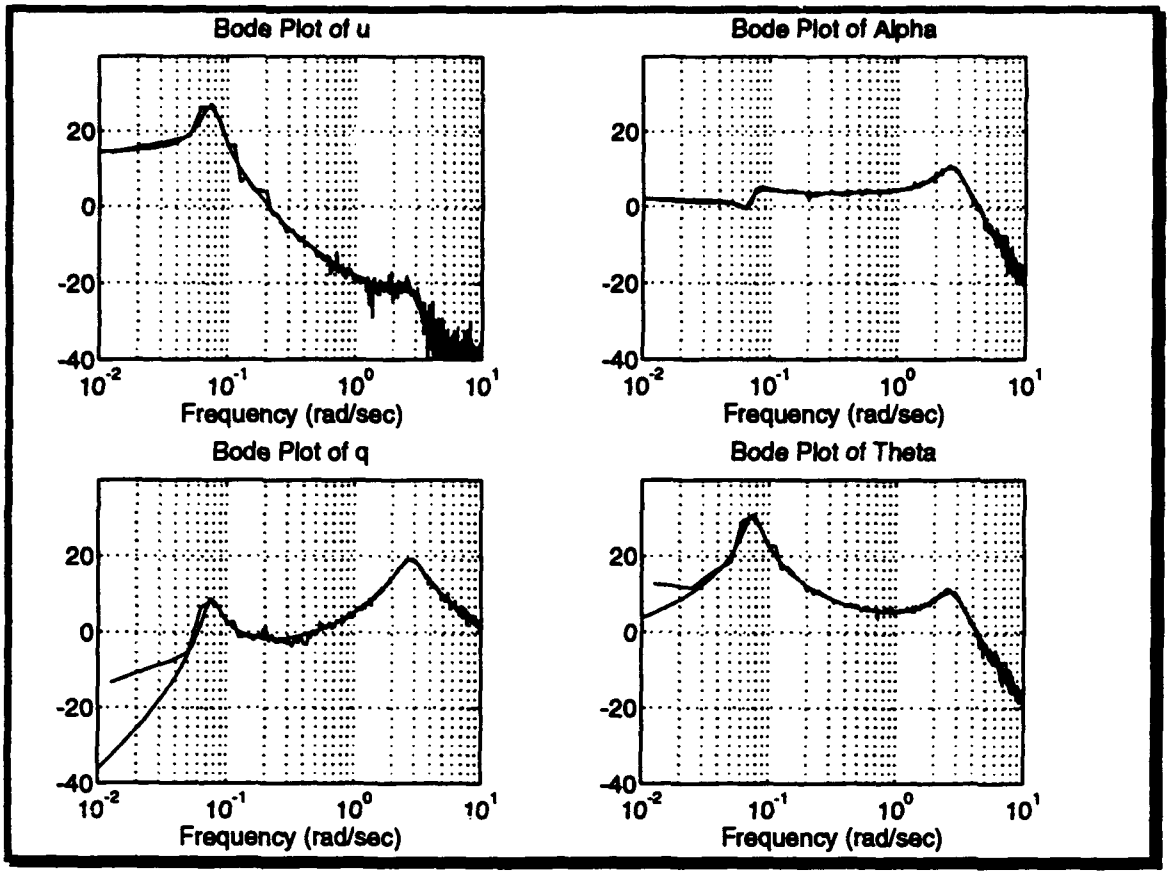


Figure 3.16 Network Trained on Damaged and Undamaged A-4D; Tested with Undamaged A-4D

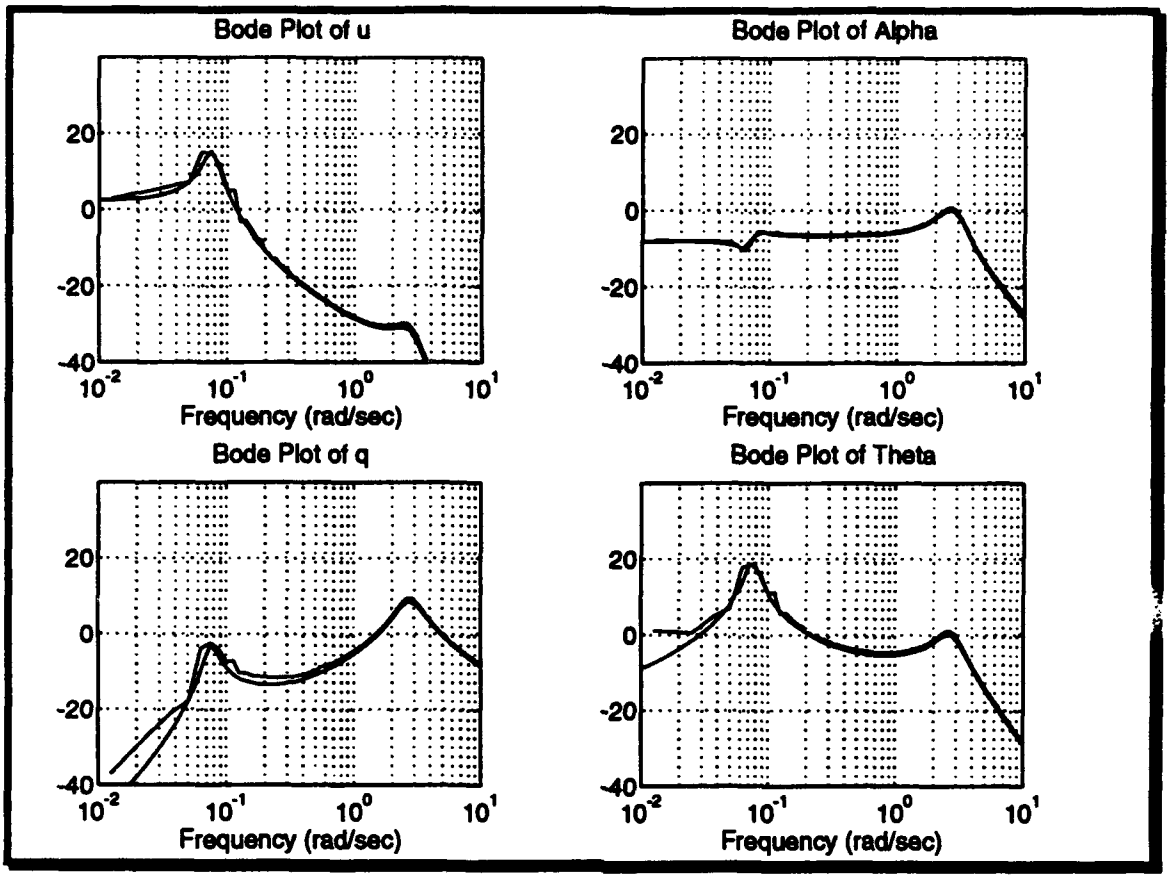


Figure 3.17 Network Trained on Damaged and Undamaged A-4D; Tested with Damaged A-4D

IV. RESULTS

Since the purpose of this thesis was to determine the robustness of the artificial neural network as a model for damaged aircraft, three concepts were evaluated--interpolating, extrapolating and detecting.

A. INTERPOLATION BETWEEN DAMAGE MECHANISMS

With the neural network designed and trained on two plants--the damaged A-4D and undamaged A-4D--the network was tested on other damaged A-4D data to determine how it responded when the damage to the elevator was less than that for which it was trained. In order to investigate this, six damaged A-4D models were created to provide the test data for the artificial neural network using 10, 20, 30, 40, 50, and 60 percent reductions in $M_{\delta e}$, respectively. Figure 4.1 is the bode plot of the undamaged A-4 plotted along with the power spectral density of the neural network's output. Figure 4.2 through Figure 4.8 shows the bode plots of the respective damaged A-4D model along with the power spectral density of the artificial neural network's output.

The spectral density plots indicate that the neural network has extrapolated the transfer functions of all of the damaged A-4D models from the two models on which it was trained. While having been trained on only one value of damage, $M_{\delta e}$ reduced 70 percent, the

network has determined the transfer function of all of the damaged plants and, therefore, does not need to be trained on them.

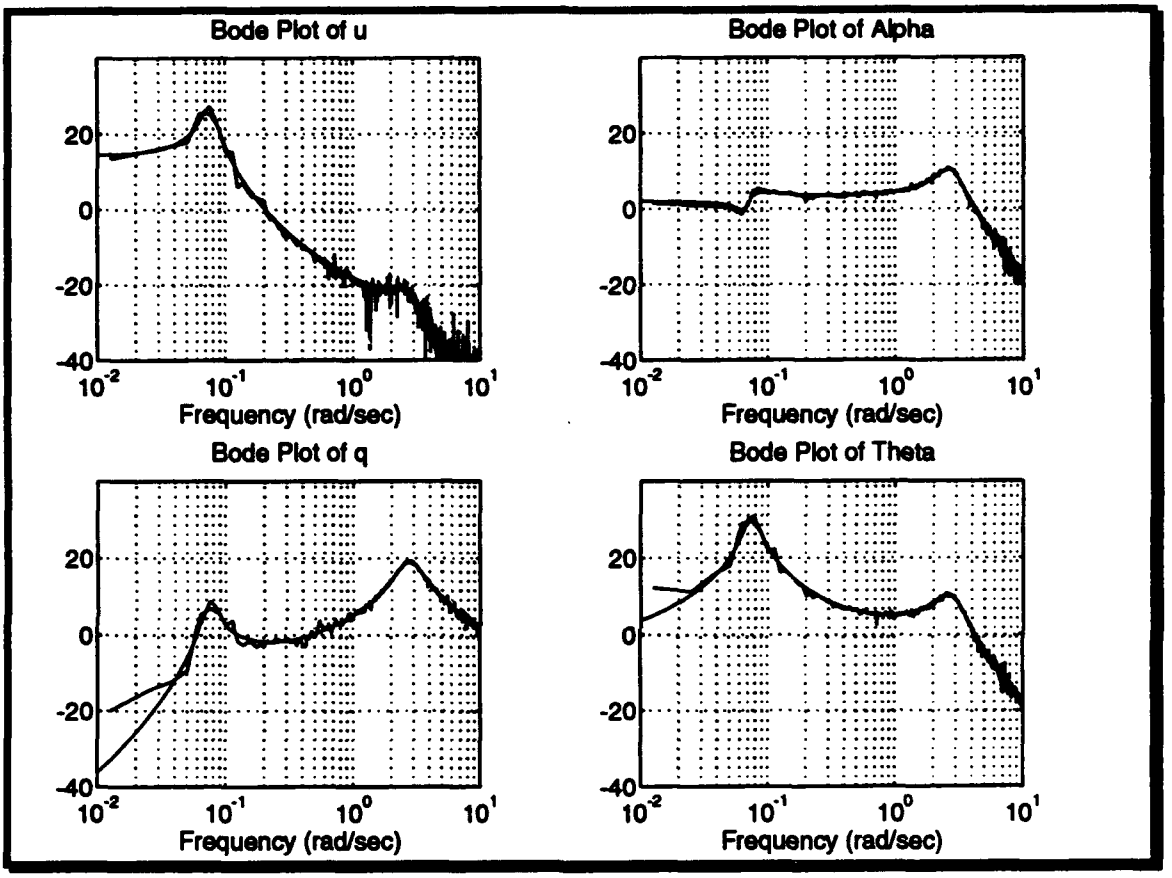


Figure 4.1 Network Trained on 70% Damage and Undamaged A-4D; Tested with Undamaged A-4D

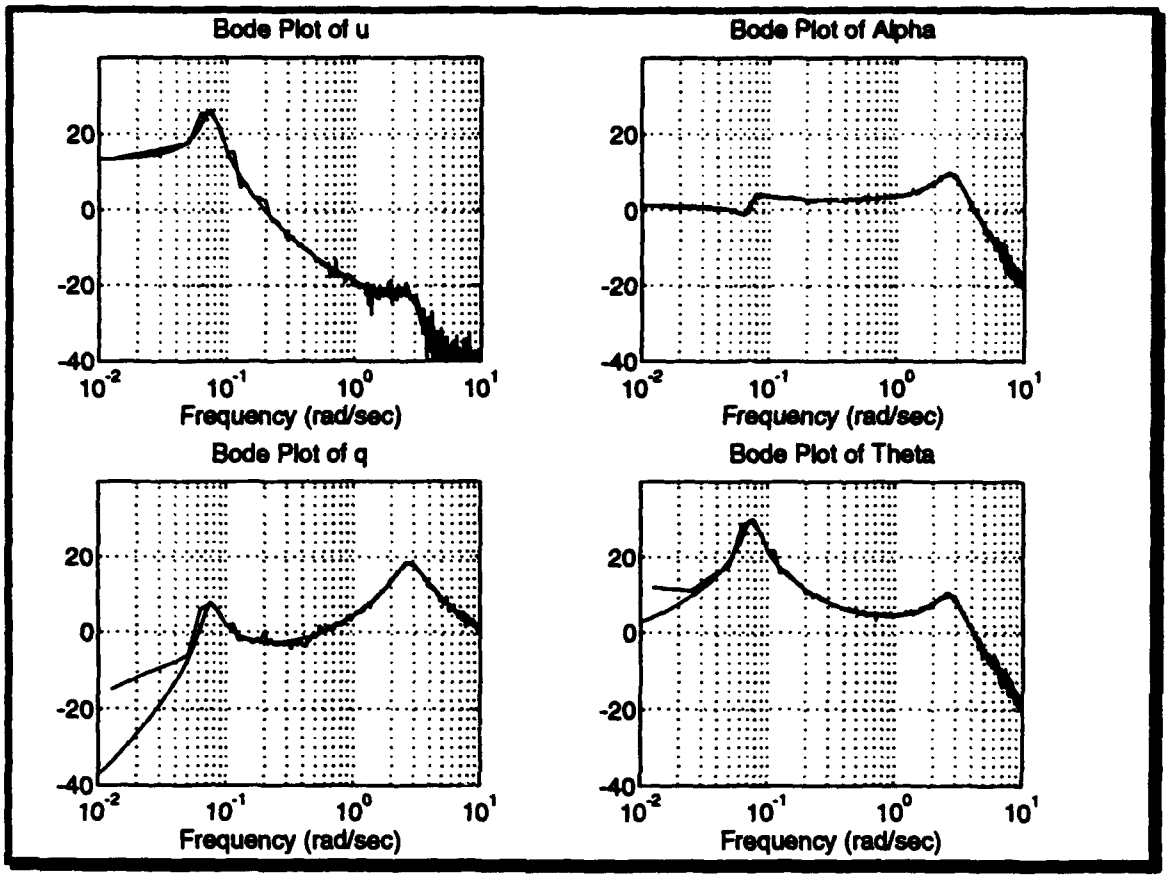


Figure 4.2 Network Trained on 70% Damage and Undamaged A-4D; Tested with 10% Damage

