Application of Neural Network in Control of a Ball-Beam Balancing System

by

Yuhong Jiang

A thesis submitted to

DUBLIN CITY UNIVERSITY

for the degree of

Master of Engineering

Supervisor: Dr. Charles McCorkell

School of Electronic Engineering

DUBLIN CITY UNIVERSITY

November 1991

I declare that the research herein was completed by the undersigned and has not been submitted for a degree to any other institutions.

Signed: Yuhong Jiang **Date:** 14 Dec. 1991

ACKNOWLEDGEMENT

I wish to express my sincere thanks to my supervisor, Dr. Charles McCorkell for the continued guidance and support throughout the research and his patient advice and encouragement during the preparation of the thesis.

I would also like to thank the staff members in the School of Electronics Engineering at Dublin City University, especial Dr J Ringwood for his advice and assistant on some of the programming Last, but not least, my sincere thanks to John Whelan for his help in the instrumentation some of the experimental work, and David Condell, who helped in the installation and maintenance of the hardware involved

Summary

Neural networks can be considered as a massively parallel distributed processing system with the potential for ever improving performance through dynamical learning. The power of neural networks is in their ability to learn and to store knowledge. Neural networks purport to represent or simulate simplistically the activities of processes that occur in the human brain. The ability to learn is one of the main advantages that make the neural networks so attractive. The fact that it has been successfully applied in the fields of speech analysis, pattern recognition and machine vision gives a constant encouragement to the research activities conducted in the application of neural networks technique to solve engineering problems. One of the less investigated areas is control engineering. In control engineering, neural networks can be used to handle multiple input and output variables of nonlinear function and delayed feedback with high speed. The ability of neural networks to control engineering processes without prior knowledge of the system dynamic very appealing to researchers and engineers in the field.

The present work concerns the application of neural network techniques to control a simple ball-beam balancing system. The ball-beam system is an inherent unstable system, in which the ball tends to move to the end of the beam. The task is to control the system so that the ball can be balance at an location of the beam within a short period of time, and the beam be kept at an horizontal position.

The state of the art of neural networks and their application in control engineering has been reviewed. The computer simulation of the control system has been performed, using both the conventional Bass-gura (chapter 3) method and the neural network method. In the conventional method the system equations were established using the Lagrangian variational

principle, and Euler method has been used to integrate the equations of movement.

Two-layered networks have been used in the simulation using neural networks, one being the action network and the other being the evaluation network. The evaluation network evaluates the system using the previous information and the action network actuates the controller according this evaluation. The error back-propagation and temporal difference algorithms have been used in the neural networks.

The implementation of both the conventional and the neural networks control systems have been carried out on the ball-beam system in the Control Laboratory in School of Electronics Engineering, Dublin City University. The control work was performed using a 80386-based personal computer system. The neural networks system, which is a parallel processing system in nature, has been implemented with a serial computer. The results and comparison show that both in simulation and the experiments, the neural networks system performed favourabl compared with the more established conventional method. This is very encouraging since in the implementation of neural networks system the system dynamics is not necessary. It is the author's believe that should the neural network system be implemented using hardware, conserving its parallell processing characteristics.

The general concept and types of neural networks are explained in Chapter 1. The common algorithms used by neural nets are also presented in this chapter. At the end of this chapter the aim of the present work has been outlined and previous work reviewed. Chapter 2 and 3 deal with the simulation of a ball-beam system using conventional method and neural network technique, respectively. The implementation of both the control method has been presented in Chapter 4. Chapter 5 draws the conclusions and supplies the author's vision of future work in the field. A list of reference has been given in the of the work.

CONTENTS

LIST OF FIGURES

NOMENCLATURES

x	Position of the ball on the beam.
α	Angular position of the beam relative to the horizontal plane.
$\dot{x}$	Velocity of the ball along the beam.
$\dot{\alpha}$	Angular velocity of the beam.
ω	The absolute angular velocity of the ball.
v	The absolute velocity of the ball.
U^*	Co-energy.
T	Potential energy.
L	Lagrangian function.
L_o	Beam length.
l	The distance of the spring.
I_a	Moment of inertia, beam.
I_b	Moment of inertia, ball.
R	Ball rolling radius.
r	Ball rolling radius.
m	Ball mass.
ρ_m	Mass density.
k_1	Stiffness of the spring.

τ	Sample time
$F(t)$	Action force applied to the beam.
g	Gravitational acceleration.
J	The system co-content.
b	Damp factor.
α_i	The close-loop eigenvector of the system, ($i=1,2,3,4$)
a_i	The characteristic polynomial of the system, ($i=1,2,3,4$)
k	The feedback vector.
r'	A failure signal from the system.
r_1	A failure signal from the evaluation network output.
a_{ij}, b_i, c_i	The weight value of the evaluation network.
d_{ij}, e_i, f_i	The first layer output of the action network.

β, β_h, ρ_e, ρ_h, ρ, γ	The parameters of the neural network learnmg algonthms.

CHAPTER 1 INTRODUCTION

1.1 Concept and Brief History of Neural Networks

An artificial neural network, as its name implies, is a network of artificial "neurons". These neurons, sometimes knows as nodes, are computational units which can perform certain simple computations, thus enabling the network as a whole to represent or simulate simplistically the activities of processes that occur in the human brain. Essentially, neural networks can be considered as a massively parallel distributed processing system with the potential for ever improving performance through dynamical learning. The power of neural networks is in their ability to learn and to store knowledge. They can be used to handle multiple input and output variables of nonlinear function, and delayed feedback, with high speed. Applications of artificial neural networks are mostly found in the area of speech analysis, pattern recognition, but with the development of fast architectures it is also very attractive to introduce the technique into control engineering.

1.1.1 Brief history

The initial steps towards artificial neural networks or simply "neural-like networks", which were motivated by a paper of McCulloch and Pitts (1943), primarily concerned computational and representational issues. The first major contribution on learning in biological neural networks was made by Hebb (1949), who suggested that learning results from the formation of neuron assemblies based on the strengthening of connections between simultaneously firing neurons. Pioneering work on artificial neural networks learning was done by Rosenblatt (1958, 1962) who introduced the perceptrons and demonstrated experimentally that these neural-like networks in principle are capable of learning. Complex networks were poorly understood at that time, and research mainly focused on the structurally restricted elementary perceptrons. Neural network research was popular in the early 1960's due mainly to the contributions of Rosenblatt (1959) and Widrow and Hoff (1960). It has been proven by Nilsson (1965) that one of the learning procedures proposed by Rosenblatt (1962), the perception convergence procedure, achieves the desired input-output behaviour of the elementary perceptions, if they at all can achieve it. Minsky and Papert (1969) provided an excellent mathematical analysis of these restricted networks; in particular, they proved that these networks have several strong computational limitations. Among the many contributors to the field Grossberg (1967, 1982) and Fukushima (1975) have maintained research efforts since the sixties with continued contributions. These limiting results, the lack of learning procedures for networks being more complex than elementary perceptions, and the growing interest in symbolic information processing caused a rapid decrease in artificial neural networks research.

Since about 1980, due to the advances in VLSI implementation techniques and the development of parallel computers, this situation has changed completely. Presently artificial neural networks are the subject of most intensive research activities, and one of the major goals is the development of learning procedures that could work efficiently even for large real-tıme tasks.

