Neural Network Identification of Quarter-Car Passive and Active Suspension Systems

by

Michael Tran

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED:

Dr. H. F. VanLandingham, Chairman

Dr. S. Bingulac

Dr. W.R. Cyre

March, 1992

Blacksburg, Virginia

Neural Network Identification of Quarter-Car Passive and Active Suspension Systems

by

Michael Tran

Dr. H. F. VanLandingham, Chairman

Electrical Engineering

(ABSTRACT)

Much research effort has been done on the design of active suspension systems based on a quarter-car model with both full-state feedback and incomplete state feedback controllers to optimize the passenger ride comfort, road handling and car controlling. Linear stochastic optimal control will be employed to design an active controller in vehicle active suspension model. The active suspension model will be simulated and compared with the car's passive suspension model. Backpropagation neural networks then will be used to identify the passive suspension model with full state or incomplete state measurements. The accurate identification of the suspension system employing neural networks is used to reduce the number of sensors needed over full-state measurements.

Acknowledgement

The author would like to express his deepest gratitude to his advisor, Dr. Hugh F. VanLandingham for his valuable time, advice, support, and guidance to make this paper possible.

The author wishes to thank Dr. Stanoje Bingulac for serving on his advisory committee. Dr. Bingulac is known for his teaching excellence, and has been helpful to the author. The author also wishes to thank Dr. Cyre for serving on his advisory committee. Dr. Cyre has given the author much helpful advise.

The author would like to dedicate this work to his parents, brothers, and sisters for their love, support, and encouragement.

The author would like to thank Thao Tran, Hoai Pham, Dinh Nguyen, Phuong Huynh, and other friends for making his years at Virginia Tech the most enjoyable ones.

Table of Contents

List of Figures

1.0 Introduction

Active suspension has recently attracted increasing attention in the automotive industry for its significant contribution to the improvement of passenger ride comfort, road handling, and car control. Stochastic optimal control theory has been used to design active controllers with both full state feedback controllers and incomplete state feedback controllers. Neural networks, which have emerged and developed strongly in recent years, can be used to identify both linear dynamical systems and nonlinear dynamical systems. Backpropagation neural networks will be employed to identify both passive and active suspension systems.

1.1 Suspension Systems

Passenger ride comfort, road handling, and car controlling performance are among the requirements of designing an effective car suspension system. Car suspension must be able to isolate environmental disturbances and vibration to provide passenger ride comfort. There are basically three types of suspension systems: passive, semi-active, and active.

The quarter car model has been widely used for design and analysis. This simple model does offer most basic features:

- A proper representation of the problem of controlling wheel load variations.
- Suspension forces are properly applied between car mass and wheel mass.

1.1.1 Passive Suspension Systems

A car passive suspension system consists of springs and passive dampers. Figure 1 shows a simple two degrees of freedom suspension model. In this system, the tire is modelled as a linear spring k_t and fixed damper b_t and is assumed to contact with the road continuously. The passive suspension system has some limitations. The ground clearance and the dynamic characteristics change with various static loadings. Passive systems only generate forces in response to the local relative motion, storing energy in the springs and dissipate energy in the dampers [1].

A heavily damped suspension can control the amplitude of vibration response when the frequency of the base disturbances is around the natural frequency of the system. The vibration performance of the heavily damped suspension is degradated considerably in the higher frequency range [2].

Non-linear springs, air springs, self-levelling systems, and shock absorbers can be used to improve some of the above limitations. Non-linear springs often result in the suspensions operating in the non-linear range and much stiffer systems with degradation of vibration isolation on rough roads combined with high speeds [1].

<fig1.drw>

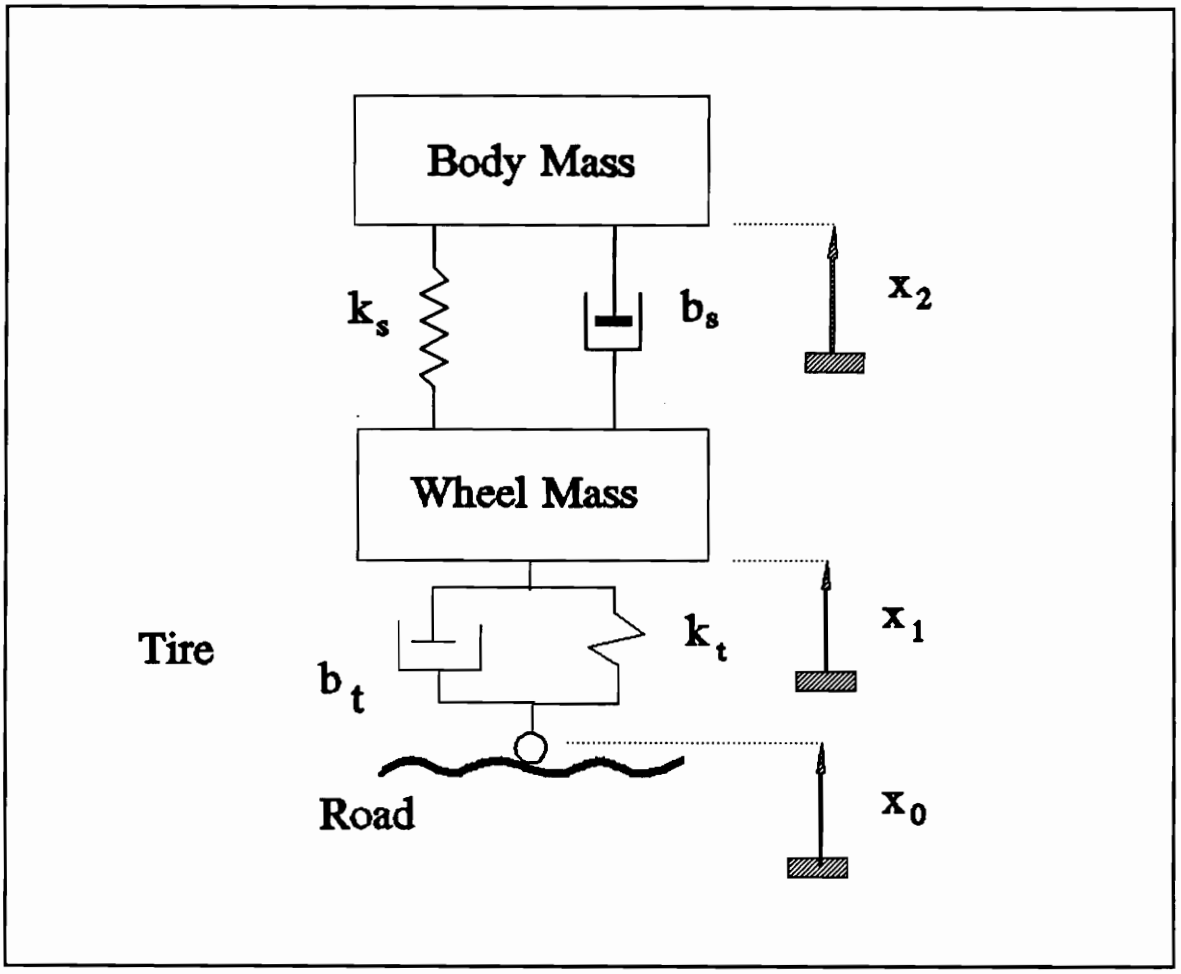

Figure 1 : Model of a passive suspension system.

Shock absorbers are designed to limit the damping force at higher velocities. The self-levelling system with time delays is designed to compensate for variations in static load [3]. However, none of the above devices can isolate the environmental vibration and provides passenger ride comfort and handling performance at the same time.

1.1.2 Active Suspension Systems

Figure 2 shows a two degrees of freedom model of the active suspension system consisting of actuators, springs, and passive dampers. Actuators are used to generate needed forces between car body and car axle in addition to supported springs and dampers.

Active suspension systems provide the following advantages as listed in [4]:

- small static deflections for maximum passenger ride comfort at low natural frequency
- low dynamic deflections under transient excitation
- high speed response to any inputs
- the maintenance of suspension characteristics under various loadings

These are made possible because active suspension can generate continuous forces which are functions of many measurements depending on the employed control laws [5]. Therefore, active suspension systems can provide passenger ride comfort while adapting to different levels of loadings, and road roughness. Slow active suspension systems can be viewed as an active system with the actuator control bandwith reduced to cover the normal range of body resonant frequencies

<fig2.drw>

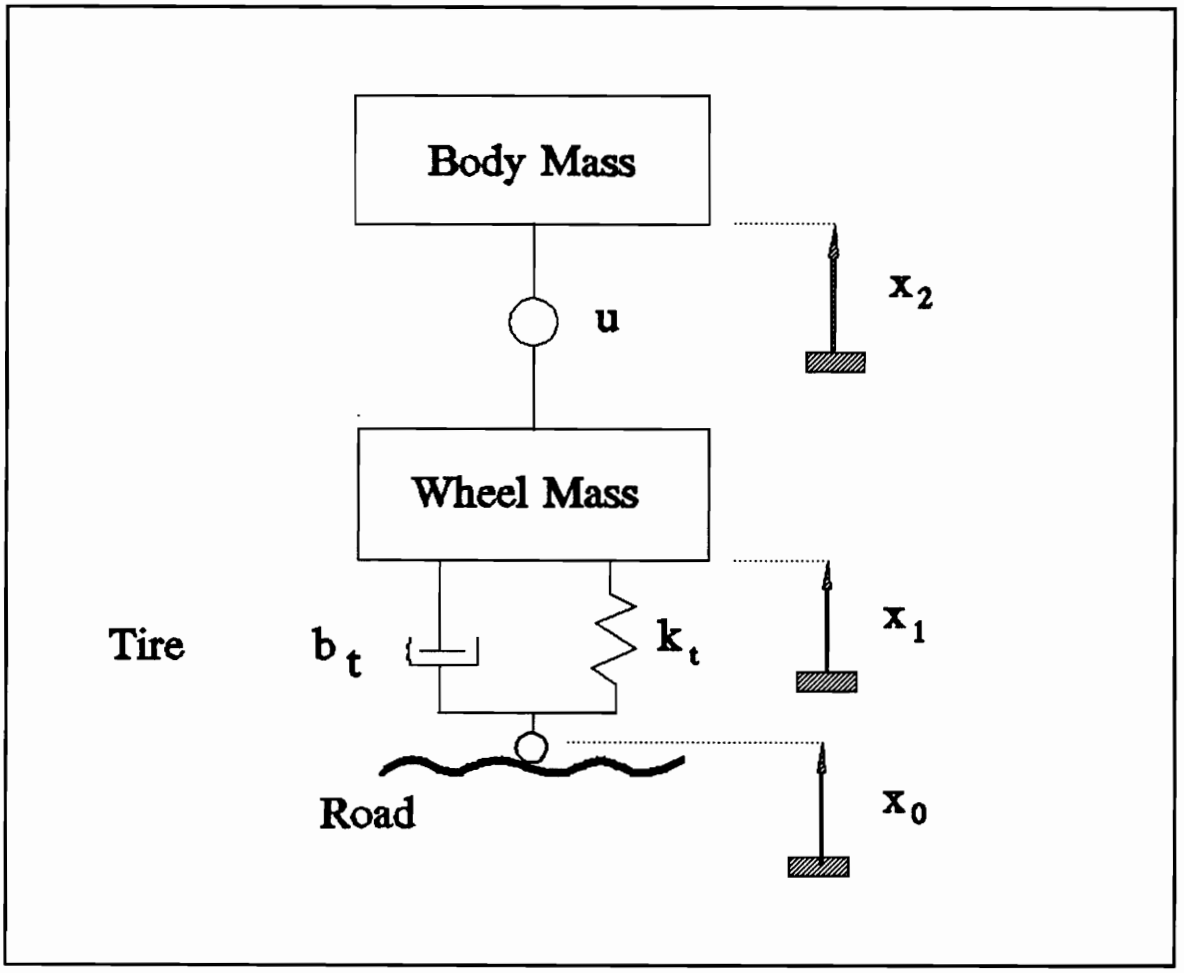

Figure 2 : Model of an active suspension system.

and as far as frequency range of steering control, but not exceeding wheel hop frequencies [3].

1.1.3 Semi-active Suspension Systems

Semi-active suspension systems consist of continuously variable dampers in parallel with springs to support the static loadings. Figure 3 shows a two degrees of freedom model of the semi-active suspension. The variable dampers provide a way to improve the passenger ride comfort of the passive suspension limitation of fixed dampers.

The variable damper force is set to zero when there is a sign opposite between actual damping forces and the desired active force [6]. This semi-active 'on-off' control can yield a large jerk between the on and the off states as in bang-bang control systems.

In general, the active suspension system has more advantages than the passive and the semi-active suspension systems. However, its realization can be complicated and expensive.

<Fig3.drw>

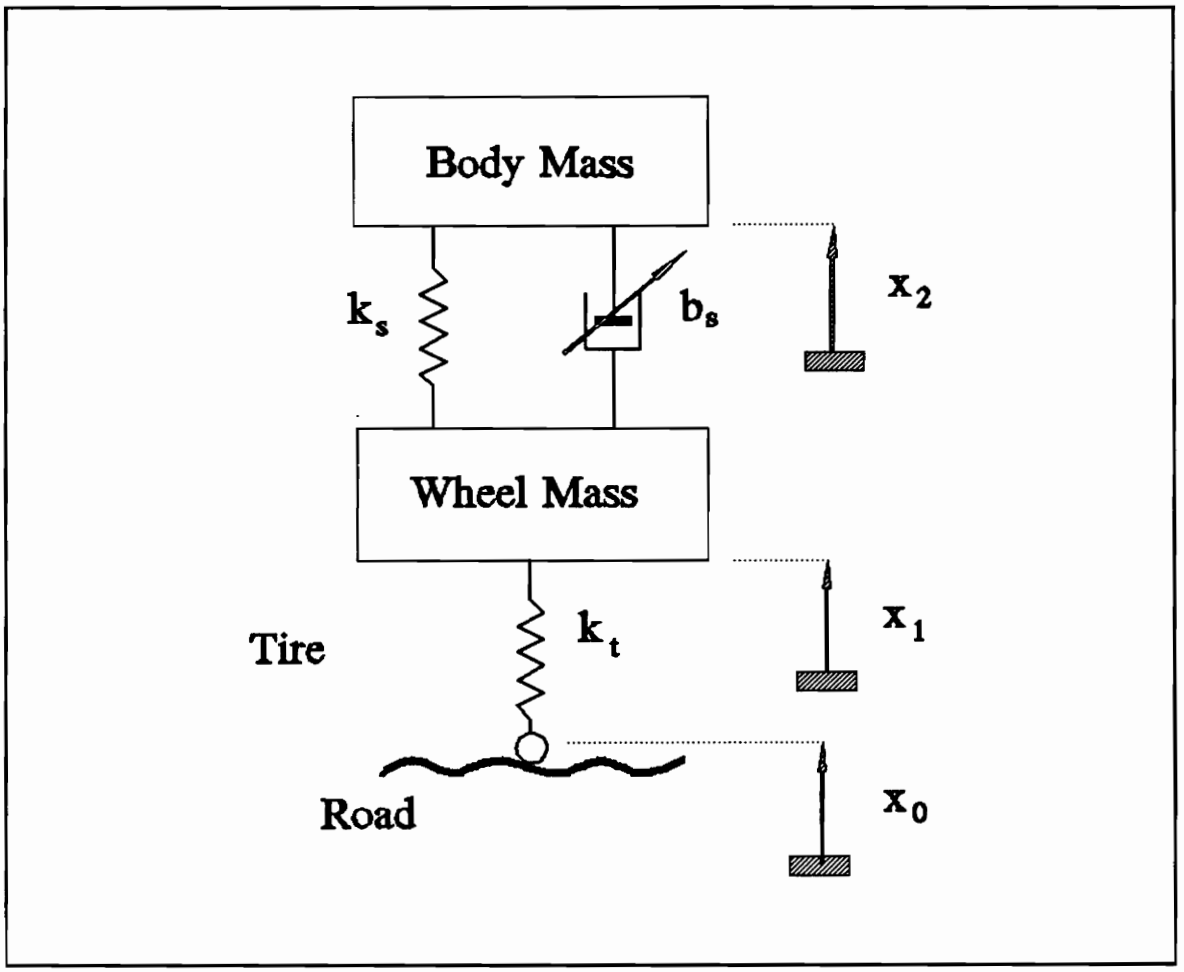

Figure 3 : Model of a semi-active suspension system.

1.1.4 Road Surface Representation

Road profile is assumed to be continuous and random with a power spectral density (P.S.D.) varying inversely with the square of frequency. For a variety of road surfaces, the P.S.D. of the corresponding time signal input to the suspension is given by [7]

$$\Phi(w) = C\frac{v}{w^2} \tag{1.1}$$

where C is a road roughness constant,

v is the velocity of the car.

Equation 1.1 represents the power spectral density of integrated white noise. The advantage of above equation is that all road roughness is represented by a single parameter. For the linear system case, the mean squared value of any output signal is related to the integral squared value of the corresponding output signal due to a unit step input [8]. The mean square response to random signal is given by [9]

$$\left\langle y_r^2 \right\rangle = Cv \int_0^\infty y_s^2(t)dt \tag{1.2}$$

where y_r is the output for a random excitation

y_s is the output for a unit step input.

From [9], the above equation shows that for a given velocity v, the above equation shows that if the system is optimal for a unit step input to minimize the integral on the right hand side, it will also be optimal for the random signal input given by equation (1.1).

1.2 Neural Networks

The working of the human brain is still a mystery to man. The brain is so powerful that it can perform a wide variety of solving many problems from thinking, talking, remembering, feelings, and learning. This powerful device of the human has amazed and inspired many scientists to attempt modeling its operations. In the brain, a neuron is the main cellular unit of the nervous system. Each neuron receives and combines signals from many other neurons and produces an output signal to the axon to perform certain actions. Neurocomputing has emerged from this inspiration.

1.2.1 History of neural networks

In 1943, a neurobiologist, Warren McCulloch, and a statistician, Walter Pitts, published a paper " A logical Calculus of Ideas Imminent in Nervous Activity". This paper has inspired a number of prominent scientists to apply the idea to different areas. Frank Rosenblatt focused his interest on computations of the eye which led to his invention of the perceptron. Marvin Minsky focused his interest on Artificial Intelligent and expert systems. John von Neumann viewed the idea as "electronic brains".

In 1957, Frank Rosenblatt developed an element called a "perceptron" which led to the wave of interest in neurocomputing. The perceptron is a pattern classification system which can identify both abstract and geometric patterns. The perceptron is capable of categorizing patterns despite of noise in the input. There are some limitations. The perceptron did not have state structures and is not

capable of solving the Exclusive OR function as pointed out by Minsky and Papert. In 1959, Bernard Widrow developed an adaptive linear element, "adaline". This development found many applications in speech recognition, character recognition, and later adaptive filters that eliminate echoes on phone lines.

In 1969, Minsky and Papert published the book, "Perceptrons", to show the incapability of a perceptron to handle inputs that are visually nonlocal. This led to the downhill of neurocomputing. However, there were some researchers who continued to work in this area. James Anderson developed a linear associator. The linear associator is a parallel, distributed associative model. Anderson later extended the linear associtor to the new development of the brain-state-in-a-box (BSB) model. Teuvo Kohonen in Finland worked on adaptive learning and associative memories in the 70's. His analysis of a large class of local adaptation rules in which weights are modified depending on the previous weight value and the post and pre-synaptic values. His another contribution is the principle of competitive learning in which processing elements compete to respond the input stimulus. The winner is allowed to modify itself to respond to that input. It is an unsupervised learning and led to his development of the self-organizing map neural network. Instead of learning on supervised examples of the mapping, the self-organizing map learns a continuous topological mapping $f : B \subset R^n \rightarrow C \subset R^m$ by mean of self-organization. Grossberg made an important contribution in the proof of the Cohen-Grossberg theorem concerning stability during recall of a general class of networks. This stability expressed in term of Lyapunov energy level can be shown that the response to any external stimulus to these networks will converge to an equilibrium.

In 1982, John Hopfield presented to the National Academy of Science his "Hopfield Model" or "crossbar associative network". This neural computing model consists of interconnected processing elements that seek an energy

minimum. This created a new wave of neural computing research interest in many institutions.

1.2.2 Neural Network Architecture

In general, the artificial neural network consists of many processing elements (nodes), numerous interconnections (weights) among processing elements, and / or some hidden layers. Figure 4 shows the architecture of a typical neural network and one of its processing elements. Each processing element sums many inputs received from either external source or from other processing elements. The combined signal goes through a transfer function to produce the output signal to be received by elements of the next layer or to the outside network. The connections between processing elements represent the weights or the synaptic strength of the neural connections.

1.2.3 Neural Network Operation

There are two distinct operation modes of neural networks. They are the Learning mode and the Recall mode.

The learning mode is the process of modifying the network weight connections in response to a set of inputs in such way that the outputs produced by the network should follow the desired outputs to an acceptable level of error. There are three types of learning : "supervised", "unsupervised", and "reinforcement". When the desired outputs are used to train the network, the learning process is called "supervised". When the outputs are not presented to train the network, the

<Fig4.drw>

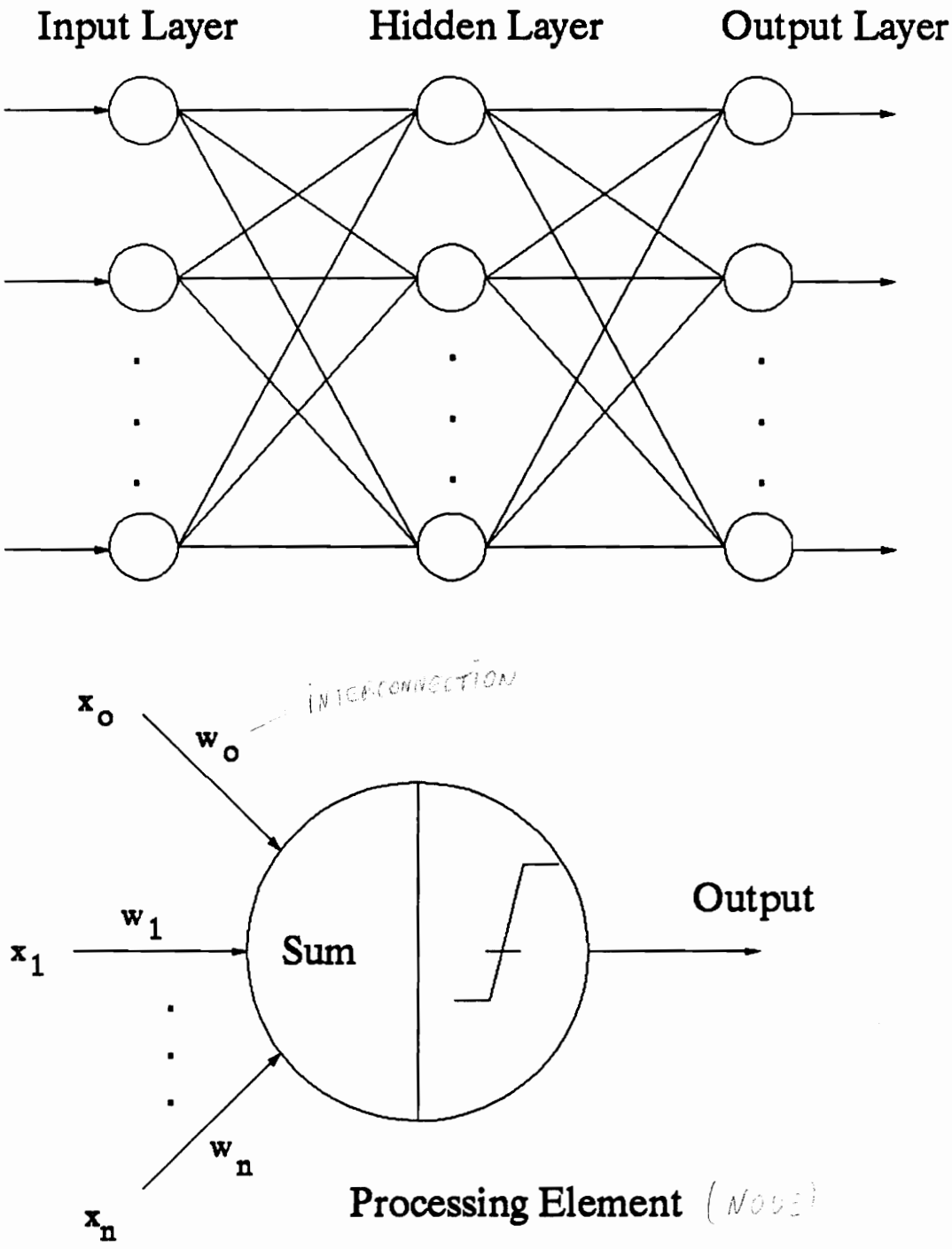

Figure 4 : General neural network architecture.

learning process is designated as "unsupervised" learning. The last type of learning is reinforcement learning where only grades of good or bad are presented to the network during training.

The recall mode is the process of finding out how well the network has learned. Often, the network will be presented with a different set of data and the outputs produce by the network will be compared against the desired outputs.

1.2.4 Learning by example

Unlike expert systems where the knowledge base is in the form of rules, neural networks create their rules by learning from examples presented to the networks. In supervised learning, there are generally three types of learning laws :

* Hebbian Learning where weights change in response to events within a processing element if both the input is high and desired output is high.
* Delta rule learning where the weights are modified in such a way to reduce the error between the desired outputs and the actual outputs.
* Competitive learning where the processing elements compete and the winner is allowed to modify its weight to reflect the input.

1.2.5 Distributed Associative Memory

The weights are the memory units of a neural network. When an example is presented to the network, that piece of information is stored distributively in many memory units of the network and shares these memory units with previously stored information.

1.2.6 Fault Tolerance

Fault tolerance refers to the fact that the response of the neural networks only change slightly if some processing elements are defective or damaged. This fault tolerance characteristic of the network comes about because the information is not stored in one place. It is distributed across numerous memory units.

1.2.7 Pattern Recognition

The nature of pattern recognition is to digest a large amount of input information and then categorize the outputs. Neural networks are superior in this behavior due to their ability of learning and the structure of the stored information after learning from examples.

1.2.8 Neural Network Applications

Neural networks have been successfully applied to many real world problems. Here are some of their applications : text-to-speech conversion, natural language processing, image compression, handwriting recognition, advanced pattern recognition, signal processing, prediction, system modeling, noise filtering, servo control, and so on.

2.0 Theoretical Foundations

The passive suspension system of Figure 1 will be used to derive the dynamic model for the system. This model will be expressed in state space representation. Linear stochastic optimal control theory will be applied to design the active controller for the active suspension system. Then backpropagation neural network will be used to identity the plant and controller for both passive suspension and active suspension systems with either full-state measurements or incomplete-state measurements.

2.1 Linear Stochastic Optimal Control

The linear stochastic optimal problem can be considered as the optimal problem with disturbances. Consider the following linear system with full state feedback

$$\dot{\mathbf{x}}(t) = A\,\mathbf{x}(t) + B\,\mathbf{u}(t) + E\,w(t) \qquad (2.1.1)$$

$$\mathbf{y}(t) = C\,\mathbf{x}(t) \qquad (2.1.2)$$

with the white noise disturbance input w(t) satisfying

$$E [w(t) w^T(t)] = V \delta (t - \tau) \qquad (2.1.3)$$

where E [] is the expectation operator,

δ is the Dirac delta function,

V is a non-negative definite symmetric matrix,

x is a state vector of n components,

y is the output vector of q components,

u is a input vector of p components,

w is a white noise disturbance input.

The steady state, linear stochastic problem requires the determination of the control **u** which minimzes the following performance index :

$$J = \lim_{t \to \infty} E [\mathbf{u}^T(t)R\mathbf{u}(t) + \mathbf{x}^T(t)Q\mathbf{x}(t)] \qquad (2.1.4)$$

where Q and R are positive definite symmetric matrices.

From [10], the solution to this linear stochastic problem involves solving the linear algebraic Riccati equation

$$PA + A^TP + Q - PBR^{-1}B^TP = 0 \qquad (2.1.5)$$

under the condition that (A,B) is stabilizable. That means all its uncontrollable modes are stable.

The optimal control law is given by

$$\mathbf{u}^* = -R^{-1}B^T P\,\mathbf{x} \tag{2.1.6}$$

where P is the unique non-negative definite solution to the Riccati equation (2.1.6).

2.1.1 Derivation of Optimal control $\mathbf{u}^*$

The Hamiltonian is

$$H(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t), t) = \mathbf{x}^T(t)Q\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t) + \mathbf{p}^T(t)A\mathbf{x}(t) + \mathbf{p}^T(t)B\mathbf{u}(t) \tag{2.1.7}$$

and the necessary conditions for optimality are

$$\dot{\mathbf{x}}^*(t) = A\mathbf{x}^*(t) + B\mathbf{u}^*(t) \tag{2.1.8}$$

$$\dot{\mathbf{p}}^*(t) = -\partial H / \partial \mathbf{x} = -Q\mathbf{x}^*(t) - A^T\mathbf{p}^*(t) \tag{2.1.9}$$

$$\partial H / \partial \mathbf{u} = \mathbf{0} = R\mathbf{u}^*(t) + B^T\mathbf{p}^*(t) \tag{2.1.10}$$

From equation (2.1.10), $\mathbf{u}^*(t)$ is given by

$$\mathbf{u}^*(t) = -R^{-1}B^T\mathbf{p}^*(t) \tag{2.1.11}$$

where $\mathbf{p}^*(t) = P\mathbf{x}^*(t)$ [12] and P satisfies the Riccati equation (2.1.5).

2.2 Backpropagation Neural Networks

Backpropagation neural network is one of the most powerful developments in neurocomputing. Backpropagation neural network (BNN) is capable of mapping bounded functions from R^n to R^m. This approximation is achieved by minimizing the least mean square error. This artificial neural network has found applications in many areas such as pattern recognition, image processing, automatic controls, system identification, and signal processing. This adaptive model free estimation learns the mapping based on training data sets rather than programming.