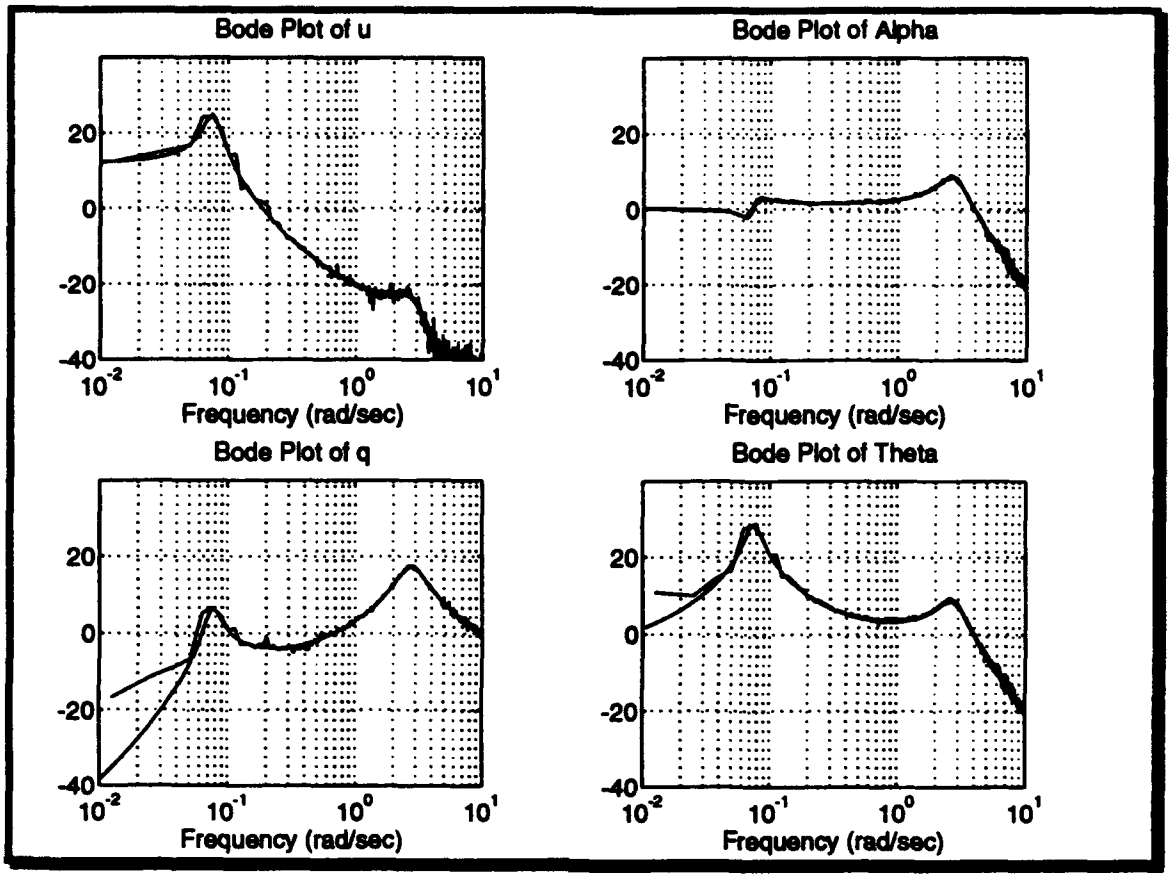


Figure 4.3 Network Trained on 70% Damage and Undamaged A-4D; Tested with 20% Damage

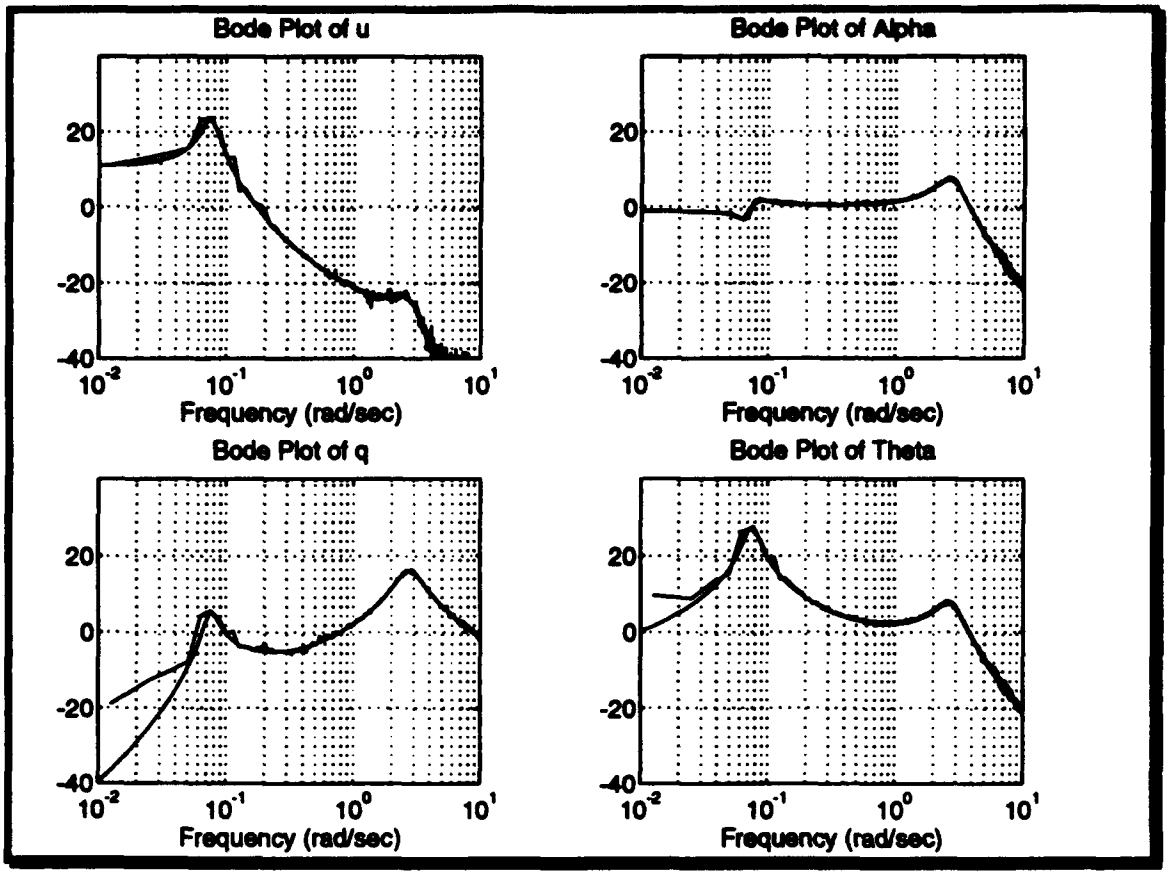


Figure 4.4 Network Trained on 70% Damage and Undamaged A-4D; Tested with 30% Damage

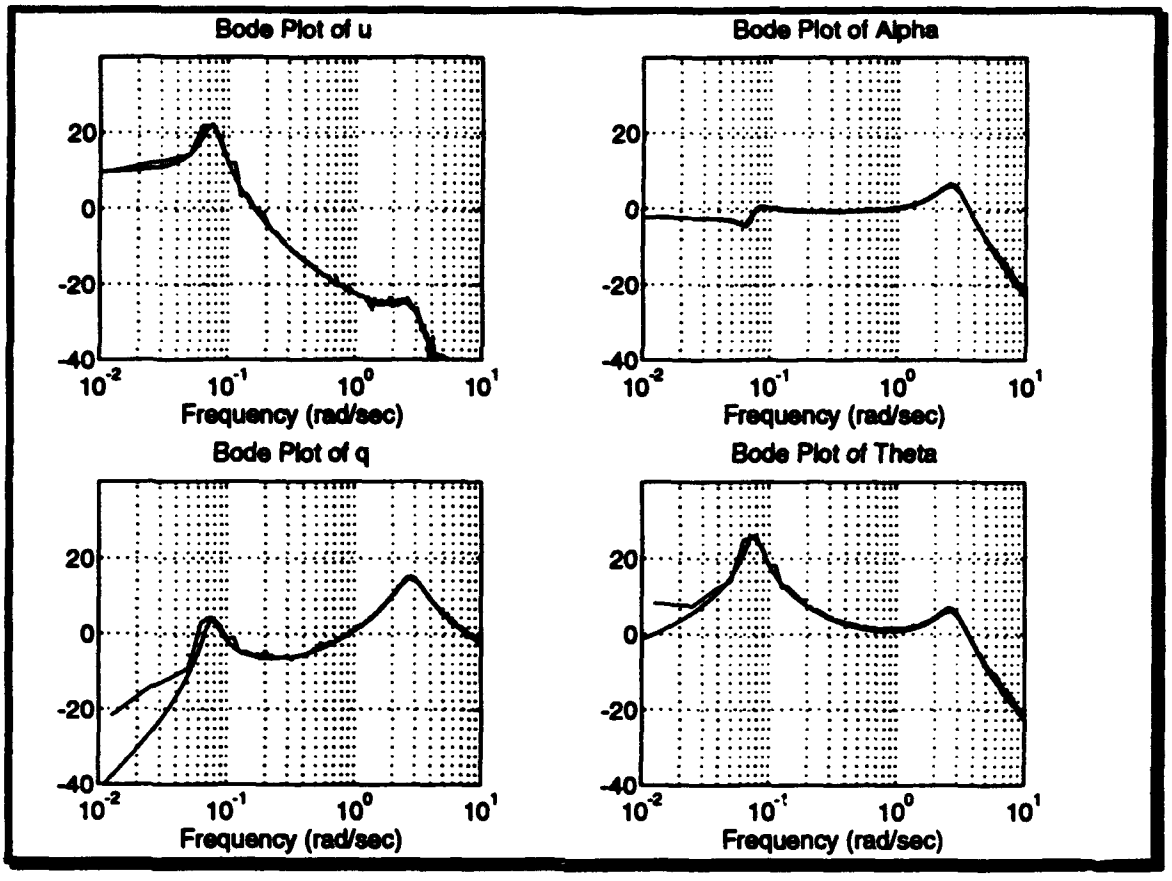


Figure 4.5 Network Trained on 70% Damage and Undamaged A-4D; Tested with 40% Damage

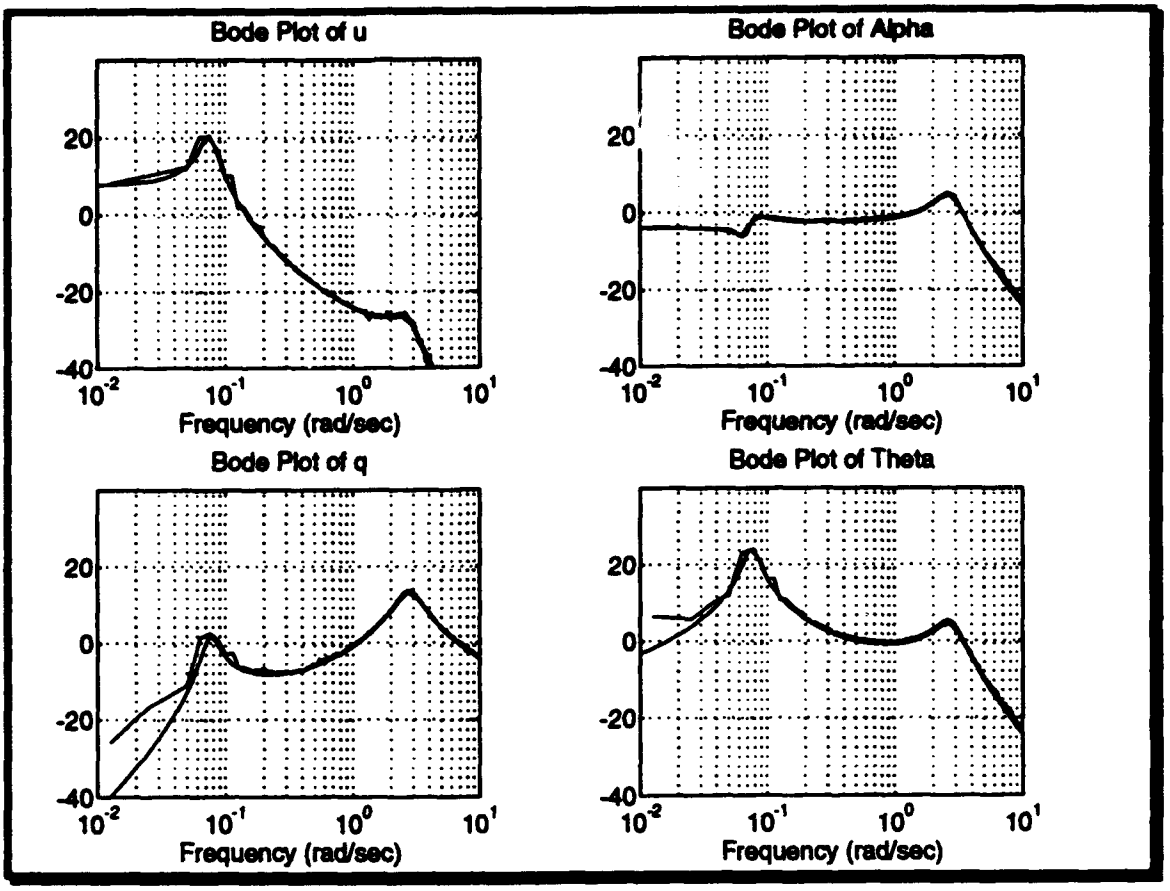


Figure 4.6 Network Trained on 70% Damage and Undamaged A-4D; Tested with 50% Damage