Among those who helped the new resurgence of actıvıtıes are Hopfield and Tank (1986, 1985). Rumelhart and Mcclelland and theır parallel dıstnbuted processıng group (1986), Hecht-Nıelsen (1986) and hıs pıoneerıng work ın neuro-computers and Kosko (1987). The renewed ınterest was due, ın part, to powerful new neural models, the multı-layer perceptıon and the feedback model of Hopfield, and to learnıng methods such as back-propagatıon, but ıt was also due to advances ın hardware that have brought wıthın reach the realızatıon of neural networks wıth very large numbers of nodes.

In the field of control engıneerıng, there have been numerous efforts to develop more heurıstıc approaches to control. Documented results are mostly sımulatıons of the controlled plant and neural network. In 1990, Kraff and Campagna reported a comparıson between CMAC neural network control and two tradıtıonal adaptıve control systems. Sannev and Akın ın 1990 reported neurmorphıc pıtch attıtude regulatıon of an underwater telerobot. The applıcatıon of neural networks ın the balancıng of a ınverted pendulum was made by Anderson (1987, 1988, 1989). The neuron-lıke adaptıve elements that can solve dıfficult learnıng control problems was made by Barto, Sutton, and Anderson (1983).

1.1.2 Artificial neural network

Artıficıal neural network (ANN) can be consıdered as massively parallel dıstrıbuted processıng systems wıth the potentıal for ever-ımprovıng performance through dynamıcal learnıng. They allow non-algorıthmıc ınformatıon processing, i.e., no "programming" is required as in more conventional algorithmic signal processing. The ultimate object of artificial neural network is to closely simulate the function of the human neural system. Indeed multi-layered networks have been shown to develop very similar structures to existing human physiological structures with no human interaction or guidance. Also, the development of fast architectures makes implementation in real time feasible unlike artificial intelligence techniques which are infamous for their lengthy computation times.

One the objectives of intelligent control is to design a system with acceptable performance characteristics over a very wide range of uncertainty The system must be robust enough to deal with unexpected occurrences, large parameter variations, unquantified data, or extremely large quantities of data. Besides the approaches of expert systems and fuzzy logic, an increasingly popular approach is to augment control systems with artificial neural networks.

An artificial neural network (ANN) is a system consisting of simple processing elements called "units" or "nodes" that interact using weighted connections. It can interact with its environment, and it may be capable of learning. A neural network can be defined by its processing behaviour, its interface to the environment, its structure, and its learning procedure.

The processing behaviour of an artificial neural network is defined by the computations performed by its units and their temporal coordination. Generally, a unit calculates three functions: an input function producing the unit's input value; an activation function producing the unit's activation value; and an output function producing the unit's output value. In most neural network models only one function called "transfer function" is assumed to be equal for all units.

In the case of stochastic transfer functions the output of an unit depends on the unit's input in a probabilistic fashion, and changing the weights means changing the probability of the output values.

Most widely used transfer functions are the linear function, the sigmoid function and the stochastic function. Denoting x_i as the inputs, y_i as the outputs, w_{ij} as the connection weights, and N as the number of inputs, the three transfer functions can be briefly outlined here In the linear function, the output of a node is the linear combination of all the inputs,

$$y_i = \sum_{j=0}^{N} w_{ij} x_j \qquad \textbf{(1-1)}$$

In the function the output is assumed unit if it is greater than zero and zero otherwise, i.e.

$$y_i = \frac{1}{(1+e^{-x_i})} \qquad \textbf{(1-2)}$$

The final values of y_i are computed according to the following

$$y_i = \begin{cases} 1 & \text{if } \; y_i > 0. \\ 0 & \text{if otherwise;} \end{cases} \qquad \textbf{(1-3)}$$

The stochastic function is used to describe the undeterministic nature of the real world problem and is given as

$$y_i = \begin{cases} 0, & \text{with probability } 1-p_i \\ 1, & \text{with probability } p_i = \dfrac{1}{1+e^{x_i}/T} \end{cases} \tag{1-4}$$

where

$$x_i = -b_i + \Sigma w_{ij}\, y_j \tag{1-5}$$

and b_i and T are real-valued parameters. The variable b_i is sometimes called the threshold or bias of x_i. This threshold can be eliminated by giving each unit an extra input line with weight $-b_i$ and constant input 1.

Linear networks, which build up the simplest class of networks, show several computational and representational limitations (Rumelhart, Hinton and McClelland, 1986) However, despite these limitations, linear networks exhibit some interesting properties and they are useful for a number of theoretical studies. An extensive analysis of linear networks is provided by Kohonen (1977, 1988). Non-linear networks overcome these limitations of the linear ones. Furthermore, with regard to propositional logic, for any logical expression there is a network of binary threshold units representing it (McCulloch and Pitts, 1943). With regard to automata theory, which was mainly influenced by the work of McCulloch and Pitts, the class of threshold-unit networks and the class of finite automata are equivalent (Kleene, 1956). Recently it has been proven that every mapping from external input to output patterns can be implemented by a finite three-layered network (Hecht, 1986, Hornik, Stinchcombe and White, 1989).

The net-environment interaction results in the distinction between input units, those receiving input from the environment, output units, those providing output to the environment, and hidden units, those being neither input units nor output units Both the input and output are called visible units; the set of all visible units, which may be time-varying, make up the network's interface to its environment.

The structure (topology, architecture) of a neural network is defined by the arrangement of its units, that means, by the set of all weighted connections between the units. Several kinds of structure exist for neural networks.

A layered or hierarchical network is one whose units are hierarchically organized into disjoint layers. Based on this hierarchical ordering, it is usual to distinguish between lower and higher layers. A bottom-up (top-down) network is a layered network whose units only affect units at the same and higher (lower) layers, and an interactive network is one having both bottom-up and top-down connections. A feed-forward network is a layered network whose units only affect units at higher layers, whose lowest layer is an input layer, whose intermediate layers are hidden layers, and whose highest layer is an output layer. A perceptron is a feed-forward network consisting of binary threshold units; a one-layered perceptron is called " elementary perceptron". A recurrent network or cyclic network or network with internal feedback is one whose external output may affect its external input. A symmetric network is a network being both symmetrically connected (c_{ij} exists if and only if c_{ji} exists) and symmetrically weighted($w_{ij} = w_{ji}$).

An artificial neural network learns by means of appropriately changing the weights of the connections and external input lines. How this is done is described by its learning procedure.

1.1.3 Basic aspects and characteristics of artificial neural learning

The central point of artificial neural learning (ANL) is to form associations between patterns. There are two variants of the association: auto-association and hetero-association. An auto-association is one in which a pattern is associated with itself. The goal of the auto-association is pattern completion: After the network has learned a pattern, whenever a part of it is presented to the network, the network has to produce the total pattern. A hetero-association is one in which two different patterns have to be associated. The goal of this is that whenever one of the two associated patterns is presented to the network, the network supplies the other one. Learning that can be viewed as special variants of the pattern association are regularity detection and pattern classification/recognition. Other variants of artificial neural learning, apart from the association, are the mapping and the modelling. Due to the former, a multidimensional mapping from the input to the output pattern has to be constructed. Due to the latter the network's environment has to be internally modelled (where the information about the environment will be encoded in the weights).

Artificial neural networks have shown some interesting and powerful features in building up and representing associations (mapping, environment models). In particular, artificial neural networks are capable of implicit generalization to new associations (see McClell, Rumelhart and Hinton ,1986, for some general considerations and Baum and Haussler, 1989, for a formal analysis of generalization and representation), thereby the patterns themselves are not stored but the connection strengths that enable the network to recollect the associated pattern are, even if it is common to speak of "storing a pattern".