2.2.1 Backpropagation Architecture

In general, a backpropagation neural network consists of many processing elements (nodes), connections (weights), and hidden layers. It is a supervised learning network in the sense of mapping the inputs to the desired outputs by approximating the unknown surface error. Figure 5 shows the architecture of a three-layer feedforward neural network. The first layer is the input layer consisting of (A) nodes that each accepts a single input u_i and propagates this signal without modification to all nodes in the hidden layer. Each hidden node will sum all its received signals (summing junction). All squashing signals at the hidden layer then propagate to each node of the output layer. Each output

<fig5.drw>

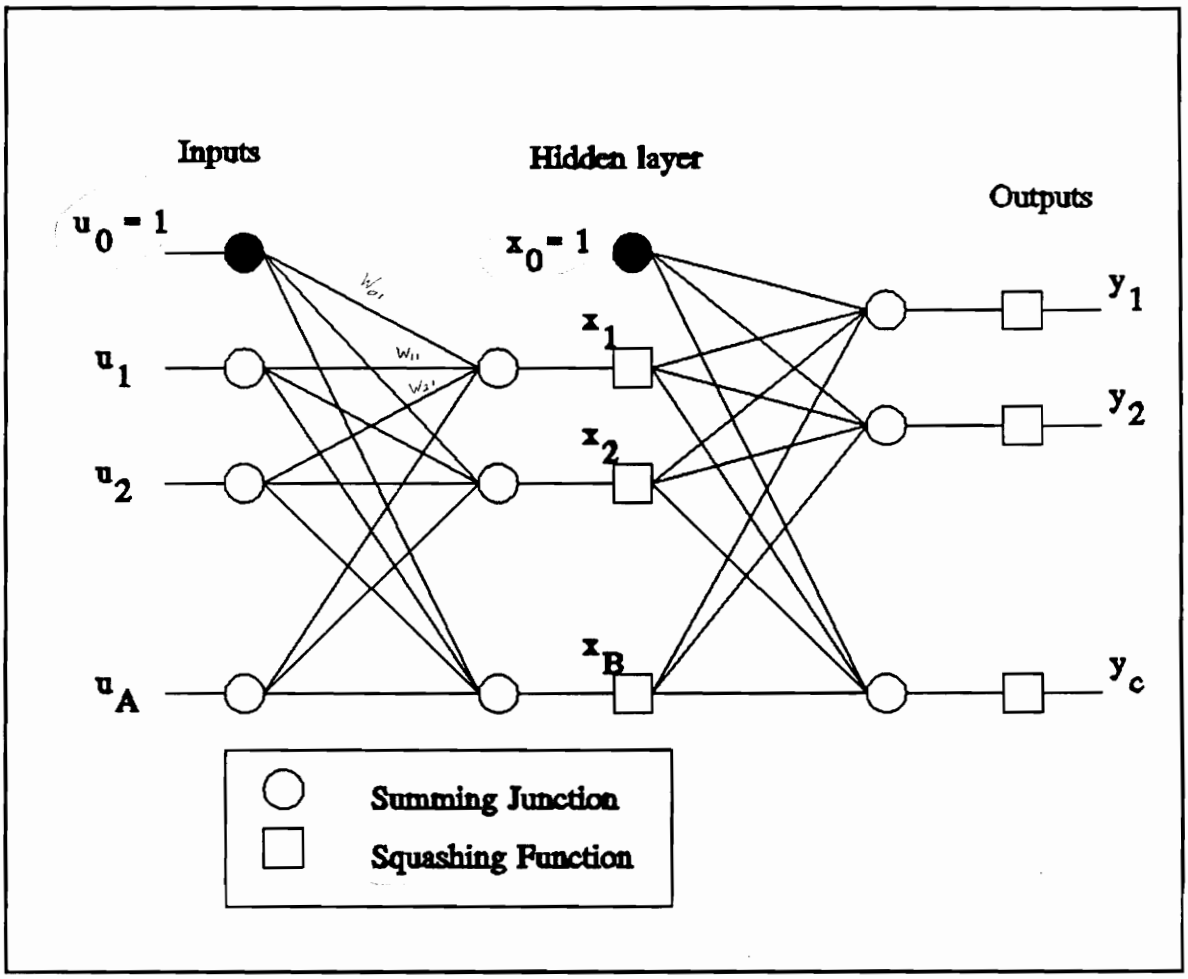

Figure 5 : A three-layer feedforward neural network.

node will sum all its received signals from the hidden layer and squashes it to produce the estimate y_i to be compared with the desired output d_i. Note that u_0 and x_0 are always equal to one. The differences between the desired outputs and the estimate outputs create an error surface. The backpropagation learning algorithm will be used to propagate the errors back and to modify the weights of the networks so that the network will approximate the function f better. The training data set will be presented to the network many times before the network can estimate the function correctly for an acceptable error. The training data set must be rich, complete, and exhaustive in order for the network to produce and estimate a function correctly. Hower, the shape of the backpropagation error surface can pose difficulty for the learning algorithm to find the global minimum. In general, backpropagation error surfaces have many global minima, many shallow slopes which lead to the difficulty for the learning algorithm to modify the weights to reduce the errors. Backpropagation error surfaces also have local minima, but they do not generally prove troublesome.

2.2.2 Backpropagation Learning Algorithm

The derivation of the backpropagation learning algorithm for the case of one hidden layer from figure 5 will be presented. One will find that the backpropagation algorithm provides an efficient way to compute the instantaneous partial derivatives. From the output errors, the algorithm modifies the weights recursively. For most cases, it is adequate to use one or two hidden layers and more nodes in each hidden layer to estimate a function.

2.2.3 Backpropagation Algorithm

- Step 1 : Define structure

$$\hat{x} = W1^T u$$
$$\hat{y} = W2^T x$$

where u is the vector of inputs : u_i for $i = 0$ to A

x is the vector of hidden nodes : x_i for $i = 0$ to B

y is the vector of outputs : y_i for $i = 0$ to C

W1, W2 are the weight matrices :

$$\hat{x}_j = \sum_{i=0}^{A} W1_{ij} u_i$$

x_0 and u_0 are both equal to unity, serving as sources of the "thresholds" for the next layers.

- Step 2 : Initialize the weights to independent random values between -0.1 and 0.1.

- Step 3 : Pick a training pair, **(u,d)** , activate inputs.

- Step 4 : Propagate to hidden layer and squash

$$x_j = f(\hat{x}_j)$$

where f is a squashing function.

Here are the examples of two types of squashing functions.

* Binary Logistic :

$$f(x) = 1 / (1 + e^{-x})$$

* Hyperbolic Tangent :

$$f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$$

- Step 5 : Propagate and squash to the output layer.

- Step 6 : Compute the output errors

$$\delta 2_j = f'(\hat{y}_j)(d_j - y_j)$$

where d_j is the desired output for j = 1 to C.

* Binary Logistic :

$$f'(\hat{y}_j) = y_j(1-y_j)$$

* Hyperbolic Tangent :

$$f'(\hat{y}_j) = (1+y_j)(1-y_j)$$

- Step 7 : Compute hidden layer errors

$$\delta 1_j = f'(\hat{x}_j)\sum_{i=1}^{C} \delta 2_i W 2_{ji}$$

for j = 1 to B.

- Step 8 : Update weight matrix W2

$$W2_{ij}(t+1) = W2_{ij}(t) + \eta\delta 2_j(t+1)\, x_i(t+1) + \alpha\Delta W2_{ij}(t)$$

for i = 0 to B and j = 1 to C,
where η is the learning rate parameter,
α is the optional momentum coefficient,
$\Delta W2_{ij}(t) = \delta 2_j(t)\, x_i(t)$.

- Step 9 : Update weight matrix W1

$$W1_{ij}(t+1) = W1_{ij}(t) + \eta\delta 1_j(t+1) u_i(t+1) + \alpha\Delta W1_{ij}(t)$$

for i = 0 to A and j = 1 to B,
where $\Delta W1_{ij}(t) = \delta 1_j(t)\, u_i(t)$.

- Step 10 : Return to step 3 until all training data pairs have been used. An epoch is a complete training data set. Many epochs may be needed to train the network to a desirable error.

3.0 Modelling and Simulation

Figure 1 will be used to model and simulate the passive suspension system. Figure 2 will be used to model and simulate the active suspension system. The wheel tire will be modelled as a spring and a damper in parallel. Linear stochastic optimal control will be applied to determine the optimal active controller in the active suspension system. Axle and body responses for both systems will be plotted for three different cases. Finally, the comparison of both suspension systems will be presented. For all cases, the following system parameters [8] will be used throughout the modelling and simulation :

m_s	=	288.90 kg	: Body mass
m_w	=	28.58 kg	: Wheel mass
k_s	=	19660.00 N/m	: Body spring constant
b_s	=	1861.00 Ns/m	: Body damper constant
k_t	=	155900.00 N/m	: Tire spring constant
b_t	=	100.00 Ns/m	: Tire damper constant

Figure 6 shows the control block diagrams of active and passive suspension models to be simulated.

<Fig6.drw>

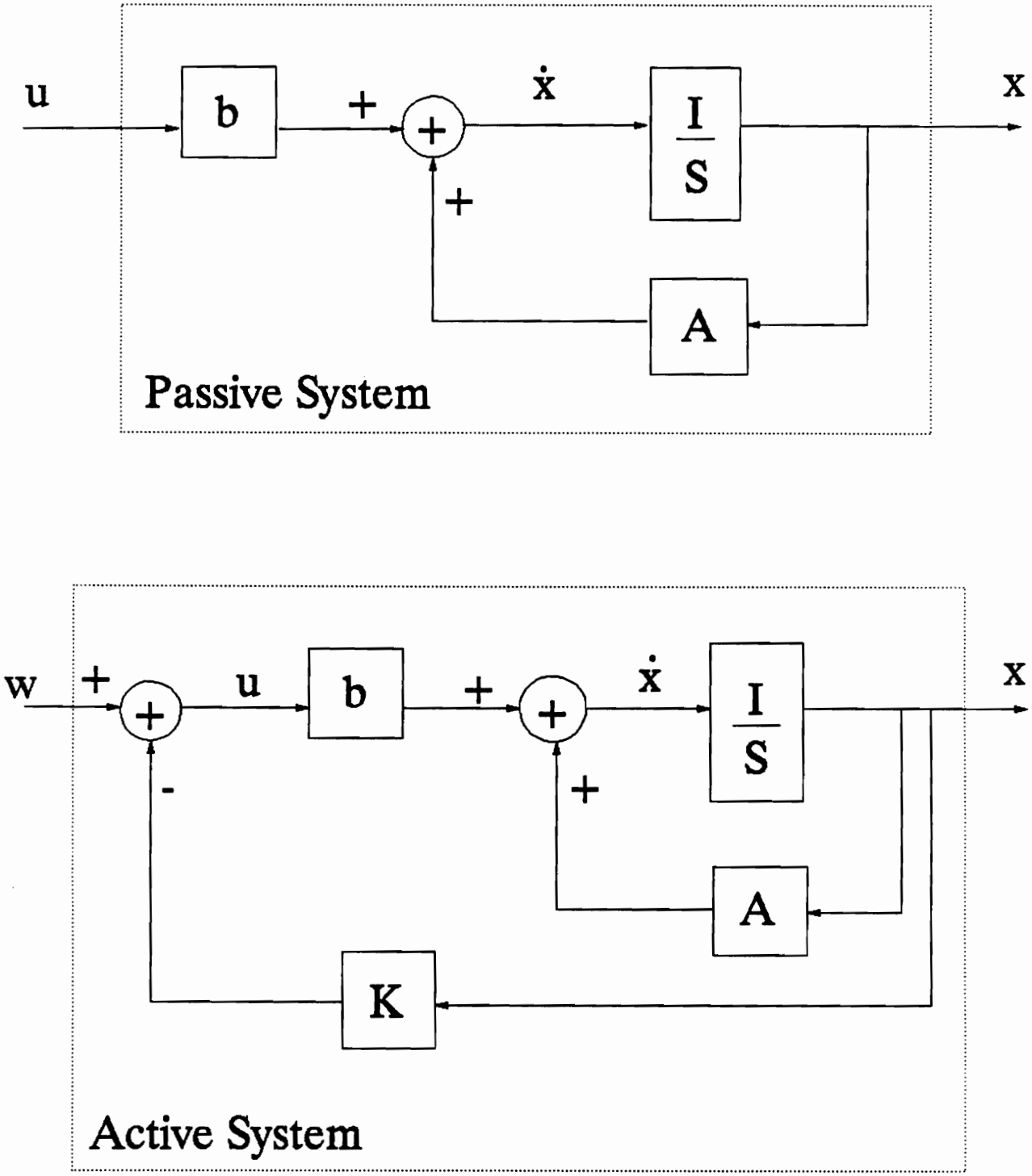

Figure 6 : Block diagrams of active and passive suspension models.

3.1 Passive Suspension Modelling

The following system dynamical equations can be obtained from figure 1 :

$$\begin{aligned} m_s\ddot{x}_2 + k_s(x_2 - x_1) + b_s(\dot{x}_2 - \dot{x}_1) &= 0 \\ m_w\ddot{x}_1 + k_t(x_1 - x_o) + b_t(\dot{x}_1 - \dot{x}_o) + k_s(x_1 - x_2) + b_s(\dot{x}_1 - \dot{x}_2) &= 0 \end{aligned} \tag{3.1}$$

where x_o is the road integrated white noise input,

x_1 is the axle vertical displacement,

x_2 is the body vertical displacement,

$dx_{1/dt}$ is the axle vertical velocity,

$dx_{2/dt}$ is the body vertical velocity,

$\ddot{x}_1$ is the axle vertical acceleration,

$\ddot{x}_2$ is the body vertical acceleration.

Now, let $x_3 = dx_{1/dt}$ and $x_4 = dx_{2/dt}$.

Then equation (3.1) can be written in the form of the state space representation of the passive suspension system.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{-(k_s + k_t)}{m_w} & \dfrac{k_s}{m_w} & \dfrac{-(b_s + b_t)}{m_w} & \dfrac{b_s}{m_w} \\ \dfrac{k_s}{m_s} & \dfrac{-k_s}{m_s} & \dfrac{b_s}{m_s} & \dfrac{-b_s}{m_s} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dfrac{k_t}{m_w} \\ 0 \end{bmatrix} x_o + \begin{bmatrix} 0 \\ 0 \\ \dfrac{b_t}{m_w} \\ 0 \end{bmatrix} \dot{x}_o \tag{3.2}$$

The equation (3.2) can be written as

$$dx(t)/dt = Ax(t) + b_1x_o + b_2dx_o/dt \quad (3.3)$$

Let $v(t) = x(t) - b_2x_o$

$$dv(t)/dt = dx(t)/dt - b_2dx_o/dt \quad (3.4)$$

Then combine (3.3) and (3.4),

$$dv(t)/dt = Av(t) + (Ab_2 + b_1) x_o.$$

$$dv(t)/dt = Av(t) + b\, x_o. \quad (3.5)$$

where $b = (Ab_2 + b_1)$.

The equation (3.5) can be written as :

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_3 \\ \dot{v}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-(k_s+k_t)}{m_w} & \frac{k_s}{m_w} & \frac{-(b_s+b_t)}{m_w} & \frac{b_s}{m_w} \\ \frac{k_s}{m_s} & \frac{-k_s}{m_s} & \frac{b_s}{m_s} & \frac{-b_s}{m_s} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} + \begin{bmatrix} \frac{b_t}{m_w} \\ 0 \\ \frac{k_t m_w - b_t(b_s+b_t)}{m^2_w} \\ \frac{b_s b_t}{m_w m_s} \end{bmatrix} x_o \quad (3.6)$$

The eigenvalues of equation (3.6) are obtained using MATLAB :

-34.8725 + 67.7753i

-34.8725 - 67.7753i

-2.6556 + 7.6039i

-2.6556 - 7.6039i

Since all eigenvalues of the passive suspension system are on the left-half plane, the system is stable. Since the input x_o is the integrated white noise, the state

space representation in equation (3.6) will be augmented to include a state dx_0 = white noise. The new state space representation will be as follow :

$$\begin{bmatrix} \dot{v}_o \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ b & A \end{bmatrix} \begin{bmatrix} v_o \\ v \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} w \tag{3.7}$$

where w is the white noise,

v_o is the integrated white noise.

The axle transfer function and the body transfer function are obtained from MATLAB as following :

$$\frac{X_1(s)}{X_2(s)} = \frac{3.499s^3 + 5477.4s^2 + 35380.2s + 376874.6}{s^4 + 75.056s^3 + 6244.88s^2 + 35380.2s + 376874.6} \tag{3.8}$$

$$\frac{X_2(s)}{X_o(s)} = \frac{22.54s^2 + 35380.2s + 376874.6}{s^4 + 75.056s^3 + 6244.88s^2 + 35380.2s + 376874.6} \tag{3.9}$$

3.2 Passive Suspension Simulation

MATLAB will be used to simulate the state space representation in equation (3.7) for three cases of random white noise.

Case 1 : Random white noise with seed = 0.

Case 2 : Random white noise with seed = 1.

Case 3 : Random white noise with seed = 4.

For each case, road input, axle response, and body response are plotted on the same graph for comparison. In general, the axle response should follow the road

roughness closely to meet our design goal that the tires are assumed to contact the road. However, the body response should be smoother to provide passenger ride comfort.

```
%  passive.m
%=============================
%
%  Author :  Michael Tran
%  Date   :  08/15/91
%
%=============================
%
%  Simulation of passive suspension system
%  with integrated white noise input
%  Axle/Body frequency response
%
%
ms = 288.9;                 %  Body mass
mw = 28.58;                 %  Wheel mass
kt = 155900;                %  Tire spring constant
bt = 100;                   %  Tire damping constant
ks = 19960;                 %  Suspension spring constant
bs = 1861;                  %  Suspension damping constant

t = 0:0.025:2.5;            %  Simulation time from 0 to 2.5 seconds
n = max(size(t'));

rand('normal');             %  Set to white noise random
rand('seed',0);             %  Set random seed
                            %  Case 1 :  random seed = 0
                            %  Case 2 :  random seed = 1
                            %  Case 3 :  random seed = 4
w = rand(1,n);              %  Generate 100 random data.

a31 = -(ks+kt)/mw;
a32 = ks/mw;
a33 = -(bs+bt)/mw;
a34 = bs/mw;
a41 = ks/ms;
a42 = -ks/ms;
a43 = bs/ms;
a44 = -bs/ms;
b13 = kt/mw;
b23 = bt/mw;

a = [0 0 1 0;0 0 0 1;a31 a32 a33 a34;a41 a42 a43 a44];
b1 = [ 0 0 b13 0 ]';
```

```
b2 = [ 0 0 b23 0 ]';

b3 = a*b2 + b1;
a = [0 0 0 0 0;b3 a];              %  Augmented A matrix to generate
                                   %  integrated white noise.
b = [1 0 0 0 0]';

[ad,bd] = c2d(a,b,0.025);          %  Convert continuous time to discrete time.

% Outputs: x0, x1, x2, x3, x4.
c = [1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;b23 0 0 1 0;0 0 0 0 1];
d = [ 0 0 0 0 0]';

x0 = [ 0 0 0 0 0 ]';               %  Set state initial condition

z = dlsim(ad,bd,c,d,w,x0);         %  Discrete time simulation

z0 = z(:,1);                       %  Get road data
z1 = z(:,2);                       %  Get axle data
z2 = z(:,3);                       %  Get body data
p = [w' z];                        %  Data to be saved for plot later

save pcaran4.dat p /ascii

plot(t,z1,'--',t,z2,'-',t,z0,':')
title('Passive suspension response for integrated white noise'),
xlabel('Time, sec'), ylabel('Displacement, m')
gtext('Body')
gtext('Axle')
gtext('Road')
gtext('Random seed = 0')
print              % Send plot to laser printer
pause
```

<fig7.drw>

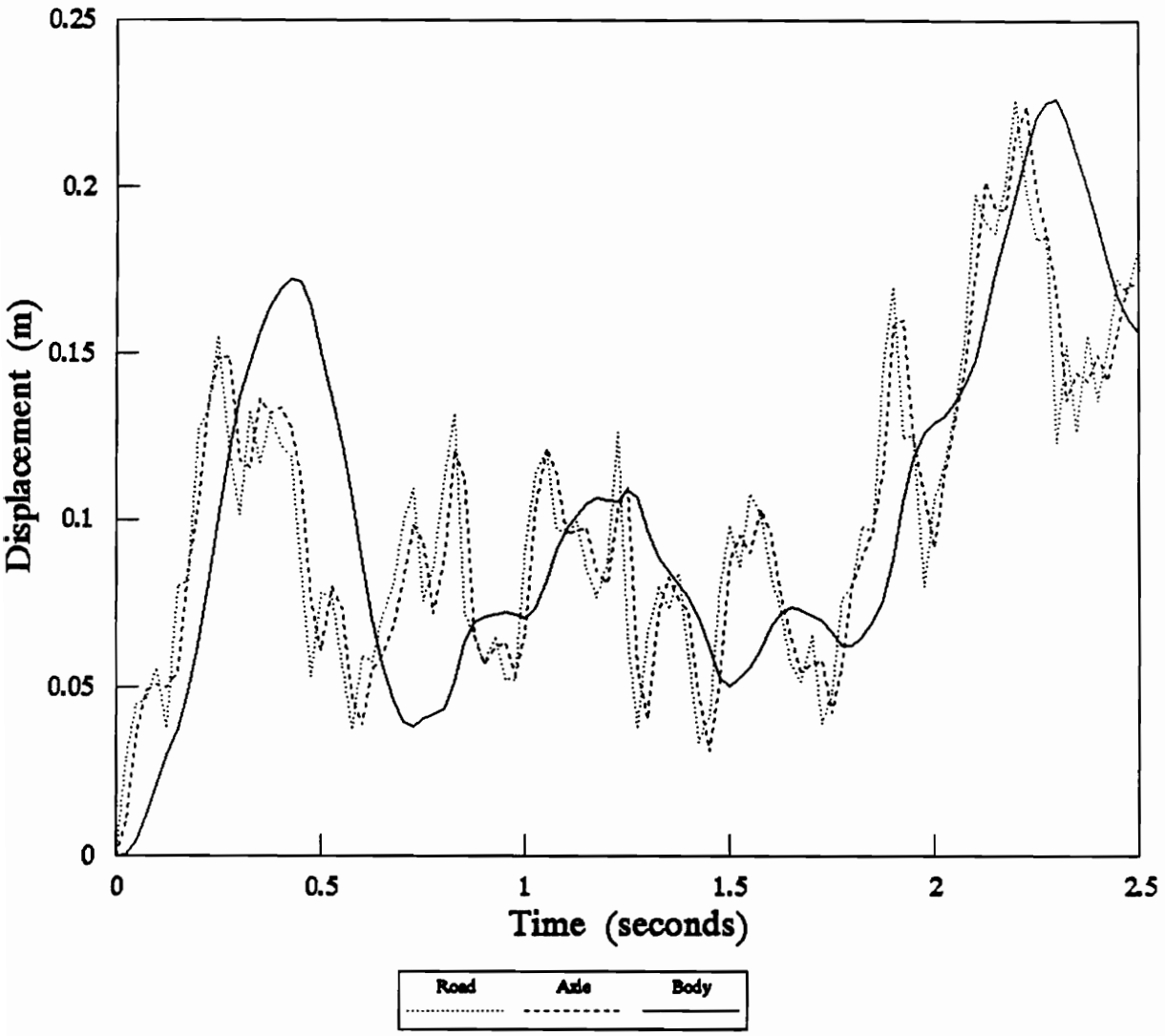

Figure 7 : Passive Suspension Response (random seed = 0).

<fig8.drw>

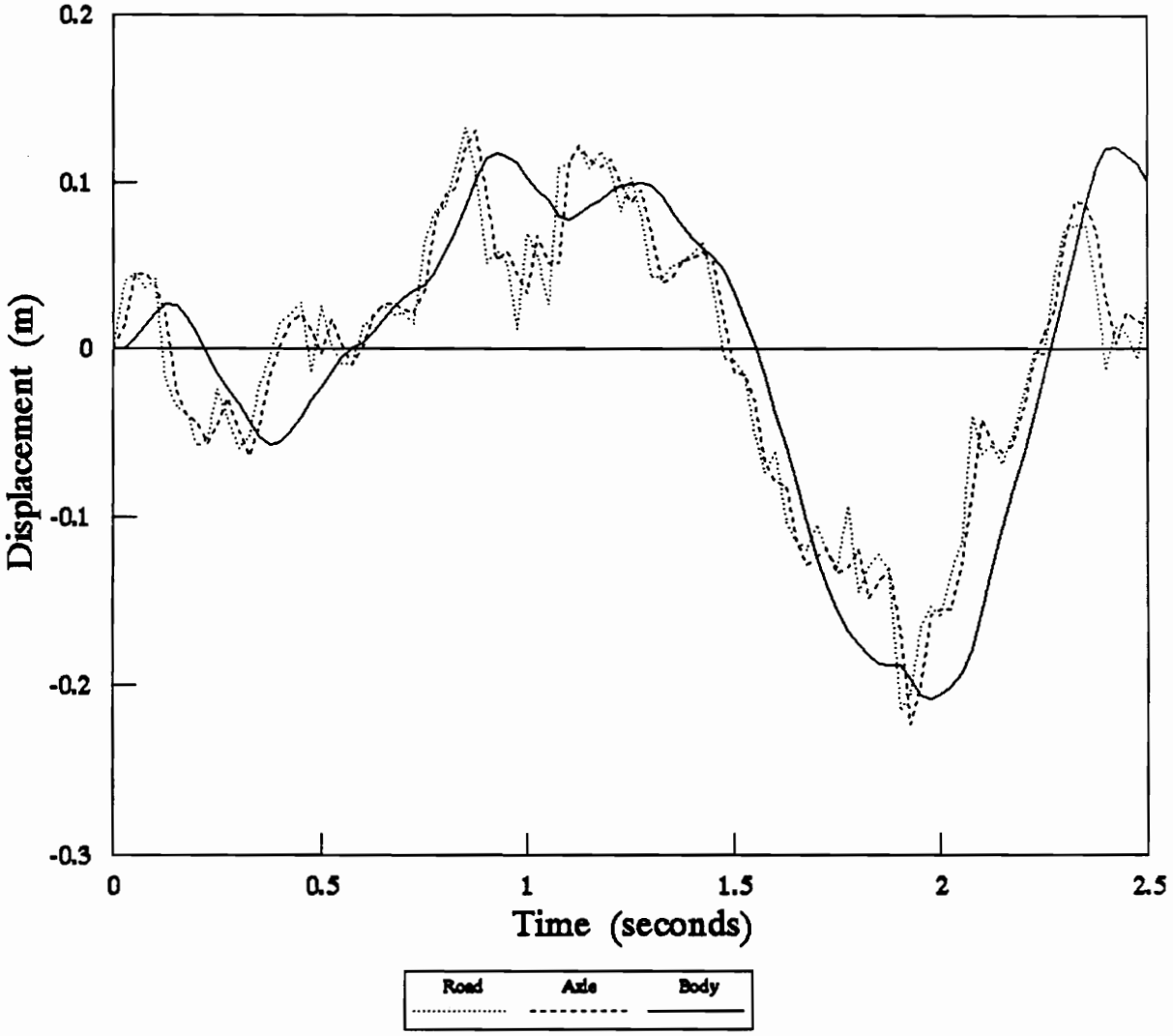

Figure 8 : Passive Suspension Response (random seed = 1).

<fig9.drw>

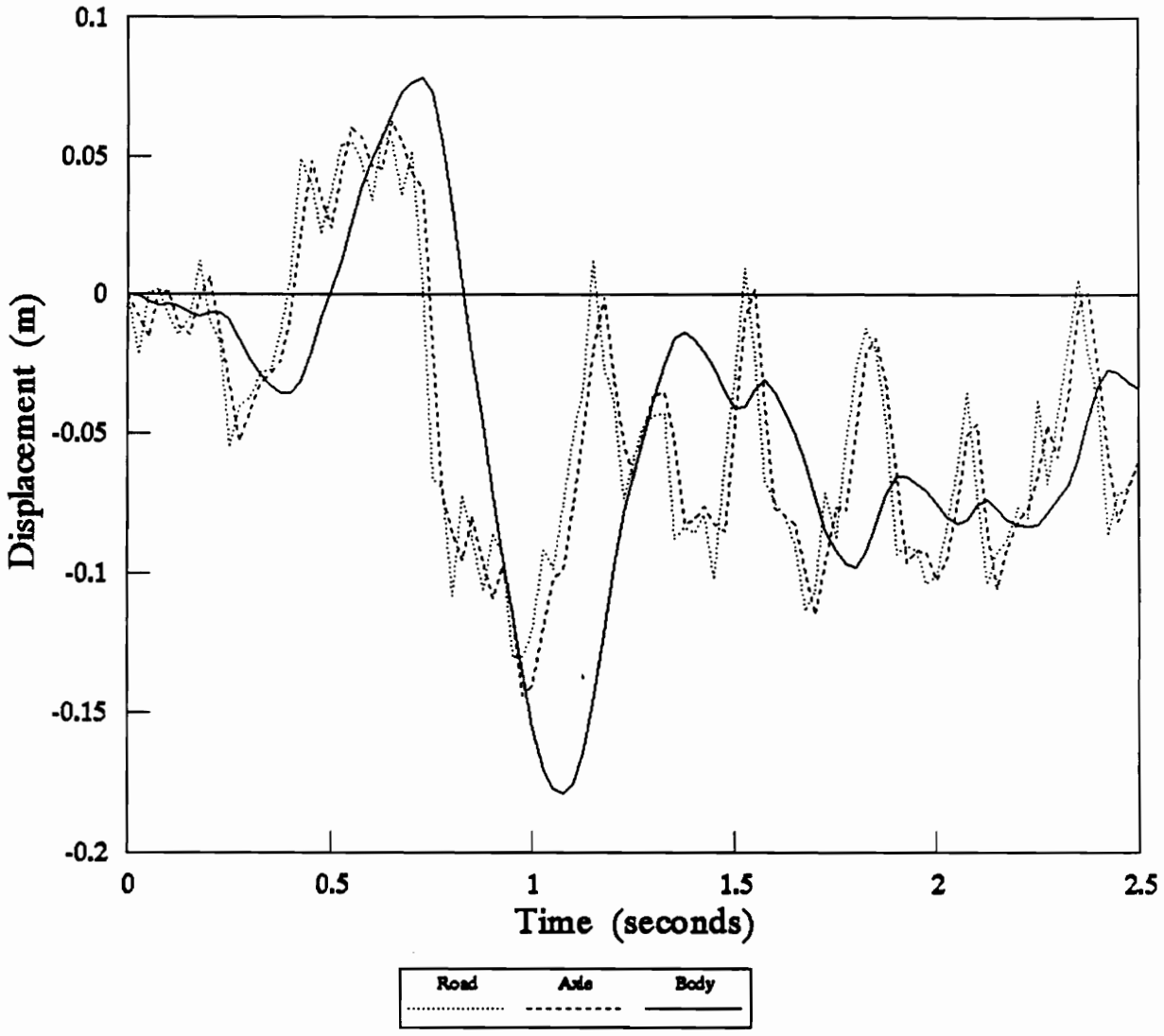

Figure 9 : Passive Suspension Response (random seed = 4).