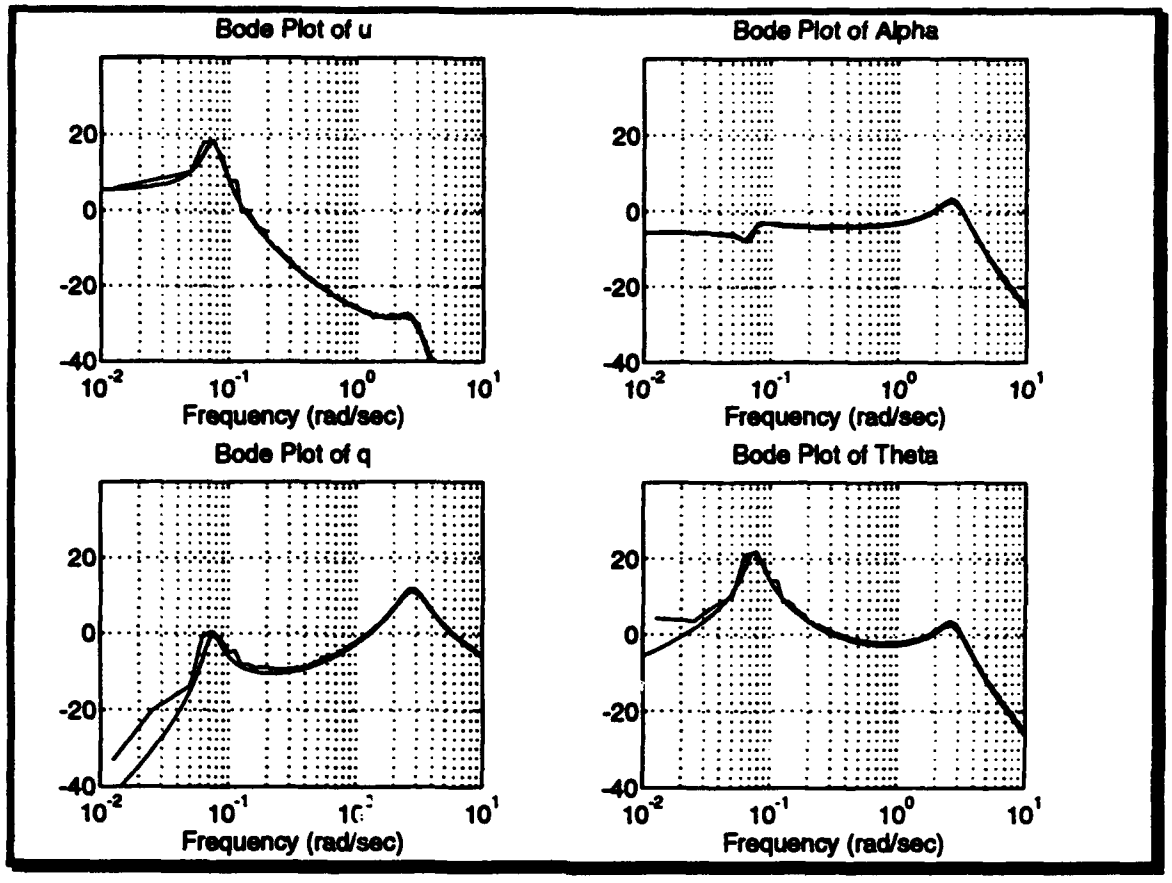


Figure 4.7 Network Trained on 70% Damage and Undamaged A-4D; Tested with 60% Damage

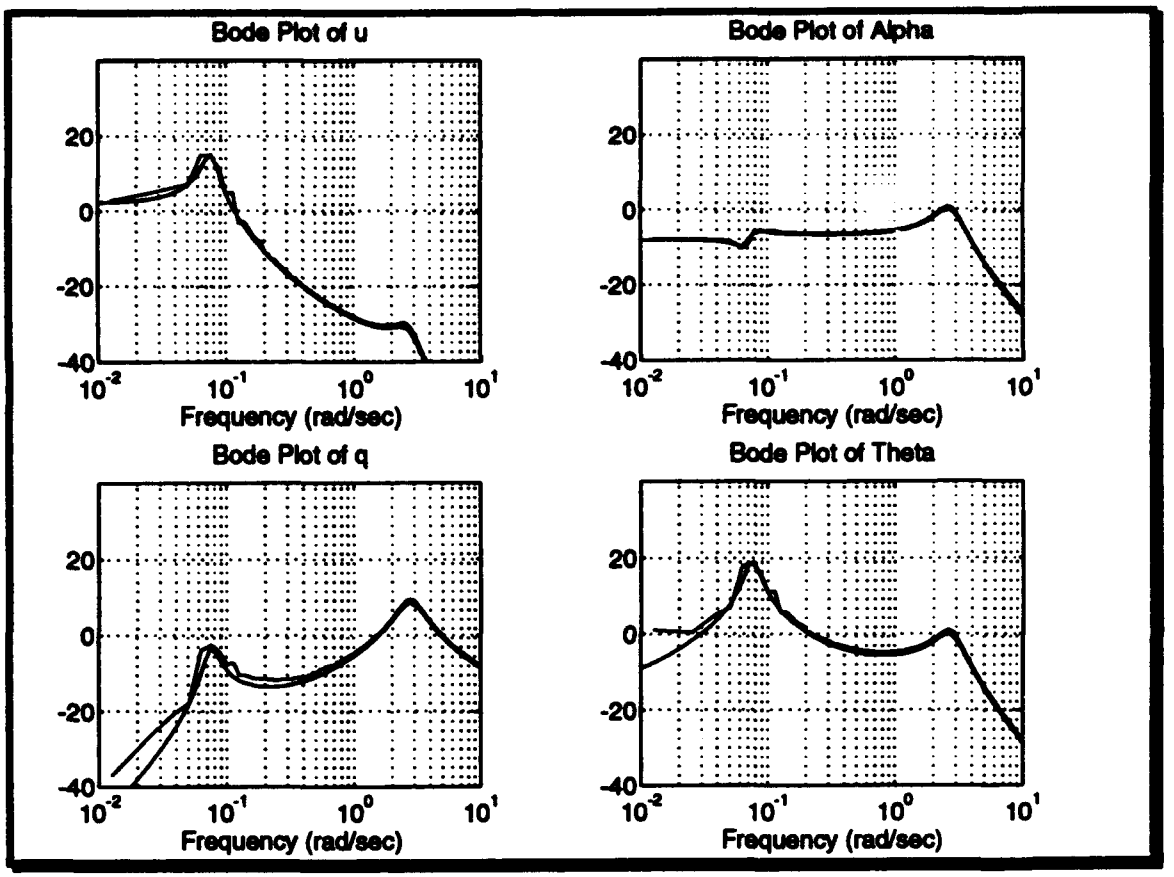


Figure 4.8 Network Trained on 70% Damage and Undamaged A-4D; Tested with 70% Damage

B. EXTRAPOLATING FROM A DAMAGED AIRCRAFT

Since the artificial neural network could interpolate between the two transfer functions on which it was trained, it was tested to see how it responded when the damage to the elevator was *more* than that for which it was trained. In order to investigate this, two damaged A-4D MATLAB models were created to provide the test data for the artificial neural network: $M_{\delta e}$ reduced 80 percent and 90 percent, respectively. Figures 4.9 and 4.10 show the power spectral density plots of the neural network's outputs along with the outputs of the test models. From Figure 4.9 it is evident that the network has extrapolated the correct transfer function. Figure 4.10, however, shows the extrapolation beginning to breakdown slightly in pitch rate. This is probably due to the extreme reduction in $M_{\delta e}$, i.e., the elevator has been reduced in size by roughly 90 perce :. This could be because there is not enough elevator surface area left to effectively control the pitching moment.

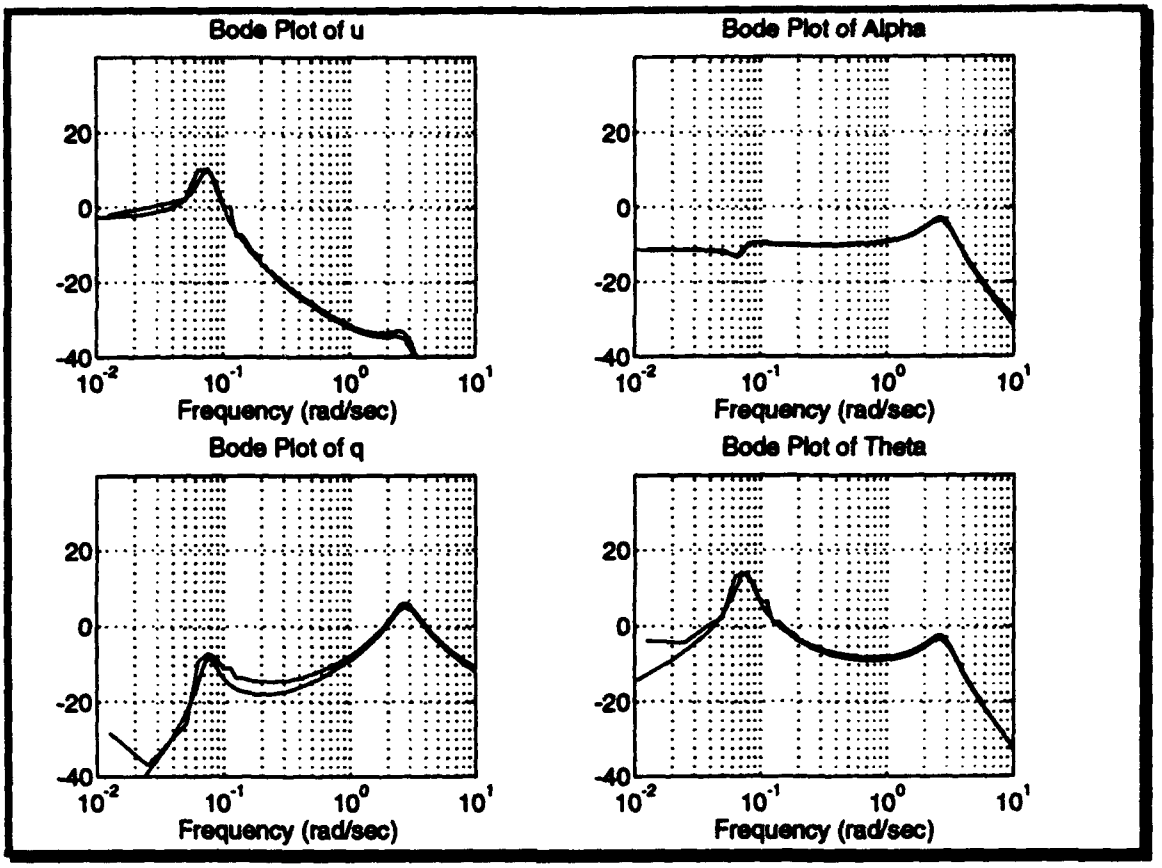


Figure 4.9 Network Trained on 70% Damage and Undamaged A-4D; Tested with 80% Damage

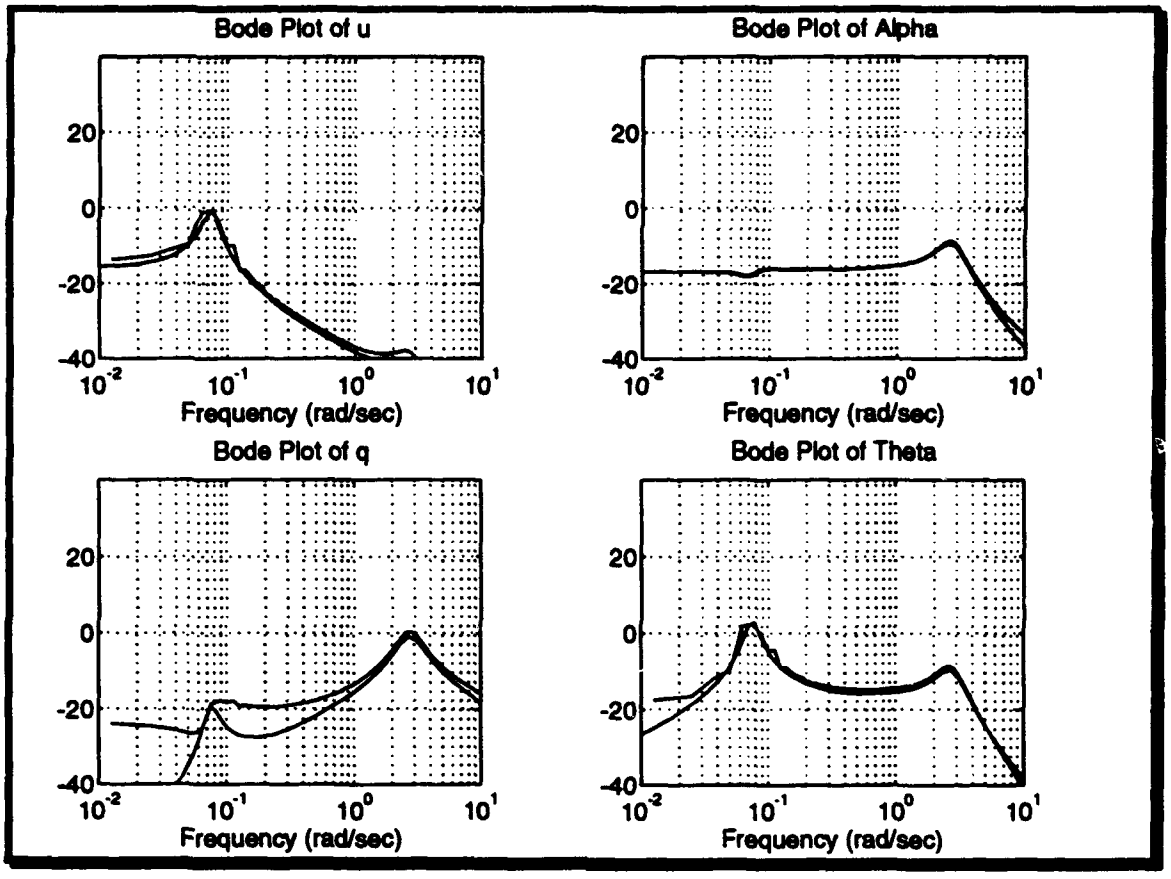


Figure 4.10 Network Trained on 70% Damage and Undamaged A-4D; Tested with 90% Damage

C. DETECTING AND RESPONDING TO DAMAGE

In the previous two sections, the neural network's output was analyzed in the frequency domain to determine whether the network could interpolate and extrapolate. Here, the time response of the network is analyzed to confirm those earlier results. In Figure 4.11, 10 seconds of the undamaged A-4D output [u, q, alpha, theta], of the neural network is plotted along with the true A-4D's output. Note that the network tracks the true output almost exactly.