A neural network learns by means of appropriately changing the weights, and this weight changing or weight adaptation happens during the so-called training phase or learning phase. In this phase the external input patterns that have to be associated with specific external output patterns or specific activation patterns across the network's units are presented to the network. The set of all these external input patterns is called the training set, and a single input pattern is called a training instance. The network may receive environmental learning feedback, and this feedback is used as additional information for determining the magnitude of the weight changes that are necessary for representing the desired associations. Typically the weight changes can be done in parallel, and this makes up one of the main characteristics of neural learning; Neural learning inherently is parallel distributed learning.

The change of the weights, in comparison with the change of the units activation states, occurs on a slow time scale. Hence, one can distinguish two kinds of network dynamics: slow dynamics constituted by the process of updating the weights and fast dynamics constituted by the process of updating the activation values of the units.

1.1.4 Construction

Neural networks are composed of many units that simulate the properties of real

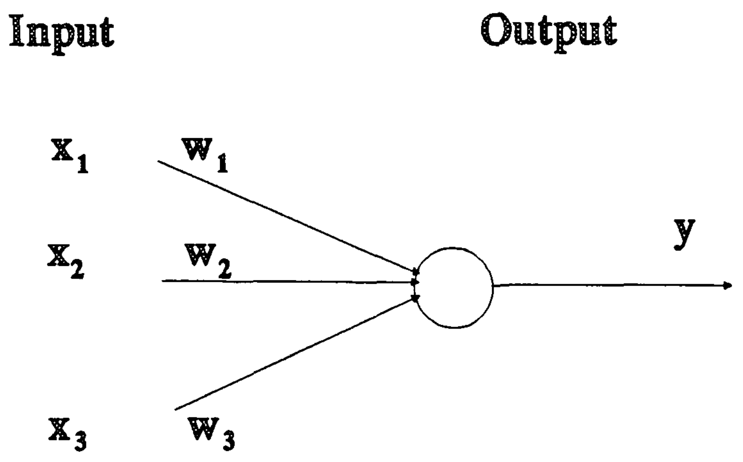


Figure 1.1 One neural node. x_i is input, y is output and w_i is the connection weights

neurons in the central nervous system Each unit or node can have many inputs and usually a single output. The input may be either inhibitory or excitatory and a node produces an output relative to a weighted sum of the inputs. The output is usually a binary state (on or off) although more complicated networks use graded outputs. Figure 1.1 shows a single node, at which the inputs x_i are passed through a non-linearity function.

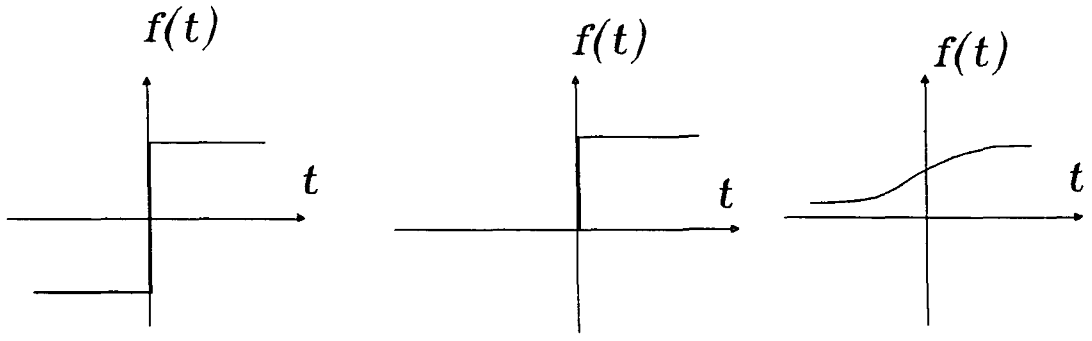
$$y=f(\zeta)=f\left(\sum_{i=0}^{N-1} w_i x_i - \theta'\right) \tag{1-6}$$

where

w_i is the weight from input i at time t.

θ' is the threshold in the output node (small random values) for typical non-linear transformations.

Computational element or node which forms a weighted sum of N input and passes the result through a non-linearity. Three representative nonlinearities are shown in Figure 1.2.



Hard limiter Threshold logic Sigmoid

Figure 1.2 Threshold functions

A neural network is composed of many such nodes connected together in a certain topology. The interconnection topology is important because its complexity determines how easily the weights may be adjusted for learning. Singly layered units consisting of only input and output nodes, as shown in Figure 1.3, have been proven to be unable to perform certain important calculations. Multi-layered networks with hidden layers between the input and output nodes, as shown in Figure 1.4, can overcome these problems.

1.1.5 Application disciplines

Artificial neural network research receives interest from several disciplines. For example, finding algorithms for determining the connections and weights of a neural networks to solve a problem, mechanizing networks using microelectronic or optical approaches and investigating the operation and structure of biological neural networks. Much of the early algorithms work has been in computationally intensive areas of signal processing, such as adaptive pattern recognition, real-time speech recognition, and image interpretation. Computer and cognitive scientists have been pursuing the

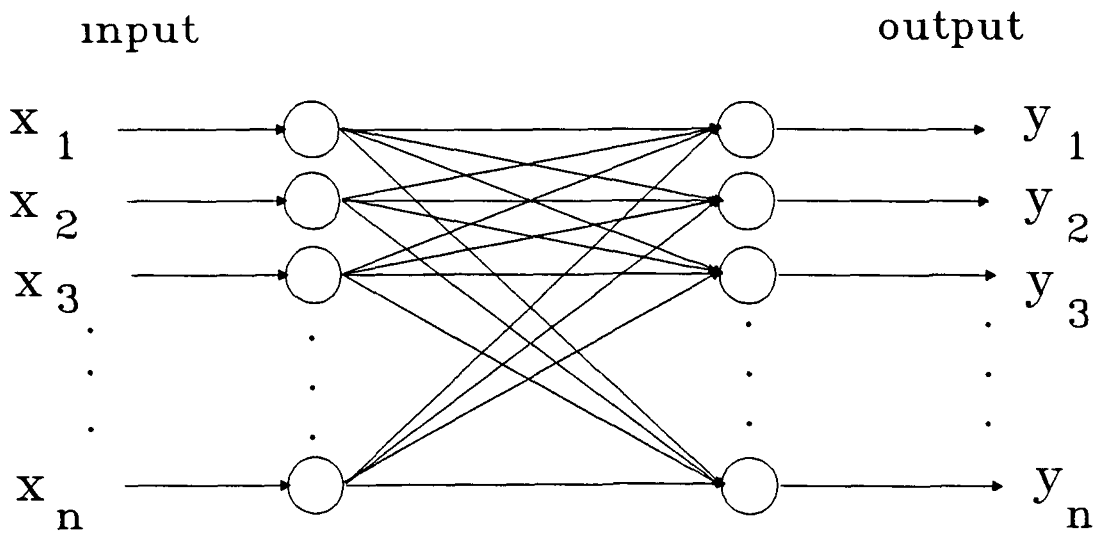


Figure 1.3 A single-layered neural network

identification of intelligence and massively parallel distributed information processing performed by biological systems.

In system and control engineering, neural networks are seen as an alternative technology by which information processing may be accomplished quickly and easily. To this end they have predominantly found application in patten recognition and signal processing (Astrom, 1987). There are also areas that are computationally intensive, such as real-time identification and control of large flexible structures in aerospace or robotics.