3.3 Active Suspension Modelling

The following state dynamical equations are derived from figure 2 of the active suspension system :

$$\begin{aligned} m_s\ddot{x}_2 &= u \\ m_w\ddot{x}_1 + k_t(x_1 - x_o) + b_t(\dot{x}_1 - \dot{x}_o) + u &= 0 \end{aligned} \tag{3.10}$$

Now, let $x_3 = dx_{1/dt}$ and $x_4 = dx_{2/dt}$, then the equation (3.10) can be written in the form of state space representation :

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_t}{m_w} & 0 & \frac{-b_t}{m_w} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{k_t}{m_w} \\ 0 \end{bmatrix} x_o + \begin{bmatrix} 0 \\ 0 \\ \frac{b_t}{m_w} \\ 0 \end{bmatrix} \dot{x}_o + \begin{bmatrix} 0 \\ 0 \\ \frac{-1}{m_w} \\ \frac{1}{m_s} \end{bmatrix} u \tag{3.11}$$

Equation (3.11) can be written in matrix form as :

$$\mathbf{dx/dt} = \mathbf{Ax} + \mathbf{b_1x_o} + \mathbf{b_2dx_{o/dt}} + \mathbf{b_3u} \tag{3.12}$$

Let $\mathbf{v} = \mathbf{x} - \mathbf{b_2x_o}$,

then $\mathbf{dv/dt} = \mathbf{dx/dt} - \mathbf{b_2dx_{o/dt}}$,

Equation (3.12) can be rewritten as :

$$\mathbf{dv/dt} = \mathbf{Av} + (\mathbf{Ab_2} + \mathbf{b_1})\,\mathbf{x_o} + \mathbf{b_3u}$$

or $$\mathbf{dv/dt} = \mathbf{Av} + \mathbf{b_4x_o} + \mathbf{b_3u}$$

where

$$b_4 = \begin{bmatrix} \frac{b_t}{m_w} \\ 0 \\ \frac{k_t m_w - b_t^2}{m_w^2} \\ 0 \end{bmatrix}$$

Since the road input is in the form of integrated white noise, the state space representation will be augmented to include dx_0 = white noise. The new state space representation will be as following :

$$\begin{bmatrix} \dot{v}_o \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ b_4 & A \end{bmatrix} \begin{bmatrix} v_o \\ v \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} w + \begin{bmatrix} 0 \\ b_3 \end{bmatrix} u \qquad (3.13)$$

where w is the white noise,

v_0 is the road input - the integrated white noise.

Now, the linear stochastic optimal control theory will be applied to find the optimal state feedback controller which minimizes the following performance index :

$$J = \lim_{t \to \infty} E [u^T(t)Ru(t) + x^T(t)Qx(t)]$$

where R = 0.8 e-09 ,

$$Q = \begin{bmatrix} 10 & -10 & 0 & 0 & 0 \\ -10 & 11 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The solution to this problem involves solving the Riccati equation :

$$A^T P + PA + Q - PBR^{-1}B^T P = 0 \qquad (3.14)$$

The optimal control law is given by

$$\mathbf{u} = \mathbf{k}\,\mathbf{x} \qquad (3.15)$$

where

$$k = -R^{-1}B^T P.$$

The optimal feedback for this system is obtained from MATLAB to be :

$$k = [\,-15741 \;\; 55614 \;\; -35355 \;\; 1291 \;\; -4084\,] \qquad (3.16)$$

The closed loop eigenvalues of the active suspension system are located at :

-26.3494 + 78.2601i

-26.3494 - 78.2601i

-6.3048 + 7.6254i

-6.3048 - 7.6254i.

Since all eigenvalues of the closed loop system are located in the left half of the s-plane, the active suspension system is stable.

3.4 Active Suspension Simulation

MATLAB will be used to simulate the active suspension system for three cases of random white noise.

Case 1 : Random white noise with seed = 0.

Case 2 : Random white noise with seed = 1.

Case 3 : Random white noise with seed = 4.

For each case, road input, axle response, and body response are plotted on the same graph for comparison. The axle of the system must follow the road closely . The body response of the active system is expected to provide maximum passenger ride comfort as well as road handling.

```
%  active.m
%=============================================
%  Author : Michael Tran
%  Date   : 8/15/91
%=============================================
%
%  Active suspension simulation with integrated white noise input
%  Axle/Body frequency response
%
ms = 288.9;              % Body mass
mw = 28.58;              % Wheel mass
kt = 155900;             % Tire spring constant
bt = 100;                % Tire damping constant
a31 = -kt/mw;
a33 = -bt/mw;
b13 = -a31;
b23 = -a33;
b1 = [ 0 0 b13 0]';
b2 = [ 0 0 b23 0]';

a = [0 0 1 0;0 0 0 1;a31 0 a33 0;0 0 0 0];
b4 = a*b2 + b1;

%  Augmented the matrix to include the input to produce integrated white noise

a=[0 0 0 0 0;b4 a ];
b=[0 0 0 -1/mw 1/ms]';

%  Finding optimal feedback k by solving the Riccati equation

h=[10 -10 0 0 0;-10 11 -1 0 0;0 -1 1 0 0;0 0 0 0 0;0 0 0 0 0];
r=0.8e-09;
g=b*b'/r;
z= aremod(a,g,h);
kv = -b'*z/r;                    %  Optimal feedback k.
a = a + b*kv;
b = [1 0 0 0 0]';
[ad,bd] = c2d(a,b,0.025);        %  Discrete time simulation

% Output x0, x1, x2, x3, x4.

c = [1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;b23 0 0 1 0;0 0 0 0 1];
d = [0 0 0 0 0]';
```

```
x0 = [0 0 0 0 0]';                  % Set initial condition to zero
t = 0:0.025:2.5;                    % Simulation time from 0 to 2.5 seconds
n = max(size(t'));

rand('normal');                     % Set to normal random
rand('seed',4);                     % Set random seed
                                    % Case 0 : random seed = 0
                                    % Case 1 : random seed = 1
                                    % Case 4 : random seed = 4
w = rand(1,n);                      % Generate 100 random data points

y = dlsim(ad,bd,c,d,w,x0);          % Discrete time simulation

y0 = y(:,1);
y1 = y(:,2);
y2 = y(:,3);

v = [w' y];                         % Save data to plot later
save acaran4.dat v /ascii

plot(t,y1,'--',t,y2,'-',t,y0,':')
title('Active suspension response for integrated white noise'),
xlabel('Time, sec'), ylabel('Displacement, m')
gtext('Body')
gtext('Axle')
gtext('Road')
gtext('Random seed = 0')
print
pause
```

<fig10.drw>

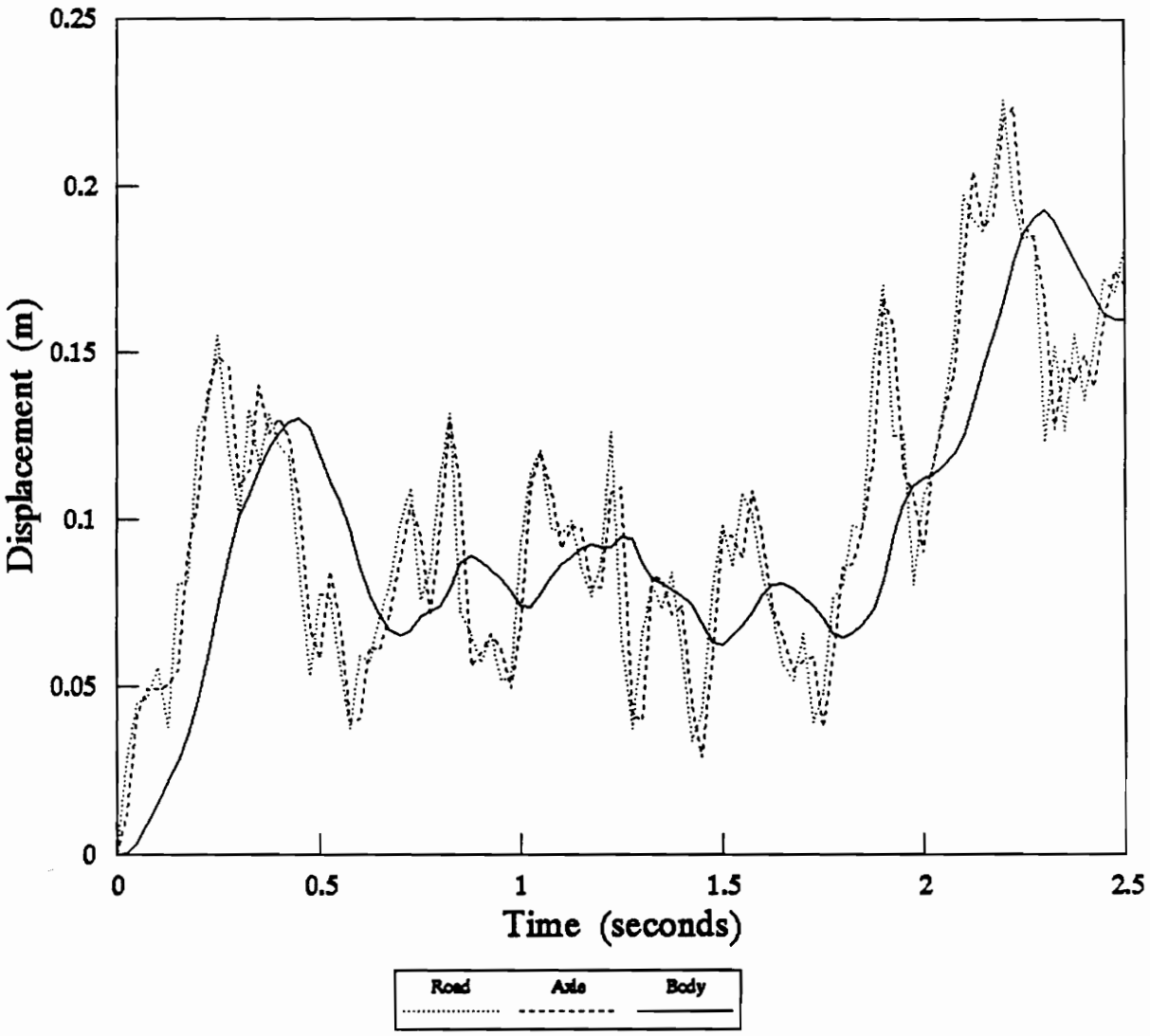

Figure 10 : Active Suspension Response (random seed = 0).

<fig11.drw>

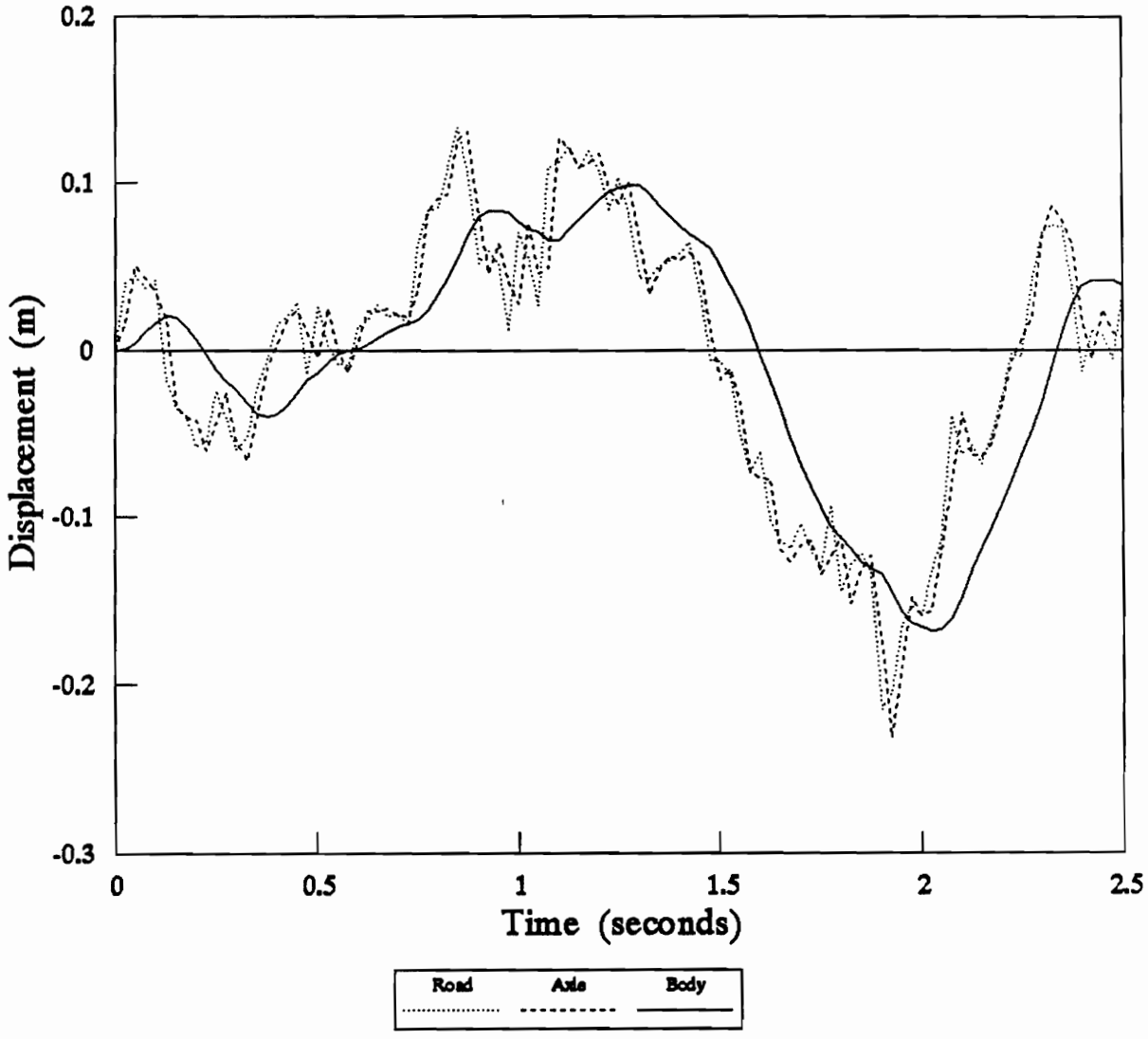

Figure 11 : Active Suspension Response (random seed = 1).

<fig12.drw>

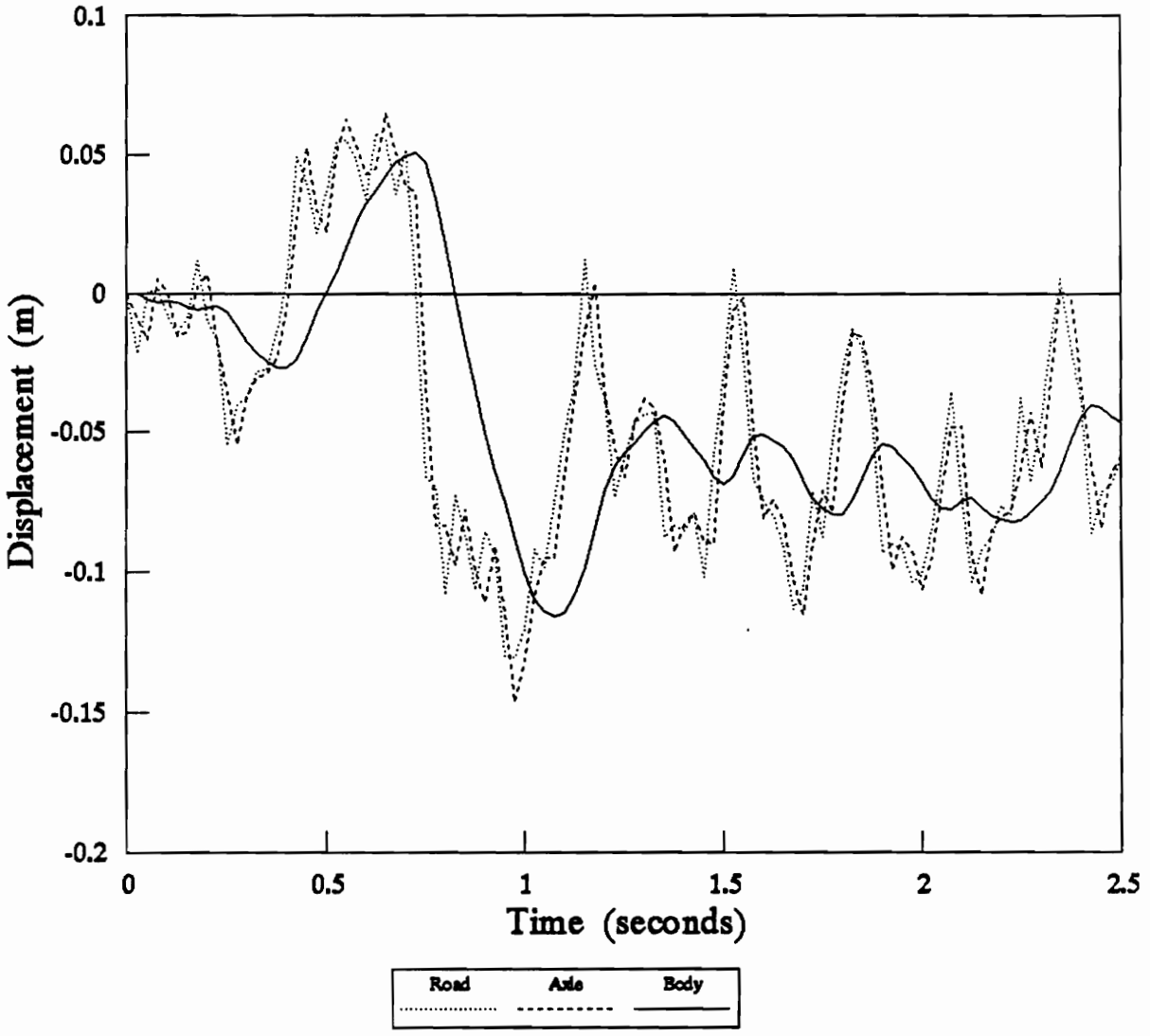

Figure 12 : Active Suspension Response (random seed = 4).

3.5 Active and Passive Comparison

The axle, and body damping ratios for the passive suspension system can be calculated from its eigenvalues as following :

$$\frac{34.8725}{\sqrt{(34.8725)^2+(67.7753)^2}} = \varsigma_{axle} = 0.4575$$

$$\frac{2.6556}{\sqrt{(2.6556)^2+(7.6039)^2}} = \varsigma_{body} = 0.3297$$

Similarly, the axle and body damping ratios for the active suspension system can be obtained from its eigenvalues :

$$\frac{26.3494}{\sqrt{(26.3494)^2+(78.2601)^2}} = \varsigma_{axle} = 0.3191$$

$$\frac{6.3048}{\sqrt{(6.3048)^2+(7.6254)^2}} = \varsigma_{body} = 0.6372$$

Since the body damping ratio of the active suspension system is bigger than that of the passive suspension system, the body response of the active system will be less overshoot than the body response of the passive system.

Figure 12 shows the comparison of the axle response between active and passive suspension systems to the road roughness. The axle response of both systems follows the road closely as desired. Figures 13, 14, 15 show the comparison of the body responses of both systems. In these figures, the body response of the active

suspension system has less overshoot than the body response of the passive suspension system. Thus, it is clear that the active suspension system is superior to the passive suspension system in terms of passenger ride comfort and road handling.

<fig13.drw>

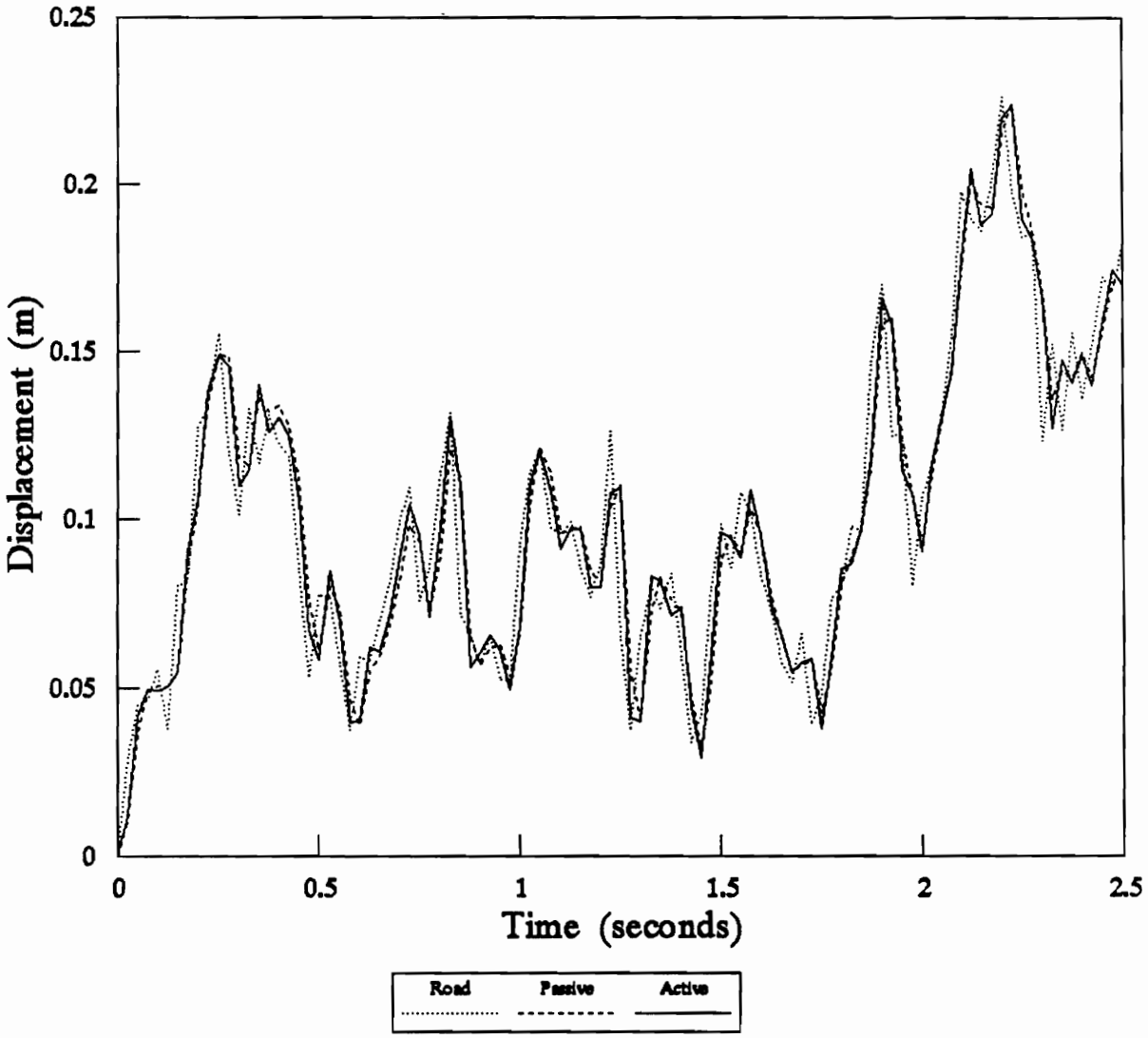

Figure 13 : Axle Response Comparison (random seed = 0).

<fig14.drw>

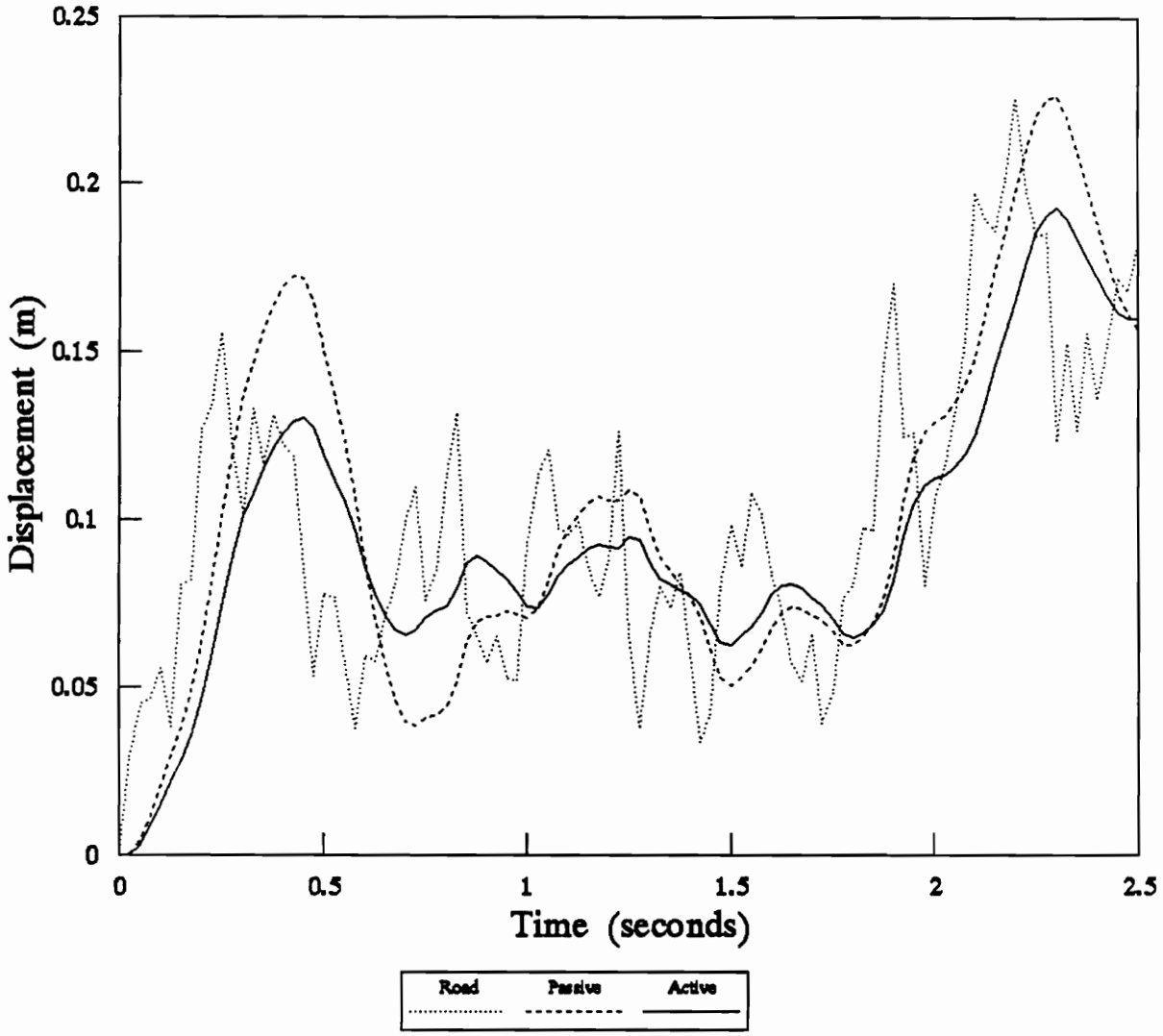

Figure 14 : Body Response Comparison (random seed = 0).

