Flexible Body Control Using Neural Networks

Final Report

March 5, 1992

Prepared by

Dr. Claire L. McCullough

University of Alabama in Huntsville

MSFC Contract NAS8-36955 D.O.127

While conventional computers must be programmed in a logical fashion by a person who thoroughly understands the task to be performed, the motivation behind neural networks is to develop machines which can train themselves to perform tasks, using available information about desired system behavior and learning from experience.

Goals of the project conducted under the this contract were threefold:

1) to evaluate various neural net methods and generate computer software to implement those deemed most promising on an IBM-compatible personal computer equipped with MATLAB;
2) to evaluate methods described in the current professional literature for system control using neural nets and to choose those most applicable to control of flexible structures;
3) to apply the control strategies chosen in 2) to a computer simulation of a test article, the Control Structures Interaction (CSI) Suitcase Demonstrator, which is a portable system consisting of a small flexible beam driven by a torque motor and mounted on springs tuned to the first flexible mode of the beam.

The first two goals have been accomplished, and work on the third is on-going. Results of each will be discussed below.

A standard neural net is composed of neurons such as that shown in Figure 1. The neuron forms a weighted sum of its inputs, which then has applied to it an activation function F to produce the system output. That is, if x_i are the neuron inputs and w_i are the corresponding weights

$$NET = \Sigma_i x_i w_i \qquad (1)$$

and

$$OUT = F(NET). \tag{2}$$

The activation function can be a simple threshold, a linear function such as

$$OUT = k\ NET, \tag{3}$$

or a nonlinear function, such as the logistic, or sigmoid, function

$$OUT = 1 / (1 + e^{-NET}) = F(NET). \tag{4}$$

These neurons are arranged in layers, as shown in Figure 2. A layer is defined as a set of weights followed by associated computation. Since the input layer serves only to distribute weights and performs no computation, it is not counted as a layer; thus the neural net shown in Figure 2 is a two-layer neural net. It is not necessary that activation functions for all layers be the same, although it is usual for all neurons in a single layer to have the same activation function. It is necessary that neural nets have at least one nonlinear layer, as it can be shown that a neural net with only linear activation functions produces no computational benefit over a single linear layer system. This type of neural net can perform only problems in which patterns are linearly separable; a simple two-input exclusive-or function violates this constraint, and as the number of system inputs increases, the chance of a given function of n variables being linearly separable (by an n-dimentional hyperplane) becomes vanishingly small. The goal of neural net training algorithms is to systematically change the weights of the network until an optimum value is reached. Optimality depends of the particular problem being addressed and how the "goodness" function for the system is chosen.

Using many references, the currently available methods for training neural nets were examined and evaluated for ease of implementation, reliability, computer requirements, and applicability to control systems. Some methods

were rejected because of the vast numbers of neurons required to work practical problems (e.g., Bidirectional Associative Memories); some, for example Boltzmann machines, because of the very large amount of computer time required to train the nets; and some, like Hopfield nets, for the extreme difficulty of implementation (in order to utilize a Hopfield net, a Lyapunov function must be generated for system "goodness" and appropriate weight adjustments based on that Lyapunov function must be determined--a procedure requiring vast "mathematical expertise and ingenuity" [1]). While there is currently no optimum method for neural nets, after careful evaluation, back-propagation was chosen as the most practical choice for implementation. Difficulties with back-propagation include possible network paralysis if neurons saturate, the possibility of reaching a local rather than a global error minimum, and long training times. However, the method is very easy to implement algorithmically, and is used in the majority of the controls applications appearing in the current literature. Methods have been proposed to fix difficulties with back-propagation, but each has its own associated problems (for example, Cauchy training eliminates the problem of convergence to local minima, but has a greater instance of network paralysis than systems using back-propagation, and a training time one hundred times that of the already lengthy back-propagation training). Thus back-propagation was chosen as the neural net training method to be implemented.

The steps used in training a neural net using back-propagation are as follows:

1) Select one training pair, consisting of an input vector and a target output vector, from the pattern set. Apply the input portion to the input of the neural net.
2) Calculate the network output.

3) Calculate the error between the actual system output and the desired value.

4) Adjust the weights so that error is decreased.

5) Repeat steps 1-4 for all available patterns until error for the entire training set is acceptably low.

Steps 1 and 2 constitute a forward pass through the network and describe the operation of the trained neural net as well. Steps 3 and 4 are a reverse pass, as error is propagated backward through the neural net beginning at the output and working back to the input layer. [1] Weights are adjusted in the following way. The weight change for the weight to neuron q in the kth layer from neuron p in the previous (jth) layer is given by

$$\Delta w_{pq,k} = \mu \, \partial_{q,k} \, OUT_{p,j} \qquad (5)$$

where μ is a training rate between 0 and 1, and $OUT_{p,j}$ is the output value from neuron p in layer j, which serves as an input to neuron q in layer k. The definition of ∂ varies depending on whether the layer being considered is an output layer or an interior (hidden) layer, and will be given presently. The new value for the weight (at time step n+1) is then given by

$$w_{pq,k}(n+1) = w_{pq,k}(n) + \Delta w_{pq,k} \qquad (6)$$

These two equations are the same regardless of whether or not the layer is hidden. The difference comes in the definition of ∂. If k is an output layer, $\partial_{q,k}$, the ∂ for neuron q in layer k, is given by

$$\partial_{q,k} = (Target_q - OUT_{q,k}) \, F'(NET_{q,k}) \qquad (7)$$

where $Target_q$ is the desired output for neuron q, $OUT_{q,k}$ is the actual output, and

$$F'(NET_{q,k}) = \frac{\partial F(NET)}{\partial NET} \qquad (8)$$

evaluated at

$$NET = NET_{q,k} \qquad (9)$$

where $NET_{q,k}$ is the NET value for neuron q. If j is a hidden layer, the ∂ for neuron p in that layer is given by

$$\partial_{p,j} = (\Sigma_q \partial_{q,k} w_{pq,k}) F'(NET_{p,j}) \qquad (10)$$

where k is the layer subsequent to j. Thus all ∂'s and weight changes are calculated first for the output layer, then for the hidden layer connecting to the output layer, and so forth until the reverse pass has been completed. It can be shown that these weight changes are proportional to the partial derivative of the system error function with respect to each weight. This approximates a gradient descent on the error surface, and therefore assures that the system, if not saturated, will eventually settle on weights that correspond to an error minimum. [5]

Using MATLAB, software was generated implementing a back-propagation trained neural net on an IBM compatible personal computer. For a given problem, number of layers and number of neurons must be "empirically determined," [2] so neural nets of several sizes and configurations were compared. Some authors have hypothesized that fewer neurons may be used for a given problem if those neurons are arranged in more layers [1]. In the initial trials conducted (using randomly chosen test data), no network was found which failed to converge eventually, so no evidence was obtained to support or disprove this hypothesis. However, empirical evidence does suggest that given that both will eventually converge to a solution, a neural net with fewer layers will converge more quickly; i.e., with fewer passes

through the training set (epochs). The error measure used in all simulations was Total Sum Square Error, given by the equation

$$\Sigma_p \Sigma_i (\text{Target}_{i,p} - \text{Actual}_{i,p})^2 \qquad (11)$$

where $\text{Target}_{i,p}$ is the desired output of neuron i in the output layer for training pattern p, and $\text{Actual}_{i,p}$ is the actual output. This is summed over all the output neurons (i) and all training pattern in a single epoch (p). Figure 3, showing the error measure versus number of training epochs for a two-layer neural net (with one nonlinear hidden layer and a linear output layer) and a three-layer net (with two nonlinear hidden layers and a linear output layer), showing faster convergence for the two-layer network, is typical of the results generated.

In the second phase of the project, recent publications in the professional literature regarding applications of neural nets to control problems were examined and compared. Methods currently available can be divided into roughly three categories:

a) methods in which the neural net generates a controller for an unknown system without human intervention [2].

b) methods in which a neural net is trained to emulate a currently existing controller, whether human or computerized (such as [3]);

c) methods in which nets generate some state or function which is then used in a standard controller design (for example, [4] in which the neural net is used to generate estimates of unknown nonlinear system parameters, which are then used in a standard adaptive controller);

Methods of each of the three types were tested under the current contract. Of the three types, the first is by far the most sophisticated, as it assumes

no mathematical knowledge of the system to be controlled, and does not require a human to be able to control the system or to generate a controller which successfully does so. This would mean that nonlinear systems which could be modeled poorly, if at all, theoretically could still be successfully controlled by a trained neural net.