1.2 Learning Algorithms

Neural networks have the ability to learn to learn and store knowledge. Both of these important functions are achieved through adaptation of the synaptic weights assigned to each node's input. The weights are adjusted by a learning algorithm. Learning algorithms fall into three general categories:

(1) Supervised learning

(2) Unsupervised learning

(3) Associative reinforcement

In this work, all error back-propagation algorithm (Lippmann, 1987) and a supervised learning method are used in action and evaluation networks. The reinforcement learning method (Barto and Sutton, 1983, Anderson, 1987) and association reinforcement are used in the action network. Temporal-difference (TD) learning method (Anderson, 1987) supervised learning is used in evaluation network.

1.2.1 Supervised learning algorithms

The more popular supervised learning techniques employ a "teacher" who presents the desired output to the network for a given input pattern. For example, the perceptron convergence procedure, the back-propagation and the Boltzmann learning".

1.2.1.1 The perceptron convergence procedure learning

In the 1950's and 1960's Rosenblatt investigated the learning behaviour of perceptrons Rosenblatt (1962). Much of his work deals with learning procedures for elementary perceptrons; one of these procedures, which is nowadays known as the perceptron convergence procedure, performs weight changes for each of its units as follows:

$$\Delta w_{ij} = \begin{cases} -\tau\, y_j \ (+\tau\, y_j) & \textit{if output is } 1 \ (0) \ \textit{ but should be } 0 \ (1) \\ 0 & \textit{if output is correct} \end{cases} \qquad (1\text{-}7)$$

where τ is the learning rate and d_i is the desired output value of unit u_i.

The well-known perceptron convergence theorem states that there is at least one set of weights such that the elementary perceptron works correctly. This theorem says nothing about the existence of such a "correct set of weights". If there is no such set then the perceptron convergence procedure might lead to unreasonable results; (Hinton, 1987) This is because the perceptron convergence procedure ignores the magnitude of the error produced by the elementary perceptron.

1.2.1.2 Back-propagation learning

The technique of back-propagation (BP) or error propagation was developed by (Werbos, 1974). Independently of Werbos' work, and Hinton and Williams(1986) all applied this technique to the task of learning in artificial neural networks. The word "back-propagation" refers to a specific type of learning procedure for supervised learning that is intensively studied within ANN research. The following consideration focus on the elementary version of back-propagation, back-propagation for semi-linear feed-forward networks.

The back-propagation method involves two phases for each input-output case to be learned, see Figure 1.4 In the first phase, the "forward pass", an external input pattern is passed through the network from the input units towards the output units,

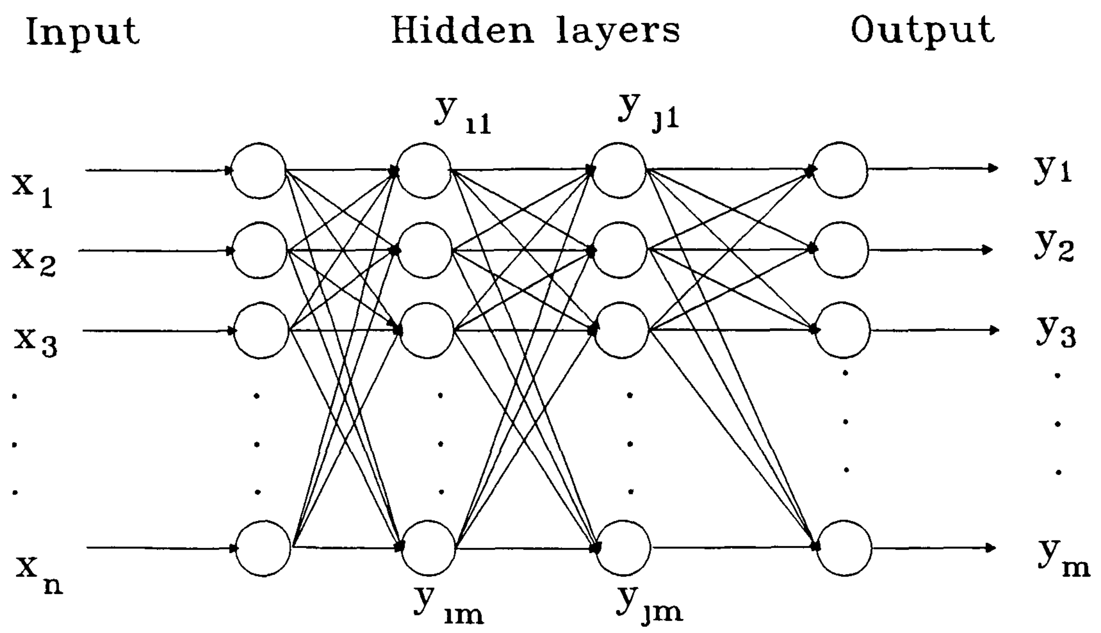


Figure 1.4 Three layers perceptron with N continuous values input

adapting to an external output pattern. This output pattern is compared with the desired external output pattern, and the error signal for each output unit is produced.

In the second phase, the "backward pass", the error signals of the output units are passed backward from the output units towards the input units. The error signals for input and hidden units are evaluated recursively, i.e., iteratively for each next-lower layer; to evaluate an error signal for an input or a hidden unit the error signals of the units to which this unit is connected has to be taken into consideration.

The back-propagation training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multi-layer feed-forward perceptron and the desired output. It requires continuous differentiable non-linearities. The following assumes that a sigmoid logistic non-linearity is used where the function f(zeta) is:

$$f(\zeta)=\frac{1}{1+e^{-(\zeta-\theta)}} \quad (1\text{-}8)$$

step 1:

Initialize weights and offsets.

Set all weights w_{ij} and node offsets to small random values θ.

step 2:

Present input and desired outputs.

Present a continuous valued input vector $x_o, x_1, x_2, \ldots\ ..x_{N-1}$ and specify the desired output $d_0, d_1, d_2 \ldots\ldots d_{M-1}$.

step 3.

Calculate actual output y_N.

$$y_j = f\left(\textstyle\sum_{i=0}^{N-1} w_{ij}\, x_i - \theta_i \right)$$

$$y_M = f\left(\textstyle\sum_{j=0}^{N-1} w_{jM}\, x_j - \theta_M \right) \quad (1\text{-}9)$$

$$0 \le M \le N-1$$

step 4:

Adapt weights

Use a recursive algorithm starting at the output nodes and working back to the first hidden layer.

Adjust weights by

$$w_{ij}\ (t+1) = w_{ij}\ (t) + \mu\ \delta_j\ x_i \quad (1\text{-}10)$$

where

w_{ij} is the weight from hidden node i.

x_i is either the output of node i or is an input.

δ_j is an error term for node j.

μ is a gain term.

If node j is an output node, then

$$\delta_j = y_j\,(1-y_j)\,(d_j - y_j) \tag{1-11}$$

If node is an internal hidden node, then

$$\delta_j = \acute{x}_j\,(1-\acute{x}_j)\,\Sigma_M\,\delta_k\,w_{jM} \tag{1-12}$$

Where

y_j is the actual output.

d_j is the desired output of node j.

k is over all nodes in the layers about node j.