<fig15.drw>

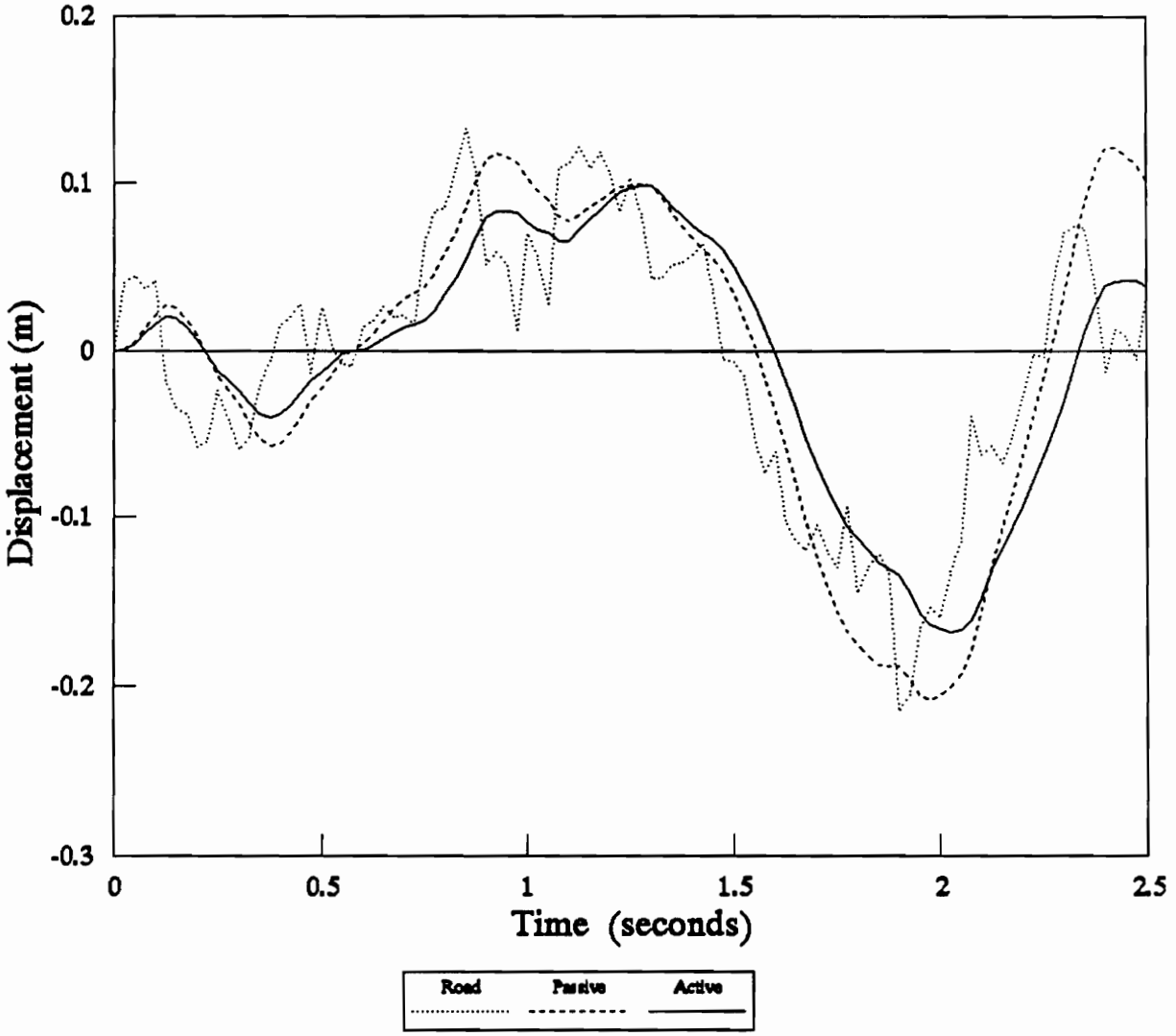

Figure 15 : Body Response Comparison (random seed = 1).

<fig16.drw>

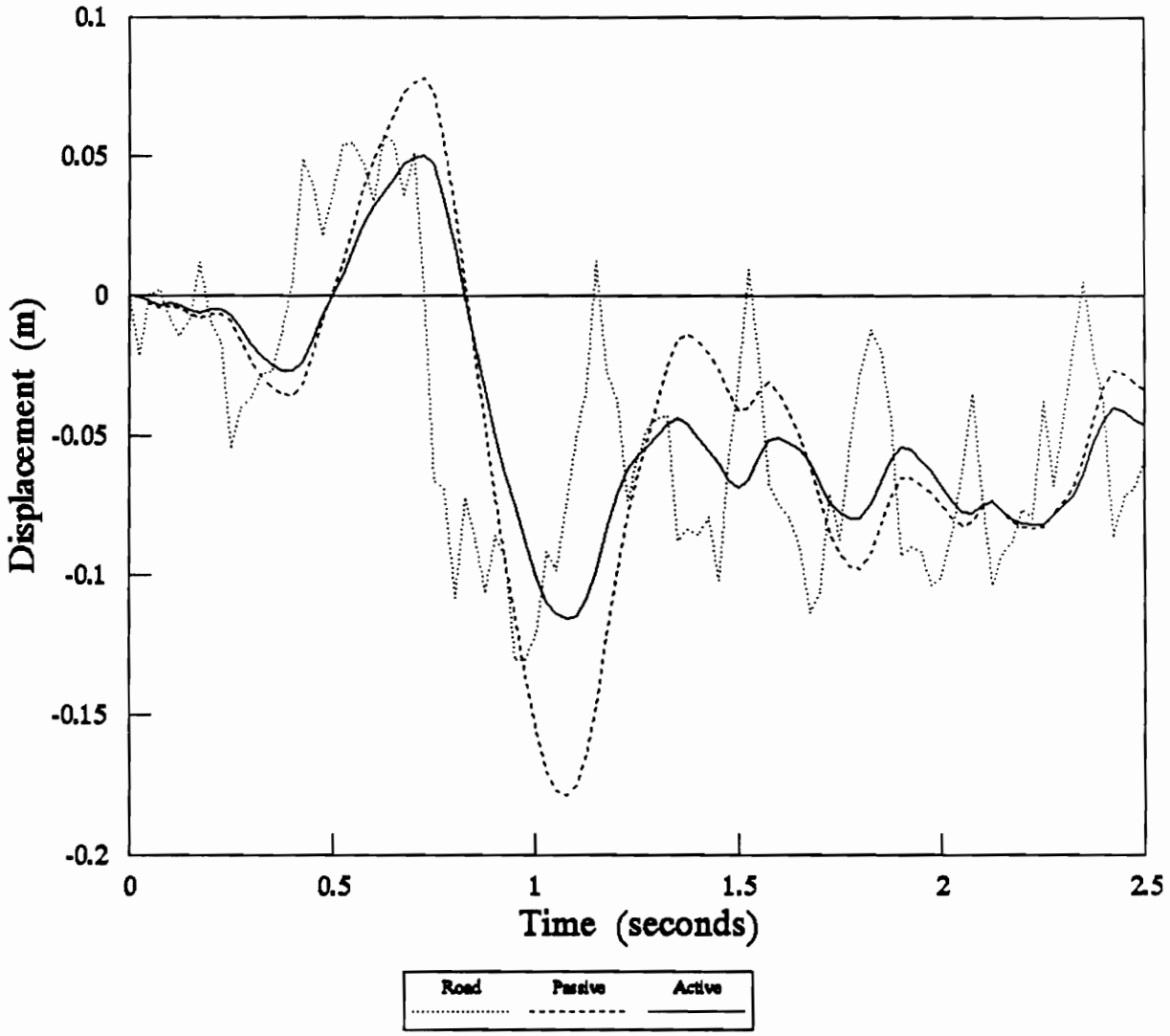

Figure 16 : Body Response Comparison (random seed = 4).

4.0 Identification of Suspension System Using Neural Networks

Backpropagation neural network will be used to identify both the passive suspension systems and the active suspension systems. The identification of suspension systems using neural networks consists of four parts :

- Identification of passive suspension system with the knowledge of full state measurements.
- Identification of passive suspension system with the partial knowledge of the system state measurements.
- Identification of active suspension system with the knowledge of full state measurements.
- Identification of active suspension system with the partial knowledge of the system state measurements.

Figures 17, and 22 show the simulation block diagrams to be used in the identification process for the passive suspension system with either full state measurements or partial state measurements. In a similar fashion, figures 27, and

32 present the simulation block diagrams to be used in the identification process for the active suspension system with full state / partial state measurements. For each part, a set of training data will be used to train the neural networks to estimate the desired outputs of the suspension system from a set of inputs. The desired outputs of the suspension system will be the axle displacement and the body displacement.

For the full state neural network identification, the inputs to the neural networks consist of the road displacement, the previous axle displacement, the previous axle velocity, the previous body displacement, the previous body velocity, and random white noise.

For the partial state neural network identification, the inputs to the neural networks consist of the road displacement, the previous axle displacement and its time-delayed version, the previous body displacement and its time-delayed version, and random white noise.

4.1 Full State Identification of Passive Suspension

Figure 17 shows a simulation block diagram for the full state neural network identification of the passive suspension system. The full state neural network identification model has the following form :

$$\hat{y}_1(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_3(k), x_4(k)\}$$
$$\hat{y}_2(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_3(k), x_4(k)\}$$

where NN stands for neural networks mapping,

$x_0(k)$ is the road displacement input,

$x_1(k)$ is the axle displacement input,

$x_2(k)$ is the body displacement input,

$x_3(k)$ is the axle velocity input,

$x_4(k)$ is the body velocity input,

$w(k)$ is the random white noise input,

$y_1(k+1)$ is the neural network axle displacement output,

$y_2(k+1)$ is the neural network body displacement output.

A set of training data will be applied to the neural networks with six inputs, one hidden layer of six hidden nodes, and two ouputs. Backpropagation neural network algorithm presented in section 2.2.3 will be used to train the networks. The backpropagation program listing is included in appendix A. Figures 18 and 19 show the plots of the desired axle / body displacements vs. the estimate axle / body displacements using neural networks from the set of training data. The neural network outputs map the desired outputs almost exactly with the absolute error of 0.000804092. Next, a different set of test data will be applied to the already-trained neural networks and its outputs will be compared against the desired outputs to see how well the neural networks perform. Figures 20 and 21 show the plots of the desired axle / body displacements vs. the estimate axle / body displacements from trained neural networks on a set of test data. The neural networks also perform exceptionally well with very, very small error. Therefore, with enough information the neural networks can be trained to identify the passive suspension system to a desired level of error.

Here is the weight connections of the trained neural networks :

Number of inputs = 6

Number of hidden layers = 1

Number of hidden nodes = 6

Number of outputs = 2

Desired error = 0.001

Actual absolute error = 0.000804092

Number of iterations = 673992

Weight connections :

$u \rightarrow h$	$h \rightarrow y$
0.00017	0.00013
0.00867	0.00042
0.64826	0.01911
0.27393	-0.00214
0.08226	0.99057
0.00873	0.00165
0.00764	0.02328

<Fig17.drw>

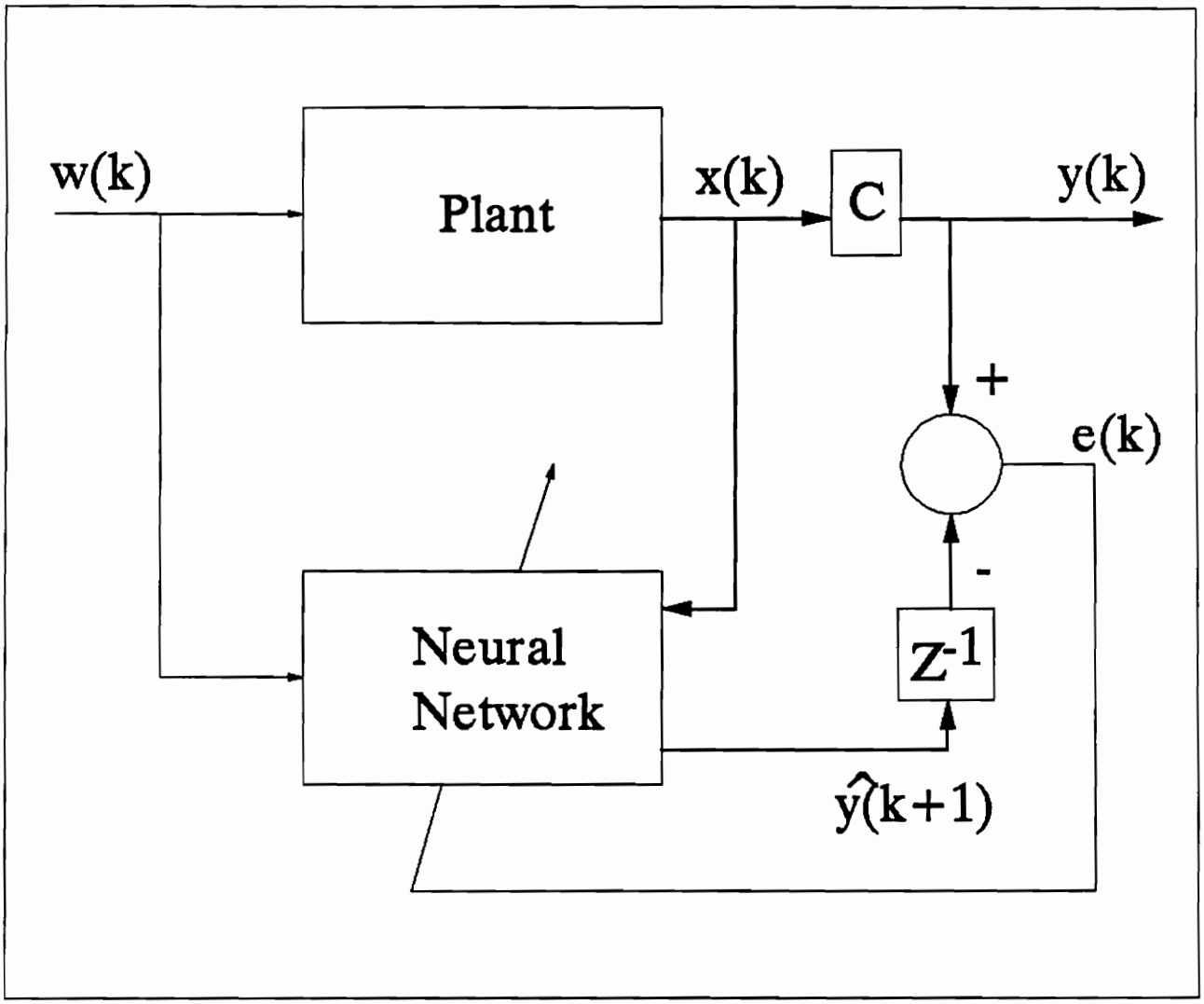

Figure 17 : Full state neural network identification of the passive suspension.

<Fig18.drw>

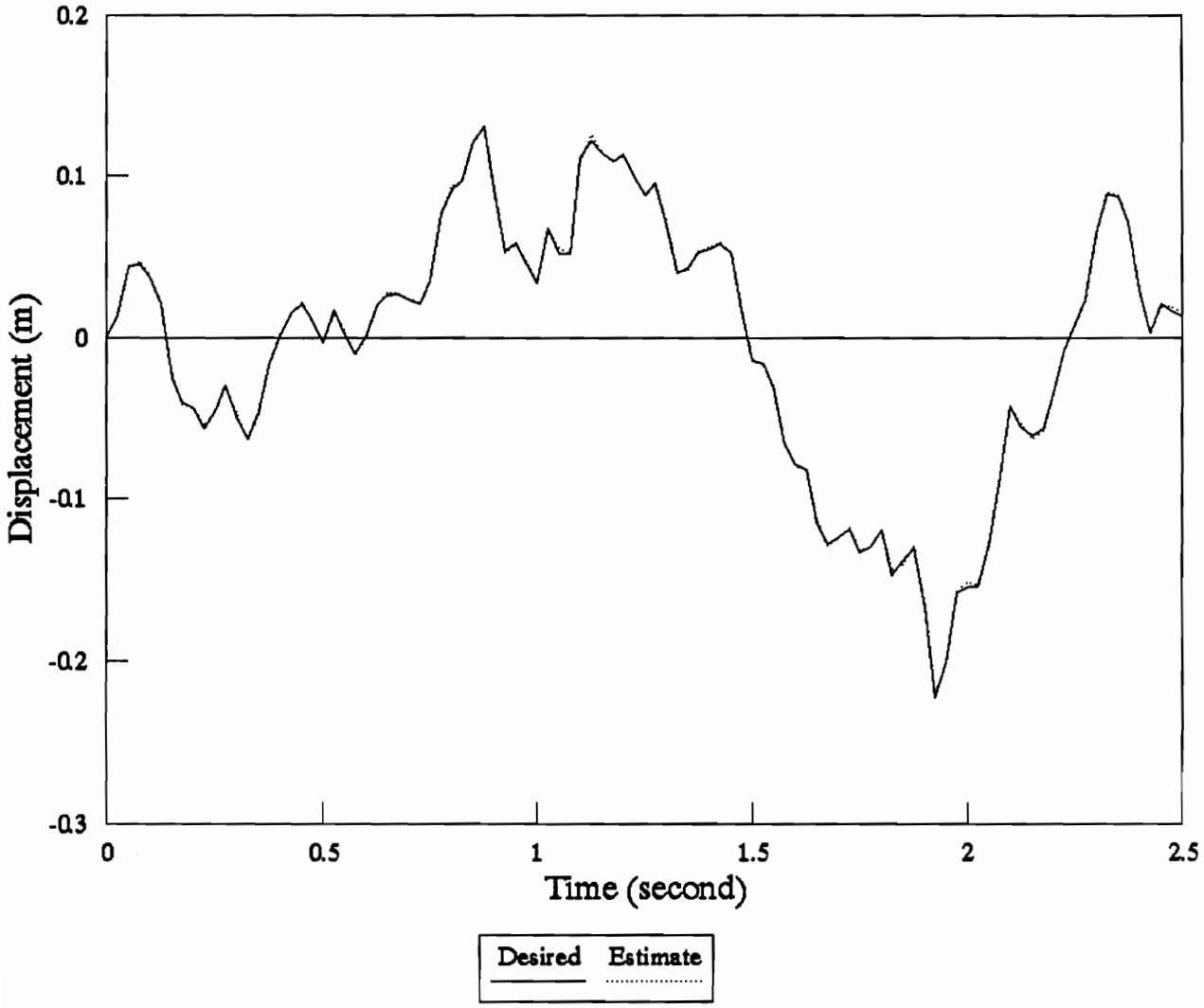

Figure 18 : Full state neural network _ passive axle training (seed = 1).

<Fig19.drw>

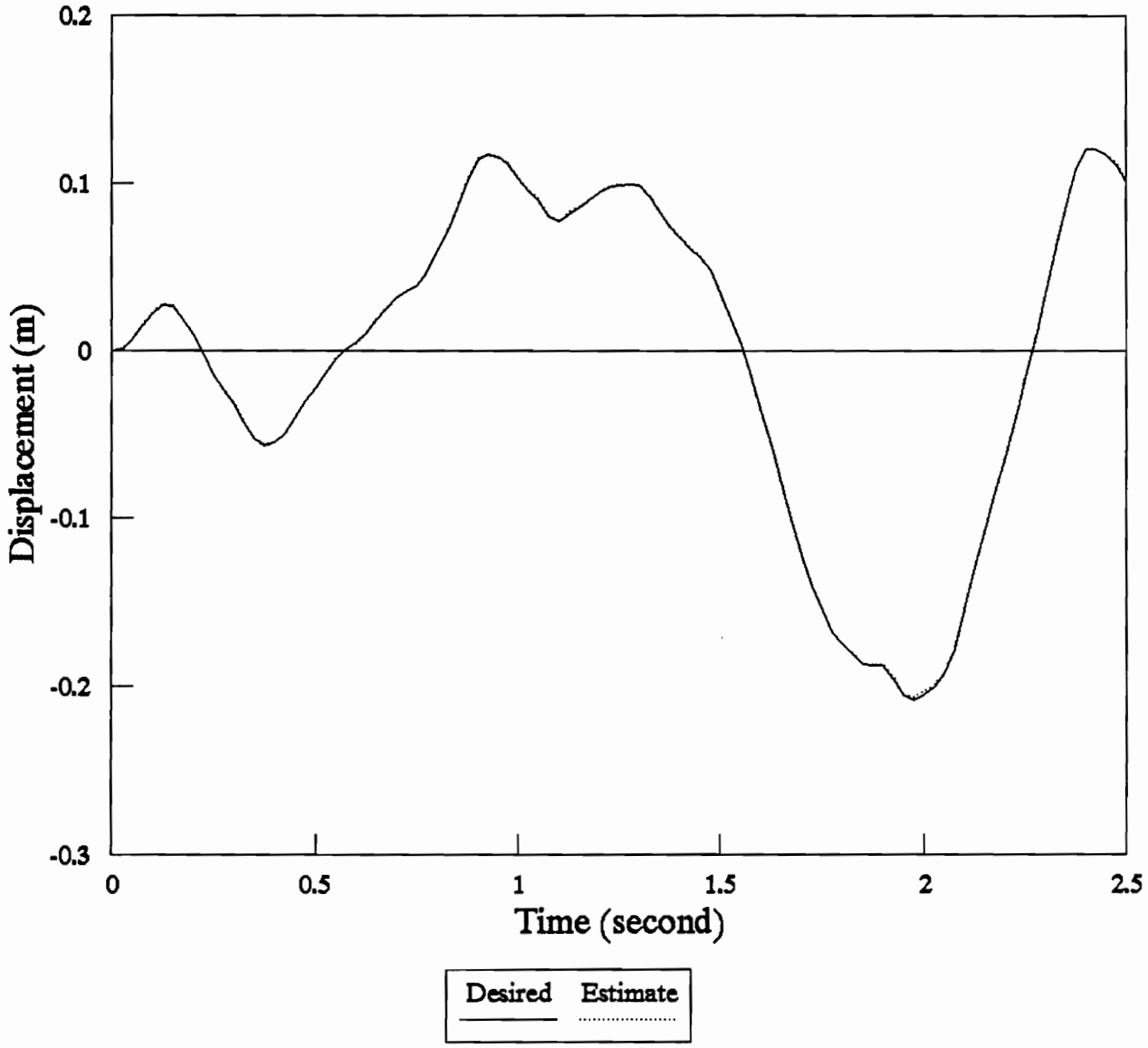

Figure 19 : Full state neural network _ passive body training (seed = 1).

<Fig20.drw>

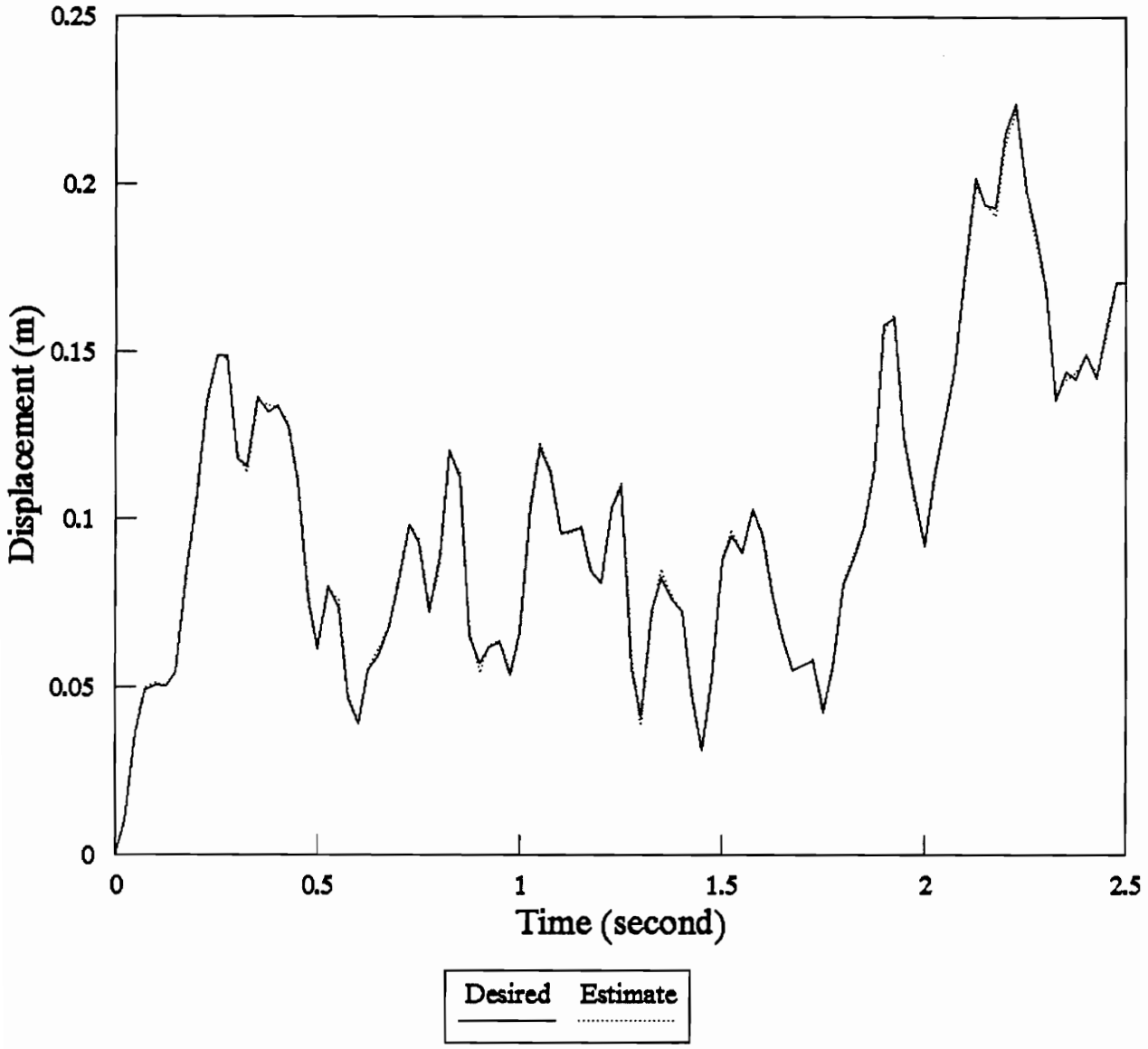

Figure 20 : Full state neural network _ passive axle testing (seed = 0).

<Fig21.drw>

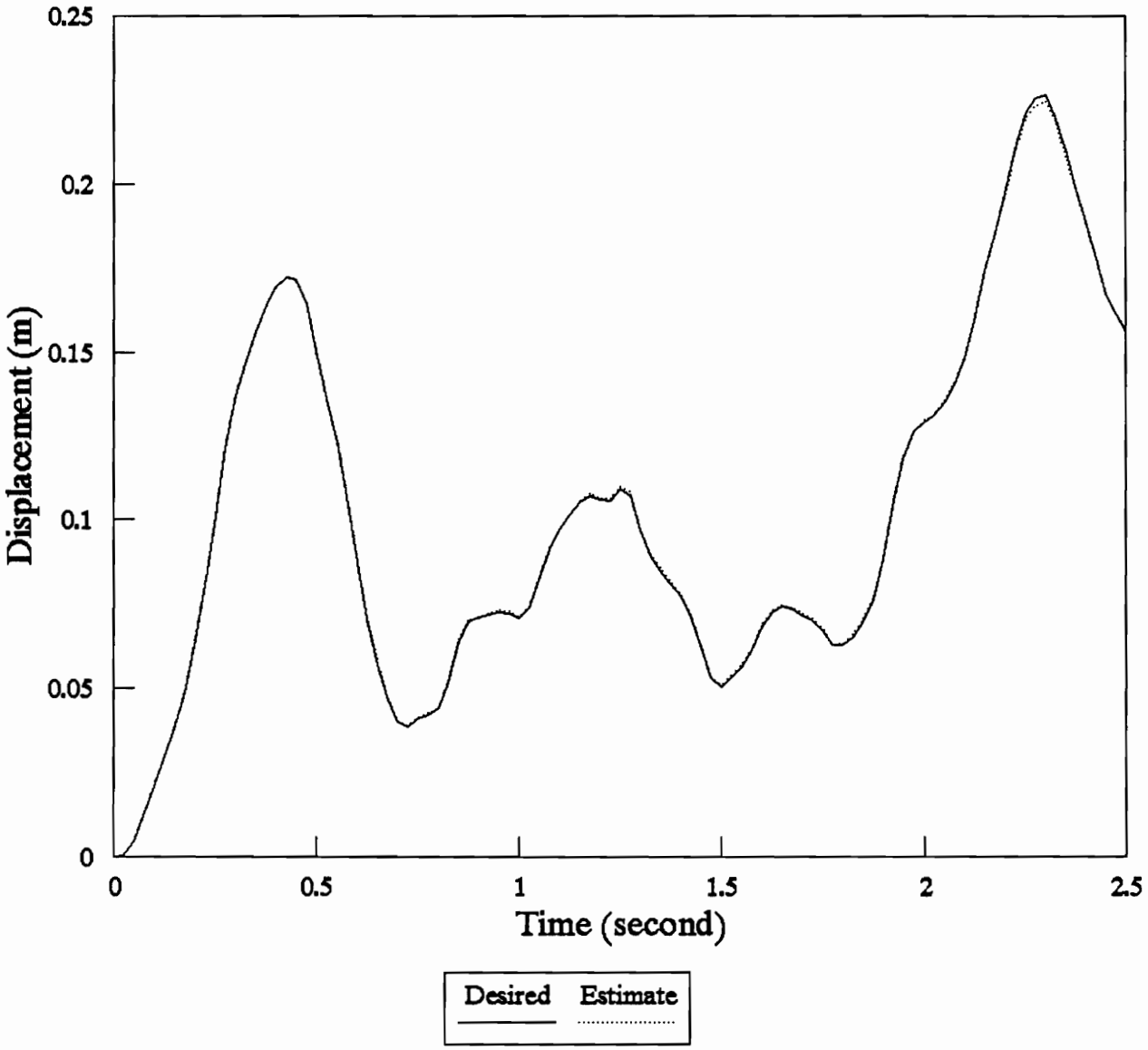

Figure 21 : Full state neural network _ passive body testing (seed = 0).

4.2 Partial State Identification of Passive Suspension

Figure 22 shows the simulation block diagram for the partial state neural network identification of the passive suspension system. The partial state neural network identification model has the following form :

$$\hat{y}_1(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_1(k-1), x_2(k-1)\}$$
$$\hat{y}_2(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_1(k-1), x_2(k-1)\}$$

where NN stands for neural network mapping,

$x_0(k)$ is the road displacement input,

w(k) is the random white noise input,

$x_1(k-1)$ is the time-delayed version of $x_1(k)$,

$x_2(k-1)$ is the time-delayed version of $x_2(k)$,

$x_1(k)$ is the axle displacement input,

$x_2(k)$ is the body displacement input,

A set of training data will be applied to a neural network consisting of above six inputs, one hidden layer of six hidden nodes, and two outputs. Figures 23, and 24 show the plots of the desired axle / body displacements versus the estimate axle / body displacement using neural network on a set of training data. As before, the neural network estimate outputs follow the desired outputs very closely with the absolute error of 0.000999904. Then, a different set of test data will be fed to the

already trained neural network to measure how well the neural network outputs perform against its desired outputs. Figures 25, and 26 show their plots on the data test set. Again, the neural network axle / body displacements closely follow the desired axle / body displacements on the data test set. This indicates that even with partial knowledge of the system states the neural network still is able to identify the system plant accurately.