The first method to be discussed is that of [2], which theoretically generates a controller without human guidance. First, a neural net must be trained to emulate the system to be controlled, which may be poorly modeled and may contain nonlinearities. After the emulator has been trained and weights fixed, it is used to back-propagate error to the emulator input, to give an estimate of controller error. These estimates are then be used to train the controller. This method as reported in [2] requires that all system states be directly measurable at the output. As this is not the case with most realistic systems, this method may fail to converge to an acceptible solution for many practical problems.

The trained emulator is used to train the controller as follows:

1) A time trajectory for system behavior is generated, with the untrained controller generating essentially random inputs to the emulator.
2) The final emulator output is compared to the desired output.
3) The error is propagated back through the emulator to generate an equivalent controller error, which is used to train the controller. That is, the emulator generates at its input a ∂u which is used in training the controller rather than $(u_{desired} - u_{actual})$, which is not available. This proportionate input error generation is the reason that the emulator is necessary to the process.
4) The process is continued, propagating back through each time step of the trajectory until the controller has been trained for all time steps

(i.e., a back-propagation through time).

5) Steps 1-4 are repeated for many trajectories.

The neural net chosen for use had one hidden nonlinear layer containing 175 neurons and a linear output layer of 10 neurons (to scale the outputs). The activation function used for the nonlinear layer is the hyperbolic tangent function , chosen because its odd symmetry about zero allows both excitatory and inhibitory outputs from a single neuron. One problem in implementing the method was difficulty in obtaining accurate training data for the CSI Demonstrator; this was done using a MATLAB simulation of the system developed by John Sharkee of Marshall Space Flight Center.

Another difficulty encountered was ill-conditioning of the data. Although it was mentioned nowhere in the literature, it was discovered that if inputs to the neural net vary by several orders of magnitude, as is the case of the Demonstrator, the nonlinear neuron layer soon saturates, so that training of that layer comes to a virtual standstill. This causes the nonlinear layer to send the same input to the linear layer regardless of the system input, causing the linear weights to grow without bound as they try to adjust to give varying outputs a constant input. This causes the error measure to grow without bound. This problem was solved by scaling the trajectories of very large system states to bring them down to the order of magnitude of the others and prevent layer saturation.

After emulator training was completed, most of the simulation on this method was done by Chris Tharpe, the research assistant funded by the project. When it was considered that the emulator was adequately, although imperfectly, trained, a multiple time stage controller was developed. Although several training methods were attempted, the error back-propagated though a few timesteps consistently rendered error values too small to affect

weights significantly; the net consequently failed to train to an adequate controller. To overcome this difficulty, a single time stage version of the method was utilized, in which the controller net was penalized, not if the control failed to drive the state to zero (which could not realistically be expected in a single time step of .01 seconds), but if the control generated by the neural net caused the error between actual and desired state values to increase rather than decrease. This net is currently training, but is, after many epochs of training, still highly unstable, as is shown in Figure 4.

The next method chosen was to train a neural net to emulate a currently existing controller for the CSI Demonstrator, in order to compare the properties of the standard, with a neural net generated, controller. The controller chosen for the neural net to emulate was the anticipatory fuzzy logic controller designed under the NASA Summer Faculty Fellowship program in 1991. This controller was chosen because of its superior performance under all tested operating conditions, including random perturbations of state matrices and addition of measurement noise. A brief description of the anticipatory fuzzy logic controller is given below; further details may be found in [6].

Fuzzy systems operate by testing variables with IF-THEN rules, which produce appropriate responses. Each rule is then weighted by a "Degree of Fulfillment" of the rule invoked; this is a number between 0 and 1, and may be thought of as a probability that a given number is considered to be included in a particular set. A wide variety of shapes is possible for fulfillment functions, triangles and trapezoids being the most popular.[7] Fulfillment functions for this study were of the form

$$fuzzy(x,m,s,p)=\exp(-(|x-m|/s)^p) \qquad (12)$$

where m,s,and p are user-chosen parameters and x is the value to be tested. This function was chosen because of its flexibility; by changing "m," "s," and "p" whole families of different functions can be obtained. The system operates by testing rules of the type

"If error is big and velocity is small, then u should be negative and big." The degree of fulfillment for such a rule is the minimum of the degrees of fulfillment of the antecedent clauses; i.e.,

$$DOF = \min [DOF_{\text{error big}}, DOF_{\text{velocity small}}] \qquad (13)$$

The total output of the control system is a weighted sum of the responses to all "n" rules

$$u = (_{i=1}\Sigma^{n}\, w_i\, (DOF_i)\, B_i^{d})/(_{i=1}\Sigma^{n}\, w_i\, (DOF_i)) \qquad (14)$$

where DOF_i is the degree of fulfilment of rule "i," B_i^{d} is the "defuzzified" output response to rule "i," and w_i is a weight indicating the relative importance of rule "i."[8]

The rules for the initial (standard fuzzy) system were of two types

Set A: If LOS error is positive-big, then u is negative-big.
(7 rules, one for each category of LOS error)

Set B: If LOS error is near-zero and velocity is positive-big, then u is negative-big.
(7 rules, one for each category of angular velocity)

The rules in set A approximate a proportional control scheme; set B approximates derivative control, but is only effective when LOS error is

small. This strategy is to drive the system to the desired output as quickly as possible, and only apply damping when the system response is close to the desired value. This system was demonstrated in [6] to perform adequately under a variety of conditions, including system mis-modeling and addition of measurement noise.

A new control strategy, called anticipatory fuzzy control, was developed under this program. This differs from traditional fuzzy control in that once fuzzy rules have been used to generate a control (as in equation (14)), a predictive routine built into the controller is called to anticipate the effect of the proposed control on the system output. If using the current control value will result in system behavior which is in some way unacceptable, additional rules are called. This method may be used to nest as many sets of rules as the designer desires. Advantages of this approach compared to standard fuzzy controllers are

1. Nesting rules allows use of only as many rules as are necessary to achieve desired system performance, resulting in savings in computer run time.
2. By predicting system performance, controls which would result in unstable or unacceptable system performance can be eliminated.

Standard predictive fuzzy control, which uses only predictive rules, requires more calls to the predictive routine than this scheme, and fails to take advantage of all system knowledge.[9] The simplest type of anticipatory system control has a single additional rule of the form

"If the current value of the control (u_c) will cause the difference between the current and anticipated values of velocity to be 'big,' then $u = u_c(1 - \beta \bullet bigt)$," (15)

where ß is a user-chosen parameter between 0 and 1, and "bigt" is the fulfillment function for the anticipated difference in velocity values (which is proportional to the predicted acceleration of the system.) A ß value of.7 was chosen for its smooth response and small settling time. Higher values of ß result in smoother responses with slightly more overshoot; lower values resemble the nonanticipatory response. In every case, the anticipatory fuzzy system results in smoother system response than the traditional fuzzy control. When plant parameters were perturbed (representing a mis-modeled system), the system exhibited a larger overshoot, but still settled to 0 within 3 seconds. In contrast, a standard linear quadratic regulator had considerably less overshoot, but failed to drive the system to 0 within 5 seconds. The anticipatory fuzzy system tolerated added state noise much better than the LQR, in which the noise caused a wide excursion from the desired LOS error value of 0.

A typical plot of the control generated by the fuzzy system and that generated by the neural net emulator is shown in Figure 5.a. The neural net used was expanded to a hidden layer of 175 neurons after a smaller net failed to train. The training level shown in the figure represents hundreds of epochs, each consisting of 500 time steps, and each beginning with randomly chosen initial conditions. Although the two responses are similar, their effect on the sytem is different, as may be seen in Figure 5.b. It can be seen that while the fuzzy system controls the CSI demonstrator adequately, for this particular initial condition the neural net exhibits even better behavior. However, for some initial conditions, the neural net controller exhibited very poor response; the difficulty here is the impossibility of including in the training set <u>every possible</u> situation the controller will encounter. In order to counter this difficulty, a fixed training set was not used for this neural

net; instead, for each epoch of 5 seconds, a random initial condition (within the constraints of possibility for the physical system) was chosen, a fuzzy control was generated for the system for each timestep, and this data was used to train the net. Thus, the net trained on thousands of situations, but never saw the same training data twice. This improved the response of the neural net over one trained on a fixed set, but it was still not possible to look at every possibility. More work needs to be done on this method to investigate robustness to noise and state model perturbations.

The method which seems to show the most promise is that in which a neural net emulator is imbedded in the fuzzy logic controller. The final method considered required training a neural net as an observer for the system, which was assumed to be imperfectly modeled. The observer output was used in the design of an existing type of controller. While this method is more assured of successful convergence than the first, it is a somewhat less powerful technique, as a form for the system model must be assumed and the form of the controller determined by a designer, rather than allowing the neural net to both identify the system and optimize the controller.