Internal node thresholds are adapted in a similar manner by assuming they are connection weights on links from auxiliary constant- valued input. Convergence is sometimes faster if a momentum term is added and weight changes are smoothed by

$$w_{ij}\,(t+1) = w_{ij}\,(t) + \mu\,\theta_j\,\acute{x}_j + \alpha\,(w_{ij}\,(t) - w_{ij}\,(t-1)) \tag{1-13}$$

$$(0 < \alpha < 1) \tag{1-14}$$

(1-15)

step 5:

Repeat by going to step 2.

There are a lot of variations and extensions of the elementary version of back-propagation, some of them are mentioned briefly below.

BP described above is only applicable to non-recurrent networks. It can be applied also to recurrent networks by taking advantage of the fact that for every recurrent network there is a non-recurrent network with identical behaviour (for a finite time); this approach, which is called "unfolding-in-time BP" or "BP through time", is described in Rumelhart, Hinton and Williams (1986). Other extensions of back propagation to learning procedures for recurrent networks are presented in the results of Almeida (1987), Pineda (1987) and Rohwer (1987).

The major problem with BP is that it requires much time to learn, and there are various attempts to cope with this problem. This method aims at beginning with a network having few units, and dynamically adding units to hidden layers whenever gradient descent in the weight error surface happens too slowly.

1.2.2 Unsupervised learning algorithms

Unsupervised learning methods do not need a "teacher", they usually employ a local gradient algorithm to adjust the networks weights based around the activity near each particular node. For example, topology-preserving feature maps and adaptive resonance theory, and development of feature analyzing cells.

1.2.2.1 Topology-preserving feature maps

Topology-preserving feature maps (TPFM) was developed by Kohonen (1982, 1988). This method has been used to the sensory modalities-visual area, auditory area, somatosensory area, etc, and to the various operational areas-speech area, motor area, etc.

Topology-preserving feature maps method has two phases. In the first phase, the input pattern $x=(x_1, x_2, ..., x_n)$ at time t is located. Denoting the weight vector W as $W = (w_1, w_2, ...w_n)$, the following is defined,

$$| X(t) - W|_E = \min_i (| x_i(t) - w_i(t) |_E) \qquad (1\text{-}16)$$

In the second phase, the weight vector $w_i(t)$ is determined by

$$w_i(t+1) = \begin{cases} w_i(t) + \alpha[x_i(t) - w_i(t)], & \text{if } i \in N_s \\ w_i(t), & \text{otherwise} \end{cases} \qquad (1\text{-}17)$$

Where α is a positive scalar constant and $|.|_E$ is a distance function.

1.2.2.2 Adaptive resonance theory

The adaptive resonance theory (ART) was developed by Grossberg (1976, 1978) and it has been used in speech and visual perceptron. They have two networks, ART1 and ART2 networks. A mathematical analysis of the fast and slow dynamics of ART1 network and ART2 network is provided by Carpenter and Grossberg (1987).

1.2.2.3 Development of feature-analyzing cells

The concept of feature-analyzing cells (DFAC) was developed by Linsker (1986, 1988) and Stotzka and Maenner (1989). The method has been successfully applied in the area of visual analysis. The method has been introduced to overcome the constraints of the supervised learning that the exact performance of each node of the network must be known for each training pattern. A type of network has been developed which requires a "critic" instead of a "teacher", thus enabling the network to adjust its performance according to the response from the critic. Methods of this nature are collectively called the associative reinforcement. Some of the examples are the associative reward-penalty method, the reinforcement-comparison method, and the temporal-difference method.

The associative reinforcement learning is achieved by giving the system a "reward" when the reinforcement signal indicates a success, and a "penalty" when the reinforcement signal indicates a failure.

1.2.2.4 The associative reward-penalty learning

The associative reward-penalty (A_{r-p}) algorithms were used in control engineering, pattern classification and system identification (Barto, 1987, Barto and Anderson, 1985, Barto and Sutton, 1981, 1982). This method recognises that environmental feedback may not be informative as to providing individualised instruction to each adaptive element. A scalar evaluation signal (critic) is used to assess the general performance (reward/penalty) of the A_{r-p}. This common scalar signal is used by all of the elements to adapt their weights. The advantage of this algorithm is that learning occurs without the need for a very knowledgable "teacher". A "critic" is sufficient which can provide a success/failure indication of the result of an applied action.

The A_{r-p} algorithm nodes output y_k is

$$y_k = \begin{cases} 0 & \textit{otherwise}; \\ 1 & \textit{if } \theta_k^T x_k + \eta_k > 0; \end{cases} \qquad (1\text{-}18)$$

Where

η_k are independent identically distributed random variables.

θ_{Kt} are weight vector values.

x_k are input values.

Then

$$p_k = 1 - \psi (-\theta_k^T x) \tag{1-19}$$

The weight vector is updated according to the following equation:

$$\theta_{k+1} - \theta_k = \begin{cases} \lambda \ \rho_k [1 - y_k - p_k] \ x_k & \text{if } r_k = 0 \ (\textit{penalty}); \\ \rho_k [y_k - p_k] \ x_k & \text{if } r_k = 1 \ (\textit{reward}); \end{cases} \tag{1-20}$$

$$\textit{Where} \quad 0 \le \lambda \le 1, \qquad \rho_k > 0.$$

The A_{r-p} algorithm is local both in space and in time, as a consequence the A_{r-p} procedure is easy to implement. The A_{r-p} algorithm has been used for learning in layered networks. A major problem with the A_{r-p} procedure is its very slow speed of learning in case of large networks. To overcome this, another type of algorithm has been developed, which is named the reinforcement learning algorithm.

1.2.2.5 Reinforcement-comparison learning

In the reinforcement-comparison learning (RCL) method the weights changes are correlated with the result of comparing the current reinforcement level with past reinforcement levels. This method has been reported by Sutton in 1984.

Reinforcement-comparison learning has two methods: elementary method and prediction method. Prediction methods can be divided into classical prediction (reinforcement learning method) method and temporal-difference (TD) method.

1.2.2.5.1 Elementary methods for reinforcement comparison (Barto, Anderson and Sutton, 1983, Barto and Sutton, 1981, Sutton, 1984) . This method uses the difference between primary reinforcement signals received by the network at different time step. The weight update rule based on the approach is

$$\Delta w_{ij} = \tau \, [r_c - r_p] \, [y_i - p_i] \, y_i \tag{1-21}$$

Where

τ	is the learning rate
r_c	is the current reinforcement signal
r_p	is the preceding reinforcement signal
p_i	is the probability that $y_i = 1$.