Here is the weight connections of the trained neural network :

Number of inputs = 6

Number of hidden layers = 1

Number of hidden nodes = 6

Number of outputs = 2

Desired error = 0.001

Actual absolute error = 0.000999904

Number of iterations = 1917777

Weight connections :

$u \rightarrow h$	$h \rightarrow o$
0.00081	0.00229
0.00875	0.00049
1.04834	0.08851
0.04259	0.05827
0.50663	1.66798
-0.18996	-0.06916
-0.40072	-0.73282

<Fig22.drw>

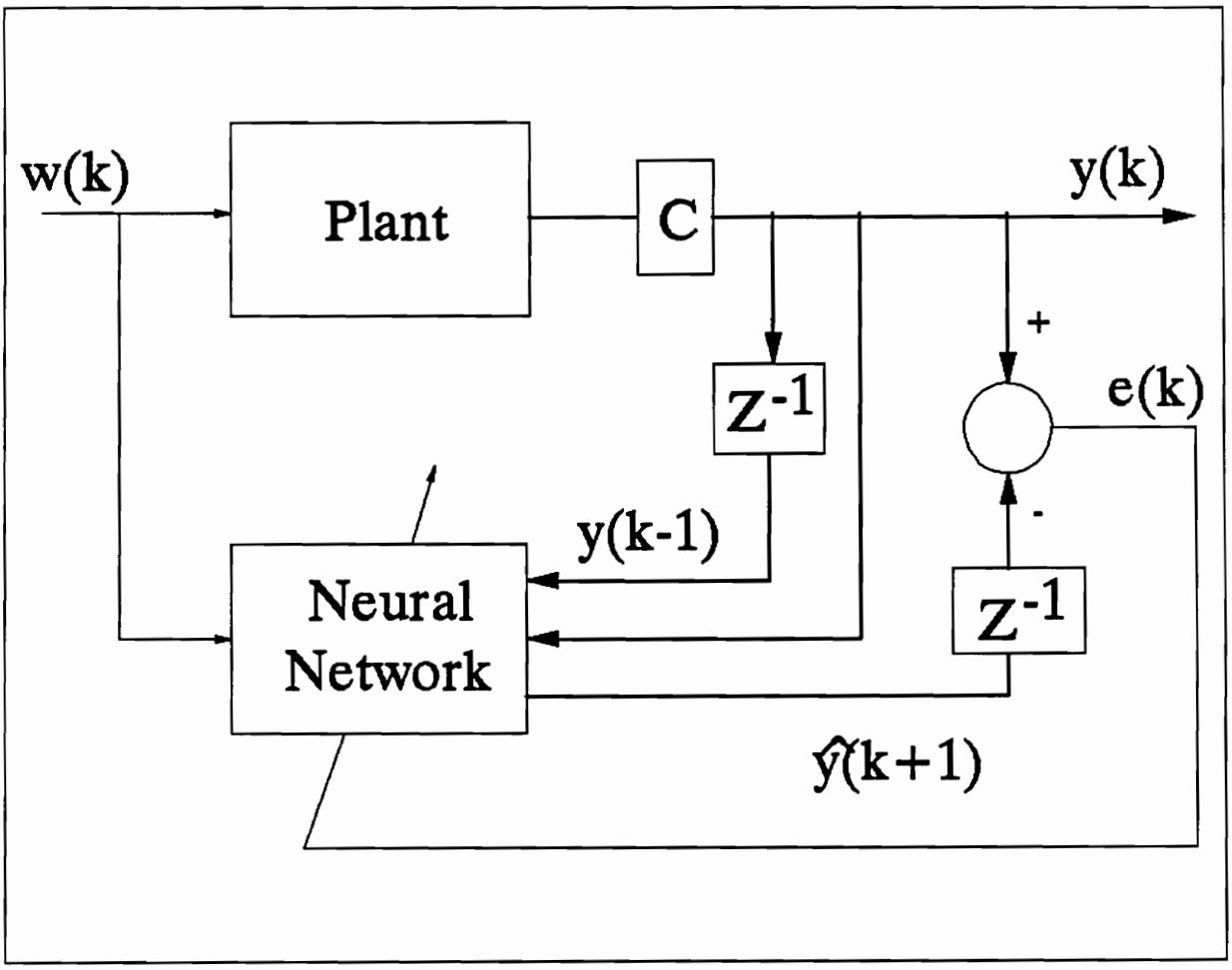

Figure 22 : Partial state neural network identification of passive suspension.

<Fig23.drw>

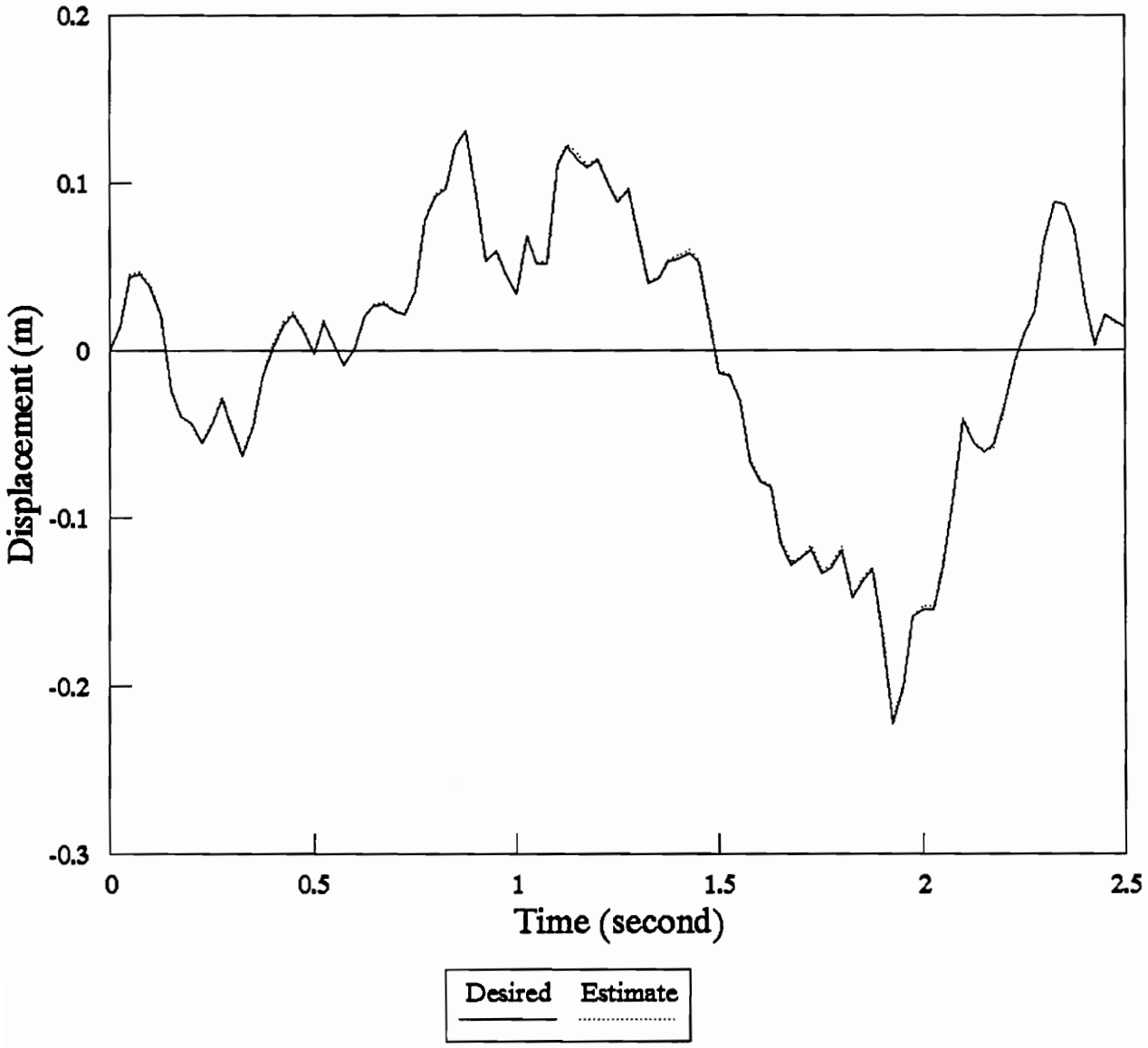

Figure 23 : Partial state neural network _ passive axle training (seed = 1).

<Fig24.drw>

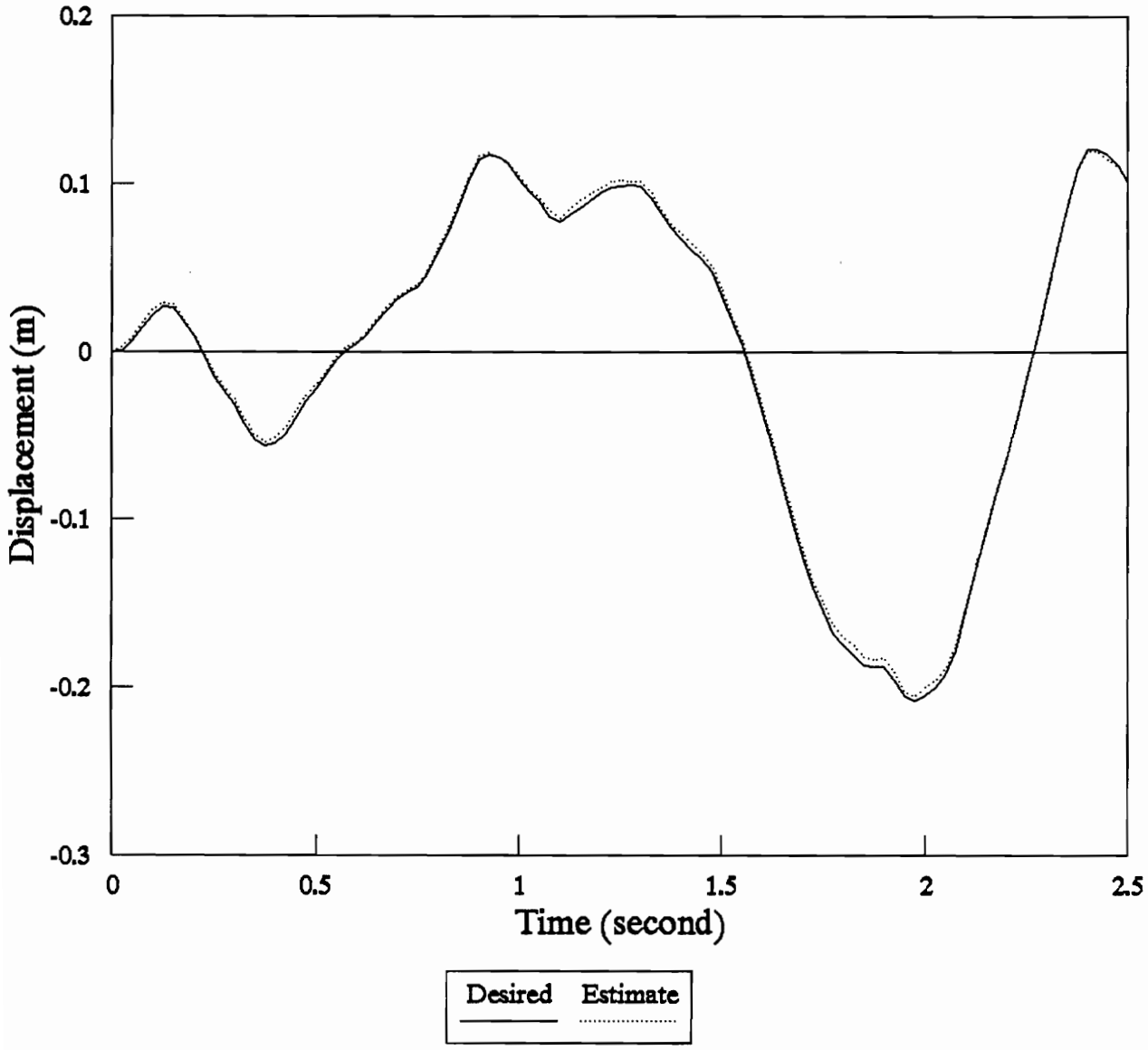

Figure 24 : Partial state neural network _ passive body training (seed = 1).

<Fig25.drw>

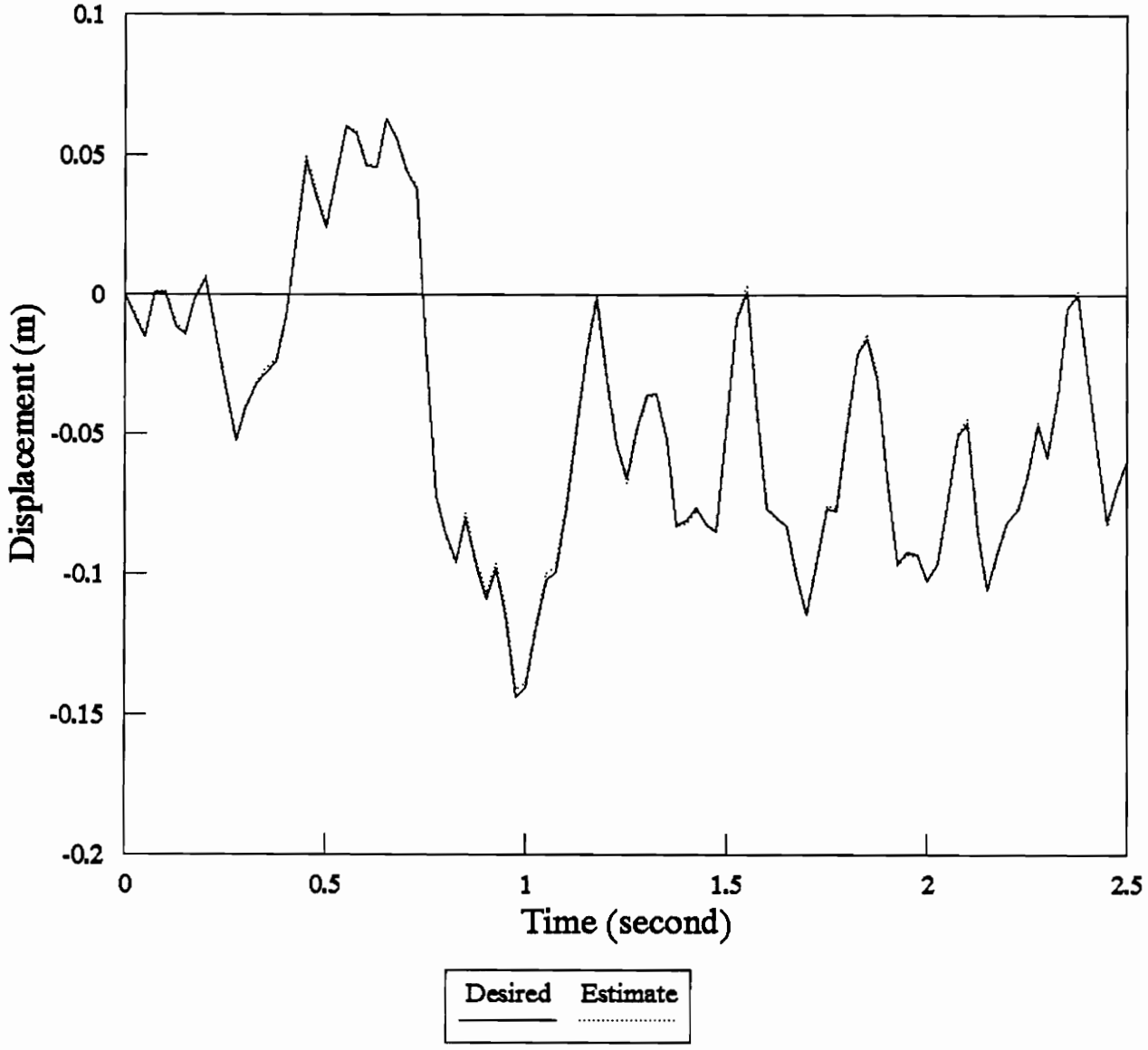

Figure 25 : Partial state neural network _ passive axle testing (seed = 4).

<Fig26.drw>

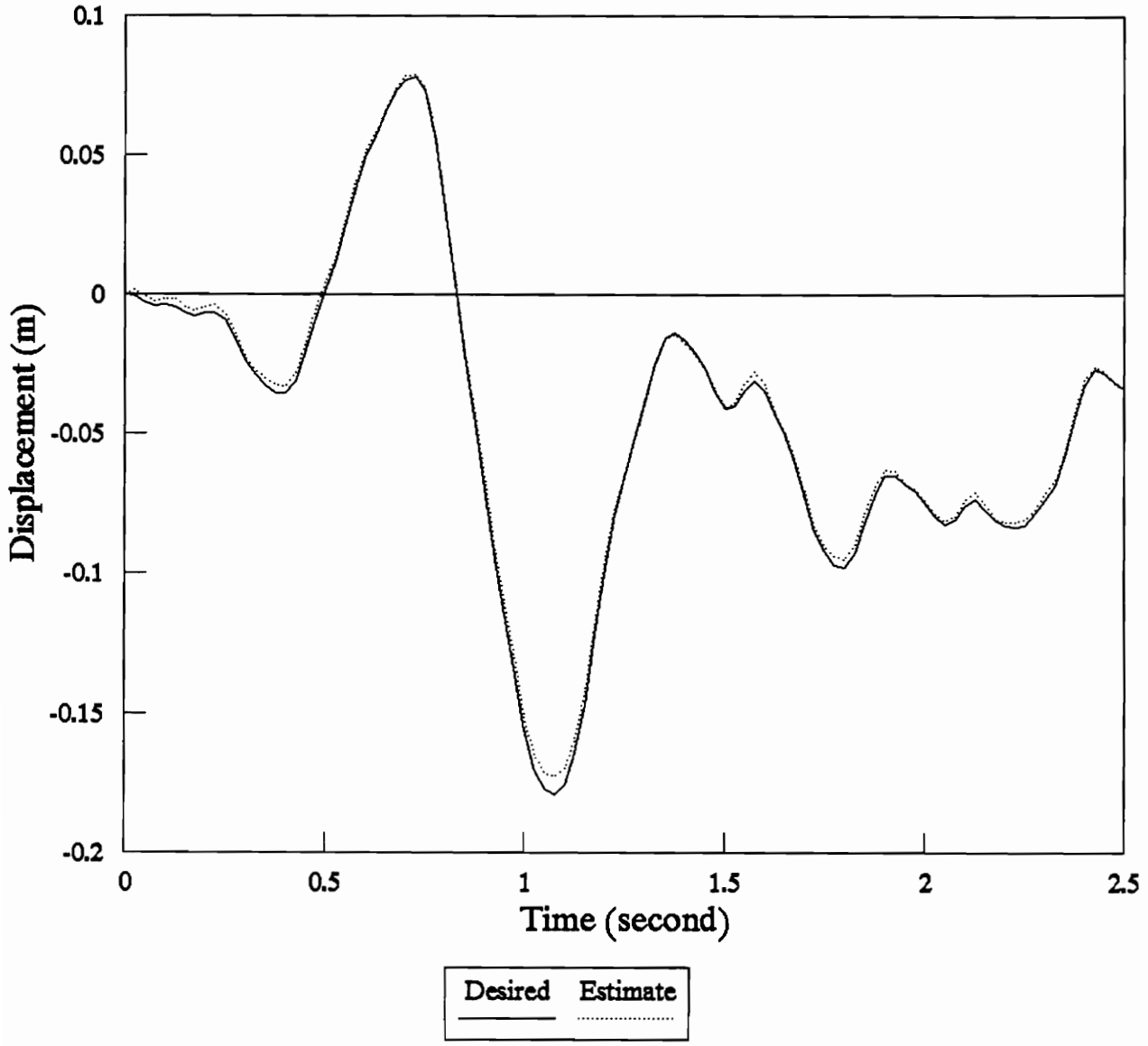

Figure 26 : Partial state neural network _ passive body testing (seed = 4).

4.3 Full State Identification of Active Suspension

Figure 27 shows a simulation block diagram for the full state neural network identification of the active suspension system. The full state neural network identification model has the following form :

$$\hat{y}_1(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_3(k), x_4(k)\}$$
$$\hat{y}_2(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_3(k), x_4(k)\}$$

A neural network consisting of one hidden layer of six hidden nodes, six inputs and two outputs will be trained to identify the active suspension system. Figures 28, 29, 30, and 31 show the plots of the neural network axle / body displacements versus the desired axle / body displacements on a set of training data and on a set of test data. The neural network outputs match the desired outputs almost exactly on both set of data to a level of error of 0.000699788.

Here is the weight connections of the trained neural network :

Inputs = 6 Hidden layer s= 1 Hidden nodes = 6 Outputs = 2

Actual absolute error = 0.000699788

Number of iterations = 277046

Weight connections :	-0.00026	0.00003
	0.01070	0.00024
	0.82894	0.03852
	0.02132	-0.00241
	0.15513	0.96963
	0.00819	0.00075
	0.02166	0.02052

<Fig27.drw>

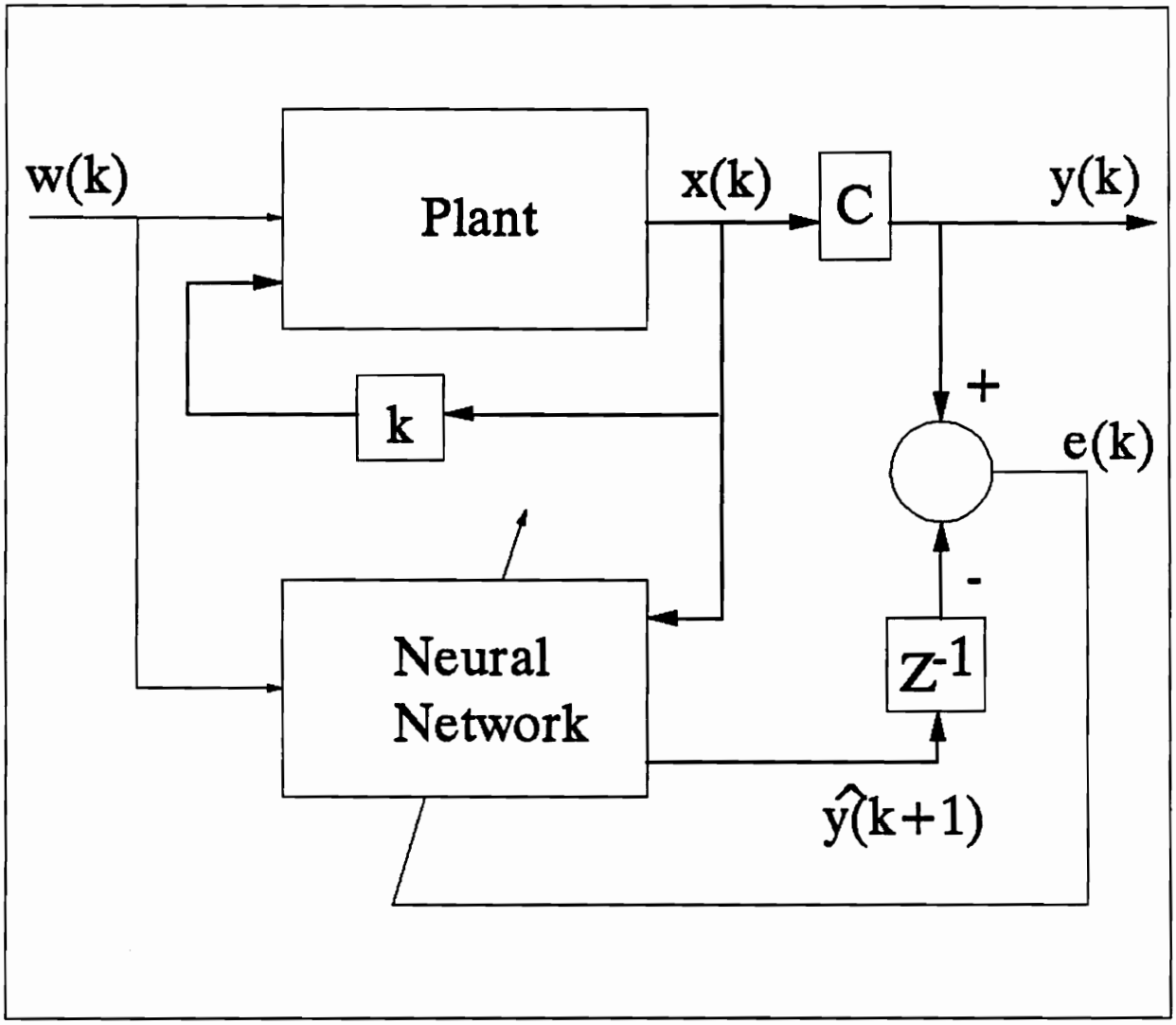

Figure 27 : Full state neural network identification of active suspension.

<Fig28.drw>

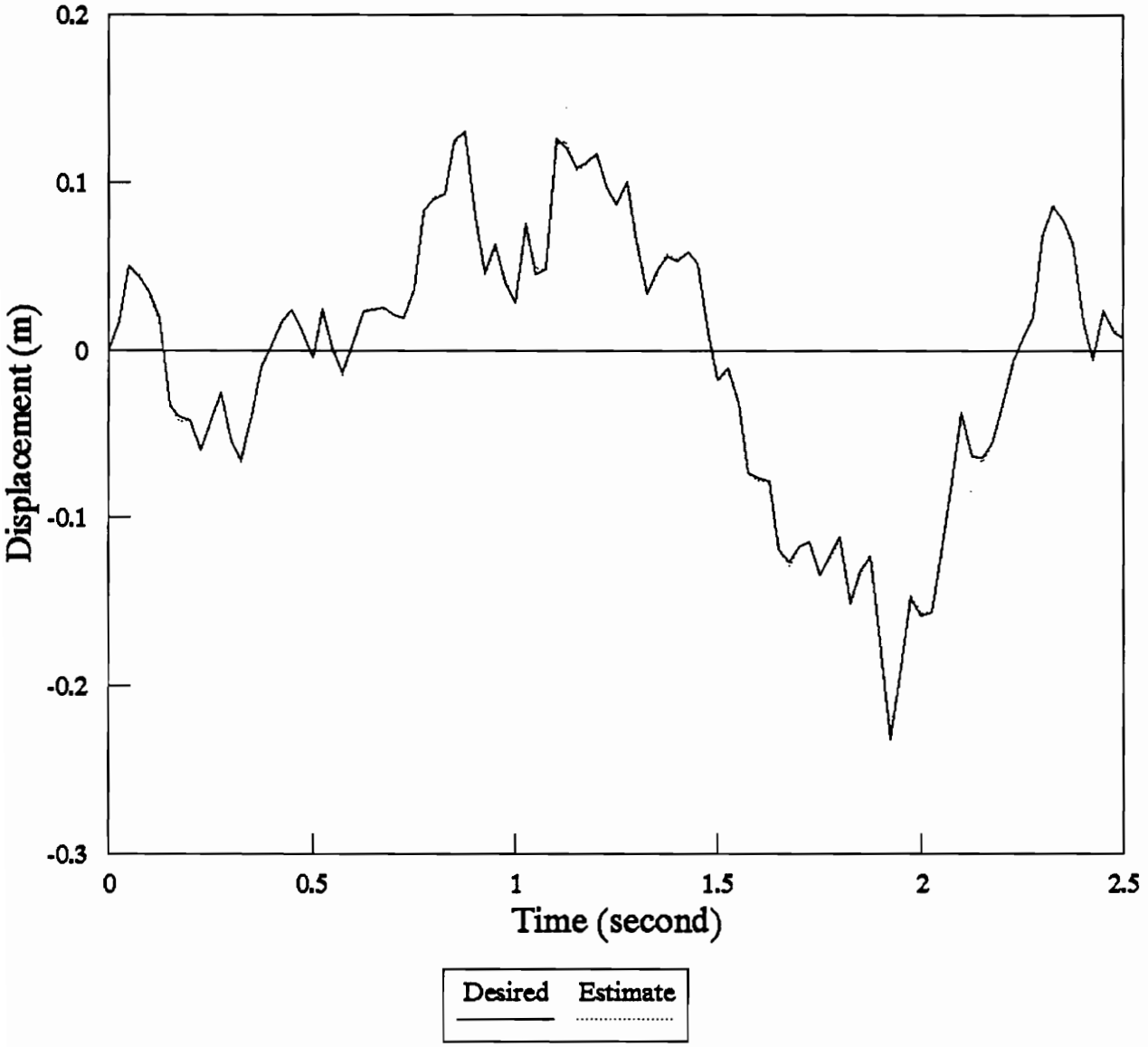

Figure 28 : Full state neural network _ active axle training (seed = 1).