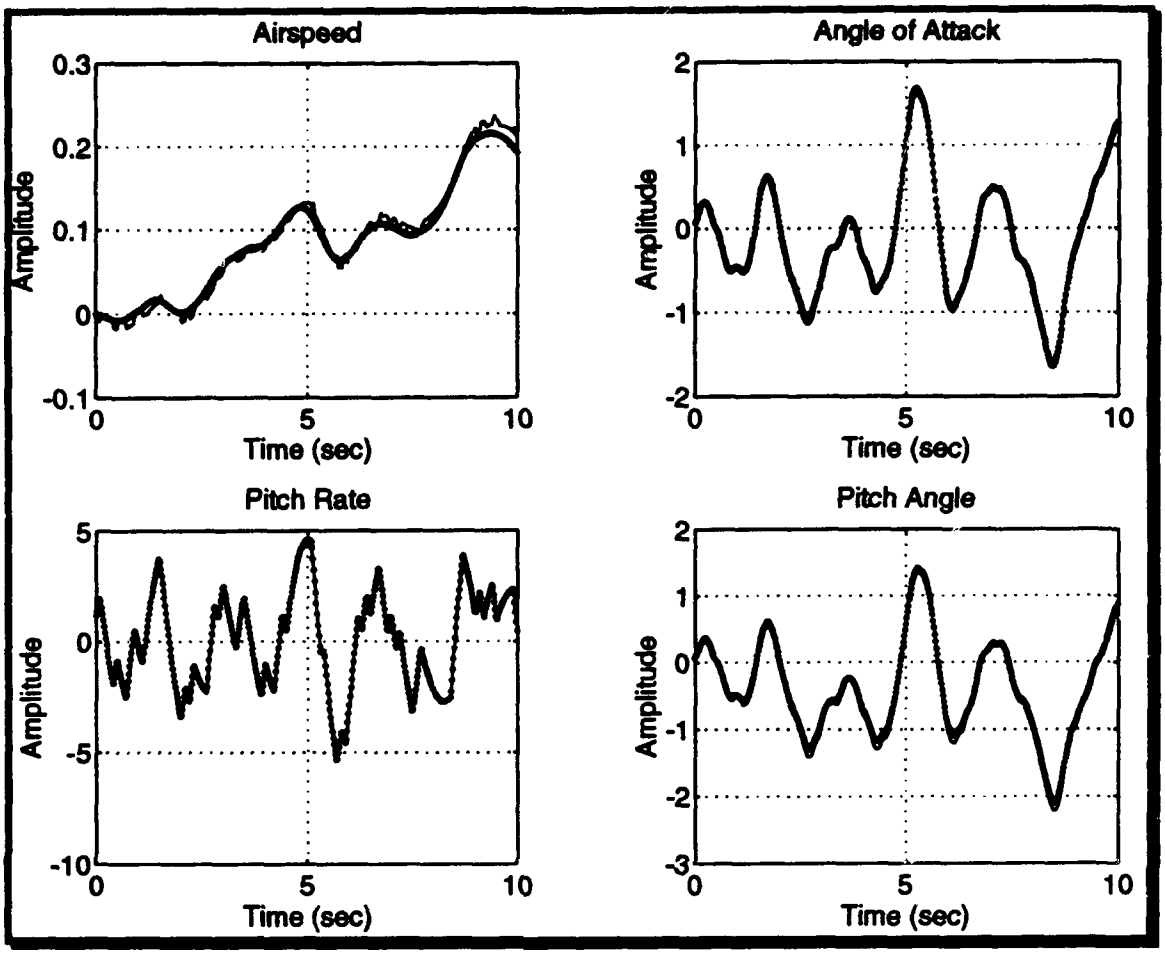


Figure 4.11 Network Trained on 70% Damage and Undamaged A-4D; 10 Seconds of Undamaged Response

In Figure 4.12 through Figure 4.20 the output of the artificial neural network is again plotted, except at t=5 seconds, the elevator is simulated to have been damaged, i.e., $M_{\delta e}$ is reduced. This was accomplished by switching the test data from the undamaged A-4D model to one of the damaged models. Plotted along with neural network's responses are the desired responses provided by the respective aircraft models. From these plots it is evident that the neural network has not only identified that damage has occurred, but it has also correctly modeled the aircraft's response to the damage.