The neural net emulator in this case was used in the implementation of an anticipatory fuzzy logic controller. A flaw in the previous method of generating anticipatory fuzzy logic control was that a mathematical simulation of the system to be controlled was required to perform the prediction necessary to the controller; this eliminated one of the primary benefits of fuzzy logic control: that a complete mathematical description of the system to be controlled is not required. This flaw was removed when the mathematical simulation was replaced by a neural net which had been trained to predict the behaviour of the system. Even when the neural net emulator was imperfectly trained, the flexibility of the fuzzy logic method allowed the

system to not only produce a control which adequately drove the LOS error of the CSI Suitcase demonstrator to zero quickly, but also demonstrated good noise rejection and robustness properties. This is illustrated by Figure 6.a, which shows the response of the fuzzy-neural system, Figure 6.b, which shows the response of the fuzzy-neural system with noise added to the measurements, and Figure 6.c, which shows the response of the system when randomly chosen parameters in the state matrices were altered by + or - 50%. When a <u>well</u> trained neural net was used in the controller, response was essentially indistinguishable from that of the original anticipatory fuzzy system with perfect prediction. Additional training was performed to test the capability of the neural net to retrain on-line in recognition of perturbed state parameters.

An additional benefit of the fuzzy-neural hybrid is the ability of the neural net to retrain on-line if its predictions cease to match actual system behavior. Figure 6.c. showed the response of the fuzzy-neural system to large system perturbations. Figure 7.a. shows the best response of the fuzzy system with no weight changes, etc. (i.e., the fuzzy system is as before, but contains a perfect mathematical model of the perturbed system for prediction). Note that the two responses have essentially the same form, but the perfect system has a shorter settling time. Figure 7.b. shows the response of the neuro-fuzzy system after the neural net has been allowed to retrain on-line (i.e., to adapt to the perturbed state matrices). Note that this is essentially identical to the perfect predictor system response. Of course, after it becomes clear that the system is not as originally perceived, rule weights, etc., may be adjusted (empirically, or using computer directed search techniques) to achieve better performance; for example, Figure 8 shows the response of the adapted neuro-fuzzy system with p=.5 and spread factors for

the fuzzy functions doubled.

The only problems encountered in performance of the work described above have all been concerned with hardware and software difficulties, which have greatly reduced the amount of computer time which could be devoted to the lengthy training necessary for neural nets. It is recommended that if neural methods are to be applied in any practical situations, a dedicated computer be employed for the training of the nets, and that a version of MATLAB later than the 3.5.j version used in this work be obtained (the version purchased has a flaw which randomly causes the program to be terminated prematurely--this is a very serious deficiency in applications such as this (where long continuous run times are required), and the makers of MATLAB are working to correct it).

The conclusion of this study is that, while control by neural nets alone (i.e., allowing the net to design a controller with no human intervention) has yielded less than optimal results, the neural net trained to emulate the existing fuzzy logic controller does produce acceptible system responses for the initial conditions examined. In addition, a neural net was found to be very successful in performing the emulation step necessary for the anticipatory fuzzy controller for the CSI Suitcase Demonstrator. The fuzzy-neural hybrid, which exhibits good robustness and noise rejection properties, shows promise as a controller for practical flexible systems, and should be further evaluated.

REFERENCES

[1] Wasserman, P. D., Neural Computing: Theory and Practice, Van Nostrand Reinhold, New York, N. Y., 1989.

[2] Nguyen, D. H. and Widrow, B., "Neural Networks for Self-Learning Control Systems," IEEE Control Systems Magazine, April 1990, pp. 18-23.

[3] Guez, A. and Selinsky, J., "A Trainable Neuromorphic Controller," Journal of Robotic Systems, vol. 5, no. 4, 1988, pp. 363-388.

[4] Chen, F. C., "Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control," IEEE Controls Systems Magazine, April 1990, pp. 44-48.

[5] McClelland, J. L. and Rumelhart, D. E., Explorations in Parallel Distributed Processing, MIT Press, Cambridge, MA, 1988.

[6] McCullough, Claire L., "Control of a Flexible Beam Using Fuzzy Logic," NASA Summer Faculty Fellowship Final Report, Marshall Space Flight Center, AL, 1991.

[7] Self, K., "Designing with Fuzzy Logic," IEEE Spectrum, November 1990.

[8] Chiu, S., Chand, S., Moore, D., and Chaudhary, A., "Fuzzy Logic for Control of Roll and Moment for a Flexible Wing Aircraft," IEEE Control Systems, June 1991.

[9] Miyamoto, S., Yasunobu, S., and Ihara, H., "Predictive Fuzzy Control and Its Application to Automatic Train Operation Systems," Analysis of Fuzzy Information, J. C. Bezdek, ed., CRC Press Inc., Boca Raton, FL, 1986.

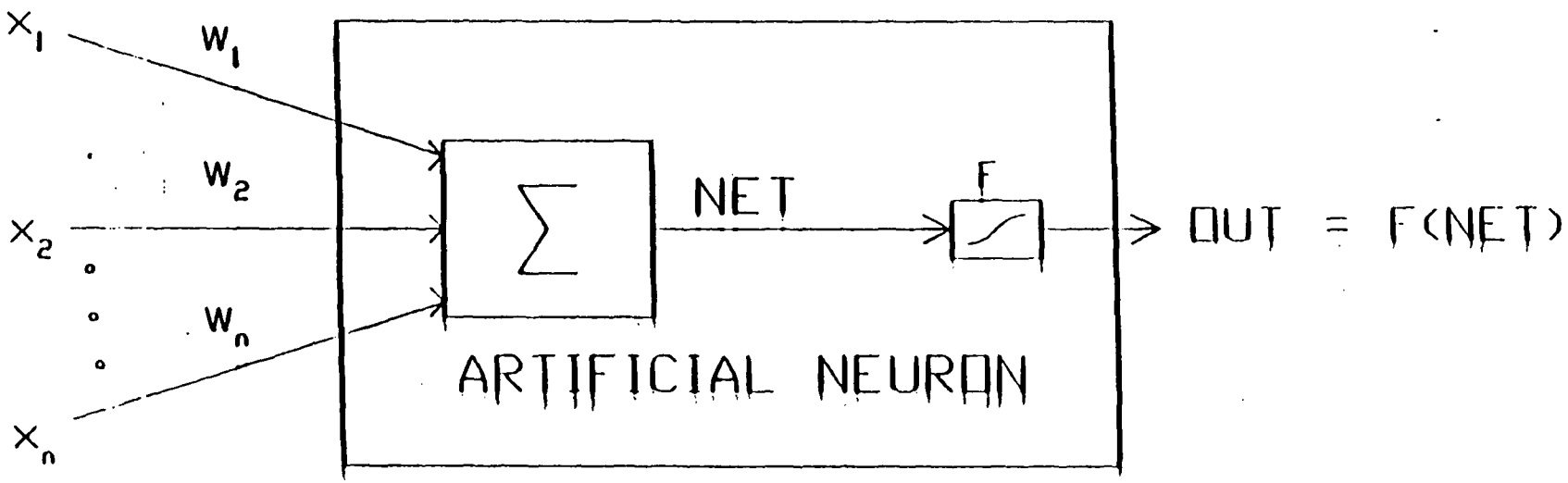
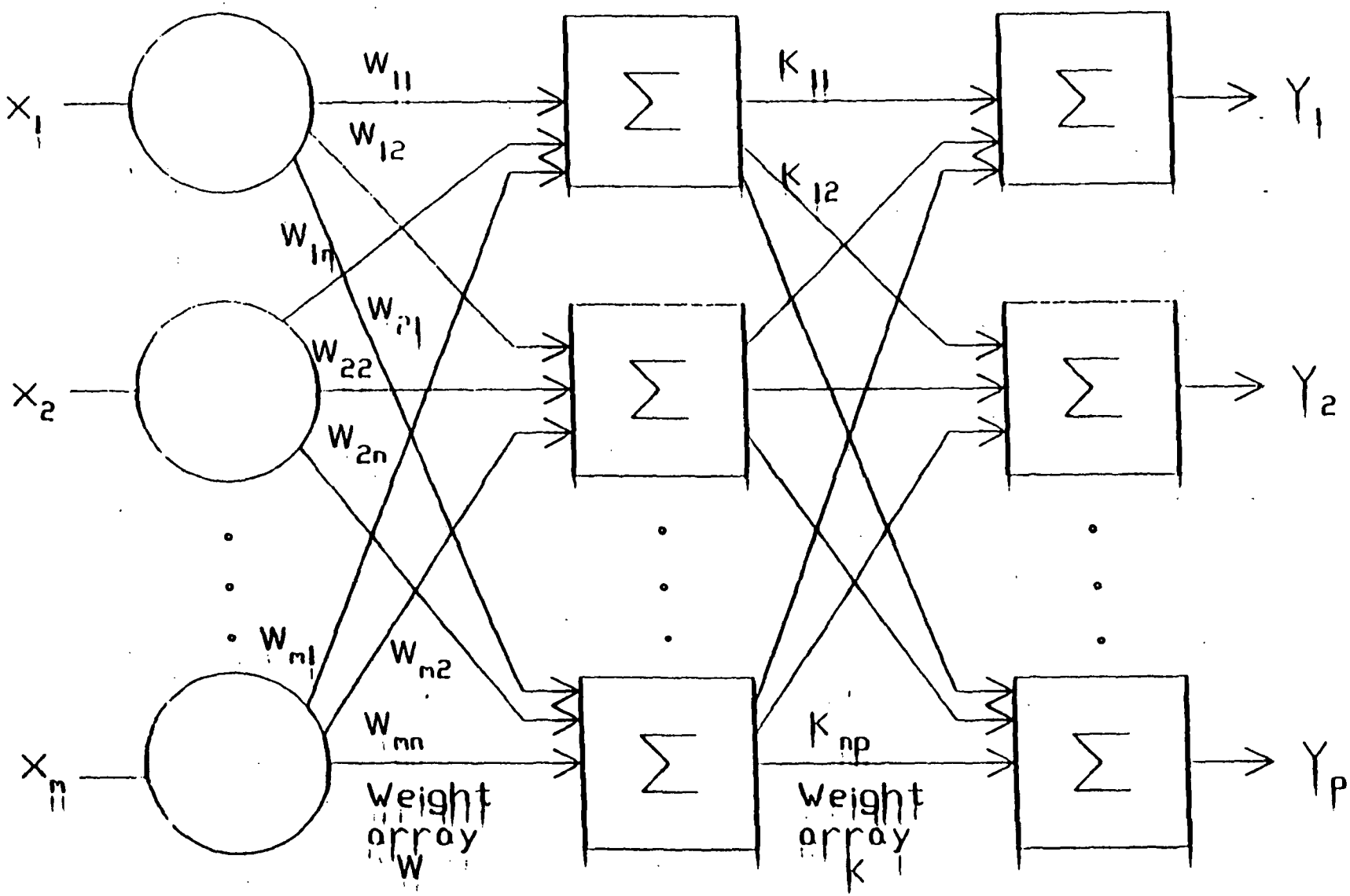