1.2.2.5.2 The classical prediction (Reinforcement learning) method (Barto, Sutton and Brouwer, 1981, Sutton, 1984).

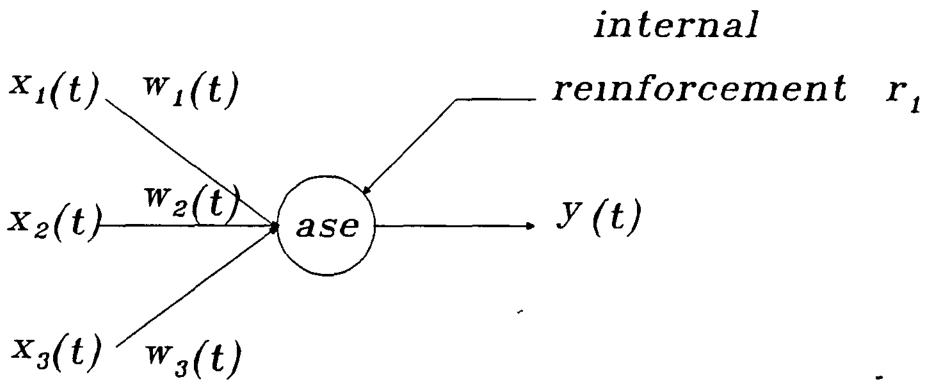


Figure 1.5 Network construction

This method uses the difference between primary and predicted reinforcement signals. A direct approach to reinforcement learning that is highly developed is the theory of learning automata, which has been extensively developed since applications were found in engineering. The weight update rule realizing a classical prediction method is given by

$$\Delta w_{ij} = \tau \, r_{heuristic} \, [\, y_i - p_i \,] \, y_i \qquad (1\text{-}22)$$
$$r_{heuristic} = r - r_{predicted}$$

This update rule has been successfully applied in the experimental studies done by Sutton (1984)

The element's output y(t) is determined from the input vector x(t)=(x$_1$(t), x$_2$(t)......x$_N$(t)) as follows:

$$y(t) = f\,[\,\sum_{i=1}^{N} w_i(t)\, x_i(t) + b(t)] \qquad (1\text{-}23)$$

where b(t) is a real random variable and f is the following threshold function.

$$f(x) = \begin{cases} -1, & \textit{if } x<0, \textit{ control down} \\ +1, & \textit{if } x\geq 0, \textit{ control up.} \end{cases} \qquad (1\text{-}24)$$

The weights w$_i$ are adjusted according to the following rules,

$$w_i(t+1) = w_i(t) + \alpha \, \hat{r}(t) \, e_i(t) \qquad (1\text{-}25)$$

For computational simplicity, we generate exponentially decaying eligibility traces e$_i$ using the following linear difference equation:

$$e_i(t+1) = \delta \, e_i(t) + (1 - \delta)\, y(t)\, x_i(t) \qquad (1\text{-}26)$$
$$(\, 0 \leq \delta < 1 \,)$$

where

α is a positive constant determining the rate of change of w$_i$

f(t) is a reinforcement value at time t.

e$_i$(t) is the eligibility at time t of input pathways i.

δ determines the trace decay rate.

Reinforcement learning involves two problems. The first problem is to construct a critic capable of evaluating plant performance in the way that is both appropriate to the actual control objective and informative enough to allow learning. The second problem is to determine how to alter controller outputs to improve performance as measured by the critic.

1.2.2.5.3 The temporal-difference (TD) methods Barto, Sutton and Anderson (1983) and Sutton (1984).

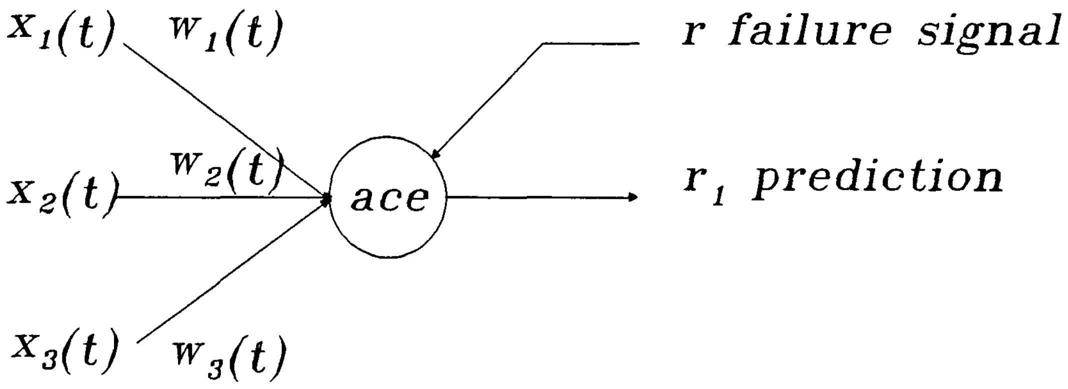


Figure 1.6 Network construction

The temporal-difference methods, as shown in Figure 1.6, demonstrate some important advantages over classical prediction method: they require less memory, all more incremental and therefore easier to compute, and produce better predictions and converge faster The temporal-difference methods learn associations among signals separated in time, such as the ball-beam system state vectors and failure signals. Through learning, the node output comes to predict the failure signal, with the strength of the prediction indicating how soon failure can be expected to occur.

The failure signal is adjusted after each step by current system states. The change of prediction is dependent on the difference between the failure signal, current system states and previous prediction.

In order to produce r^(t), the ACE must determine a prediction P(t) of eventual reinforcement that is a function of the input vector x(t), we let

$$p(t) = \sum_{i=1}^{N} v_i(t)\, x_i(t) \tag{1-27}$$

and seek a means of updating the weights v_i so that p(t) converges to an accurate prediction. The updating rule we use is

$$v_i(t+1) = v_i(t) + \beta\,[r(t) + \gamma\, p(t) - p(t-1)]\, \bar{x}_i(t)$$
$$(\, \bar{x}_i(t) = \bar{U}_i(t)\,) \tag{1-28}$$

where

β is a positive constant determining the rate of change of v_i.

r(t) is reinforcement signal supplied by the environment at time t.

$\bar{U}_i(t)$ is the value at time t of a trace of the input variable x_i.

λ determines the trace decay rate $(0 \leq \lambda \leq 1)$.

$\hat{j}$ is node ACE's output, this is a prediction value.

$$\bar{x}_i(t+1) = \lambda\, \bar{x}_i(t) + (1-\lambda)\, x_i(t) \tag{1-29}$$

The ACE's output, the improved or internal reinforcement signal, is computed from these predictions as follows.

$$\hat{j}(t) = \hat{r}(t) = r(t) + \gamma\, p(t) - p(t-1) \tag{1-30}$$

1.3 Types of Neural Network

There are many different types of networks possible in artificial neural networks and some of the more commonly encountered ones are bnefly descnbed below.

1.3.1 Associative search network

Associative search networks (Barto, Sutton and Brouwer 1981) combine

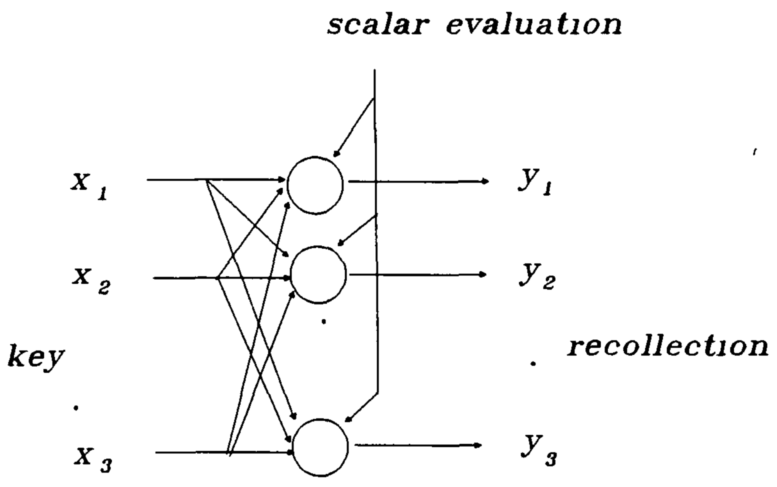


Figure 1.7 Associative search network

associative memory with a process that searches for associations worth storing according to an evaluation cntenon. It is interesting that this can be done simply by modifying the type of adaptive element used in the network. Figure 1.7 shows the organisation of an ASN. The ASN and the environment interact m a closed loop. The environment provides the ASN with a key, x_k, at each discrete time step k. This results in a recollection or output, y_k, emitted from the ASN ($y_k \in \{0,1\}$). The result of the action y_k is evaluated and a reinforcement signal $r_k \in \{0,1\}$ generated where 0 and 1 indicate "Penalty" and "Reward" respectively.