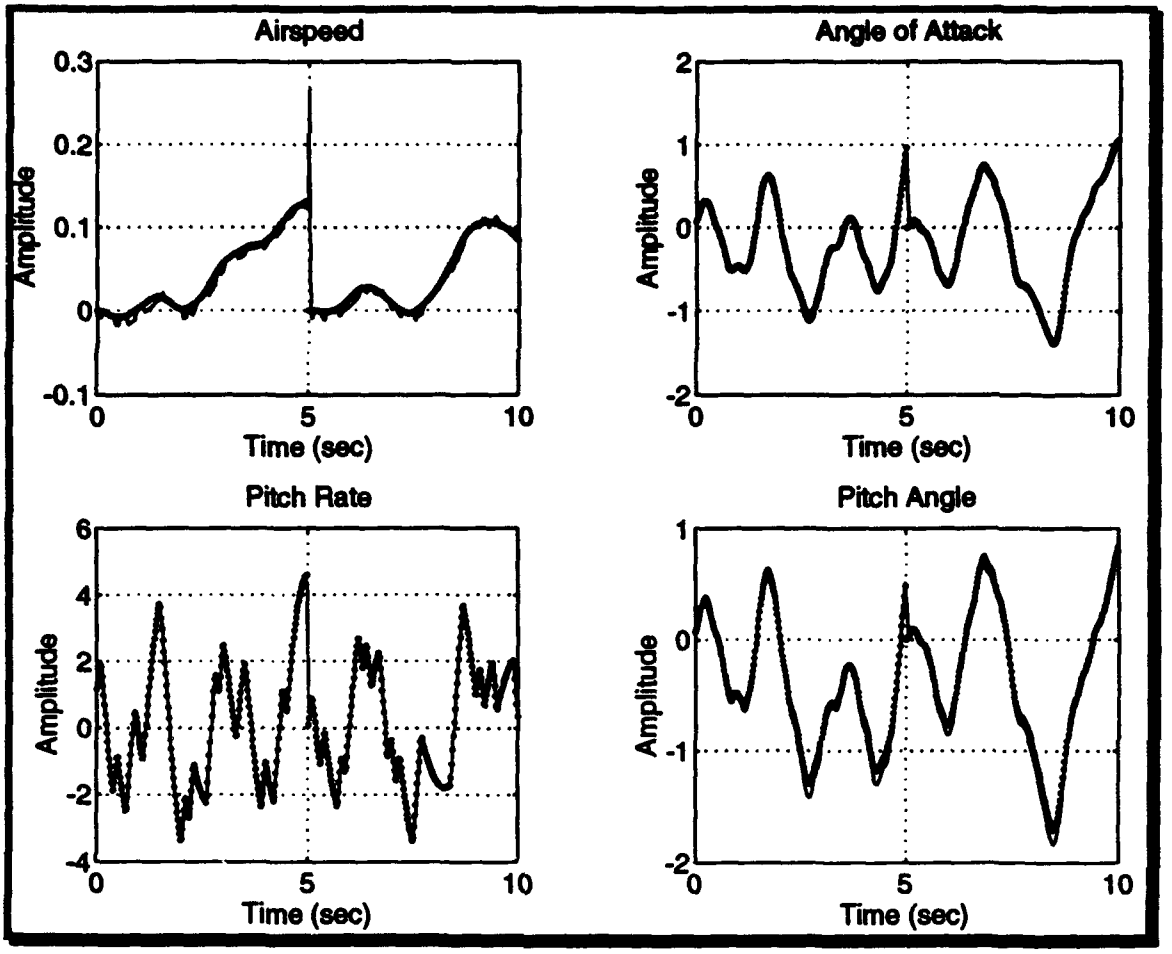


Figure 4.12 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 10% at t=5

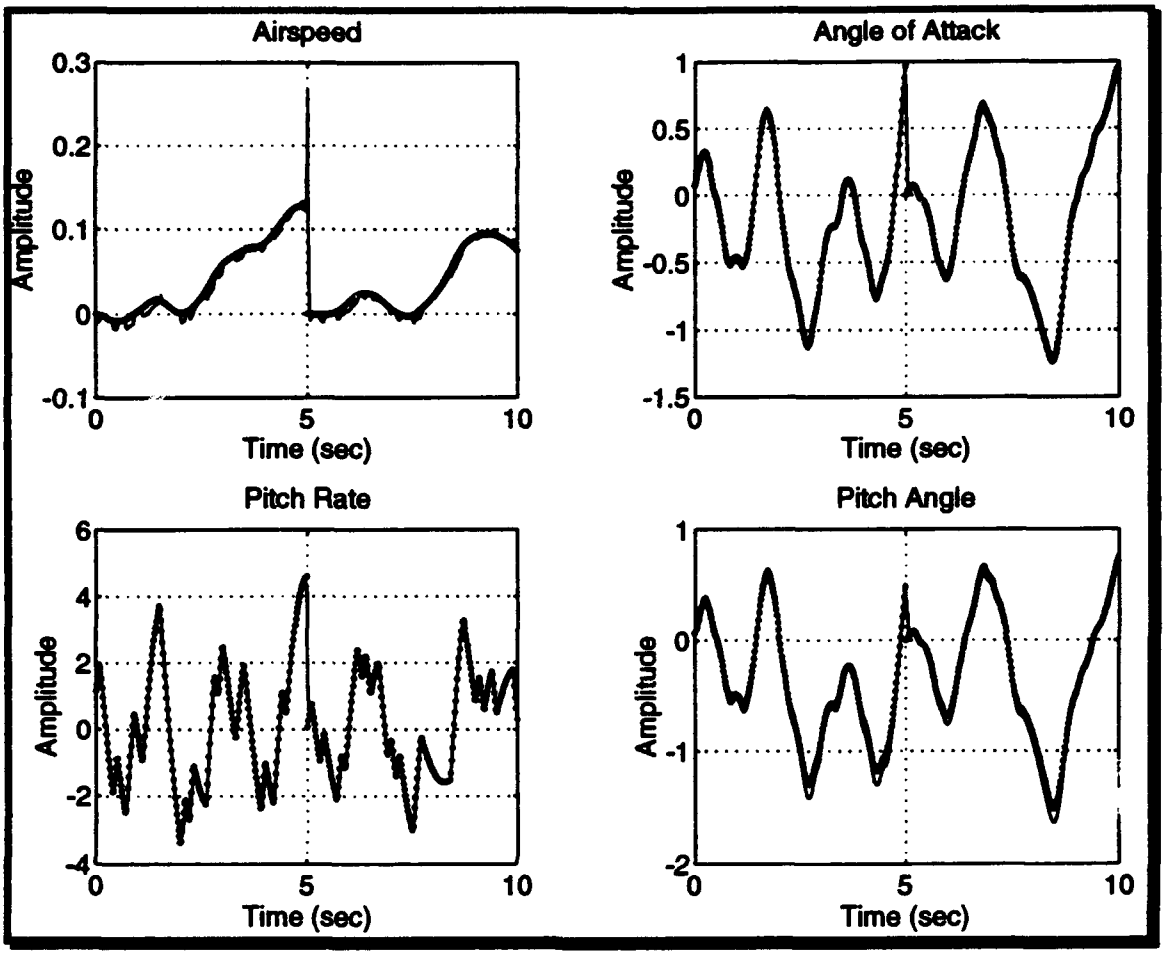


Figure 4.13 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 20% at t=5

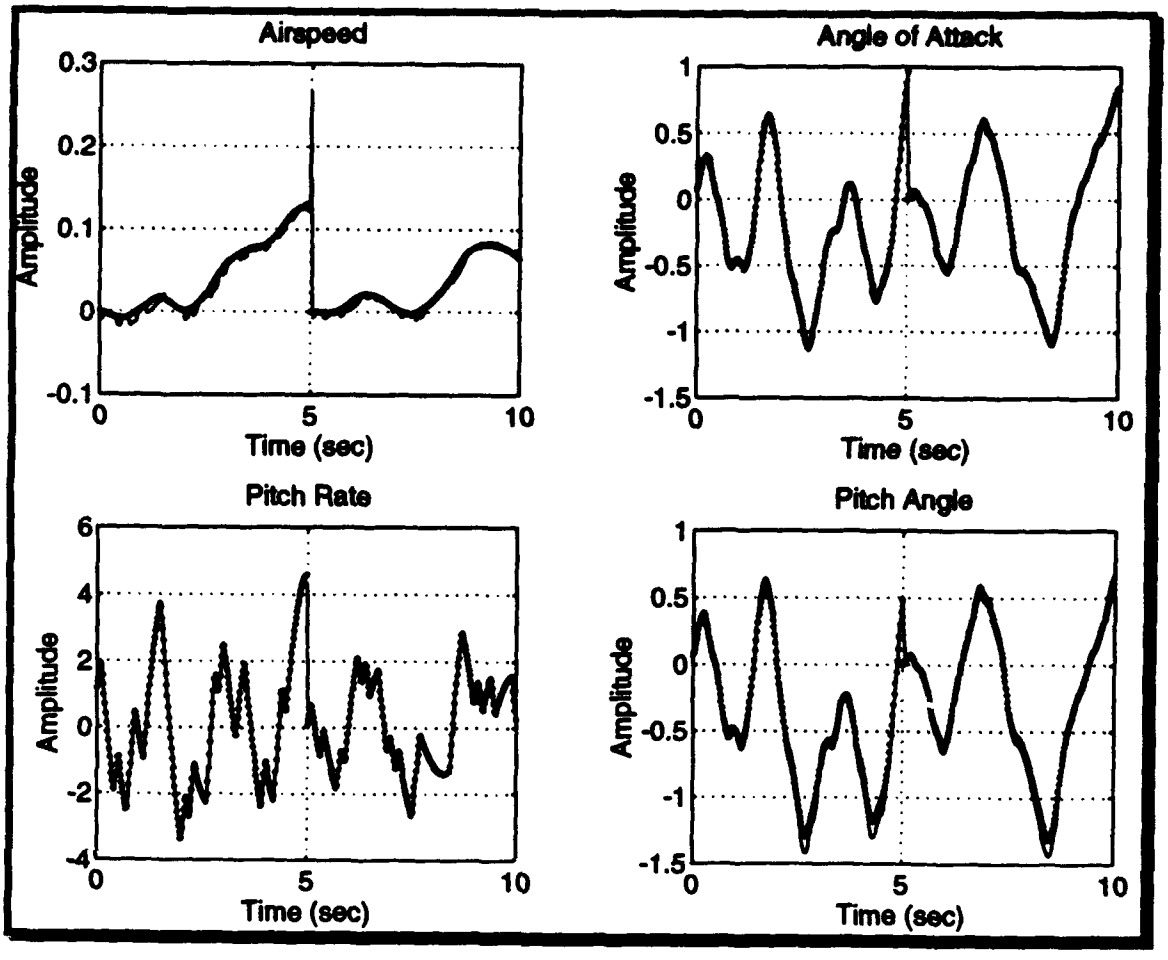


Figure 4.14 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 30% at t=5

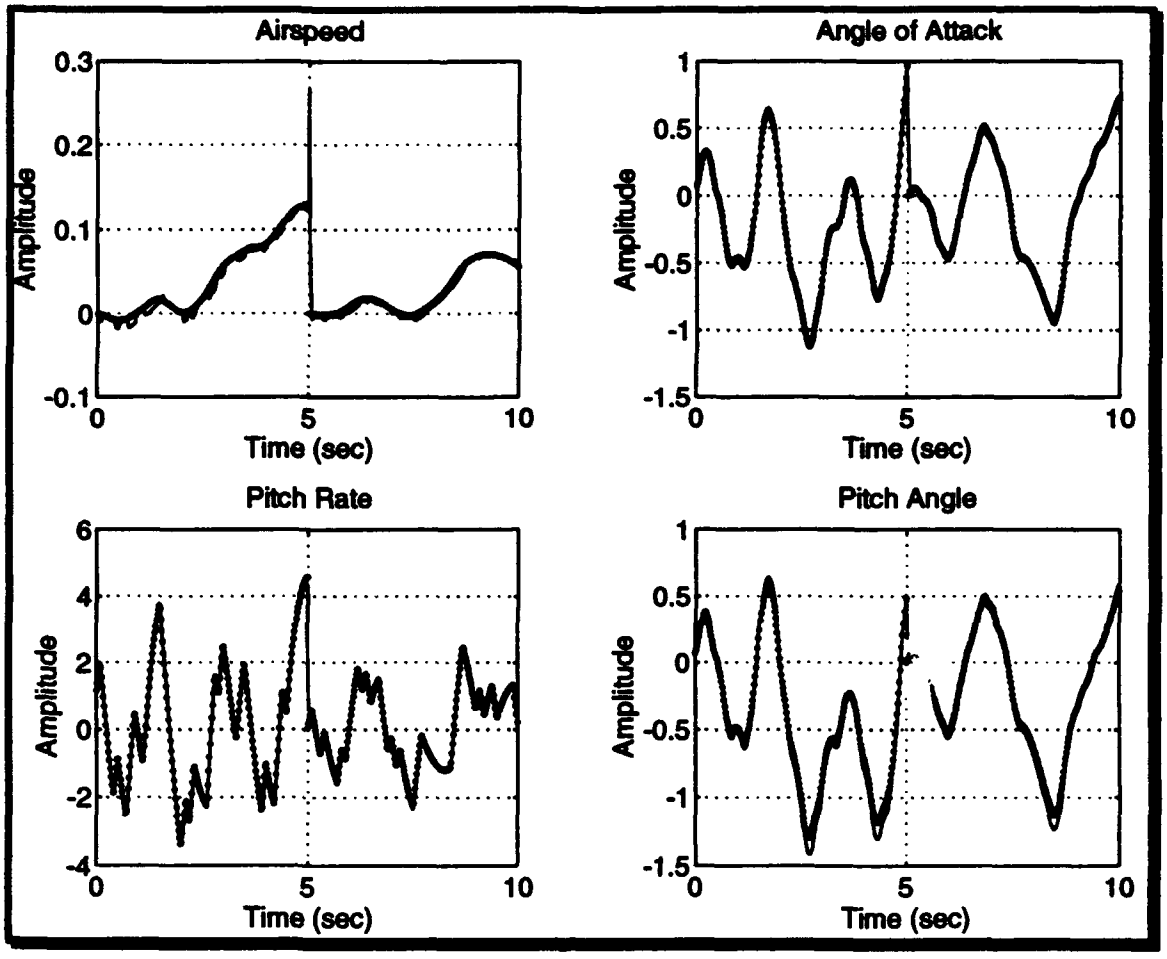


Figure 4.15 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 40% at t=5

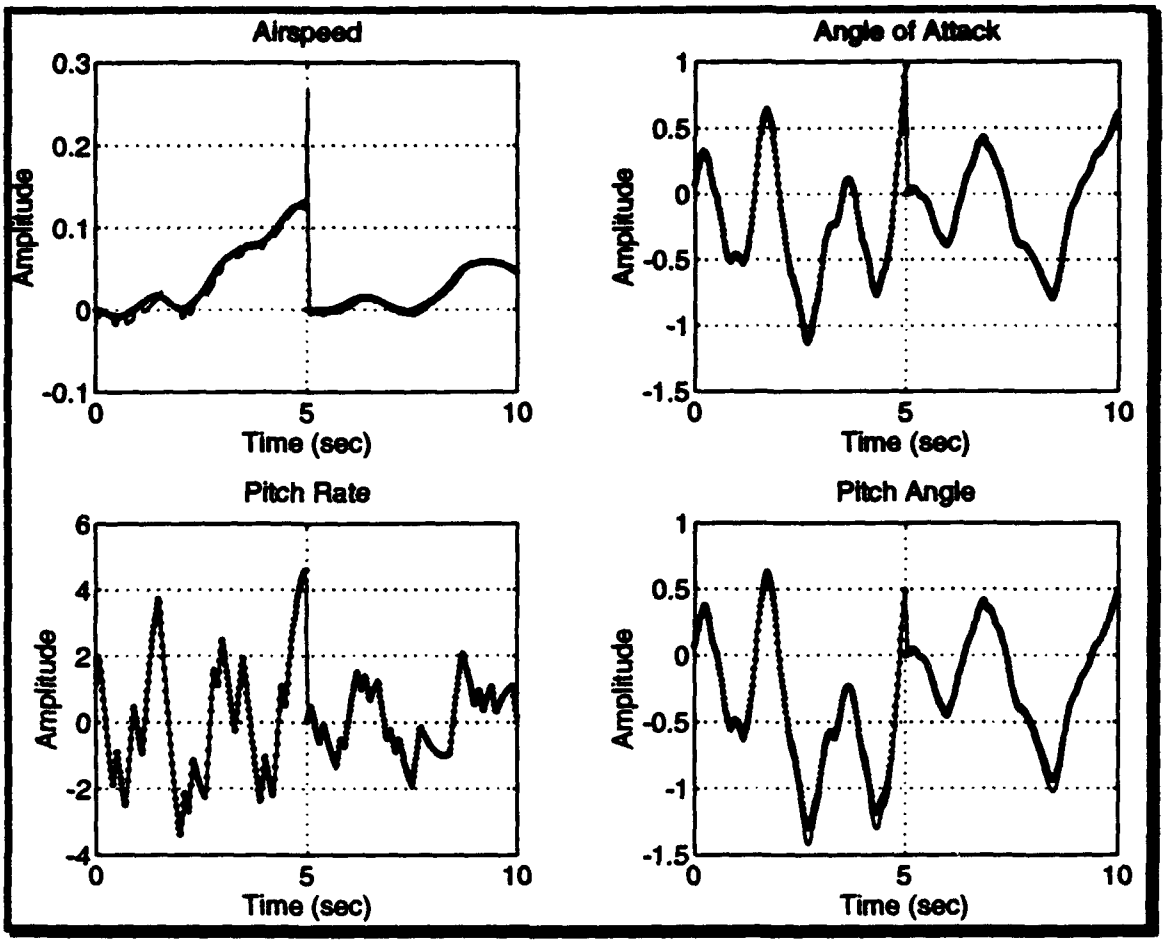


Figure 4.16 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 50% at t=5

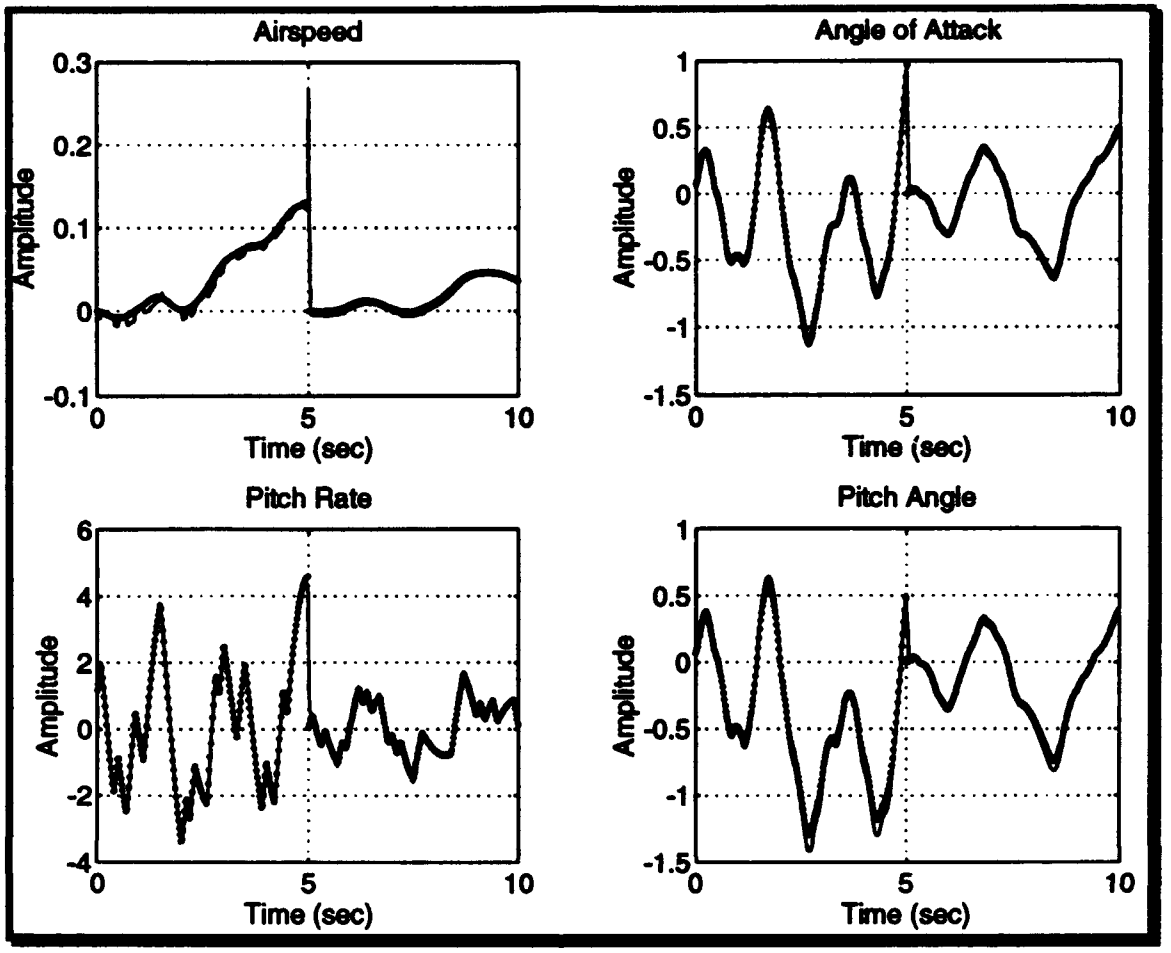


Figure 4.17 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 60% at t=5

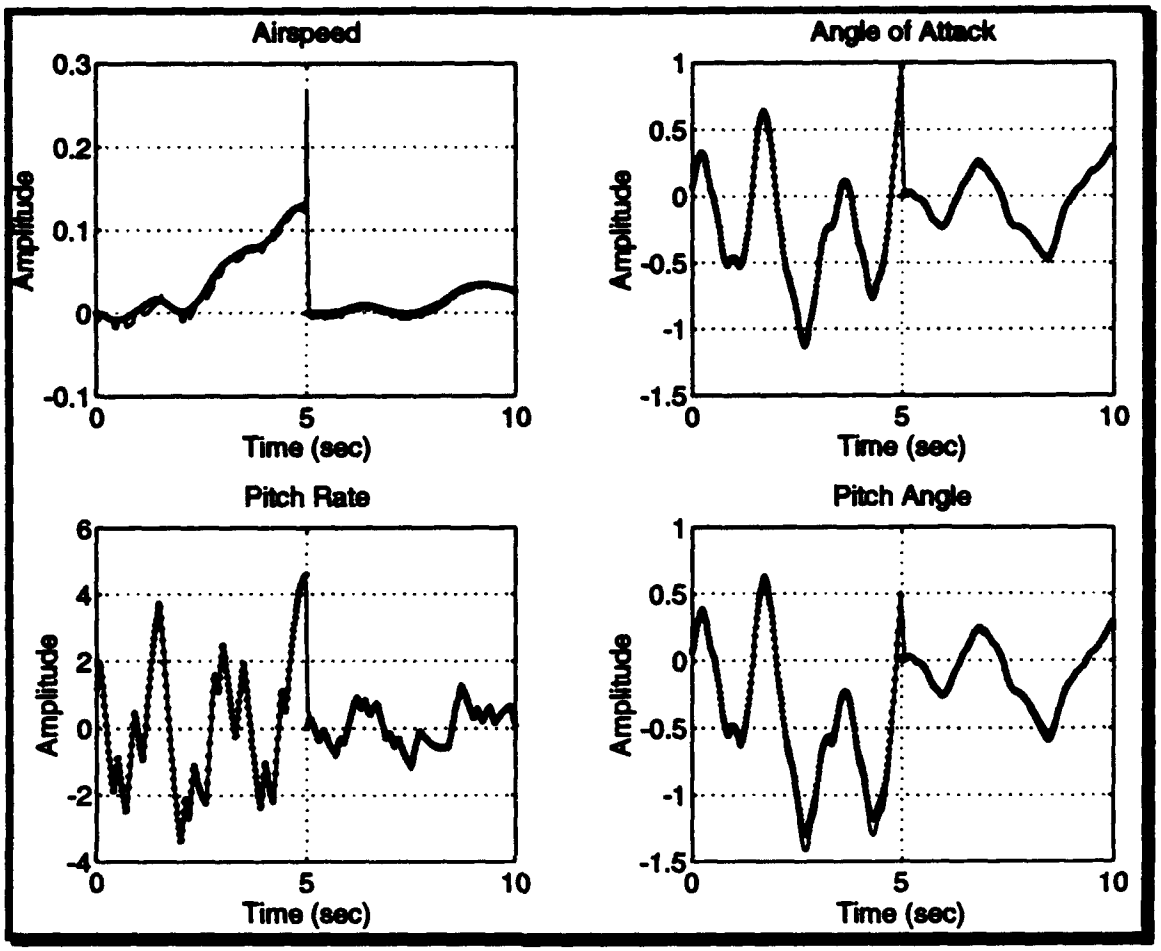


Figure 4.18 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 70% at t=5

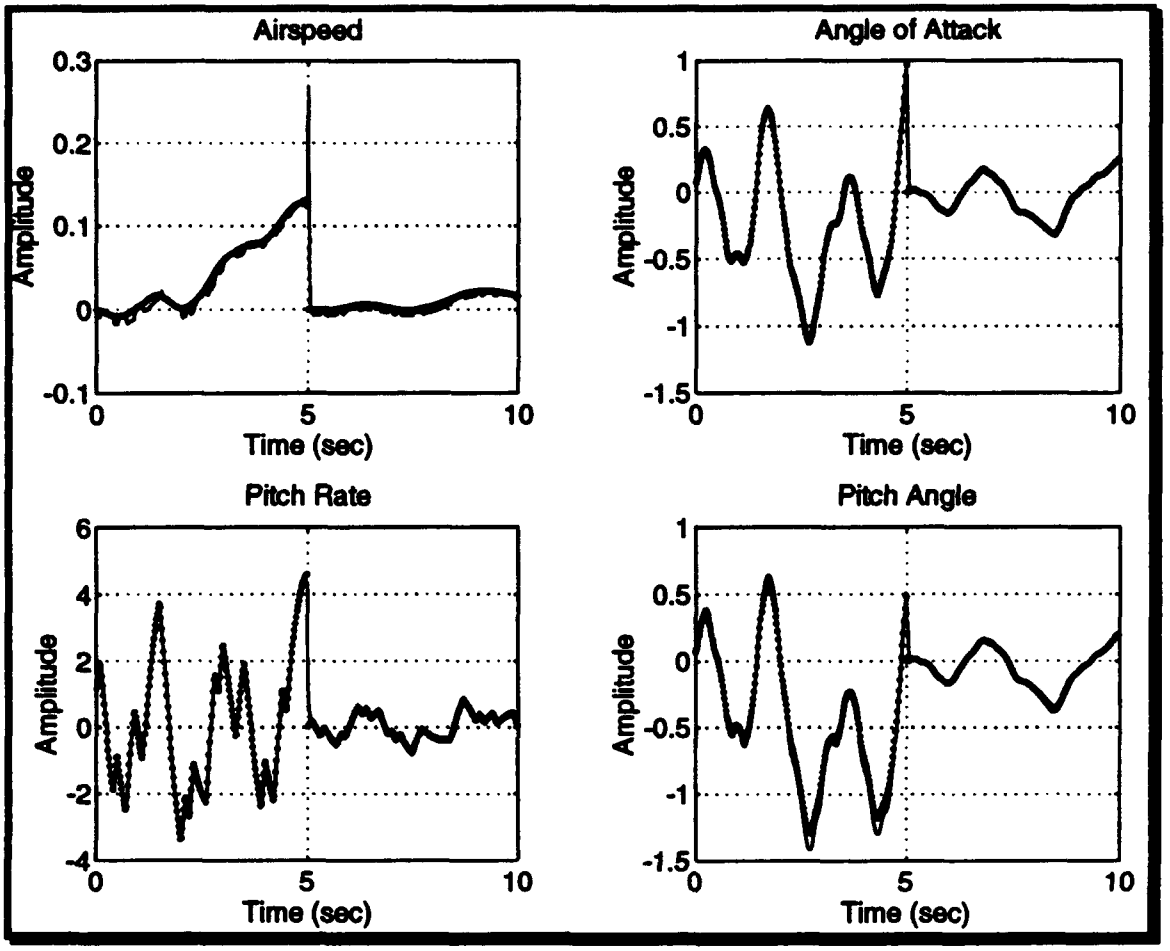


Figure 4.19 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 80% at t=5

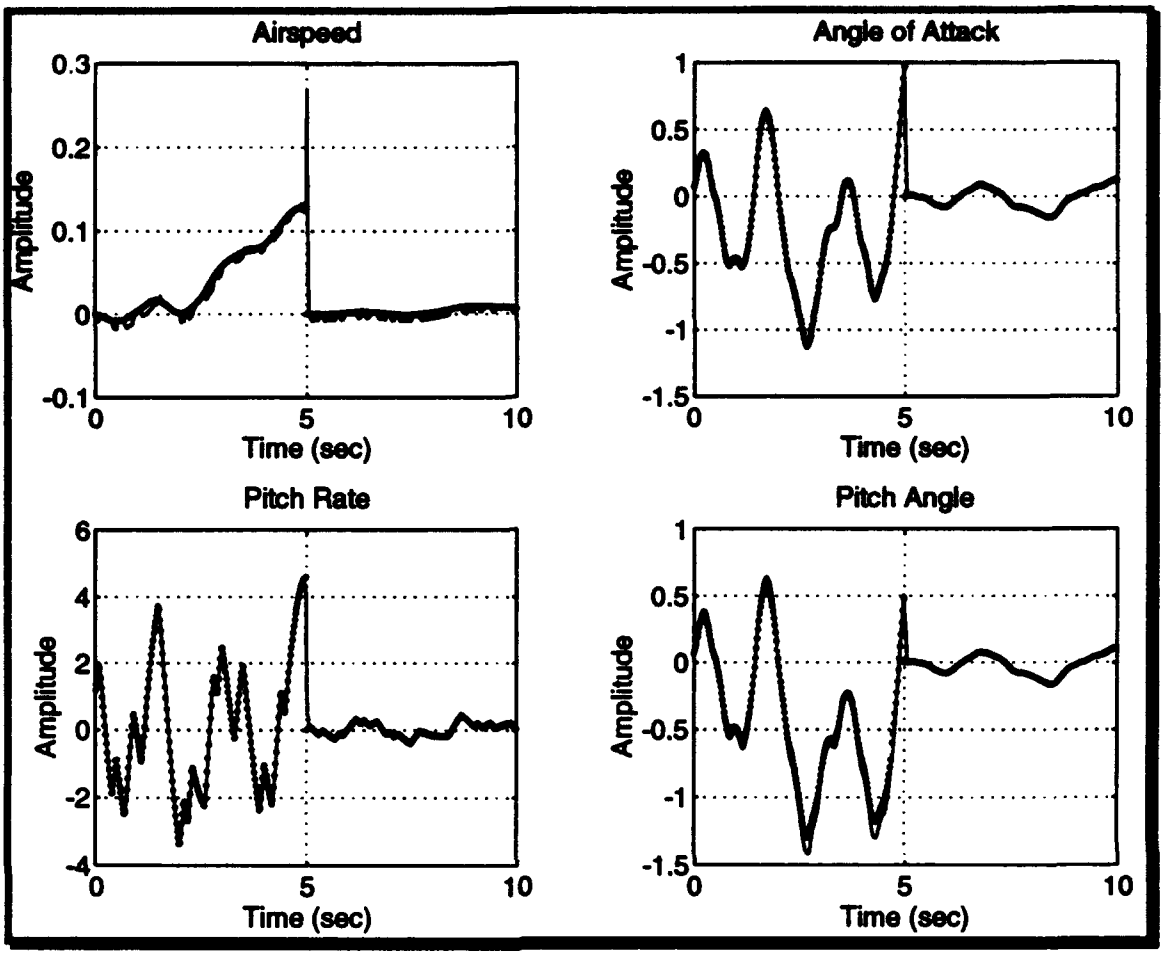


Figure 4.20 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 90% at t=5

In Figure 4.21, an expanded section the time plot with $M_{\delta e}$ reduced 60% is shown. Notice that the artificial neural network takes four time steps of 0.02 seconds or 0.08 seconds to correctly respond to the damage. Since the modeled A-4D is a fourth order plant, this is consistent. From the plot, a spike in airspeed is observed at the moment the damage is applied. Although the spike appears large, the vertical scale is small, therefore, the spike is insignificant. The spike is due to the fact that at t=5 seconds the network is using only one damaged data