FIGURE 1. [1]


X_1
X_2
X_n
W_{11}
W_{12}
W_{1n}
W_{21}
W_{22}
W_{2n}
W_{m1}
W_{m2}
W_{mn}
Weight array W
K_{11}
K_{12}
K_{np}
Weight array K
Y_1
Y_2
Y_p

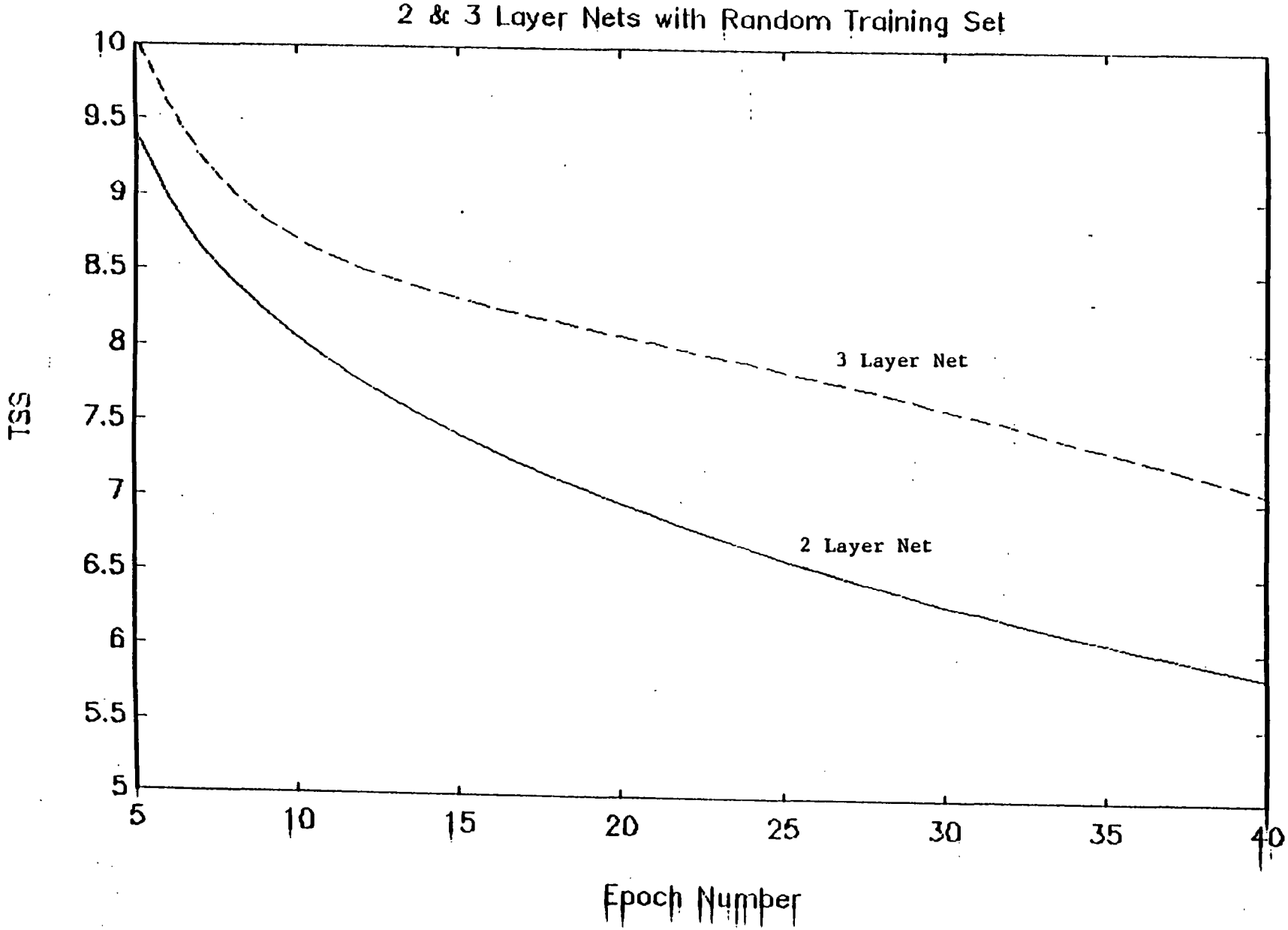
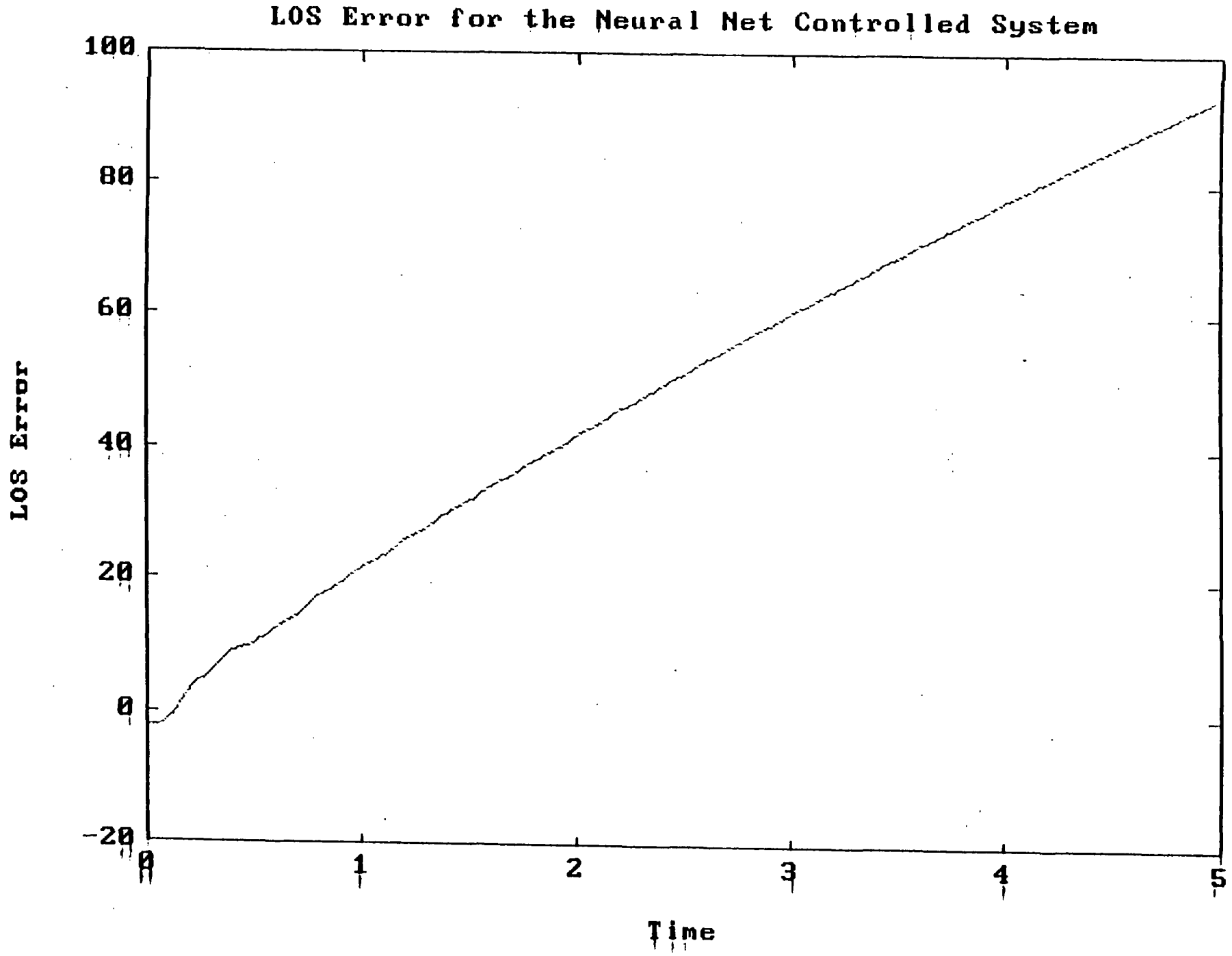