The output y_k is:

$$y_k = \begin{cases} 0, & otherwise \\ 1, & if\ \theta_k^T x_k + \mu_k > 0; \end{cases} \qquad (1\text{-}31)$$

Where μ_k are independent identically distributed random variables, each having distribution function Ψ, let p_k denote Pr{ $y_k=1$, $x_k=x$ }, then

$$p_k = P_r\,(\theta_k^T x + \eta_k > 0\,) = 1-\psi\,(-\theta_k^T x) \qquad (1\text{-}32)$$

The weight vector is updated according to the following equation:

$$\theta_{k+1} - \theta_k = \begin{cases} \lambda\ p_k\,[\,1-y_k - p_k\,]\,x_k, & if\ r_k = 0\ (penalty) \\ p_k\,[\,y_k - p_k\,]\,x_k, & if\ r_k = 1\ (reward\,) \end{cases} \qquad (1\text{-}33)$$

$$(\,0 \le \lambda \le 1\ ,\ p_k > 0)$$

The advantage of this algorithm is that the learning occurs without the need for a very knowledgable "teacher", a "critic" is sufficient which can provide a success/failure indication of the result of an applied action.

1.3.2 Hopfield neural network

Hopfield network (Hopfield,1982, 1984), as shown in Figure 1.8, can be used as a content addressable memory, an associative memory, a classifier and to solve optimization problem The operation of this network is described below.

The neuron state is assessed by

$$w_{ij} = \begin{cases} 0 & i=j. \\ \sum_{i=0}^{N-1} x_j^2\, x_i^2, & i \ne j. \end{cases} \qquad (1\text{-}34)$$

$$0 \le i,\ j \le N-1.$$

and the connection weights are updated by

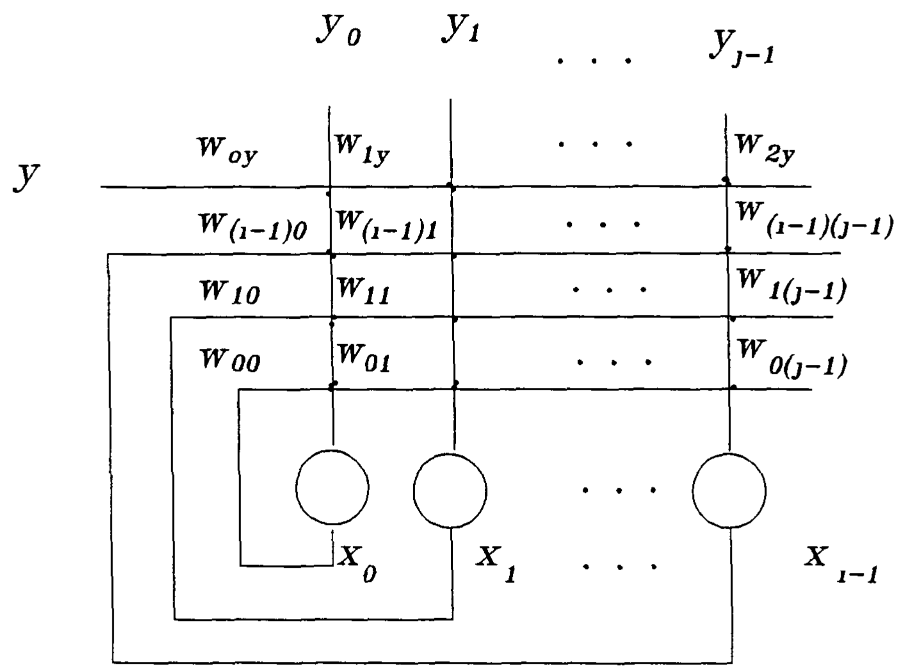


Figure 1.8 Hopfield neural network

$$w_{ij} = \begin{cases} 0 & i=j \\ \sum_{i=0}^{N-1} x_j^2 x_i^2 & i \neq j,\ (i \geq 0,\ j \leq N-1) \end{cases} \qquad (1\text{-}35)$$

where

(1) the non-linearity f_n is a sigmoid curve

$f(x) = 1/[1+\exp(-x)]$.

(2) w_{ij} are the connection weights from node i to node j.

(3) x_j (which can be +1 or -1) is the output of node i at time t.

The weights are determined by defining a quadratic energy function and adapting the weights to minimise the energy It has been shown that the rate of convergence toward a steady state is essentially independent of the number of neurons in the network.

1.3.3 Multi-layers perceptron

Multi-layer perceptron Lippmann (1987) are feed-forward networks with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the input and output nodes. A three-layer perceptron with two layers of hidden units is shown in Figure 1.4.

Multi-layered perceptrons overcome many of the limitations of single-layer, but were generally not used in the past because effective training algorithms were not available. This has recently changed with the development of new training algorithms, they have been shown to be successful for many problems of interest.

1.4 Application of Neural Networks in Control Engineering

The literature of neural networks in control system applications is expanding rapidly. In 1988, Kawato *et al* reported on hierarchical neural network models for voluntary movement with application to robotics. In order to control voluntary movements, the central nervous system must solve the following two computational problems at different levels.

(1) determination of a desired trajectory in the visual coordinates.

(2) generation of motor commands. Based on physiological information and previous models, computational theories are proposed for the first two problems, and a hierarchical neural network model is introduced to deal with motor command. The application of this approach to robotics is outlined.

In 1990, Kraff and Campagna reported a comparison between CMAC neural network control and two traditional adaptive control systems. This article compares a neural network-based controller similar to the cerebellar model articulation controllers, a self-tuning regulator, and a Lyapunov-based model reference adaptive controller. The three systems are compared conceptually and through simulation studies on the same low-order control problem. Results are obtained for the case where noise is added to the system, and for the case where a nonlinear system is controlled. Comparisons are made with respect to closed-loop system stability, speed of adaptation, noise rejection, the number of required calculations, system reaching performance, and the degree of theoretical development. The results indicate that the neural network approach functions well in noise, works for linear and nonlinear systems, and can be implemented very efficiently for large scale system.

Borto, Sutton, and Anderson 1983 reported neuron-like adaptive elements that can solve difficult learning control problems. The task is to balance a pole that is hinged to a movable cart by applying forces to the cart's base. The two single-layer networks were used in control.

The application of neural networks in the balancing of a inverted pendulum was made by Anderson (1987,1986,1989). An inverted pendulum is simulated as a control task with the goal of learning to balance the pendulum with no a priori knowledge of the dynamics. In contrast to other applications of neural networks in the inverted pendulum task performance feedback is assumed to be unavailable on each step, appearing only as a failure signal when the pendulum falls or reaches the bounds of a horizontal track. To solve this task, the controller must deal with issues of delayed performance evaluation, learning under uncertainty, and the learning of non-linear functions. Reinforcement and temporal-difference learning methods were used to deal with these issues in order to avoid unstable conditions and balance the pendulum.