point and 3 undamaged data points to compute an output. Since all of the inputs to the network were high before the damage occurred, with a sudden drop at t=5 seconds, the airspeed output spiked high. In a practical sense, if an aircraft were to have sudden decrease in angle of attack, and pitch rate, airspeed would increase. This is what the artificial neural network has determined also. Since the network has determined that pitch rate and angle of attack are high frequency modes of the A-4D, a sudden change in these values over ride sudden changes in airspeed and pitch angle, which are low frequency modes. The network expects fast changes in pitch rate and angle of attack, and responds to them, rather than to fast changes in pitch angle and airspeed, which it does not expect. When the same data was run with the damage occurring at t=6 seconds, there was a downward spike in airspeed.

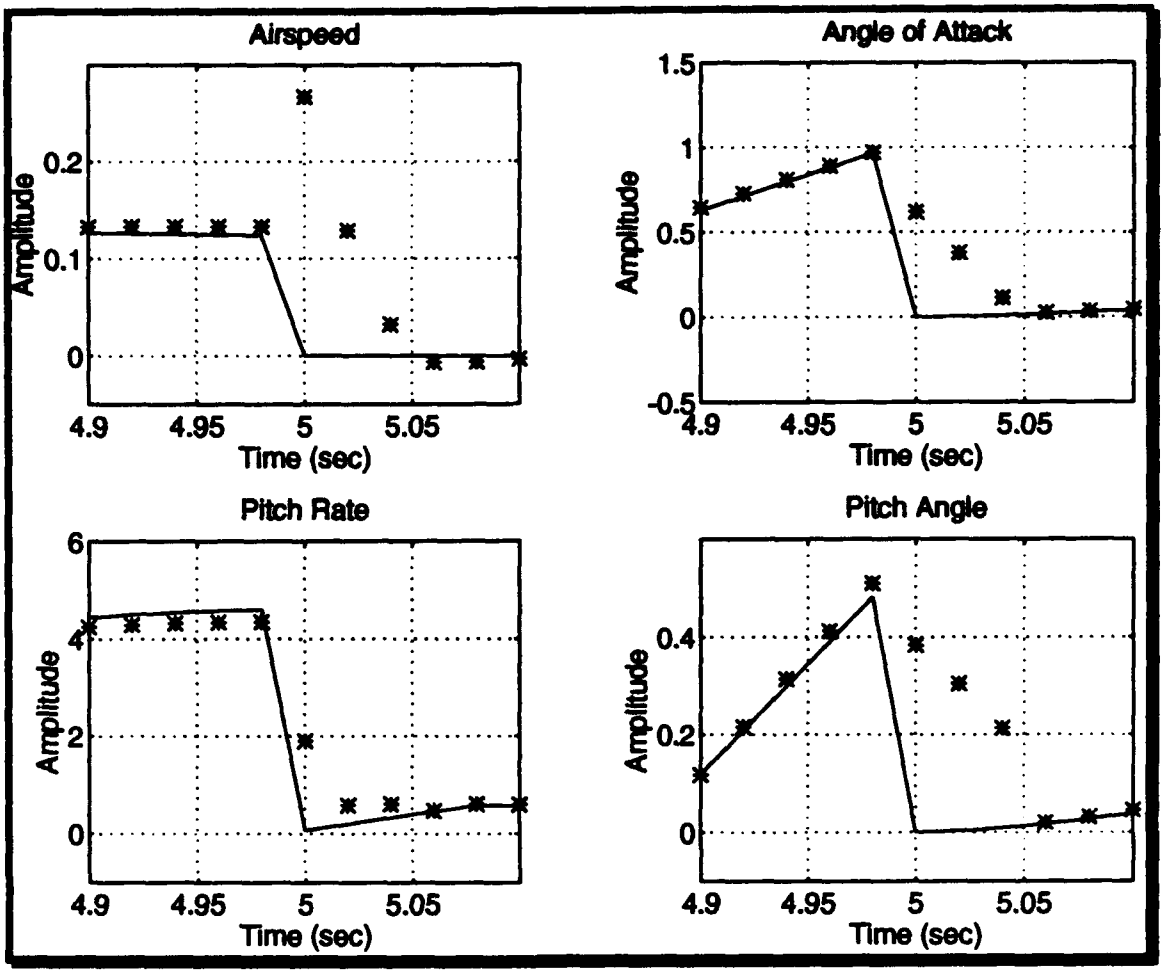


Figure 4.21 Network Trained on 70% Damage and Undamaged A-4D; $M_{\delta e}$ Reduced 60% at t=5 (Time expanded)

V. CONCLUSIONS

The designed artificial neural network was excellent at modeling a damaged aircraft. It interpolated between a good aircraft and a damaged aircraft, and it extrapolated from a single damage parameter. Only one damaged aircraft model was required to correctly model all of the damage mechanisms to $M_{\delta e}$ thus making the network a robust model. It determine when damage occurred automatically without any outside detection device and it responded rapidly and correctly to the new aircraft dynamics with the response time determined by the order of the plant and the sampling period.