<Fig29.drw>

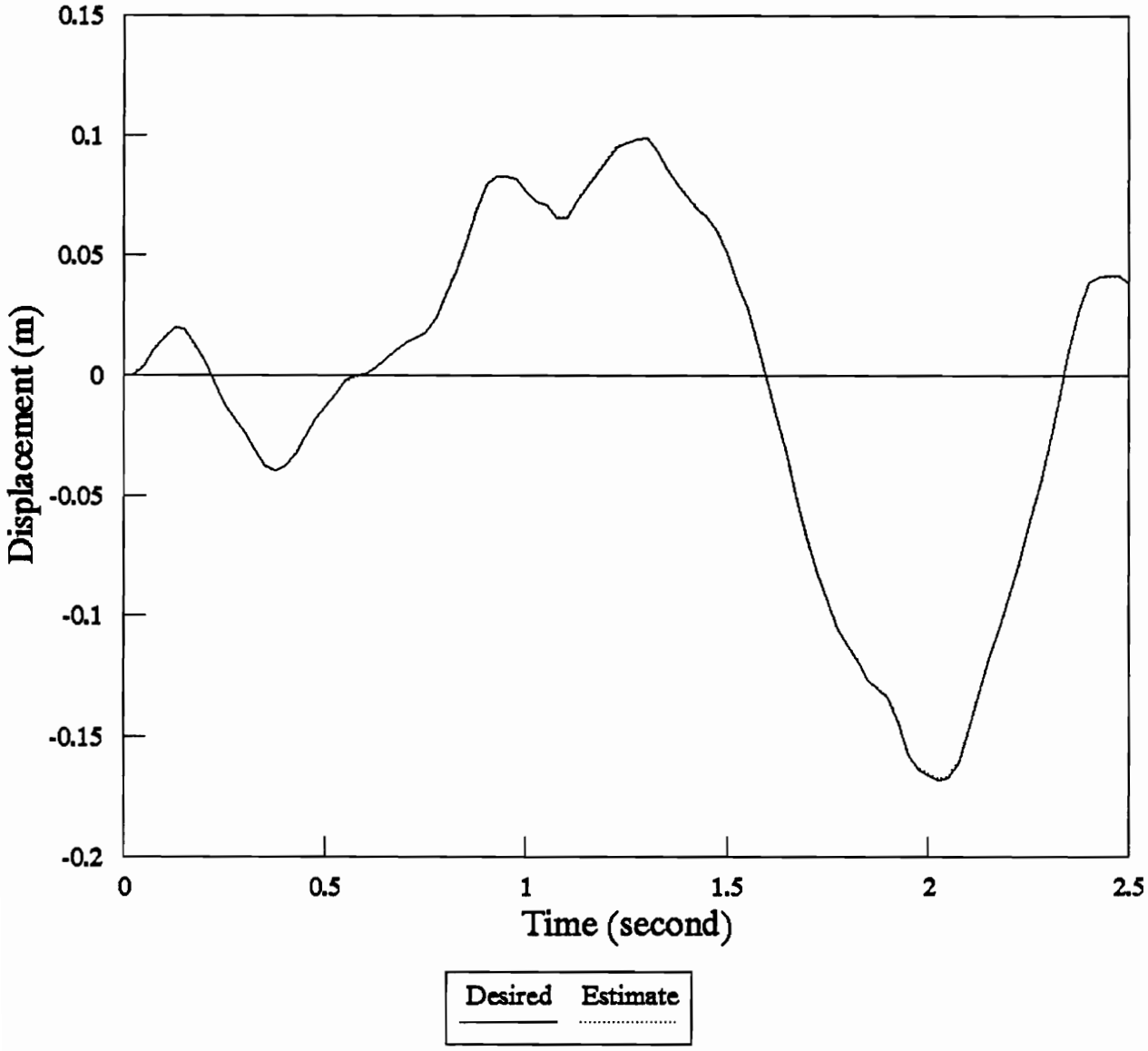

Figure 29 : Full state neural network _ active body training (seed = 1).

<Fig30.drw>

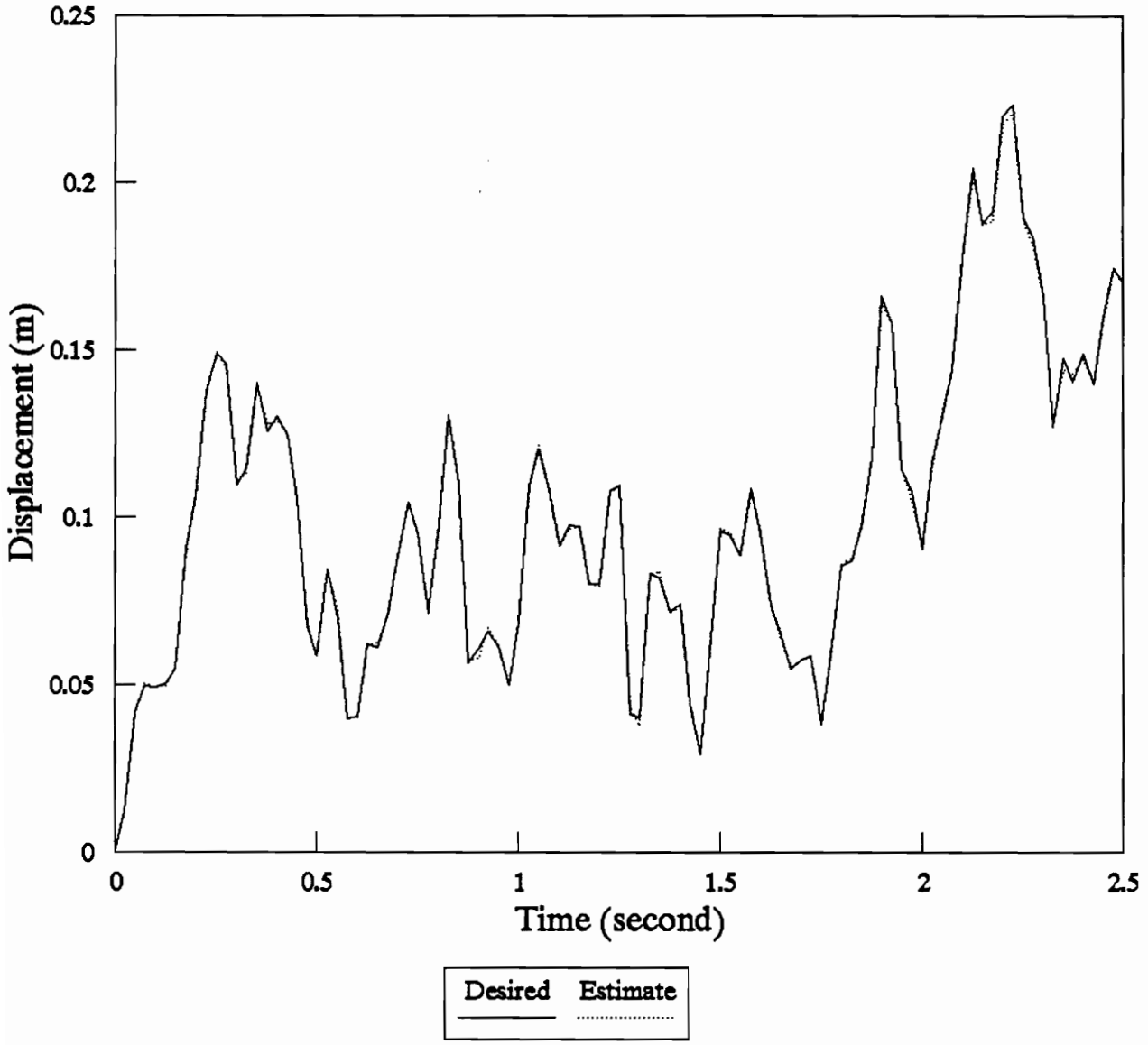

Figure 30 : Full state neural network _ active axle testing (seed = 0).

<Fig31.drw>

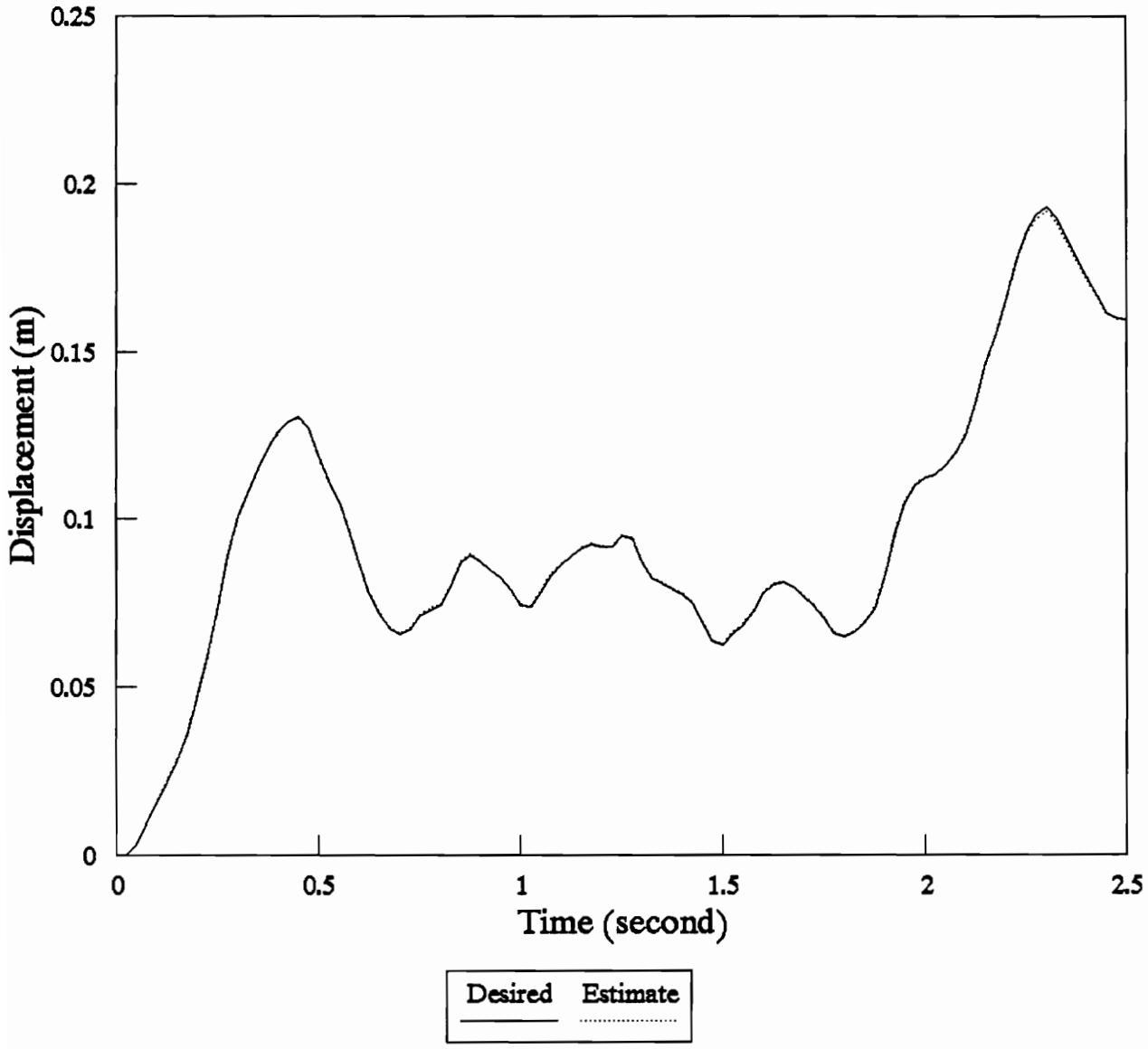

Figure 31 : Full state neural network _ active body testing (seed = 0).

4.4 Partial State Identification of Active Suspension

Figure 32 shows a simulation block diagram for the partial state neural network identification of the active suspension system. The partial state neural network identification model has the following form :

$$\hat{y}_1(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_1(k-1), x_2(k-1)\}$$
$$\hat{y}_2(k+1) = NN\{w(k), x_0(k), x_1(k), x_2(k), x_1(k-1), x_2(k-1)\}$$

where $x_1(k-1)$ is the time-delayed version of $x_1(k)$,

$x_2(k-1)$ is the time-delayed version of $x_2(k)$,

$x_1(k)$ is the axle displacement input,

$x_2(k)$ is the body displacement input.

A neural network consisting of one hidden layer of six hidden nodes, six above inputs, and two outputs will be trained to identify the active suspension system with partial knowledge of the system states. The neural network outputs will be compared against the desired outputs to determine how well the network performs on the partial state measurements. Figures 33, and 34 present the plots of the desired axle / body displacements versus the neural network estimate axle / body displacements on a data training set. As expected, the neural network outputs follow the desired outputs very well to a level of error of 0.00218508 on a training data set. Then, a data test set is fed to the network to determine how well it performs. Figures 35, and 36 also show an exceptional performance of the neural

network estimate outputs against their desired outputs. This indicates that the neural network identification of the active suspension system still performs well even with partial knowledge of the system states. The error between the network outputs and the desired outputs are very small.

Here is the weight connections of the trained neural network :

Number of inputs = 6

Number of hidden layers = 1

Number of hidden nodes = 6

Number of outputs = 2

Actual absolute error = 0.00218508

Number of iterations = 831227

Weight connections :

$u \rightarrow h$	$h \rightarrow o$
-0.00029	-0.00030
0.01063	0.00027
1.29045	0.05904
-0.17693	0.09295
0.33231	0.67607
-0.12434	0.08977
-0.29906	0.10381

<Fig32.drw>

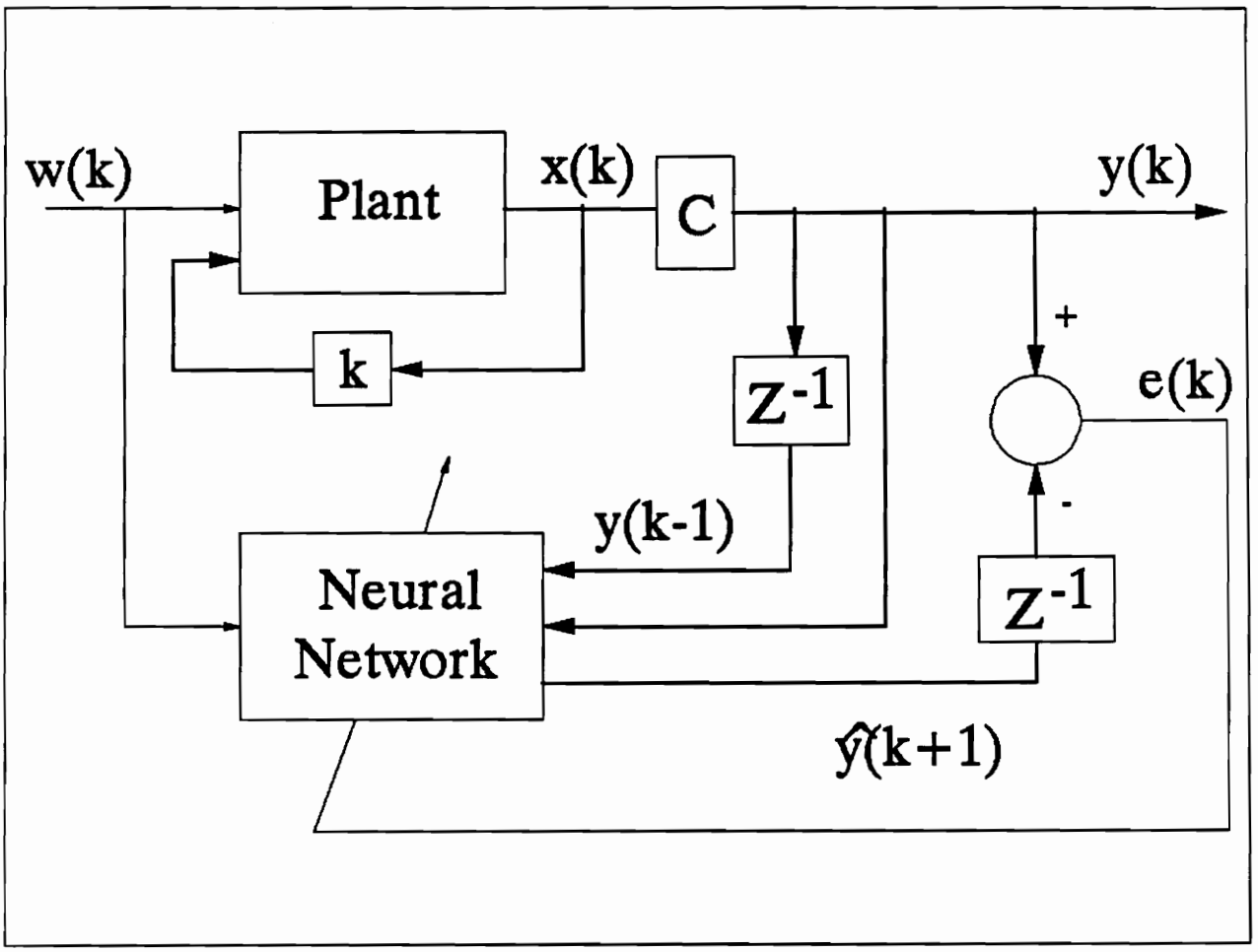

Figure 32 : Partial state neural network identification of active suspension.

<Fig33.drw>

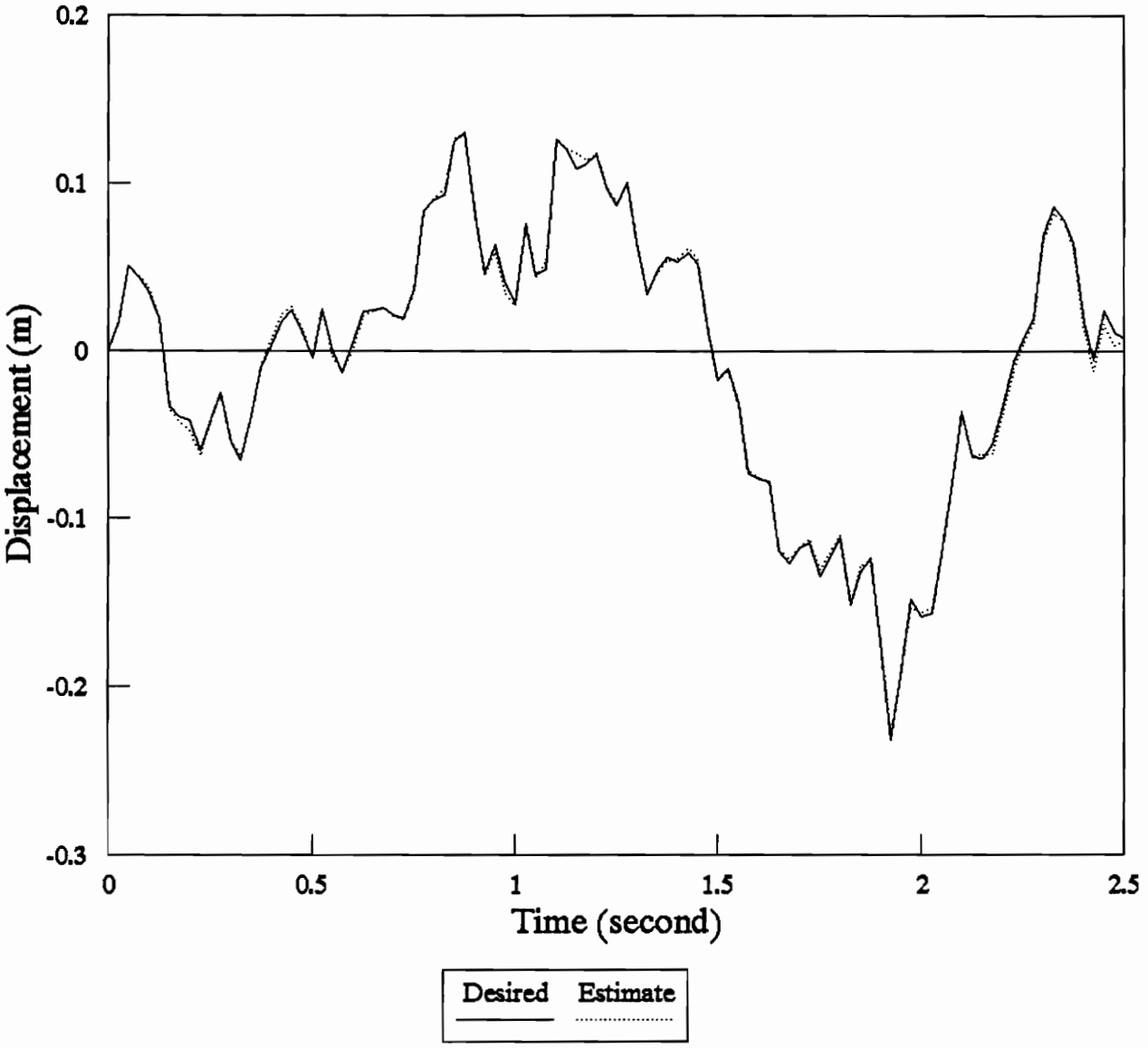

Figure 33 : Partial state neural network _ active axle training (seed = 1).

<Fig34.drw>

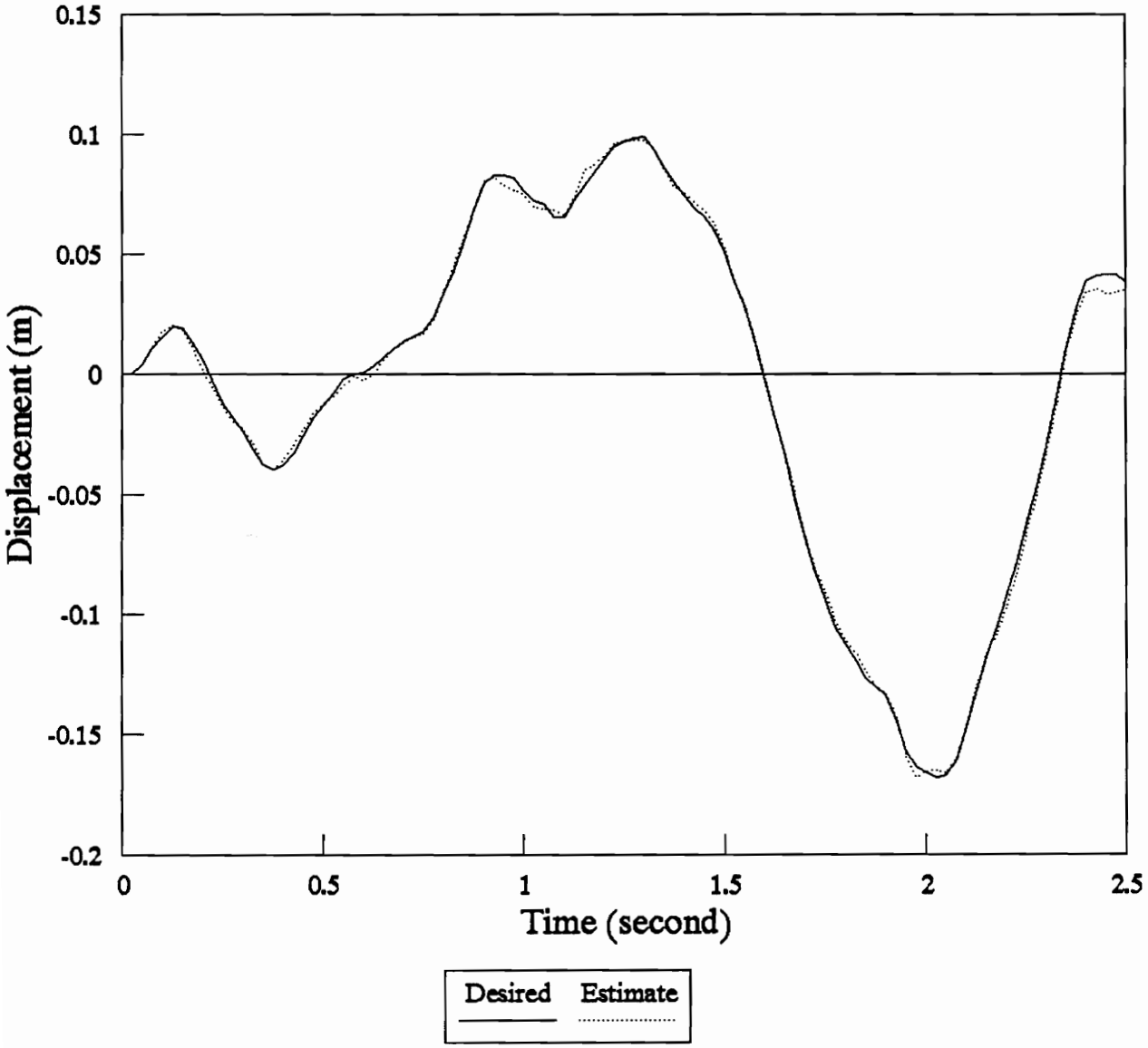

Figure 34 : Partial state neural network _ active body training (seed = 1).

<Fig35.drw>

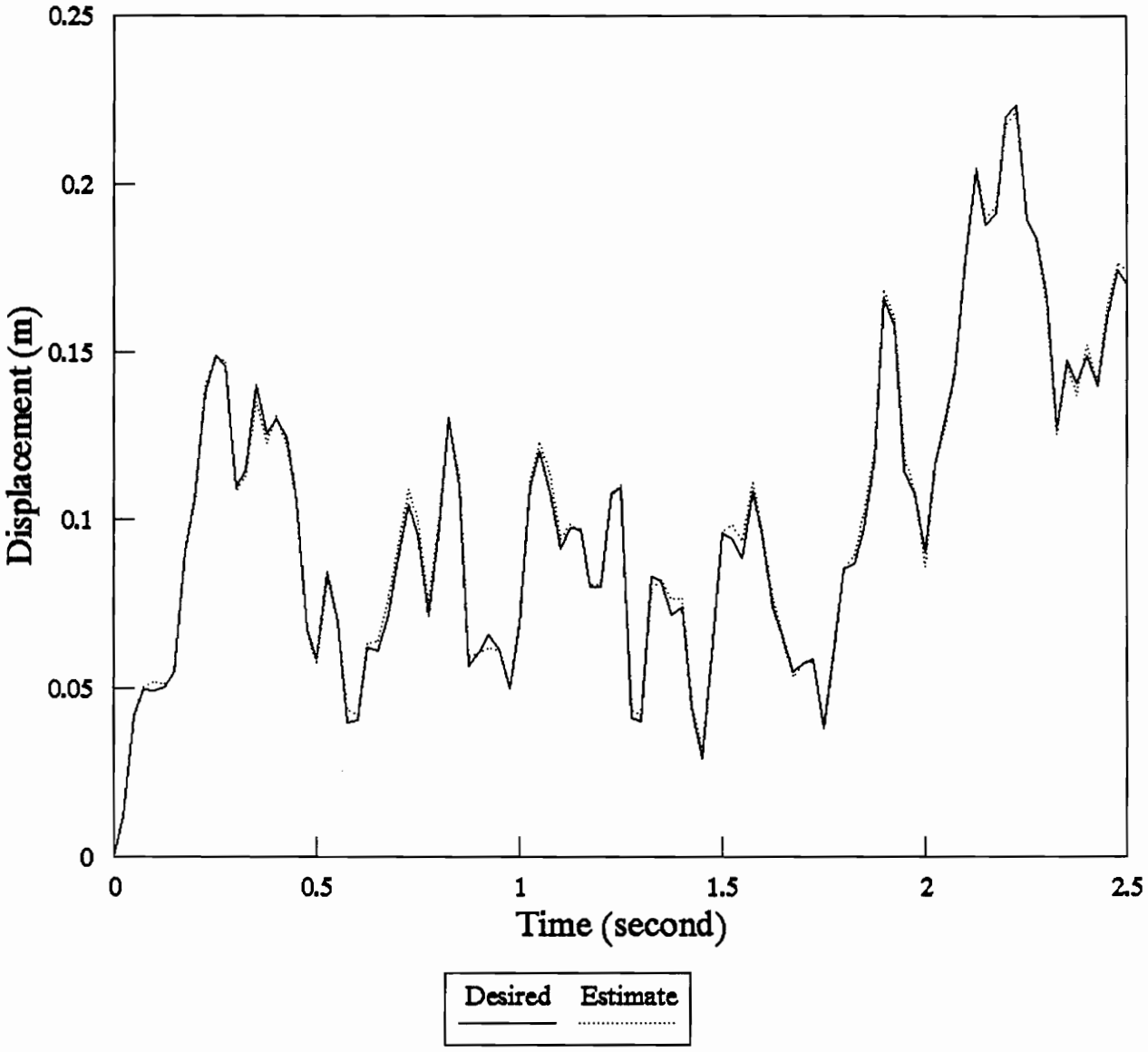

Figure 35 : Partial state neural network _ active axle testing (seed = 0).

<Fig36.drw>

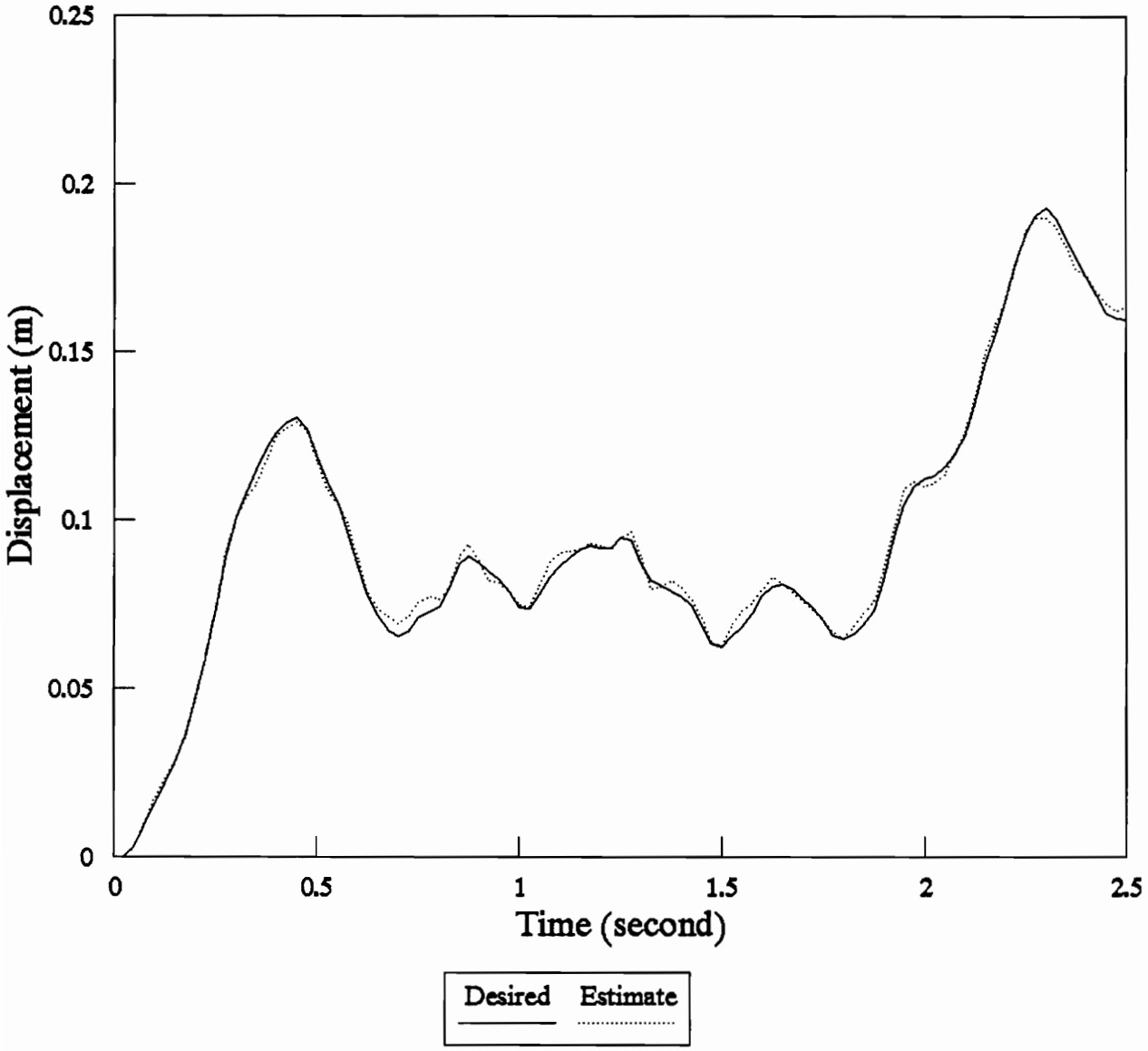

Figure 36 : Partial state neural network _ active body testing (seed = 0).