2 & 3 Layer Nets with Random Training Set
TSS
Epoch Number
3 Layer Net
2 Layer Net
10
9.5
9
8.5
8
7.5
7
6.5
6
5.5
5
5
10
15
20
25
30
35
40

FIGURE 3.


LOS Error for the Neural Net Controlled System
LOS Error
100
80
60
40
20
0
-20
0
1
2
3
4
5
Time

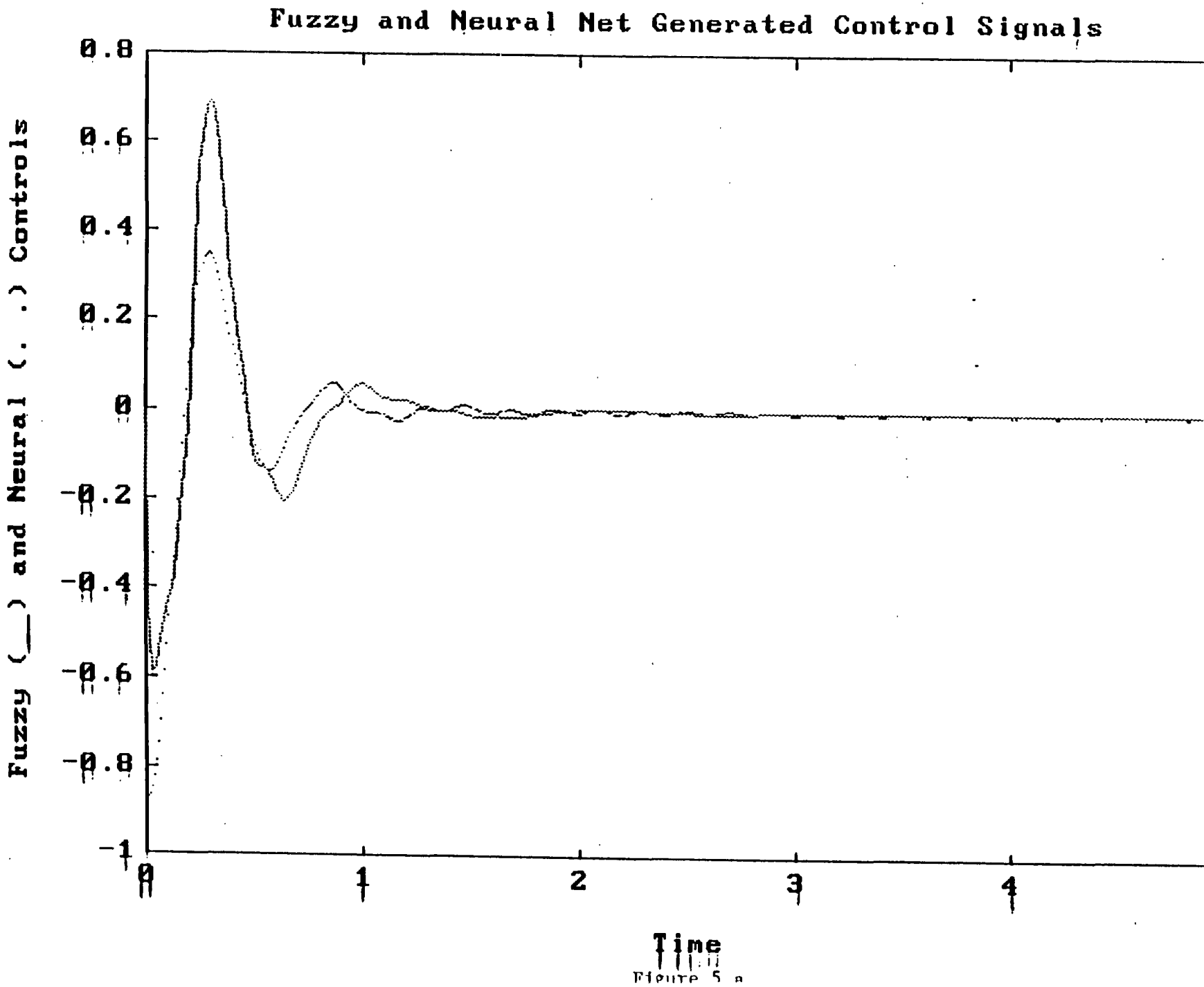
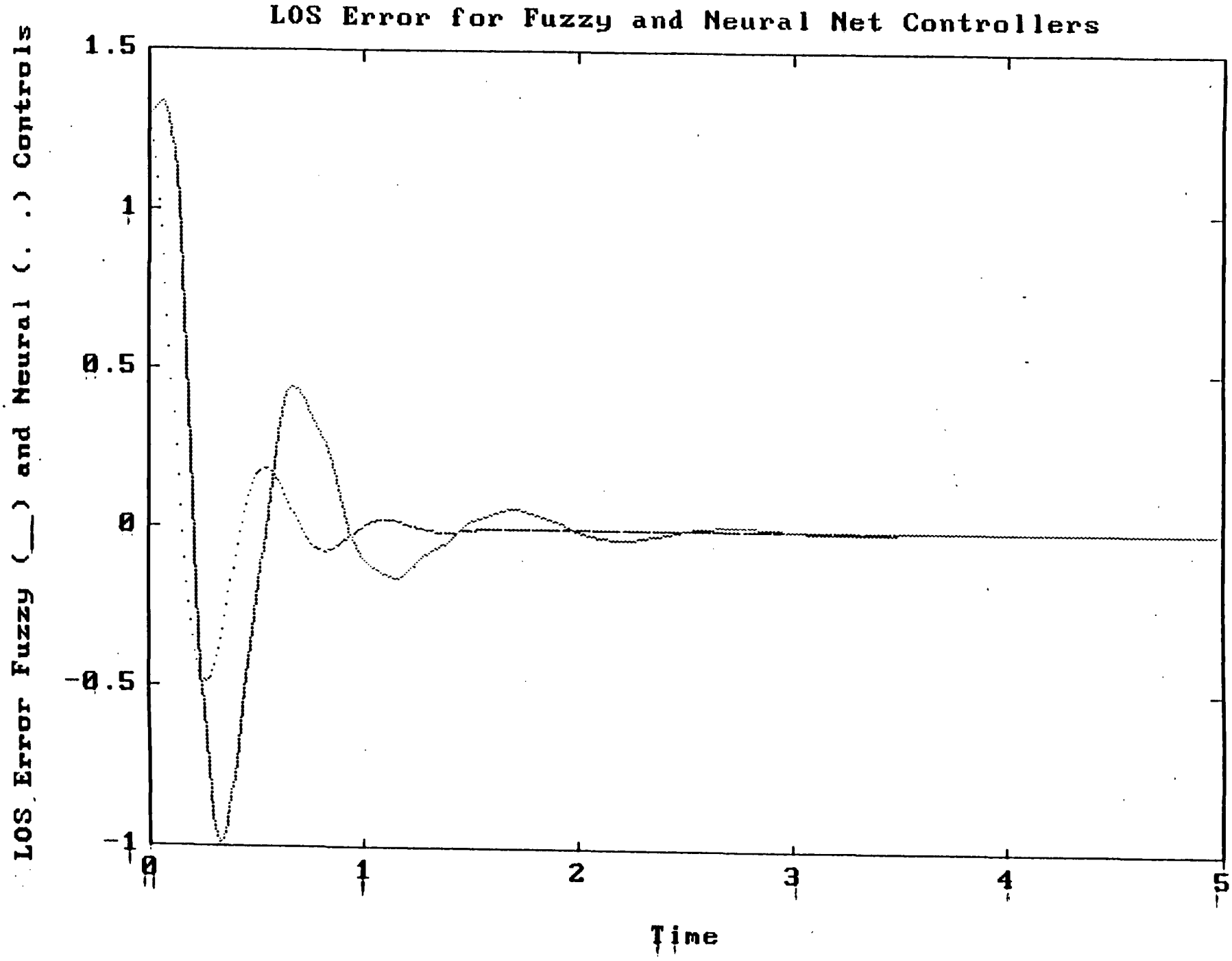
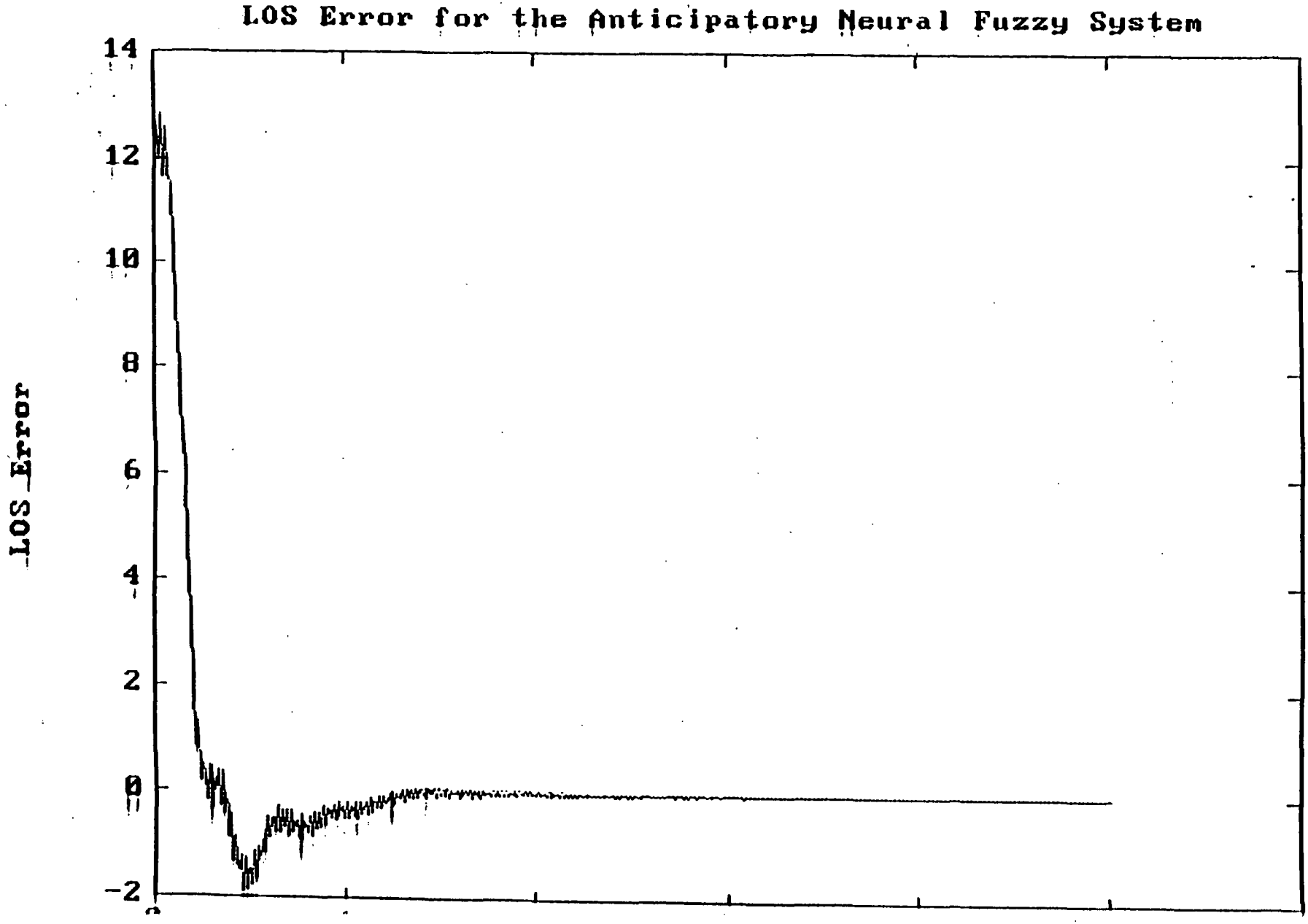
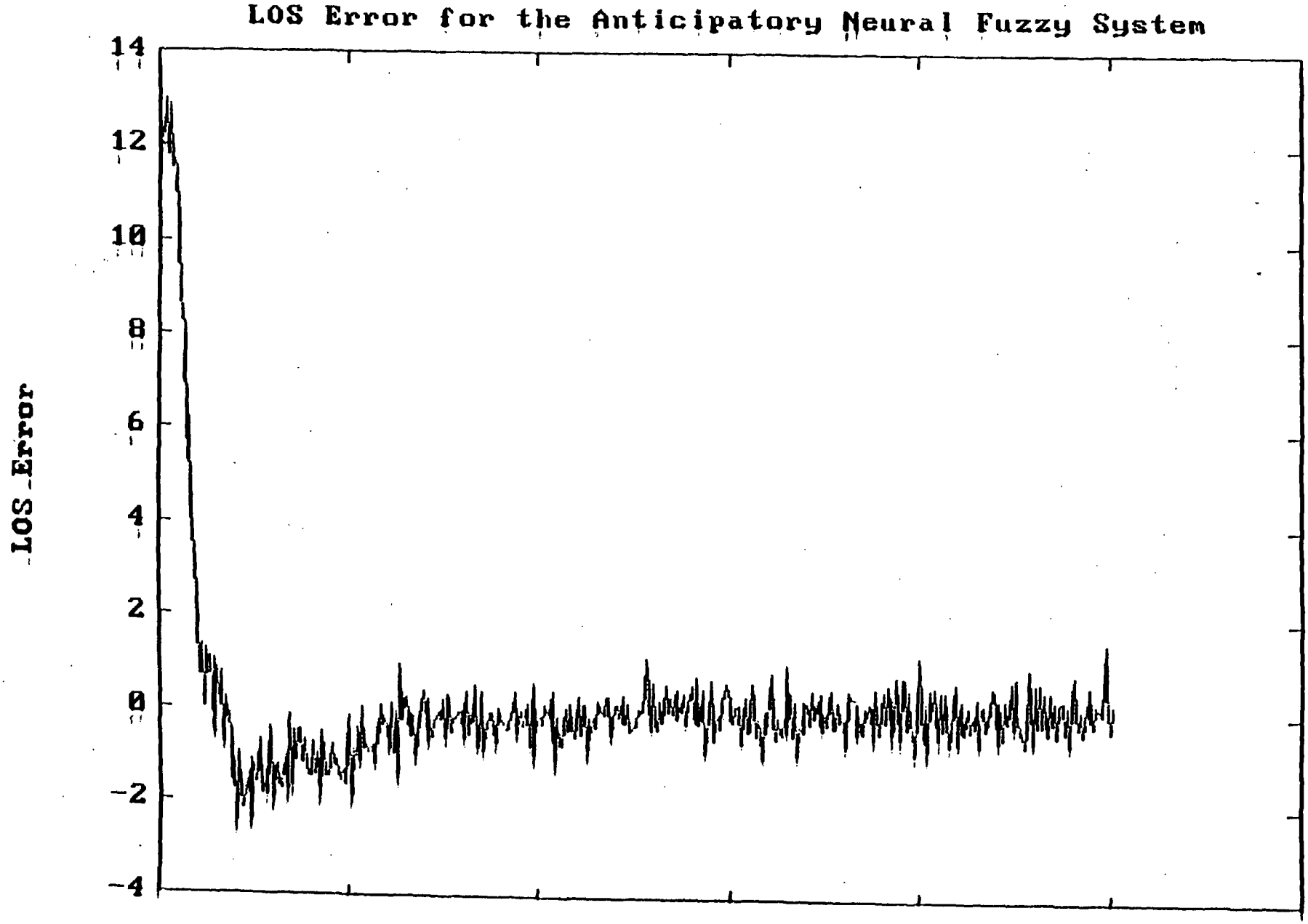
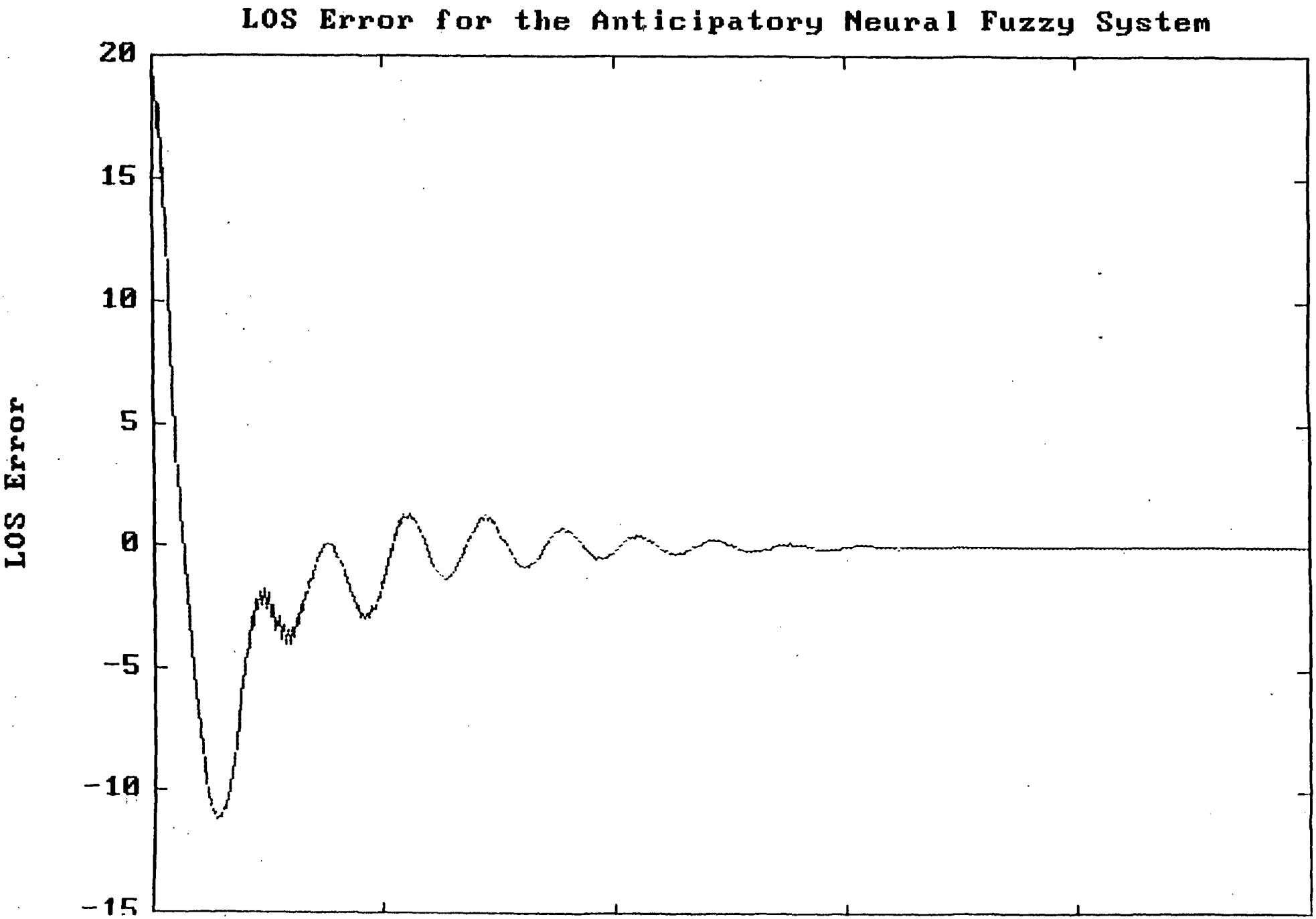