Further training of the network on damage mechanisms effecting the rest of the A-4D's aerodynamic derivatives could lead to a network capable of responding to any possible aircraft damage, thus making the neural network an ideal robust aircraft model.

APPENDIX A: MATLAB PROGRAMS

```
%*****************************************************************
% Squashing.m
%*****************************************************************
% This programs plots the two squashing funtions, sigmoid and hyperbolic tangent
% Along with their derivaitves.
%*****************************************************************

subplot(211)x=-8:.1:8;
fx=1.0./(1+exp(-x));
plot(x,fx)
grid
xlabel('x')
ylabel('F(x)')
title('Sigmoid Function')

subplot(212)
x=-8:.1:8;
fx=exp(-x)./(1+exp(-x)).^2;
plot(x,fx)
grid
xlabel('x')
ylabel('F'(x)')
title('Derivative of Sigmoid Function')

print -deps sigmoid

subplot(211)
x=-4:.1:4;
fx=(exp(x)-exp(-x))./(exp(x)+exp(-x));
plot(x,fx)
grid
xlabel('x')
ylabel('F(x)')
title('Hyperbolic Tangent Function')

subplot(212)
x=-4:.1:4;
fx=1.0-((exp(x)-exp(-x))./(exp(x)+exp(-x))).^2;
plot(x,fx)
grid
xlabel('x')
ylabel('F'(x)')
title('Derivative of Hyperbolic Tangent Function')

print -deps hyperbolic
```

```
%*****************************************************************
%  A4_cond3.m
%*****************************************************************
%  Data for A-4D flight condition 3
%*****************************************************************
Thetao=0;
UUo=0;
Alphao=0;
qo=0;

h=35000;
M=.7;
Q=171;
W=17578;
m=546;
Ix=8030;
Iy=25900;
Iz=29250;
Ixz=-891;
alphatrim=5.9*pi/180;
g=32.174;
U=681;
Xu=-.01521;
Xa=-16.68;
Xd=0;
Zu=-.1013;
Za=-309.17;
Zadot=-.613;
Zd=-33.3;
Zq=0;
Mu=.000542;
Ma=-7.423;
Madot=-.206;
Mq=-.592;
Md=-11.33;
Yb=-83.2;
Yp=0;
Yr=0;
Yda=-.979;
Ydr=14.11;
Lb=-21.35;
Lp=-.816;
Lr=.523;
Lda=11.74;
Ldr=4.73;
Nb=11.3;
Np=.00845;
Nr=-.321;
Nda=.0837;
Ndr=-4.69;
```

```
%*****************************************************************
%  A4_matrix.m
%*****************************************************************
%  This program was written to construct the State Space matrices for the
%  longitudinal mode of the A4-D.  Using the dimensional derivatives, the
%  plant is constructed and digitized. States are:  u, alpha, q, theta.
%*****************************************************************
%  A Matrix

kk=U-Zadot;
A=
[Xu                          Xa/U              0                    -g*cos(Thetao)/U;
U*Zu/kk                      Za/kk             (U+Zq)/kk            -g*sin(Thetao)/kk;
U*Mu+(Madot*U*Zu)/kk         Ma+Madot*Za/kk    Mq+Madot*(U+Zq)/kk   0;
 0                           0                 1                    0];

%*****************************************************************
%  B Matrix

B=[Xd/U                      Zd/kk          Md+Madot*Zd/kk          0]';

%*****************************************************************
%  C Matrix

C=  [1 0 0 0
     0 1 0 0
     0 0 1 0
     0 0 0 1];

%*****************************************************************
%  D Matrix

D=zeros(size(B));

%*****************************************************************
%  Min sampling frequency = Nyquist freq(fn)= 2 X highest freq
%   present.   f=w/(2*pi);  fn=2*w/(2*pi)=w/pi.   for  w=10
%  fs= 10/pi. I will ues a decade higher, max freq 100 rad/sec.

ts=.02;
fs=1/ts;

%*****************************************************************
%  Digitizing the Continuous Plant with sampling frequency of 50 Hz

[Ad,Bd,Cd,Dd]=bilinear(A,B,C,D,fs);
```

```
%*****************************************************************
% A4_bode.m
%*****************************************************************
% This program was written to construct plot the bode response for the
% longitudinal mode for the A4-D. The bode of the two plants are then
% compared.
%*****************************************************************
clear
clg
%*****************************************************************
% Data

A4_cond3
A4_matrix
%*****************************************************************
% Setting up the frequency range.

points=800;                              %I have 200 pts/decade
w=logspace(-2,1,points);                 %3 decades
%*****************************************************************
% Check Frequency response of the A-4

subplot(221)
[mag  phase]=bode(A,B,C(1,:),D(1),1,w);
semilogx(w,20*log10(mag)),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of u')
axis([.01,10,-40,40])

subplot(222)
[mag  phase]=bode(A,B,C(2,:),D(2),1,w);
semilogx(w,20*log10(mag)),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of Alpha')
axis([.01,10,-40,40])

subplot(223)
[mag  phase]=bode(A,B,C(3,:),D(3),1,w);
semilogx(w,20*log10(mag)),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of q')
axis([.01,10,-40,40])

subplot(224)
[mag  phase]=bode(A,B,C(4,:),D(4),1,w);
semilogx(w,20*log10(mag)),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of Theta')
axis([.01,10,-40,40])

print -deps bode_A4
%*****************************************************************
% Check Frequency response of the Continuous Plant and the Digitized Plant

subplot(221)
```

```
[mag  phase]=bode(A,B,C(1,:),D(1),1,w);
[magd  phased]=bode(A,B,C(1,:),D(1),1,w);
semilogx(w,20*log10(mag),'-',w,20*log10(magd),'-'),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of u')
axis([.01,10,-40,40])

subplot(222)
[mag  phase]=bode(A,B,C(2,:),D(2),1,w);
[magd  phased]=bode(A,B,C(2,:),D(2),1,w);
semilogx(w,20*log10(mag),'-',w,20*log10(magd),'-'),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of Alpha')
axis([.01,10,-40,40])

subplot(223)
[mag  phase]=bode(A,B,C(3,:),D(3),1,w);
[magd  phased]=bode(A,B,C(3,:),D(3),1,w);
semilogx(w,20*log10(mag),'-',w,20*log10(magd),'-'),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of q')
axis([.01,10,-40,40])

subplot(224)
[mag  phase]=bode(A,B,C(4,:),D(4),1,w);
[magd  phased]=bode(A,B,C(4,:),D(4),1,w);
semilogx(w,20*log10(mag),'-',w,20*log10(magd),'-'),grid
 xlabel('Frequency (rad/sec)'),title('Bode Plot of Theta')
axis([.01,10,-40,40])

print -deps bode_both
```

```
%*****************************************************************
% Rand_seq.m
%*****************************************************************
% This program computes a random binary sequence with a frequency content of
% f_low to f_high (.001 to 5 Hz)
%*****************************************************************
% damp(eig(A))

%  Eigenvalue          Damping     Freq. (rad/sec)
%  -0.6245 + 2.6988i   0.2254          2.7701
%  -0.6245 - 2.6988i   0.2254          2.7701
%  -0.0088 + 0.0747i   0.1177          0.0752
%  -0.0088 - 0.0747i   0.1177          0.0752


ts=.02;

f_high=5;                   % 1 decade higher than highest freq of interest (.45 Hz)

f_low=.001;                 % 1 decade lower than lowest freq of interest (.012 Hz)

time=1/f_low;               % time is the period of the lowest freq, this sets the
                            % minimum length of the random signal

points=2*time*f_high;
Rbnm=2*round(rand(points+1,1)')-1;% creates a random binary sequence of +1 or
- 1

Rbnm=ones(5,1)*Rbnm;        % the random binary signal needs to be sampled at a
                            % faster rate than it was produced, therefore each
                            % value of Rbnm is duplicated by an amount equal to
                            % fs/(t_high*2) which must be a whole number, in
                            % this case 5.

Rbnm=Rbnm(:);               % converting the row matrix to a column matrix

t=0:ts:(points+1)/(2*5)-ts;
t=t';
```

```
%*****************************************************************
%  Rand_plot.m
%*****************************************************************
%  This program computes a random binary sequence with a frequency content of
%  f_low to f_high and plots 3 seconds.
%*****************************************************************

axis([1,2,3,4]),axis
ts=.02;

f_high=5;                  % 1 decade higher than highest freq of interest (.5 Hz)

f_low=.001;                % 1 decade lower than lowest freq of interest (.016 Hz)

time=1/f_low;              % time is the period of the lowest freq, this sets the
                           % minimum length of the random signal

points=2*time*f_high;
Rbnm=2*round(rand(points+1,1)')-1;  % creates a random binary sequence of +1 or
- 1

Rbnm=ones(5,1)*Rbnm;       % the random binary signal needs to be sampled at a
                           % faster rate than it was produced, therefore each
                           % value of Rbnm is duplicated by an amount equal to
                           % fs/(t_high*2) which must be a whole number, in
                           % this case 5.

Rbnm=Rbnm(:);              % converting the row matrix to a column matrix

% Plotting the first five seconds of the random binary sequence

t=0:ts:3;
t=t';
Rbnm=Rbnm(1:length(t),1);

Displot(t,Rbnm),grid
axis([0,3,-2,2]);

print -deps RBN

axis
```

```
%*****************************************************************
%  Rand_short_seq.m
%*****************************************************************
%  This program computes 5 sec of a random binary sequence with a frequency
% content of f_low to f_high.
%*****************************************************************
ts=.02;
points=50;                                  % 5 sec
Rbnm=2*round(rand(points+1,1)')-1;          % creates a random binary sequence of +1
or  -1

Rbnm=ones(5,1)*Rbnm;        % the random binary signal needs to be sampled at a
                            % faster rate than it was produced, therefore each
                            % value of Rbnm is duplicated by an amount equal to
                            % fs/(t_high*2) which must be a whole number, in
                            % this case 5.

Rbnm=Rbnm(:);               % converting the row matrix to a column matrix

t=0:ts:(points+1)/(2*5)-ts;
t=t';
```

```
% Spectrum_Rbnm.m
% Specplot2, plots the output of the SPECTRUM function along with the bode plot
% of the plant for comparison.
%  SPECPLOT(P,Fs,state,A,B,C,D), uses:
%
%  P          the output of SPECTRUM,
%  state              the desired state(1,2,3 or 4)
%  Fs                 the sample frequency,
%  A,B,C,D            State State representation of plant
%
%  to plot:
%
%     1. Transfer Function Magnitude from Spectral Analysis.
%     2. Transfer Function Magnitude from Bode of plant
%
[n,m]=size(P);
f  =  (1:n-1)/n*Fs/2;

plot(f,20*log10(abs(P(2:n,1)))),grid,xlabel('Frequency   (Hz)'),
axis([.01,10,-40,40])
```

```
%*************************************************************
%  Spectrum_NN_d.m
%*************************************************************
% This program does the spectrum analysis of the output of the Neural Network.
%*************************************************************

clear
A4_cond3
Md=.1*Md;
A4_matrix
load test.nnr
input=test(:,1);

u_out=test(:,25);
P=spectrum(input,u_out,25000);
Specplot1(P,50,1,A,B,C,D)
title('Bode Plot of u')

alpha_out=test(:,28);
P=spectrum(input,alpha_out,25000);
Specplot1(P,50,2,A,B,C,D)
title('Bode Plot of Alpha')

q_out=test(:,26);
P=spectrum(input,q_out,25000);
Specplot1(P,50,3,A,B,C,D)
title('Bode Plot of q')

theta_out=test(:,27);
P=spectrum(input,theta_out,25000);
Specplot1(P,50,4,A,B,C,D)
title('Bode Plot of Theta')

print -deps Md_1.eps
```

```
%*****************************************************************
% Gen_A4_data.m
%*****************************************************************
%  This program calls sub-programs in order to generate the learning,
%  testing and minmax file to be used by the neural network.
%*****************************************************************

clear
A4_cond3
A4_matrix
rand('seed',1)

%*****************************************************************
% Generate file "learn.nna" (learn data for NN)

Rand_seq
linsim('A4_diag',length(t)*ts);
Organize                                % returns output vector
minmaxlearn=[min(output_vector');max(output_vector')]';

fid=fopen('learn.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)

%*****************************************************************
%  Generate "test.nna" data (test data for NN)

rand('seed',0)

Rand_seq
linsim('A4_diag',length(t)*ts);
Organize
minmaxtest=[min(output_vector');max(output_vector')]';

fid=fopen('test.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)

%*****************************************************************
% Generate "minmax" data (minmax data for NN)

minmax_A4=[min([minmaxlearn  minmaxtest]');max([minmaxlearn  minmaxtest]')]';

fid=fopen('minmax_A4.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',minmax_A4)
fclose(fid)
```

```
%*****************************************************************
% Gen_damaged_data.m
%*****************************************************************
%  This program calls sub-programs in order to generate the learning,
%  testing and minmax file to be used by the neural network for the damaged A4.
%*****************************************************************

clear
A4_cond3
rand('seed',1)

%*****************************************************************
% Generate file "learn_damaged.nna" (learn damaged data for NN)

Md=.3*Md;    % Make elevator 30% of original size.  Elevator reduced by 70%
A4_matrix
Rand_seq
linsim('A4_diag',length(t)*ts);
Organize
minmaxlearn=[min(output_vector');max(output_vector')]';

fid=fopen('learn_damaged.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)

%*****************************************************************
% Generate "test_damage.nna" data (test data for NN)

rand('seed',0)

Rand_seq
linsim('A4_diag',length(t)*ts);
Organize
minmaxtest=[min(output_vector');max(output_vector')]';

fid=fopen('test_damage.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)

%*****************************************************************
% Generate "minmax" data (minmax data for NN)

minmax_damaged=[min([minmaxlearn  minmaxtest]');max([minmaxlearn
minmaxtest]')]';

fid=fopen('minmax_damaged.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',minmax_damaged)
fclose(fid)
```

```
%*****************************************************************
% A4_step.m
%*****************************************************************
% This program was written to look at the step responses of all 4 states with different
% values of Md.
%*****************************************************************
A4_cond3
A4_matrix
T=1:1:400;
[y1,x1,t1]=step(A,B,C,D,1,T);

A4_cond3
Md=.7*Md;
A4_matrix
[y2,x2,t2]=step(A,B,C,D,1,T);

A4_cond3
Md=.3*Md;
A4_matrix
[y3,x3,t3]=step(A,B,C,D,1,T);

subplot(221)
plot(T,y1(:,1),'-',T,y2(:,1),':',T,y3(:,1),'-.'),  ylabel('Amplitude')
xlabel('Time (sec)'),title('Step Response in Airspeed'), grid

subplot(222)
plot(T,y1(:,2),'-',T,y2(:,2),':',T,y3(:,2),'-.'),  ylabel('Amplitude'),
xlabel('Time (sec)'),title('Step Response in Angle of Attack'),grid

subplot(223)
plot(T,y1(:,3),'-',T,y2(:,3),':',T,y3(:,3),'-.'),  ylabel('Amplitude'),
xlabel('Time (sec)'),title('Step Response in Roll Rate'), grid


subplot(224)
plot(T,y1(:,4),'-',T,y2(:,4),':',T,y3(:,4),'-.'),  ylabel('Amplitude'),
xlabel('Time (sec)'),title('Step Response in Pitch Angle'), grid

print step_plot -deps

subplot(111)
```

```
%*****************************************************************
% Organize.m
%*****************************************************************
% This program organizes the data and computes the output file to be used by
% the neural network.
%*****************************************************************

input_Rbnm=   [[Rbnm;0;0;0] [0;Rbnm;0;0] [0;0;Rbnm;0] [0;0;0;Rbnm]];
input_u=      [[u;0;0;0]    [0;u;0;0]    [0;0;u;0]    [0;0;0;u]];
input_q=      [[q;0;0;0]    [0;q;0;0]    [0;0;q;0]    [0;0;0;q]];
input_theta=  [[theta;0;0;0] [0;theta;0;0] [0;0;theta;0] [0;0;0;theta]];
input_alpha=  [[alpha;0;0;0] [0;alpha;0;0] [0;0;alpha;0] [0;0;0;alpha]];
output=       [[u;0;0;0]    [q;0;0;0]    [theta;0;0;0] [alpha;0;0;0]];

clear Rbnm
clear u
clear q
clear theta
clear alpha

output_vector=[[input_Rbnm],[input_u],[input_q],[input_theta],[input_alpha],[outp
ut]];
output_vector=output_vector(6:length(output_vector)-3,:)';

clear input_Rbnm
clear input_u
clear input_q
clear input_theta
clear input_alpha
clear output
```

```
%*****************************************************************
% Gen_both_data.m
%*****************************************************************
% This program loads the learn file generated by the undamaged plant and
% the learn file generated by the damaged plant and interleaves them.
%*****************************************************************
clear
load learn.nna
load learn_damaged.nna
load minmax_A4.nna
load minmax_damaged.nna

learn=learn';
learn_damaged=learn_damaged';
%*****************************************************************
% Generate file "learn_both.nna" (combined learning data for NN)

[m,n]=size(learn);
output_vector=zeros(m,2*n);
mix=20;

for  l=1:mix;
     output_vector(:,l:2*mix:2*n)=learn(:,l:mix:n);
     output_vector(:,l+mix:2*mix:2*n)=learn_damaged(:,l:mix:n);
end

fid=fopen('learn_b.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)

minmaxfiles=[minmax_A4;minmax_damaged];

minmax_b=[min(minmaxfiles)   max(minmaxfiles)];

fid=fopen('minmax_b.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',minmax_b)
fclose(fid)
```

```
%*****************************************************************
% Specplot1.m
%*****************************************************************
% Specplot1, plots the output of the SPECTRUM function along with the bode plot
% of the plant for comparison.
%  SPECPLOT1(P,Fs,state,A,B,C,D), uses:
%
%  P              the output of SPECTRUM,
%  state          the desired state(1,2,3 or 4)
%  Fs             the sample frequency,
%  A,B,C,D        State State representation of plant
%
%  to plot:
%
%     1. Transfer Function Magnitude from Spectral Analysis.
%     2. Transfer Function Magnitude from Bode of plant
%
%*****************************************************************
function specplot1(P,Fs,state,A,B,C,D)

[n,m]=size(P);
f  =  (1:n-1)/n*Fs/2;

[mag phase w]=bode(A,B,C(state,:),D(state));

subplot(2,2,state)

semilogx(2*pi*f,20*log10(abs(P(2:n,4))),'-',w,20*log10(mag),'-'),grid
 xlabel('Frequency (rad/sec)')
axis([.01,10,-40,40])
```

```
%*****************************************************************
%  Specplot2.m
%*****************************************************************
%  Specplot2, plots the output of the SPECTRUM function along with the bode plot
%  of the plant for comparison.
%   SPECPLOT(P), uses:
%
%    P, the output of SPECTRUM
%
%  Transfer Function Magnitude from Spectral Analysis.
%
%*****************************************************************

function  specplot2(P)
Fs=50;
[n,m]=size(P);
f  =  (1:n)/n*Fs/2;
f=f';

loglog(f,P(:,1)),grid
  xlabel('Frequency  (Hz)')
ylabel('Magnitude')
axis([.001,30,.0001,20])
```

```
%*****************************************************************
% Gen_response_data.m
%*****************************************************************
% This program calls sub-programs in order to generate 10 seconds of test data
% for plotting the output of the undamanged A4.
%*****************************************************************
% Generate 10 seconds of undamaged A4

clear
A4_cond3
A4_matrix
rand('seed',0)
%*****************************************************************
ts=.02;
points=100;                    % 10 sec
Rbnm=2*round(rand(points+1,1)')-1;   % the random binary signal needs to be
                                     % sampled at a faster rate than it was
                                     % produced, therefore each value of Rbnm
                                     % is duplicated by an amount equal to
                                     % fs/(t_high*2) which must be a whole
                                     % number, in this case 5.

Rbnm=Rbnm(:);                  % converting the row matrix to a column matrix

t=0:ts:(points+1)/(2*5)-ts;
t=t';

linsim('A4_diag',length(t)*ts);
Organize                       % returns output_vector
switched=output_vector;
%*****************************************************************

fid=fopen('switched.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',switched)
fclose(fid)
```

```
%*****************************************************************
% Gen_Md_data.m
%*****************************************************************
% This program is used to generate the test files with various values of Md.
%*****************************************************************

clear
A4_cond3
rand('seed',0)
%*****************************************************************
percent = .1                                    % change .1 for 1.0 to 0.1 as necessary
Md=percent*Md;
A4_matrix
Rand_seq
linsim('A4_diag',length(t)*ts);
Organize

fid=fopen('damaged1.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',output_vector)
fclose(fid)
```

```
%*****************************************************************
% Gen_switched.m
%*****************************************************************
% This program generates 5 seconds of undamaged A4 data followed by 5 seconds
% of damaged A4 data.
%*****************************************************************

%
clear
A4_cond3
A4_matrix
rand('seed',0)

Rand_short_seq
linsim('A4_diag',length(t)*ts);

Rbnm1=Rbnm;
u1=u;
q1=q;
theta1=theta;
alpha1=alpha;

Md=0.1*Md;
A4_matrix
Rand_short_seq
linsim('A4_diag',length(t)*ts);

Rbnm=[Rbnm1;Rbnm];
u=[u1;u];
q=[q1;q];
theta=[theta1;theta];
alpha=[alpha1;alpha];

Organize                          % returns output_vectorP
switched=output_vector;

fid=fopen('switched.nna','w')
fprintf(fid,'%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g %g %g %g\n',switched)
fclose(fid)
```

```
%*****************************************************************
%  Switch_plot.m
%*****************************************************************
%  Thsis program plots the output of the neural network after 10 seconds of data
%  have been supplied to the network.
%*****************************************  ************************
clear
ts=.02;
load switched.nnr;

u_out=switched(1:500,25);
alpha_out=switched(1:500,28);
q_out=switched(1:500,26);
theta_out=switched(1:500,27);

U=switched(1:500,21);
Alpha=switched(1:500,24);
Q=switched(1:500,22);
Theta=switched(1:500,23);

T=0:ts:10-ts;

subplot(221)
plot(T,u_out,'-',T,U,'.'),  ylabel('Amplitude')
xlabel('Time  (sec)'),title('Airspeed'),  grid

subplot(222)
plot(T,alpha_out,'-',T,Alpha,'.'),  ylabel('Amplitude'),
xlabel('Time  (sec)'),title('Angle  of  Attack'),grid

subplot(223)
plot(T,q_out,'-',T,Q,'.'),  ylabel('Amplitude'),
xlabel('Time  (sec)'),title('Pitch  Rate'),  grid

subplot(224)
plot(T,theta_out,'-',T,Theta,'.'),  ylabel('Amplitude'),
xlabel('Time  (sec)'),title('Pitch  Angle'),  grid

print -deps switched1
subplot(111)
```