4.5 Conclusion

In this paper, layered neural networks are used to identify both passive and active suspension systems with the full state measurements or partial state measurements. The simulation results suggest that the neural networks are powerful tools for system identification.

The advantage of neural networks is that it is a model-free approximator. That is, knowledge of the actual plant model is not needed. Neural networks learn to approximate the model from a set of input-output data. It can approximate most linear as well as nonlinear systems provided enough hidden layers and hidden nodes together with a rich and compact data set. Another advantage of neural networks is its distributed associative memory. That is, each piece of information is stored distributively in many memory units so that a failure of one connection will not affect much of its performance.

The disadvantage of neural networks is that training the network is time-consuming. Many backpropagation error surfaces have extensive flat areas and troughs to greatly slow down the rate of convergence. Otherwise, neural networks are quite powerful tools for system identification.

References

1. R.M. Goodall and W. Kortum, "Active Controls in Ground Transportation - A Review of the State-of-the-Art and Future Potential", Vehicle System Dynamics, Vol. 12, pp. 225-257, 1983.

2. S. Rakheja, Hong Su and T.S. Sankar, "Analysis of a Passive Sequential Hydraulic Damper for Vehicle Suspension", Vehicle System Dynamics, Vol. 19, pp. 289-312, 1990.

3. R.S. Sharp and D.A. Crolla, "Road Vehicle Suspension System Design - a review", Vehicle System Dynamics, Vol. 16, pp. 167-192, 1987.

4. H.B. Sutton, "The Potential for Active Suspension Systems", Automotive Engineer (London), Vol. 4, No. 2, pp. 21-24, April-May 1979.

5. J.K. Hedrick, D.N. Wormley, "Active Suspensions for Ground Transport Vehicles - A State of the Art Review", ASME AMD, Vol. 15, pp. 21-40, 1975.

6. D.L. Margolis and M. Goshtasbpour, "The Chatter of Semi-Active On-Off Suspensions and its Cure", Vehicle System Dynamics, Vol. 13, pp. 129-144, 1984.

7. D.C. Karnopp, "Vehicle Response to Stochastic Roadways", Vehicle System Dynamics, Vol. 7, pp. 97-109, 1978.

8. A.G. Thompson, "An Active Suspension with Optimal Linear State Feedback", Vehicle System Dynamics, Vol. 5, pp. 187-203, 1976.

9. A.G. Thompson, "Quadratic Performance Indices and Optimum Suspension Design", Proc. Inst. of Mechanical Engineers, Vol. 187(9), pp. 129-139, 1973.

10. H. Kwakernaak and R. Sivan, "Linear Optimal Control Systems", Wiley-Interscience, 1972.

11. A.E. Bryson and Y.C. Ho, "Applied Optimal Control", Revised Printing, Hemisphere Publishing Corp., Washington, DC, 1975.

12. D.E. Kirk, "Optimal Control Theory : An Introduction", Prentice Hall Inc., 1970.

13. B. Kosko, "Neural Networks and Fuzzy Systems : A Dynamical Systems Approach To Machine Intelligence", Prentice Hall Inc., 1991.

14. R. Hecht-Nielsen, "Neurocomputing", Addison-Wesley Publishing Co., 1990.

15. K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks", IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4-27, March 1990.

16. "Neural Computing : NeuralWorks Professional II/Plus and NeuralWorks Explorer", NeuralWare, Inc., 1991.

17. S.G. Kong and B. Kosko, "Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems", submitted to IJCNN-90, June 1990.