Fuzzy and Neural Net Generated Control Signals
Fuzzy (__) and Neural (. . .) Controls
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
Time

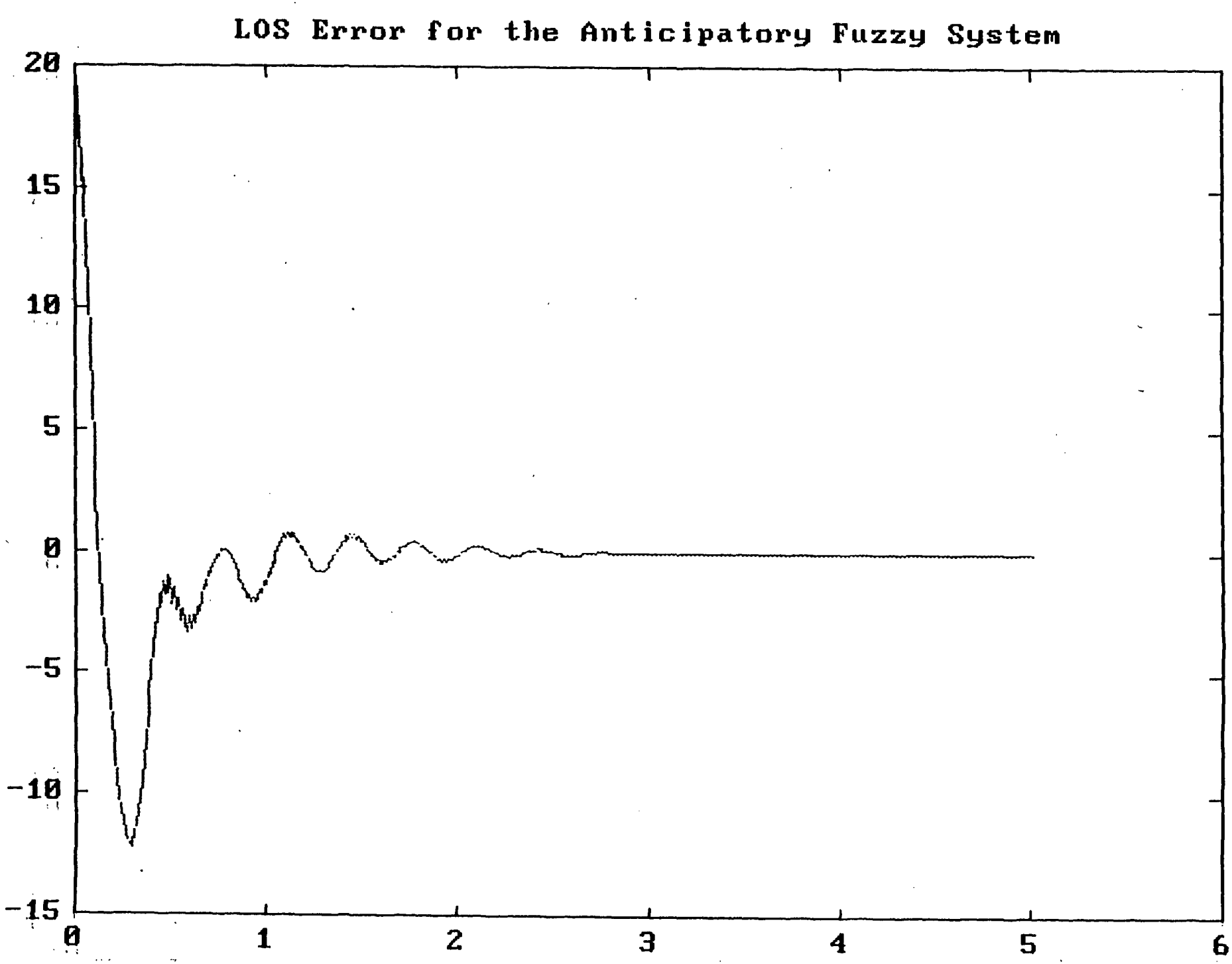
Figure 5 a

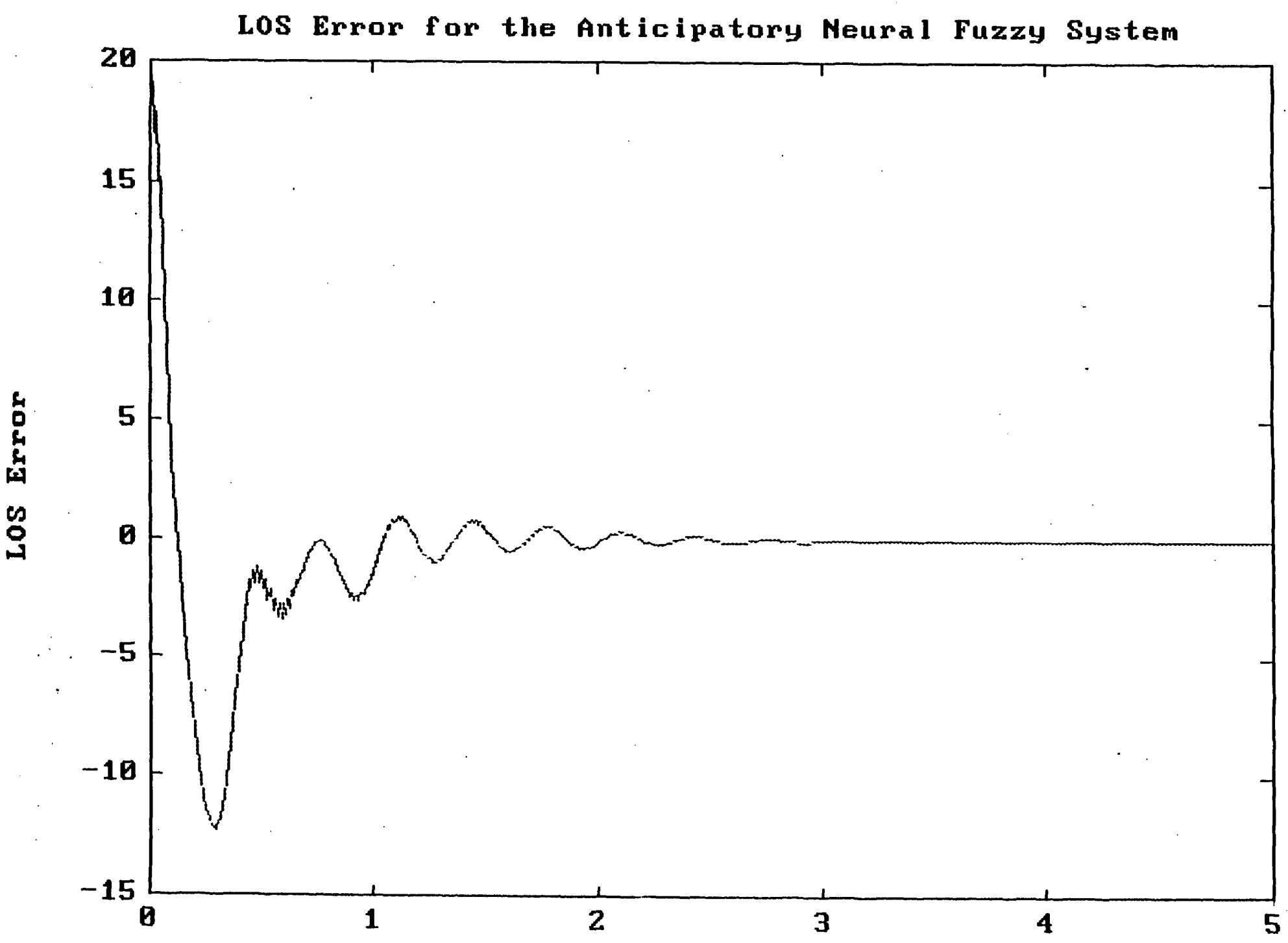

LOS Error for Fuzzy and Neural Net Controllers
LOS Error Fuzzy (__) and Neural (. .) Controls
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
Time

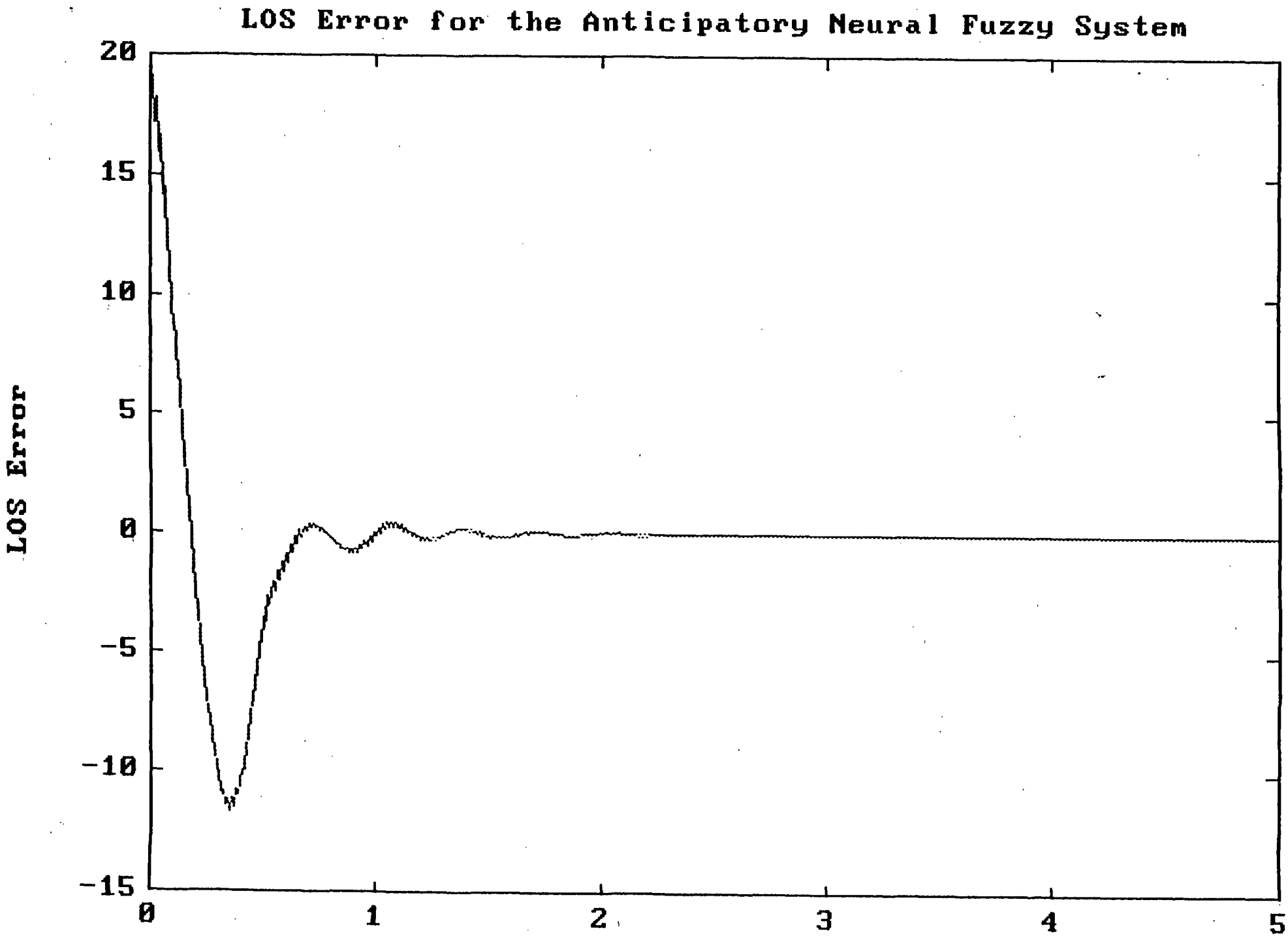

LOS Error for the Anticipatory Neural Fuzzy System
LOS Error
14
12
10
8
6
4
2
0
-2


LOS Error for the Anticipatory Neural Fuzzy System
LOS_Error
14
12
10
8
6
4
2
0
-2
-4


LOS Error for the Anticipatory Neural Fuzzy System
LOS Error
20
15
10
5
0
-5
-10
-15


LOS Error for the Anticipatory Fuzzy System
20
15
10
5
0
-5
-10
-15
0
1
2
3
4
5
6
LOS Error


LOS Error for the Anticipatory Neural Fuzzy System
LOS Error
20
15
10
5
0
-5
-10
-15
0
1
2
3
4
5


LOS Error for the Anticipatory Neural Fuzzy System
LOS Error
20
15
10
5
0
-5
-10
-15
0
1
2
3
4
5

Lawrence Smith EM11

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

FLEXIBLE BODY CONTROL USING NEURAL NETWORKS
Final Report

5. Report Date

3/4/92

6. Performing Organization Code

7. Author(s)

Dr. Claire L. McCullough

8. Performing Organization Report No.

10. Work Unit No.

9. Performing Organization Name and Address

Electrical and Computer Engineering Dept.
University of Alabama in Huntsville
Huntsville, AL 35899

11. Contract or Grant No.

NAS8-36955 D.O. 127

13. Type of Report and Period Covered

Draft Final; 6/91-3/92

12. Sponsoring Agency Name and Address

NASA
Marshall Space Flight Center, AL

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Final report for progress made under contract to control the Control Structures Interaction Suitcase Demonstrator (a flexible structure) using Neural Networks and Fuzzy Logic.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Neural nets; flexible structures; fuzzy logic	unclassified; unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
unclassified	unclassified	29	

NASA FORM 1626 OCT 86