APPENDIX B: NEURALWORKS SETUP SPECIFICS

In order to provide a means of repeating the thesis research, the specific values used in the setup of NeuralWorks are described below.

The learning rate coefficient, η was selected to be 0.3 for the hidden layer and 0.15 for the output layer. The input layer is a buffer and has no learning rate coefficient. The momentum was selected to be 0.4, the learning rate coefficient ratio was selected to be 0.5., the offset was chosen to be 0.1, and the transition point was chosen to be 10,000. Although, a transition point of 50,000 (the length of the data files) was initially chosen, the transition point of 10,000 provided better results during learning. "Hyperbolic tangent" was selected as the activation function (bipolar data), and "delta rule" was selected as the Learn Rule. Although the presence of noise during learning supposedly helps the network to generalize to the test data, this did not prove to be the case. As a result the network was trained and tested without noise added by NeuralWorks. An epoch of 1 was chosen and "fast learning," was also selected. (Although epoch does not apply to the Delta Rule during *learning*, it does apply to the output files or instruments created by NeuralWorks, i.e. an epoch of 5 and an *input* file 50000 long will generate an *output* of file 10000 long. This will cause the output data to be averaged over five inputs resulting in a lost of accuracy.)

For "I/O" (input/output) parameters, "file sequence" was selected for both the learn and test files. This setting requires the data files to be read in sequential order, rewinding once the end of the file is encountered. Although random file selection would appear to have been a better choice, the network always saturated during learning. This could have been due to the fact the input file was longer than 32768 records. According to NeuralWorks, data from files less than this size are automatically selected randomly until all of the records are presented and then the file is rewound, ensuring each data record was used the same number of times. Data from files greater that this size have their records selected randomly in a gausian distribution.

The minmax table is used by the network to normalize the data. As mentioned earlier, the data sets generated by Matlab were too large for NeuralWorks to automatically generate the required minmax table, so a minmax table was generated using Matlab. Selecting $\pm$ 0.8 in "network ranges" scales the squashing function for the network. Descaling of the output is done automatically. If the data is not scaled properly, the network well saturate and learning will cease.

LIST OF REFERENCES

[DROR 92] Dror, Shahar, "Identification and Control of Non-Linear Time-Varying Dynamical Systems Using Artificial Neural Networks," Doctoral Thesis, Naval Postgraduate School, Monterey, California, 1992.

[GROS 73] Grossberg, S., *Studies in Applied Mathematics*, New York: Academic Press, 1973.

[HECH 89] Hecht-Nielsen, Robert, *Neurocomputing*, Addison-Wesley, 1991.

[MCRU 93] McRuer (Duane), Ashkenas (Irving), and Graham (Dustan), *Aircraft Dynamics and Automatic Control*, Princeton University Press, Princeton, New Jersey, 1973.

[MINS 69] Minsky, Marvin, and Papert, Seymore S., *Preceptrons*, MIT Press: Cambridge MA, 1969.

[MOLE 90] Moler, C., J. Little, and S. Bayert, *MATLAB User's Manual*, The Mathworks, Inc., 1990.

[NEUR A 93] NeuralWare, *Neural Computing*, Technical Publications Group, 1993.

[NEUR B 93] NeuralWare, *Reference Guide*, Technical Publications Group, 1993.

[NEUR C 93] NeuralWare, *Using Neuralworks*, Technical Publications Group, 1993.

[PARK 85] Parker, David P., *Learning -Logic*, Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, 1985.

[RUMM 85] Rumelhart, D.E. and McClelland, J.L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[SCOT 89] Scott, Russel W., "Application of Neural Networks to Adaptive Control, "Engineer's Thesis, Naval Postgraduate School, Monterey, California, 1989.

[WASS 89] Wasserman, Philip D., *Neural Computing Theory and Practice,* Van Nostrand Reinhold, 1989.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, California 93943-5100	2
3. Chairman, Code AA Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	4
4. D.J. Collins, Code AA/Co Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	2
5. Richard M. Howard, Code AA/Ho Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	1
6. H. Titus, Code EC/Ts Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	1
7. LCDR(s) Clifford A. Brunger 6941 Enborne Lane San Diego, California 92139	5