Appendix A. Backpropagation source code

```
/*****************************************************************************
*   Program      :  Backpropagation Neural Network
*   Date         :  12/01/1991
*   Programmer   :  Michael Tran
*   Description  :  This backpropagation neural network is programmed to train a network with
                     unlimitted number of hidden layers as well as unlimitted number of hidden
                     nodes.  It is mouse interfaced menu-driven and very friendly to use.  The users
                     will be asked for required information about the training network and provide a
                     training data file.  The program is preset other parameters.  These parameters
                     can be viewed and changed during the training by the users.
*****************************************************************************/

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#include <dos.h>
#include <alloc.h>

#define File_left 20
#define Select_left 80
#define Run_left 160
#define View_left 220
#define ESC 27
#define SPACE 32
#define BACKSPACE 8
#define RETURN 13
#define LEFTKEY 75
#define RIGHTKEY 77
#define DOWN 80
#define UP 72
#define YES 1
#define NO 0
#define HALF_PI 1.5708

/*  Mouse call prototype  */
void mouse_call (int regax, int regbx, int regcx, int regdx);
void Mouse_display (void);

/*  Screen save and screen restore prototype  */
```

```
void *Save_screen (int Left, int Top, int Right, int Bottom);
void Restore_screen (int Left, int Top, void *Screen);
void Restore ( int L );

/*  Initialize graphics and mouse menu setups  */
void Graph_initialize (void);
void Menu1_setup (void);
void Menu2_setup (void);

void Select ( int Position );
void Sub_menu (int Left, int Top, int Right, int N_cel, char Sub [5][20]);
void Get_menu ( void );
void Status ( int Cell);
void Cell_select(int CL, int CT, int Cel, int CR, char Sub[5][20], void *Sel);
void C_box ( int CL, int CT, int R, int CB );
void Weight_init (void);


/*  File select menu  */
void File_select (void);
    void Load_weight (void);
    void Save_weight (void);
    void Report (void);
    void New_network (void);
    void Quit (void);

/*  Select select menu  */
void Select_select (void);
    void Feed_thru (void);
    void Batch_size (void);
    void Data_file (void);
        int File_lenght (void);
        void Get_data (void);
    void Signal_type (void);
        void F_assign (void);
    void Error_type (void);

/*  Run select menu  */
void Run_select (void);
    void Test (void);
    void Train (void);
       void Bkp_algo ( int m );
          void In_to_out ( int t );
          void DeltaUXY ( int t );
          void Weight_update ( int t);
    void Restart (void);
    void Continue (void);
//  void Display (void);

/*  View select menu  */
void View_select (void);
    void Fast_rate (void);
    void Learn_rate (void);
    void Momentum (void);
```

```
/*  Line plotting prototype  */
void Line_plot (double *y, int l, int t, int r, int b, double Min, double Max);
void Out_plot (void);       //  Output plot
void Err_plot (void);       //  Error plot

/*  Dynamic memory reserve, delete, and check  */
void Memleft(void);
void Delete_3D ( int Row, int Wid, double ***S );
void Delete_2D ( int Row, double **S );
double ***Mem_3D ( int Row, int Wid, int Height );
double **Mem_2D (int Row, int Wid);

/*  Screen save or restore locations  */
void *Menu;
void *Temp;
void *Shell;
void *Temp1;

/*  Types of squashing functions  */
double Sigmoid ( double x );
double Linear ( double x );
double Bipolar ( double x );
double TanH ( double x );
double Sine ( double x );

/*  Derivative version of squashing functions  */
double D_sigmoid ( double y );
double D_linear ( double y );
double D_bipolar ( double y );
double D_tanh ( double y );
double D_sine ( double y );

/*  Error types prototypes  */
double Abs_error ( void );
double Sq_error ( void );
double Norm_max_error (void);

/*  Pointers to functions  */
double (*Squash) (double);
double (*D_squash) (double);
double (*E_select) (void);

/*****************************************************************************
*   Program parameters :
*     *Lcoef       : Dynamic array of learning coefficients.
*     *Mcoef       : Dynamic array of momentum coefficients.
*      Fcoef       : Fast learning coefficient.
*     Desire_error  : Desire level of error.
*     **U          :  Dynamic arrays of inputs.
*     **Y          :  Dynamic arrays of outputs.
*     ***W         :  Three dimensional dynamic arrays of connection weights.
*     ***W_old     :  To store previous connection weights.
*     *YO          :
```

```
*    *YD          :
*    *E_local     :  Dynamic array of local errors.
*    Avg_error    :  Output average error.
*    Y_max        :  Set maximum value of the output plots.
*    Y_min        :  Set minimum value of the output plots.
*    **X          :  Two dimensional dynamic array of nodes in hidden layers.
*    **Delta      :  Weight change direction.
*   *Hd_nodes     :  Dynamic array of hidden nodes.
*   Hd_layers     :  Number of hidden layers.
*   In_nodes      :  Number of input nodes to the network.
*   Out_nodes     :  Number of output nodes to the network.
*   Out_disp      :  Output to be plotted is initially set to 1 and can be change in the menu.
*********************************************************************************/

double *Lcoef, *Mcoef, Fcoef, Desire_error;      // double *Fcoef
double **U, **Y, ***W, ***W_old, *YO, *YD, *E_local, Avg_error = 1000;
double Y_max, Y_min, **X, **Delta;

int *Hd_nodes, Hd_layers, In_nodes, Out_nodes, Out_disp = 1;

/*  Set  Feedthru, Batchsize, Err_flag, Continue_flag  */
int Feedthru = NO, Batchsize = 0, Err_flag = NO, Continue_flag = NO;

/*  Define mouse registers, signal transfer types, error types  */
int globax, globbx, globcx, globdx, F_type, E_type, Data_flag = NO;
int exit_status = NO, Dat_pts, Max_nodes, New_flag = NO, Repeat_rate = 0;
long Iter;
unsigned long Mem;

/*  Define output and data file pointers  */
FILE *O_file;
FILE *D_file;

char *Datafile;
char Buffer[25];
char Buf[25];
char *Value (int Cnt, int Tp, int RR);

/*  Initialize list of sub-menus  */
char File_menu [5][20] =  { "Load weight",
                            "Save weight",
                            "Report",
                            "New network",
                            "Quit"  } ;
char Select_menu [5][20] = { "Feedthru",
                             "Batch size",
                             "Data file",
                             "Signal type",
                             "Error type"  } ;
char Run_menu [5][20] =   { "Test ",
                            "Train ",
                            "Restart ",
                            "Continue ",
                            "Display "  } ;
```

```
char View_menu [5][20] =   { "Weight ",
                             "Momentum ",
                             "Learn rate",
                             "Fast rate",
                             "Input data"  };
char Signal_menu [5][20] = { "Linear",
                             "Sigmoid",
                             "Bipolar",
                             "TanH",
                             "Sine" };
char Error_menu [3][20]  = { "Absolute error",
                             "R.M.S. error",
                             "Norm max error" };
/*  Main Program */

void main ()
{
  Graph_initialize ();          // Initialize the graphics.
  Menu1_setup ();               // General menu setup.
  Mouse_display();              // Mouse setup.
  Get_menu ();                  // Get to menu programs.
  closegraph();                 // Close the graphics.
  clrscr();                     // Clear screen.
}

/*  This subroutine is used to train the neural network   */
void Train (void)
{
  int k, i, c, Past = -1, Plot_flag = 1, Ratio;
  double Error;
/*  Re-initialize these parameter if continue training is not selected   */
  if ( Continue_flag == NO )
  {
    X = Mem_2D (Hd_layers,Max_nodes);
    YO = new double [Dat_pts];
    YD = new double [Dat_pts];
    E_local = new double [Dat_pts];
    Delta = Mem_2D (Hd_layers ,Max_nodes );
    for ( k = 0; k < Hd_layers; k++ )
        X[k][0] = 1.0;
  }
/*  Set up the rest of the menu   */
  Menu2_setup();
  setcolor(WHITE);
  sprintf(Buffer,"%g", Desire_error);
  outtextxy (537,333,Buffer);
/*  Looping until output error less than desire error or user selects to exit   */
  do
  {
    for ( k = 0; k < Dat_pts; k++ )
    {
        Bkp_algo(k);                          // Call backpropagation algorithm
/*  Determine repeated rate   */
        Ratio = (int) (E_local[k]/Avg_error);
```

```
        if (E_local[k] > (1.5*Avg_error))
        {
          for ( i = 0; i < Repeat_rate; i++ )
            Bkp_algo(k);
        }
        else
        {
          if ( Ratio > Repeat_rate ) Ratio = Repeat_rate;
          for ( i = 0; i < Ratio; i++)
            Bkp_algo(k);
        }
    }
    Avg_error = E_select();                  // Call to calculate average error.
    Error = Avg_error - Desire_error ;
/******* Plot ******/
    setfillstyle(SOLID_FILL,MAGENTA);
    bar (530,20,600,38);
    sprintf(Buffer,"%li", Iter);             // Set buffer to display the number of iterations.
    setcolor(WHITE);
    outtextxy(540,26,Buffer);
    setfillstyle(SOLID_FILL,CYAN);
    bar (348,330,488,340);
    sprintf(Buffer,"%g",Avg_error);          // Set buffer to display the average error value.
    outtextxy(352,333,Buffer);
    if ( Plot_flag == -1 && Past == 1)
        outtextxy(340,26,"Freeze");
    if ( Plot_flag == 1 )                    // Update the plot if Plot_flag = 1.
    {
        Out_plot();
        Err_plot();
    }

    if ( (kbhit()) )                         // User selections.
    {
        c = getch();
        if ( c == ESC )
          break;
        if ( c == 70 || c == 102 )
        {
          setfillstyle(SOLID_FILL,MAGENTA);
          bar (338,25,438,38);
          Plot_flag = -Plot_flag;
          Past = -Past;
        }
    }
    if ( Error < 0.0 )
        break;
  } while (1);
}

/*  Backpropagation neural network algorithm with any numbers of hidden layers and hidden nodes  */
void Bkp_algo ( int m )
{
  Iter++;
```

```
  In_to_out (m);
  YD[m] = Y[m][Out_disp];
  YO[m] = X[Hd_layers-1][Out_disp];
  E_local[m] = fabs(YD[m] - YO[m]);
  DeltaUXY (m);
  Weight_update (m);
}

/*  To test after training the neural networks   */
void Test ( void )
{
  int i;
  FILE *f;
  char *Testfile;
  Temp1 = Save_screen(260,24,608,51);
  mouse_call(2,0,0,0);
  sprintf(Buffer,"Enter filename : ");
  Testfile = Value (0, 25, 260);
  Restore_screen(260,24,Temp1);
  delete(Temp1);
  for ( i = 0; i < Dat_pts; i++ )
  {
    In_to_out (i);
    YD[i] = Y[i][Out_disp];
    YO[i] = X[Hd_layers-1][Out_disp];
    E_local[i] = fabs(YD[i] - YO[i]);
  }
  Avg_error = E_select();
  setfillstyle(SOLID_FILL,CYAN);
  setcolor(WHITE);
  bar (348,330,488,340);
  sprintf(Buffer,"%g",Avg_error);
  outtextxy(352,333,Buffer);
  Out_plot();
  Err_plot();
  if ( (f = fopen (Testfile, "w")) == NULL)
  {
    outtextxy(300,200,"Can't open file");
  }
  else
  {
    fprintf(f,"Desire and actual outputs\n");
    for ( i = 0; i < Dat_pts; i++)
        fprintf(f,"%4i %12.8f %12.8f\n", i, YD[i], YO[i]);
    fclose(f);
  }
  mouse_call(1,0,0,0);
}

/*  Connection weights update   */
void Weight_update (int t)
{
  int i, j, k, m, n;
```

```
  for ( i = 0; i < In_nodes; i++)
  {
    for ( j = 1; j < Hd_nodes[1]; j++)
    {
          W[0][i][j] += Lcoef[0] * Delta[0][j] * U[t][i]
                        + Mcoef[0] * W_old[0][i][j];
          W_old[0][i][j] = Delta[0][j] * U[t][i];
    }
  }
  m = Hd_layers;
  for ( n = 1; n < m; n++)
  {
    for ( i = 0; i < Hd_nodes[n]; i++ )
    {
          for ( j = 1; j < Hd_nodes[n+1]; j++ )
          {
            W[n][i][j] += Lcoef[n] * Delta[n][j] * ( X[n-1][i] + Fcoef *
                          Delta[n-1][i]) + Mcoef[n] * W_old[n][i][j];
            W_old[n][i][j] = Delta[n][j] * X[n-1][i];
          }
    }
  }
}

/*  Calculate the weight change direction for output and hidden layers.   */
void DeltaUXY ( int t )
{
  int i, j, n, m;
  double S, T;
  for ( j = 1; j < Out_nodes; j++ )
  {
    S = X[Hd_layers-1][j];
    Delta[Hd_layers-1][j] = D_squash(S)*(Y[t][j] - S);
  }
  m = Hd_layers;
  for ( n = 1; n < m; n++)
  {
    for ( j = 1; j < Hd_nodes[m-n]; j++ )
    {
          S = 0.0;
          for ( i = 1; i < Hd_nodes[m-n+1]; i++ )
          {
            S += (Delta[m-n][i] * W[m-n][j][i]);
          }
          T = X[m-n-1][j];
          Delta[m-1-n][j] = D_squash (T) * S;
    }
  }
}

/*  Propagation from input layer to output layer   */
void In_to_out ( int t )
{
  int i, j, n;
```

```
  double S;
  for ( j = 1; j < Hd_nodes[1]; j++ )
  {
    S = 0.0;
    for ( i = 0; i < In_nodes; i++ )
    {
          S += (W[0][i][j] * U[t][i]) ;
    }
    X[0][j] = Squash ( S );
  }
  for ( n = 1; n < Hd_layers; n++)
  {
    for ( j = 1; j < Hd_nodes[n+1]; j++ )
    {
          S = 0.0;
          for ( i = 0; i < Hd_nodes[n]; i++ )
          {
            S += (W[n][i][j] * X[n-1][i]) ;
          }
          X[n][j] = Squash ( S );
    }
  }
}

/*   Plot the local error   */
void Err_plot (void)
{
  int i, ya, yd;
  double E_max = -1.0;
  setviewport (311,353,638,478,1);
  clearviewport();
  setviewport (0,0,getmaxx(),getmaxy(),1);
  setcolor (YELLOW);
  for ( i = 0; i < Dat_pts; i++)
  {
    if ( E_local[i] > E_max )
          E_max = E_local[i];
  }
  Err_flag = YES;
  Line_plot (E_local,311,353,638,478,0,E_max);
  Err_flag = NO;
}

/*   Plot the estimated output and the desire output     */
void Out_plot (void)
{
  setviewport (311, 41, 638, 299, 1);
  clearviewport();
  setviewport (0,0,getmaxx(),getmaxy(),1);
  setcolor (LIGHTRED);
  Line_plot (YD,311,41,638,299, Y_min, Y_max);
  setcolor (LIGHTGREEN);
  Line_plot (YO,311,41,638,299, Y_min, Y_max);
}
```

```
/**** Line_plot ****/

void Line_plot (double *y, int l, int t, int r, int b, double Min, double Max)
{
  int i, xo, yo, xn, yn, ya, yd;
  double X_scale, Y_scale, Len, Wid;
  Len = r-l ;
  Wid = b-t ;
  X_scale = Len / (Dat_pts - 1) ;
  Y_scale = Wid / ( Max - Min );
  yo = (int) (Wid - Y_scale * ( y[0] - Min ));
  xo = 0;
  setviewport ( l, t, r, b, 1 );
  for ( i = 1; i < Dat_pts; i++ )
  {
    yn = (int) (Wid - Y_scale * ( y[i] - Min ));
    xn = (int) (i * X_scale);
    line ( xo, yo, xn, yn );
    xo = xn;
    yo = yn;
  }
  if (Err_flag == YES)
  {
    ya = (int) (Wid - Y_scale * Avg_error);
    setcolor(LIGHTMAGENTA);
    line ( 0,ya,Len,ya);
    yd = (int) (Wid - Y_scale * Desire_error);
    setcolor(LIGHTCYAN);
    line ( 0,yd,Len,yd);
  }
  setviewport ( 0, 0, getmaxx(), getmaxy(), 1 );
}

/*   Menu display and selection    */
void Get_menu ( void )
{
  int Location, Previous = 0, Status_cell;
  Menu = Save_screen(1,1,299,24);
  do
  {
    mouse_call (3,0,0,0);
    Status_cell = (globdx>369)*(globdx<430)*(globcx<299)*((globdx-349)/20);
    Location = ((globcx / 73) + 1) * (globdx < 26);
    if ( Location != Previous )
          Select ( Location );
    Previous = Location;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch ( Location )
          {
            case (1) : File_select(); break;
            case (2) : Select_select(); break;
```

```
            case (3) : Run_select(); break;
            case (4) : View_select(); break;
            default  : break;
          }
          Status(Status_cell);
    }
  } while (exit_status != 1);
  delete (Menu);
}

/*   Neural network status display and selection    */
void Status ( int Cell )
{
  setfillstyle(SOLID_FILL,DARKGRAY);
  setcolor(CYAN);
  mouse_call(2,0,0,0);
  switch ( Cell )
  {
    case (1) : F_type = (F_type+1)*(F_type<4); bar(220,370,299,390);
                  outtextxy(225,375,Signal_menu[F_type]); F_assign(); break;
    case (2) : Feedthru = (!Feedthru); bar(220,390,299,410);
                  if (Feedthru == 1) {outtextxy(225,395,"Yes");}
                  else outtextxy(225,395,"No");
                  break;
    case (3) : Repeat_rate = (Repeat_rate+1)*(Repeat_rate<20);
                  bar(220,410,299,430); sprintf(Buffer,"%i",Repeat_rate);
                  outtextxy(225,415,Buffer); break;
    default  : break;
  }
  mouse_call(1,0,0,0);
}
/*   File menu selection    */
void File_select (void)
{
  int Past = 0, Cell, Cel_L, Cel_R, Exit = NO;
  Cel_L = File_left - 15;
  Cel_R = File_left + 100;
  Temp = Save_screen ( Cel_L, 25, Cel_R, 150 );
  setcolor(WHITE);
  setfillstyle(SOLID_FILL,BLUE);
  Sub_menu ( Cel_L, 25, Cel_R, 5, File_menu );
  Shell = Save_screen (Cel_L, 25, Cel_R, 150);
  do
  {
    mouse_call (3,0,0,0);
    Cell = (globdx/25+1)*(globcx <= Cel_R)*(globcx >= Cel_L);
    if ( Cell == 1 && globcx >= (Cel_L-20) )
          Cell = 0;
    if ( Cell != Past )
          Cell_select ( Cel_L, 25, Cell, Cel_R, File_menu, Shell );
    Past = Cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
```

```
        switch (Cell)
        {
          case (1) : break;
          case (2) : Load_weight(); break;
          case (3) : Save_weight(); break;
          case (4) : Report(); break;
          case (5) : New_network(); break;
          case (6) : Quit(); break;
          default  : Exit = YES; break;
        }
    }
  } while ( (Exit || exit_status) != 1);
  Restore (Cel_L);
}

/*    Save a report of the network    */
void Report (void)
{
  int Exit = NO, k, i, j;
  char *Reportfile;
  Temp1 = Save_screen(120,74,468,101);
  mouse_call(2,0,0,0);
  do
  {
    sprintf(Buffer,"Enter filename : ");
    Reportfile = Value (0, 75, 120);
    if ( (O_file = fopen (Reportfile, "w")) == NULL)
    {
        outtextxy(300,200,"Can't open file");
        break;
    }
    else
    {
        fprintf (O_file,"Hidden layers = %i\n", Hd_layers-1);
        fprintf (O_file,"Input nodes   = %i\n", In_nodes-1);
        fprintf (O_file,"Output nodes  = %i\n", Out_nodes-1);
        for ( k = 1; k < Hd_layers; k++)
          fprintf(O_file,"Hidden %i nodes = %i\n", k, Hd_nodes[k]-1 );
        for ( k = 0; k < Hd_layers; k++)
          fprintf(O_file,"Layer %1i learning rate = %f\n", k, Lcoef[k]);
        for ( k = 0; k < Hd_layers; k++)
          fprintf(O_file,"Layer %1i momentum rate = %f\n", k, Mcoef[k]);
        fprintf (O_file,"Fast rate = %f\n", Fcoef);

        fprintf(O_file,"Desire error = %f\n", Desire_error);
        fprintf(O_file,"Actual error = %f\n", Avg_error);
        fprintf(O_file,"Iterations = %li\n", Iter);
        for ( k = 0; k < Hd_layers; k++)
        {
          fprintf(O_file,"\n");
          fprintf(O_file,"Weight matrix for layer %i\n", k+1);
          for ( i = 0; i < Hd_nodes[k]; i++)
          {
            for ( j = 1; j < Hd_nodes[k+1]; j++)
```

```
                fprintf (O_file,"%12.7f ",W[k][i][j]) ;
            fprintf (O_file, "\n");
          }
        }
        fprintf(O_file,"Desire and actual outputs\n");
        for ( k = 0; k < Dat_pts; k++)
          fprintf(O_file,"%i %12.8f %12.8f\n", k, YD[k], YO[k]);

        fclose (O_file);
        Exit = YES;
    }
  } while ( Exit != YES );
  Restore_screen(120,74,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
}


/*   Save connection weight to a file   */
void Save_weight (void)
{
  int Exit = NO, i, j, k;
  char *Weightfile;
  Temp1 = Save_screen(120,49,468,76);
  mouse_call(2,0,0,0);
  do
  {
    sprintf(Buffer,"Enter filename : ");
    Weightfile = Value (0, 50, 120);
    if ( (O_file = fopen (Weightfile, "w")) == NULL)
    {
        outtextxy(300,200,"Can't open file");
        break;
    }
    else
    {
        for ( k = 0; k < Hd_layers; k++)
        {
          fprintf(O_file,"\n");
          for ( i = 0; i < Hd_nodes[k]; i++)
          {
            for ( j = 1; j < Hd_nodes[k+1]; j++)
                fprintf (O_file,"%12.7f ",W[k][i][j]) ;
            fprintf (O_file, "\n");
          }
        }
        fclose (O_file);
        Exit = YES;
    }
  } while ( Exit != YES );
  Restore_screen(120,49,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
```

```
}

/*   Load connection weight from a file   */
void Load_weight (void)
{
  int Exit = NO, i, j, k;
  char *Weightfile;
  float P;
  Temp1 = Save_screen(120,24,468,51);
  mouse_call(2,0,0,0);
  do
  {
    sprintf(Buffer,"Enter filename : ");
    Weightfile = Value (0, 25, 120);
    if ( (O_file = fopen (Weightfile, "r")) == NULL)
    {
          outtextxy(300,200,"Can't open file");
          break;
    }
    else
    {
          for ( k = 0; k < Hd_layers; k++)
          {
            for ( i = 0; i < Hd_nodes[k]; i++)
            {
              for ( j = 1; j < Hd_nodes[k+1]; j++)
              {
                    fscanf (O_file,"%f ", &P) ;
                    W[k][i][j] = P;
              }
            }
          }
          fclose (O_file);
          Exit = YES;
    }
  } while ( Exit != YES );
  Restore_screen(120,24,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
}

/*   Select menu selection   */
void Select_select (void)
{
  int Past = 0, Cell, Cel_L, Cel_R, Exit = NO;
  Cel_L = Select_left - 15;
  Cel_R = Select_left + 100;
  Temp = Save_screen ( Cel_L, 25, Cel_R, 150 );
  setcolor(WHITE);
  setfillstyle(SOLID_FILL,BLUE);
  Sub_menu ( Cel_L, 25, Cel_R, 5, Select_menu );
  Shell = Save_screen (Cel_L, 25, Cel_R, 150);
  do
  {
```

```
    mouse_call (3,0,0,0);
    Cell = (globdx/25+1)*(globcx <= Cel_R)*(globcx >= Cel_L);
    if ( Cell == 1 && globcx >= (Cel_L-20) )
          Cell = 0;
    if ( Cell != Past )
          Cell_select ( Cel_L, 25, Cell, Cel_R, Select_menu, Shell );
    Past = Cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch (Cell)
          {
            case (1) : break;
            case (2) : Feed_thru(); break;
            case (3) : Batch_size(); break;
            case (4) : Data_file(); break;
            case (5) : Signal_type(); break;
            case (6) : Error_type(); break;
            default  : Exit = YES; break;
          }
    }
  } while (Exit != YES);
  Restore (Cel_L);
}

/*   Run menu selection   */
void Run_select (void)
{
  int Past = 0, Cell, Cel_L, Cel_R, Exit = NO;
  Cel_L = Run_left - 15;
  Cel_R = Run_left + 100;
  Temp = Save_screen ( Cel_L, 25, Cel_R, 150 );
  setcolor(WHITE);
  setfillstyle(SOLID_FILL,BLUE);
  Sub_menu ( Cel_L, 25, Cel_R, 5, Run_menu );
  Shell = Save_screen (Cel_L, 25, Cel_R, 150);
  do
  {
    mouse_call (3,0,0,0);
    Cell = (globdx/25+1)*(globcx <= Cel_R)*(globcx >= Cel_L);
    if ( Cell == 1 && globcx >= (Cel_L-20) )
          Cell = 0;
    if ( Cell != Past )
          Cell_select ( Cel_L, 25, Cell, Cel_R, Run_menu, Shell );
    Past = Cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch (Cell)
          {
            case (1) : break;
            case (2) : Test(); break;
            case (3) : Train(); break;
            case (4) : Restart(); break;
```

```
            case (5) : Continue(); break;
//        case (6) : Display(); break;
            default  : Exit = YES; break;
          }
    }
  } while (Exit != YES);
  Restore (Cel_L);
}

/*   To restart training the neural network   */
void Restart ( void )
{
  Delete_3D ( Hd_layers, Max_nodes, W );
  Delete_3D ( Hd_layers, Max_nodes, W_old );
  Continue_flag = YES;
  Weight_init();
  Iter = 0;
  Train();
}

/*   To continue training the neural network   */
void Continue ( void )
{
  Continue_flag = YES;
  Train();
}

/*   View menu selection   */
void View_select (void)
{
  int Past = 0, Cell, Cel_L, Cel_R, Exit = NO;
  Cel_L = View_left - 15;
  Cel_R = View_left + 100;
  Temp = Save_screen ( Cel_L, 25, Cel_R, 150 );
  setcolor(WHITE);
  setfillstyle(SOLID_FILL,BLUE);
  Sub_menu ( Cel_L, 25, Cel_R, 5, View_menu );
  Shell = Save_screen (Cel_L, 25, Cel_R, 150);
  do
  {
    mouse_call (3,0,0,0);
    Cell = (globdx/25+1)*(globcx <= Cel_R)*(globcx >= Cel_L);
    if ( Cell == 1 && globcx >= (Cel_L-20) )
          Cell = 0;
    if ( Cell != Past )
          Cell_select ( Cel_L, 25, Cell, Cel_R, View_menu, Shell );
    Past = Cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch (Cell)
          {
            case (1) : break;
//        case (2) : Weight(); break;
```

```
            case (3) : Momentum(); break;
            case (4) : Learn_rate(); break;
            case (5) : Fast_rate(); break;
//          case (6) : Input_data(); break;
            default  : Exit = YES; break;
          }
    }
  } while (Exit != YES);
  Restore (Cel_L);
}

/*  View and change fasting learning rate   */
void Fast_rate (void)
{
  Temp1 = Save_screen(120,220,468,247);
  sprintf (Buffer,"Fcoef  layers  = %6.4f",Fcoef);     // Can be set differently
  sprintf (Buf,"%6.4f",Fcoef);                         // for different layers
  Fcoef = atof (Value (6,221,120) );
  Restore_screen(120,220,Temp1);
  delete Temp1;
}

/*   View and change the learning rate   */
void Learn_rate (void)
{
  int i;
  Temp1 = Save_screen(120,220,468,247);
  if ( Hd_layers == 1 )
  {
    sprintf (Buffer,"Lcoef U -> Y   = %6.4f",Lcoef[0]);
    sprintf (Buf,"%6.4f",Lcoef[0]);
    Lcoef[0] = atof (Value (6,221,120) );
  }
  if ( Hd_layers > 1 )
  {
    sprintf (Buffer,"Lcoef U -> H1  = %6.4f",Lcoef[0]);
    sprintf (Buf, "%6.4f",Lcoef[0]);
    Lcoef[0] = atof (Value (6,221,120));
    for (i = 1; i < Hd_layers-1; i++)
    {
          sprintf (Buffer,"Lcoef H%1d -> H%1d = %6.4f", i, i+1,Lcoef[i]);
          sprintf (Buf, "%6.4f",Lcoef[i]);
          Lcoef[i] = atof (Value (6,221,120));
    }
  sprintf(Buffer,"Lcoef H%1d -> Y  = %6.4f", Hd_layers-1,Lcoef[Hd_layers-1]);
    sprintf (Buf, "%6.4f", Lcoef[Hd_layers-1]);
    Lcoef[Hd_layers-1] = atof (Value (6,221,120));
  }
  Restore_screen(120,220,Temp1);
  delete Temp1;
}

/*  View and change  the momentum rate   */
void Momentum (void)
```

```
{
  int i;
  Temp1 = Save_screen(120,220,468,247);
  if ( Hd_layers == 1 )
  {
    sprintf (Buffer,"Mcoef U -> Y   = %6.4f",Mcoef[0]);
    sprintf (Buf,"%6.4f",Mcoef[0]);
    Mcoef[0] = atof (Value (6,221,120) );
  }
  if ( Hd_layers > 1 )
  {
    sprintf (Buffer,"Mcoef U -> H1  = %6.4f",Mcoef[0]);
    sprintf (Buf, "%6.4f",Mcoef[0]);
    Mcoef[0] = atof (Value (6,221,120));
    for (i = 1; i < Hd_layers-1; i++)
    {
          sprintf (Buffer,"Mcoef H%1d -> H%1d = %6.4f", i, i+1,Mcoef[i]);
          sprintf (Buf, "%6.4f",Mcoef[i]);
          Mcoef[i] = atof (Value (6,221,120));
    }
  sprintf(Buffer,"Mcoef H%1d -> Y  = %6.4f", Hd_layers-1,Mcoef[Hd_layers-1]);
    sprintf (Buf, "%6.4f", Mcoef[Hd_layers-1]);
    Mcoef[Hd_layers-1] = atof (Value (6,221,120));
  }
  Restore_screen(120,220,Temp1);
  delete Temp1;
}

/*   To read a string of value from screen   */
char *Value (int Cnt, int Tp, int RR)
{
  char c, Sp;
  char Bf[2];
  int Top, Next, Chk = 32, i= 0, Count;
  Sp = (char) SPACE;
  Top = Tp + 1;
  setcolor (LIGHTRED);
  rectangle (RR,Tp-1,RR+348,Tp+26);
  rectangle (RR+1,Tp,RR+347,Tp+25);
  setfillstyle(SOLID_FILL, WHITE);
  setcolor(GREEN);
  bar (RR+2,Top,RR+346,Top+23);
  outtextxy (RR+10, Top+9, Buffer);
  do
  {
    Next = RR + 146 + Cnt * 8;
    outtextxy ( Next, Top+9, "_");
    Count = 0;
    do { Count++; } while ( !(kbhit() || Count > 1000 ));
    bar ( Next, Top, Next+8, Top+23 );
    Count = 0;
    do { Count++; } while ( !(kbhit() || Count > 1000 ));
    if (kbhit())
    {
```

```
            c = getch();
            Chk = (int) c;
            if ( (i == 0)*(Chk != RETURN) )
            {
              bar(RR+146, Top, RR+346, Top+23);
              Cnt = 0;
            }
            if ( Chk == BACKSPACE && Cnt > 0 )
            {
              Buf[--Cnt] = Sp;
              bar( Next, Top, Next+8, Top+23 );
              bar( Next-8, Top, Next, Top+23);
            }
            switch ( Chk )
            {
              case (ESC)      : break;
              case (RETURN)   : break;
              case (BACKSPACE) : break;
              default         : if (Cnt < 24) { Buf[Cnt] = c; Cnt++;
                                sprintf (Bf, "%c", c);
                                outtextxy (Next, Top+9, Bf);}  break;
            }
            i++;
      }
      if ( Chk == ESC )
            break;
    } while (Chk != RETURN);
    if ( i == 1 )
      return (Buf);
    else
      return (Buf);
}

/*  Exiting the program  */
void Quit (void)
{
  int c;
  sprintf (Buffer,"Are you sure ? (y/n)");
  Temp1 = Save_screen (120,124,321,151);
  C_box (120,124,321,151);
  c = getch();
  if ( c == 89 || c == 121 )
  {
    exit_status = YES;
  }
  Restore_screen(120,124,Temp1);
  delete(Temp1);
}

void C_box ( int CL, int CT, int R, int CB )
{
  setcolor (LIGHTRED);
  rectangle (CL,CT,R,CB);
  rectangle (CL+1,CT+1,R-1,CB-1);
```

```
  setfillstyle(SOLID_FILL, WHITE);
  setcolor(GREEN);
  bar (CL+2,CT+2,R-2,CB-2);
  outtextxy (CL+10,CT+10,Buffer);
}

/*  Restore the save image  */
void Restore ( int L )
{
  Restore_screen ( L, 25, Temp );
  Restore_screen ( 1, 1, Menu );
  delete (Temp);
  delete (Shell);
}

/*  Select and highlight at the mouse position of each cell in the sub-menu  */
void Cell_select(int CL, int CT, int Cel, int CR, char Sub[5][20], void *Sel)
{
  int T, L, ST, NT, CB;
  Restore_screen (CL,CT,Sel);
  T = CL + 15;
  L = CL + 1;
  setfillstyle(SOLID_FILL, MAGENTA);
  setcolor(YELLOW);
  mouse_call (2,0,0,0);
  ST = CT + 1 + (Cel -2)*25;
  NT = CT + 10 + (Cel -2)*25;
  CB = CT - 1 + (Cel -1)*25;
  switch (Cel-1)
  {
  case (1): bar(L,ST,CR-1,CB); outtextxy (T,NT,Sub [0]); break;
  case (2): bar(L,ST,CR-1,CB); outtextxy (T,NT,Sub [1]); break;
  case (3): bar(L,ST,CR-1,CB); outtextxy (T,NT,Sub [2]); break;
  case (4): bar(L,ST,CR-1,CB); outtextxy (T,NT,Sub [3]); break;
  case (5): bar(L,ST,CR-1,CB); outtextxy (T,NT,Sub [4]);break;
  default : break;
  }
  mouse_call (1,0,0,0);
}

/*  Select the sub-menu in menu bar  */
void Select ( int Position )
{
    Restore_screen (1,1,Menu);
    setcolor(YELLOW);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    setfillstyle(SOLID_FILL,DARKGRAY);
    mouse_call (2,0,0,0);
    switch ( Position )
    {
        case ( 1 ): bar (12,3,60,22); outtextxy (20,0,"File"); break;
        case ( 2 ): bar (72,3,136,22); outtextxy (80,0,"Select"); break;
        case ( 3 ): bar (152,3,200,22); outtextxy (160,0,"Run"); break;
        case ( 4 ): bar (212,3,266,22); outtextxy (220,0,"View"); break;
```

```
            default : break;
        }
        mouse_call (1,0,0,0);
        settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
}

/*   Sub-menu cell display   */
void Sub_menu (int Left, int Top, int Right, int N_cel, char Sub [5][20])
{
  int i, CT, Bottom;
  Bottom =  Top + 25 * N_cel;
  mouse_call (2,0,0,0);
  bar ( Left+1, Top+1, Right-1, Bottom-1 );
  rectangle (Left, Top, Right, Bottom);
  for ( i = 2; i <= N_cel; i = i + 2)
  {
    CT = Top + 25*(i-1);
    rectangle ( Left, CT, Right, CT+25 );
  }
  settextstyle ( DEFAULT_FONT, HORIZ_DIR, 1 );
  for ( i = 0; i < N_cel; i++ )
  {
    CT = Top + 10 + i*25;
    outtextxy ( Left+15, CT, Sub [i] );
  }
  mouse_call (1,0,0,0);
}

/*   To save the screen   */
void *Save_screen( int Left, int Top, int Right, int Bottom)
{
  void *Screen;
  unsigned imsize;
  mouse_call (2,0,0,0);
  imsize = imagesize(Left,Top,Right,Bottom);
  if ( !(Screen =  new[imsize]) )
  { outtextxy (200,330,"Not enough memory");
    getch();
    exit(1);
  }
  getimage(Left,Top,Right,Bottom,Screen);
  mouse_call (1,0,0,0);
  return( Screen );
}

/*   To restore  to screen   */
void Restore_screen( int Left, int Top, void *Screen)
{
  mouse_call (2,0,0,0);
  putimage (Left,Top,Screen,COPY_PUT);
  mouse_call (1,0,0,0);
}

/*  Display the initial menu setup   */
```

```
void Menu1_setup ( void )
{
  clearviewport();
  rectangle (0,0,300,25);
  rectangle (0,25,300,300);
  settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
  setfillstyle(SOLID_FILL,LIGHTGRAY);
  bar(1,1,299,24);
  setcolor(RED);
  outtextxy(20,0,"File");
  outtextxy(80,0,"Select");
  outtextxy(160,0,"Run");
  outtextxy(220,0,"View");
}

/*  Display the rest of the setup  */
void Menu2_setup (void)
{
  setcolor (WHITE);
  rectangle (310,0,639,40);
  rectangle (310,40,639,300);
  rectangle (0,310,300,335);
  rectangle (0,335,300,479);
  rectangle (310,310,639,350);
  rectangle (310,350,639,479);
  setfillstyle(SOLID_FILL,CYAN);
  bar(311,311,638,349);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  setcolor(YELLOW);
  switch (E_type)
  {
   case (0): outtextxy(335,317,"Absolute error      Desire error");break;
   case (1): outtextxy(335,317,"R.M.S. error        Desire error");break;
   case (2): outtextxy(335,317,"Norm max error      Desire error");break;
  }
  settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
  setfillstyle(SOLID_FILL,MAGENTA);
  bar(311,1,638,39);
  setcolor(YELLOW);
  outtextxy(330,0,"Output          Iterations");
  setfillstyle(SOLID_FILL,BLUE);
  bar(1,311,299,334);
  setcolor(WHITE);
  outtextxy(120,310,"Status");
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  setfillstyle(SOLID_FILL,DARKGRAY);
  bar(2,337,298,477);
  setcolor(GREEN);
  outtextxy(14,355,"Number of hidden layers : ");
  outtextxy(14,375,"Signal function type   : ");
  outtextxy(14,395,"Linear feedthru        : ");
  outtextxy(14,415,"Maximum repeated rate  : ");
  outtextxy(14,435,"Training algorithm     : ");
  outtextxy(14,455,"Available heap memories : ");
```

```
  setcolor(CYAN);
  sprintf(Buffer,"%i",Hd_layers -1);
  outtextxy(225,355,Buffer);
  outtextxy(225,375,Signal_menu[F_type]);
  if (Feedthru == 1) {outtextxy(225,395,"Yes");}
  else outtextxy(225,395,"No");
  sprintf(Buffer,"%i",Repeat_rate);
  outtextxy(225,415,Buffer);
  outtextxy(225,435,"Gradient");
  Memleft();
  outtextxy(225,455,Buffer);
}

/*   To initialize the graphics   */
void Graph_initialize (void)
{
  int gdriver = DETECT, gmode, errorcode;
  registerbgidriver (EGAVGA_driver);
  registerbgifont (triplex_font);
  initgraph (&gdriver, &gmode, "");
  errorcode = graphresult();
  if (errorcode != grOk )
  {
    printf("Graphics System Error : %s\n", grapherrormsg(errorcode));
    exit( 1 );
  }
}

/*    Display  the mouse pointer     */
void Mouse_display ( void )
{
  mouse_call(0,0,0,0);
  mouse_call(1,0,0,0);
}

/*  This function is used to communicate with the mouse  */
/*  via interrupt 0x33h.                                 */
void mouse_call(int regax, int regbx, int regcx, int regdx)
{
union REGS inregs, outregs;
inregs.x.ax = regax;
inregs.x.bx = regbx;
inregs.x.cx = regcx;
inregs.x.dx = regdx;
int86(0x33,&inregs,&outregs);
globax = outregs.x.ax;
globbx = outregs.x.bx;
globcx = outregs.x.cx;
globdx = outregs.x.dx;
}

/*   Error type selection   */
void Error_type (void)
{
```

```
  void *Tmp;
  int i, T, cell, Past = 0, Exit = NO;
  Temp1 = Save_screen(195,124,336,201);
  setfillstyle(SOLID_FILL, WHITE);
  bar(197,126,334,199);
  setcolor (LIGHTRED);
  rectangle (195,124,336,201);
  rectangle (196,125,335,200);
  rectangle (195,150,335,175);
  setcolor(GREEN);
  for ( i = 0; i < 3; i++)
  {
    T = 135 + i * 25;
    outtextxy (211,T,Error_menu[i]);
  }
  Tmp = Save_screen(196,125,335,200);
  do
  {
    mouse_call (3,0,0,0);
    cell =((globdx-126)/25+2)*(globcx < 336)*(globcx > 195)*(globdx < 200)
                                                    *(globdx > 125);
    if ( cell != Past )
          Cell_select ( 196, 125, cell, 335, Error_menu, Tmp );
    Past = cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch (cell-1)
          {
            case (1) : E_type = 0; E_select = Abs_error; break;
            case (2) : E_type = 1; E_select = Sq_error; break;
            case (3) : E_type = 2; E_select = Norm_max_error; break;
            default  : E_type = 0; E_select = Abs_error; break;
          }
          Exit = YES;
    }
  } while (Exit != YES);
  Restore_screen(195,124,Temp1);
  delete(Temp1);
  delete(Tmp);
}

/*   Read Batch_size value   */
void Batch_size (void)
{
  Temp1 = Save_screen(180,49,528,76);
  mouse_call(2,0,0,0);
  sprintf (Buffer,"Batch size     = 0");
  sprintf (Buf,"0");
  Batchsize = atoi( Value (1, 50, 180));
  Restore_screen(180,49,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
}
```

```
/*  To select feedthru   */
void Feed_thru (void)
{
  int c;
  mouse_call(2,0,0,0);
  sprintf (Buffer,"Feedthru .. ?  (y/n)");
  Temp1 = Save_screen (180,24,381,51);
  C_box (180,24,381,51);
  c = getch();
  if ( c == 89 || c == 121 )
  {
    Feedthru = YES;
  }
  else Feedthru = NO;
  Restore_screen(180,24,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
}

/*  Signal tranfer type selection   */
void Signal_type (void)
{
  void *Tmp;
  int i, T, Cell, Past = 0, Exit = NO;
  Temp1 = Save_screen(195,99,336,226);
  setfillstyle(SOLID_FILL, WHITE);
  bar(197,101,334,224);
  setcolor (LIGHTRED);
  rectangle (195,99,336,226);
  rectangle (196,100,335,225);
  rectangle (195,125,335,150);
  rectangle (195,175,335,200);
  setcolor(GREEN);
  for ( i = 0; i < 5; i++)
  {
    T = 110 + i * 25;
    outtextxy (211,T,Signal_menu[i]);
  }
  Tmp = Save_screen(196,100,335,225);
  do
  {
    mouse_call (3,0,0,0);
    Cell =((globdx-101)/25+2)*(globcx <= 336)*(globcx >= 195)*(globdx < 225)
                                                    *(globdx >= 100);
    if ( Cell != Past )
          Cell_select ( 196, 100, Cell, 335, Signal_menu, Tmp );
    Past = Cell;
    mouse_call (5,0,0,0);
    if ( globbx >= 1 )
    {
          switch (Cell-1)
          {
            case (1) : F_type = 0; break;
```

```
            case (2) : F_type = 1; break;
            case (3) : F_type = 2; break;
            case (4) : F_type = 3; break;
            case (5) : F_type = 4; break;
            default  : F_type = 3; break;
          }
          F_assign();
          Exit = YES;
      }
    } while (Exit != YES);
    Restore_screen(195,99,Temp1);
    delete(Temp1);
    delete(Tmp);
}

/*  Set pointer to the signal function selection    */
void F_assign (void)
{
    switch (F_type)
    {
      case (0) : Squash = Linear; D_squash = D_linear; break;
      case (1) : Squash = Sigmoid; D_squash = D_sigmoid; break;
      case (2) : Squash = Bipolar; D_squash = D_bipolar; break;
      case (3) : Squash = TanH; D_squash = D_tanh; break;
      case (4) : Squash = Sine; D_squash = D_sine; break;
    }
}

/*   Clear all parameters for new network selection and enter network parameters  */
void New_network ( void )
{
    int i;
    if ( New_flag == YES )
    {
      Feedthru = NO;
      Continue_flag = NO;
      Data_flag = NO;
      Delete_3D (Hd_layers, Max_nodes, W);
      Delete_3D (Hd_layers, Max_nodes, W_old);
      Delete_2D (Hd_layers, Delta);
      Delete_2D (Hd_layers, X);
      Delete_2D (Dat_pts, U);
      Delete_2D (Dat_pts, Y);
      delete E_local;
      delete YO;
      delete YD;
      delete Hd_nodes;
      delete Lcoef;
      delete Mcoef;
      Iter = 0;
    }
    Temp1 = Save_screen(120,99,468,126);
    sprintf (Buffer,"Hidden layers  = 1");
    sprintf (Buf,"1");
```

```
  Hd_layers = atoi( Value (1,100,120)) + 1;
  Hd_nodes = (int *) new int [Hd_layers+1];
  Lcoef = (double *) new double [Hd_layers];
  Mcoef = (double *) new double [Hd_layers];
//  Fcoef = (double *) new double [Hd_layers);
  for (i = 1; i < Hd_layers; i++)
  {
    sprintf (Buffer,"Hidden %1i nodes = 2", i);
    sprintf (Buf,"2");
    Hd_nodes[i] = atoi( Value (1,100,120) ) + 1;
  }
  if ( Hd_layers == 1 )
  {
    sprintf (Buffer,"Lcoef U -> Y  = 0.5");
    sprintf (Buf,"0.5");
    Lcoef[0] = atof (Value (3,100,120) );
    sprintf (Buffer,"Mcoef U -> Y  = 0.9");
    sprintf (Buf,"0.9");
    Mcoef[0] = atof (Value (3,100,120) );
  }
  if ( Hd_layers > 1 )
  {
    sprintf (Buffer,"Lcoef U -> H1  = 0.7");
    sprintf (Buf, "0.7");
    Lcoef[0] = atof (Value (3,100,120));
    sprintf (Buffer,"Mcoef U -> H1  = 0.9");
    sprintf (Buf, "0.9");
    Mcoef[0] = atof (Value (3,100,120));
    for (i = 1; i < Hd_layers-1; i++)
    {
          sprintf (Buffer,"Lcoef H%1d -> H%1d = 0.5", i, i+1);
          sprintf (Buf, "0.5");
          Lcoef[i] = atof (Value (3,100,120));
          sprintf (Buffer,"Mcoef H%1d -> H%1d = 0.9", i, i+1);
          sprintf (Buf, "0.9");
          Mcoef[i] = atof (Value (3,100,120));
    }
    sprintf (Buffer,"Lcoef H%1d -> Y  = 0.5", Hd_layers-1);
    sprintf (Buf, "0.5");
    Lcoef[Hd_layers-1] = atof (Value (3,100,120));
    sprintf (Buffer,"Mcoef H%1d -> Y  = 0.9", Hd_layers-1);
    sprintf (Buf, "0.5");
    Mcoef[Hd_layers-1] = atof (Value (3,100,120));
  }
  sprintf (Buffer,"Fcoef  layers  = 0.0");     // Can be set differently
  sprintf (Buf,"0.0");                         // for different layers
  Fcoef = atof (Value (3,100,120) );
  sprintf (Buffer,"Inputs        = 2");
  sprintf (Buf, "2");
  In_nodes = atoi ( Value (1,100,120)) + 1;     // + 1
  Hd_nodes[0] = In_nodes;
  sprintf (Buffer,"Outputs       = 1");
  sprintf (Buf, "1");
  Out_nodes = atoi (Value (1,100,120)) + 1;     // + 1
```

```
  Hd_nodes[Hd_layers] = Out_nodes;
  sprintf (Buffer,"Desire error   = 0.01");
  sprintf (Buf,"0.01");
  Desire_error = atof (Value (4,100,120));
  Weight_init();
  Restore_screen(120,99,Temp1);
  delete(Temp1);
  New_flag = YES;
}

/*   Initialize connection weights   */
void Weight_init (void)
{
  int i, j, k, Max;
  Max = -1;
  for ( i = 0; i < Hd_layers+1; i++)
  {
    if ( Hd_nodes[i] > Max )
          Max = Hd_nodes[i];
  }
  Max_nodes = Max;
  W = Mem_3D(Hd_layers, Max_nodes, Max_nodes);
  W_old = Mem_3D(Hd_layers, Max_nodes, Max_nodes);

  for ( k = 0; k < Hd_layers; k++)
  {
    randomize();
    for ( i = 0; i < Hd_nodes[k]; i++)
    {
          for ( j = 1; j < Hd_nodes[k+1]; j++)
          {
            W[k][i][j] = (double) rand()/163835.0 - 0.1;
            W_old[k][i][j] = 0.0;
          }
    }
  }
}

/*   Determine and read the number of training points   */
void Data_file (void)
{
  int Exit = NO;
  Temp1 = Save_screen(180,74,528,101);
  mouse_call(2,0,0,0);
  do
  {
    sprintf(Buffer,"Enter filename : ");
    Datafile = Value (0, 75, 180);
    if ( (D_file = fopen (Datafile, "r")) == NULL)
    {
          outtextxy(300,200,"Can't open file");
          break;
    }
    else
```

```
    {
        Dat_pts = File_lenght();
        fclose (D_file);
        D_file = fopen ( Datafile, "r");
        Get_data();
        fclose (D_file);
        Exit = YES;
    }
  } while ( Exit != YES );
  Restore_screen(180,74,Temp1);
  delete(Temp1);
  mouse_call(1,0,0,0);
}

/*   Get the input training  data   */
void Get_data (void)
{
  int i, j;
  double Max, Min;
  float Z;
  Max = -1.0e7;
  Min = 1.0e7;
  if ( Data_flag == YES )
  {
    Delete_2D ( Dat_pts, U );
    Delete_2D ( Dat_pts, Y );
  }
  U = Mem_2D ( Dat_pts, In_nodes );
  Y = Mem_2D ( Dat_pts, Out_nodes );

  for ( i = 0; i < Dat_pts; i++)
  {
    U[i][0] = 1.0;
    for ( j = 1; j < In_nodes; j++)
    {
        fscanf ( D_file, "%f", &Z );
        U[i][j] = Z;
    }
    for ( j = 1; j < Out_nodes; j++)
    {
        fscanf ( D_file, "%f", &Z );
        Y[i][j] = Z;
    }
  }
  for ( i = 0; i < Dat_pts; i++)
  {
    if ( Y[i][Out_disp] > Max ) Max = Y[i][Out_disp];
    if ( Y[i][Out_disp] < Min ) Min = Y[i][Out_disp];
  }
  Y_max = Max + 0.1 * fabs(Max);
  Y_min = Min - 0.1 * fabs(Min);
  Data_flag = YES;
}
```

```
/*  Determine the file lenght  */
int File_lenght (void)
{
  char String[100];
  int Cnt = -1;
  do
  {
    Cnt++;
  } while ( fgets ( String, 3000, D_file) != NULL );
  return ( Cnt );
}

/*  Delete the two dimensional dynamic array  */
void Delete_2D (int Row, double **S)
{
  int i;
  for ( i = 0; i < Row; i++ )
    delete (*(S+i));
  delete S;
}

/*  Delete the three dimensional dynamic array  */
void Delete_3D (int Row, int Wid, double ***S)
{
  int i,j;
  for ( i = 0; i< Row; i++)
  {
    for ( j = 0; j < Wid; j++ )
          delete (*(*(S+i)+j));
    delete (*(S+i));
  }
  delete S;
}

/*  Display memory available  */
void Memleft (void)
{
//  int Memtest;
  Mem = (unsigned long) coreleft();
  sprintf(Buffer,"%li", Mem);
//  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
//  Memtest = atoi (Value(10,400,100));
}

/*  Two dimensional dynamic memory setup  */
double **Mem_2D ( int Row, int Wid)
{
  double **Space;
  int i;
  Space = (double **) new double [Row];
  for ( i = 0; i < Row; i++)
    Space[i] = (double *) new double [Wid];
  return (Space);
}
```

```
/*   Three  dimensional dynamic memory setup   */
double ***Mem_3D( int Row, int Wid, int Height)
{
  double ***Space;
  int i, j;
  Space = (double ***) new double [Row];
  for ( i = 0; i < Row; i++)
  {
    Space[i] = Mem_2D( Wid, Height );
  }
  return (Space);
}

/*  Error type calculations    */
double Abs_error (void)
{
  int i;
  double error = 0.0;
  for ( i = 0; i < Dat_pts; i++ )
  {
    error += E_local[i];
  }
  return ( error / Dat_pts );
}

double Sq_error (void)
{
  int i;
  double error = 0;
  for ( i = 0; i < Dat_pts; i++ )
  {
    error += E_local[i] * E_local[i];
  }
  return ( sqrt(error/Dat_pts) );
}

double Norm_max_error (void)
{
  int i;
  double Max_error = -10000.0;
  for ( i = 0; i < Dat_pts; i++ )
  {
    if ( E_local[i] > Max_error )
          Max_error = E_local[i];
  }
  return ( Max_error);
}

/*  Derivative of signal function type calculation    */
double D_sigmoid ( double y )
{
  return ( y * (1.0 - y) );
}
```

```
double D_linear ( double y )
{
  return ( y / y );
}

double D_bipolar ( double y )
{
  return ( 0.5 * (1.0 + y) * (1.0 - y) );
}

double D_tanh ( double y )
{
  return ( (1.0 + y) * (1.0 - y) );
}

double D_sine ( double y )
{
  return ( sqrt (1.0 - y*y) );
}

/*   Signal transfer function calculations    */
double Sigmoid ( double x )
{
  return ( 1.0 / (1.0 + exp (-x)) );
}

double Linear ( double x )
{
  if ( x >= -1.0 || x <= 1.0 )
    return (x);
  else if ( x < -1.0 )
    return (-1.0);
  else return (1.0);
}

double Bipolar ( double x)
{
  return ( 2.0 / ( 1.0 + exp (-x)) - 1.0 );
}

double TanH ( double x )
{
  return ( tanh(x) );
}

double Sine ( double x )
{
  if ( x >= -HALF_PI || x <= HALF_PI )
    return ( sin(x) );
  else if ( x < -HALF_PI )
    return (-1.0);
  else return (1.0);
}
```

Vita

Born in 1964 in Saigon, Vietnam, Michael Tran came to the United States in 1979. He graduated from Wakefield High School in 1983 with distinction. Upon graduation, he attended Virginia Tech in September 1983 and finished his Bachelor of Science degree in Electrical Engineering with honors in August 1986. In September 1986, he joined Texas Instruments as an electrical design engineer and later earned the Texas Instruments Quality Excellence Award in 1987. In September 1988, he enrolled in the Electrical Engineering graduate program at Virginia Tech. He expects to complete the requirements for his Master of Science degree in Electrical Engineering in March 1992. He also expects to complete his Master of Business Administration and will continue on for his Ph.D. in Electrical Engineering. Mr. Tran is the recipient of a Pratt Presidential Fellowship, and the Renssenlaer Medal Award. He is a member of Tau Beta Pi and Eta Kappa Nu. He enjoys traveling, collecting postcards, volleyball, and roller-skating.

Michael Tran