The ability to learn is one of the main advantages that make the neural networks so attractive. The benefits are most dramatic when a large number of nodes are used. Some examples of the approaches taken to apply neural networks to control are below.

Sanner and Akin (1990) experimental results are a follow-up of their previous work involving computer simulations only. The neural networks performed as predicted in simulations. It was observed that unacceptable delays can be introduced if a single serial microprocessor implementations of neural networks are seen as necessary.

The control of robots is the topic addressed by Nagata, Sekiguchi and Asakawa (1990). Neural networks are used to process data from many sensors for the real time control of robots and to provide the necessary learning and adaptation capabilities for responding to the environmental changes in real time. This approach is applied to several areas of robot research.

The comparison of neural networks control and conventional control is the topic addressed Chu, Shoureshi and Fenorio (1990). Kraft and Campagna (1990). Anderson (1988, 1989) controlled the inverted pendulum system using action network and evaluation network. There are a lot of neural networks control applications appearing, among them are the investigations carried out by Antsaklis (1989) and Shriver (1988). It has been widely recognized that neural networks are a potential powerful tool for the control engineering.

evaluation network. There are a lot of neural networks control applications appearing, among them are the investigations carried out by Antsaklis (1989) and Shriver (1988). It has been widely recognized that neural networks are a potential powerful tool for the control engineering.

1.5 Present Work

The control problem studied in this work is to balance a ball on a beam, as shown in Figure 1.9. The movement of both ball and beam is constrained to the vertical plane. The state of this system is given by the beam's angle and angular velocity and the ball's horizontal position and velocity. The only available control action is to exert forces of fixes magnitude on the beam that push it to move up or move down.

The event of the beam falling past a certain angle or the ball running into the bounds of its track is called a failure. A sequence of forces must be applied to avoid failure as much as possible by balancing the ball at the given position on the beam. The beam and ball system is reset to its initial state after each failure and the controller must learn to balance the system for as long as possible.

The present work involves the use of neural networks and conventional control methods in the control of the beam and ball system. The Bass-Gura control method has been used in conventional controller design as compared with neural networks control. In the neural networks system, two networks have been used. One of them is an

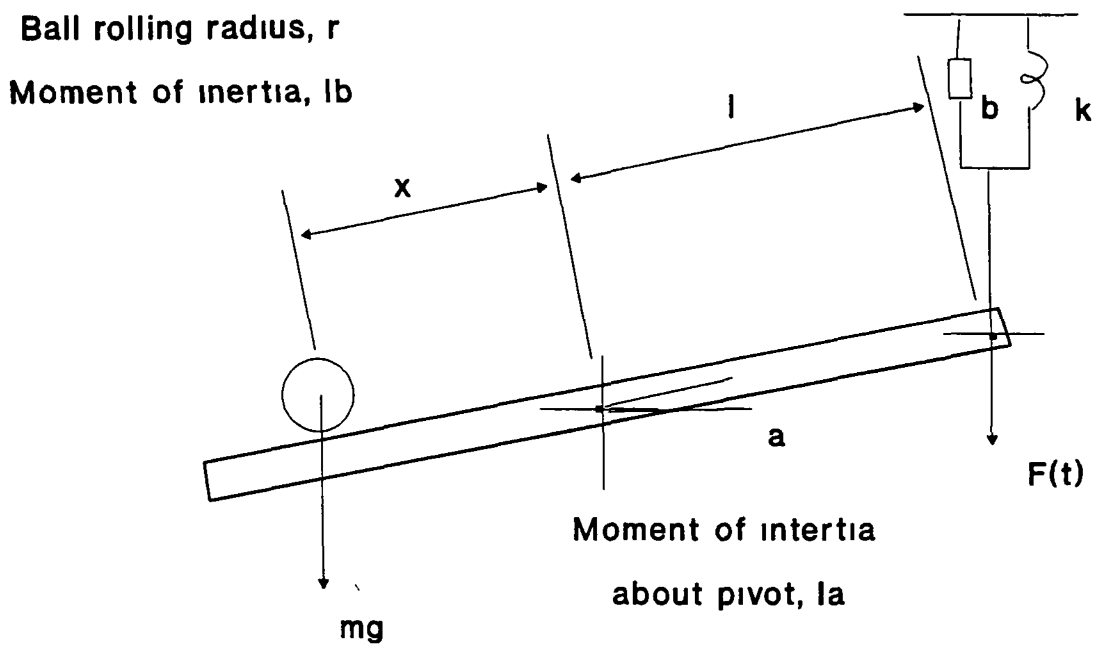


Figure 1.9 The ball beam balancing system

evaluation network, which maps the current state into an evaluation of that state. The evaluation network is used to assign credit to individual action. The error back propagation and the temporal different algorithms are used in it. The other is an action network, which maps the current state into control actions. Back propagation and reinforcement algorithms are used in it. The two networks having a similar structure are used to learn the action and evaluation function.

In nature, a neural network is a parallel processing system. This has been simulated serially on an IBM PC 80386. Both the simulation and experimental results show that the neural network control is favourable compared with the conventional control.

CHAPTER 2 SYSTEM MODELLING

As mentioned in Chapter 1, the problem studied in the present work is to balance a ball on a pivoted beam. The system is unstable in nature, in the sense that given any initial condition it will not stay in the balanced state. To stabilize the system certain feed-back control techniques are necessary.

The apparatus, shown in Figure 2.1 and 2.2, consists of a light aluminium T section approximately 1 1 m long, two insulated bridge pieces are mounted 1.15 m apart on the beam onto which two wires, 1.3 cm apart, are tautly stretched. The hybrid beam is fixed on a cradle which in turn is mounted, via a bearing block, to a rigid back plate. The beam is pivoted about the axis of rotation and is driven via a universal joint coupling by means of a vertically mounted moving coil actuator.

The angle of the beam is measured by a precision servo potentiometer mounted on axis. The position of the ball on the beam is measured by the potentiometer method in which the ball replaces the wiper blade in Figure 2.5. A small voltage is developed across the ends AB of one wire, a voltage V_x proportional to the position of the ball is measured by connecting one end C of the free wire to an operational amplifier. A particular problem is the disturbance introduced into the measurement scheme by the intermittent contact made by the ball as it rolls along the two wires.

The inputs and outputs of the apparatus are made using the instrument action box shown in Figure 2.4. This allows for an input drive voltage in the range of ± 10 volts to be applied to the actuator. The measured ball position is presented as a voltage in the range of ± 10 volts. The measured beam angle is brought out to the front panel and appears as a voltage between ± 5 volts, a null control is provided for the latter measurement in case the beam is used on a non-level surface.

2.1 Instrumentation

The moving coil actuator consists of a light electrical coil, which is suspended by a spring in the field of a permanent magnet, see Figure 2.3. The coil is constrained to move at right angles to the magnetic field, such that when a current is passed through the coil, a proportional force occurs which is parallel to the axis of the coil. Because of the spring suspension system, this force is perceived as a displacement parallel to the coil axis and proportional to the coil current. Since the apparatus is intended to be driven by voltage signals from operational amplifiers, the actuator drive circuit is configured such that a voltage applied at the 'actuator input' terminal produces a proportional current. The actuator characteristic therefore relates input voltage to actuator shaft displacement in a linear manner, with a maximum displacement being set by mechanical stops inside the actuator. Moreover, because of friction and the mass of the coil, the actuator has dynamical properties which are discussed in the modelling section. The output force, too, is presented as a voltage in the range of ≤ ±6 volts.

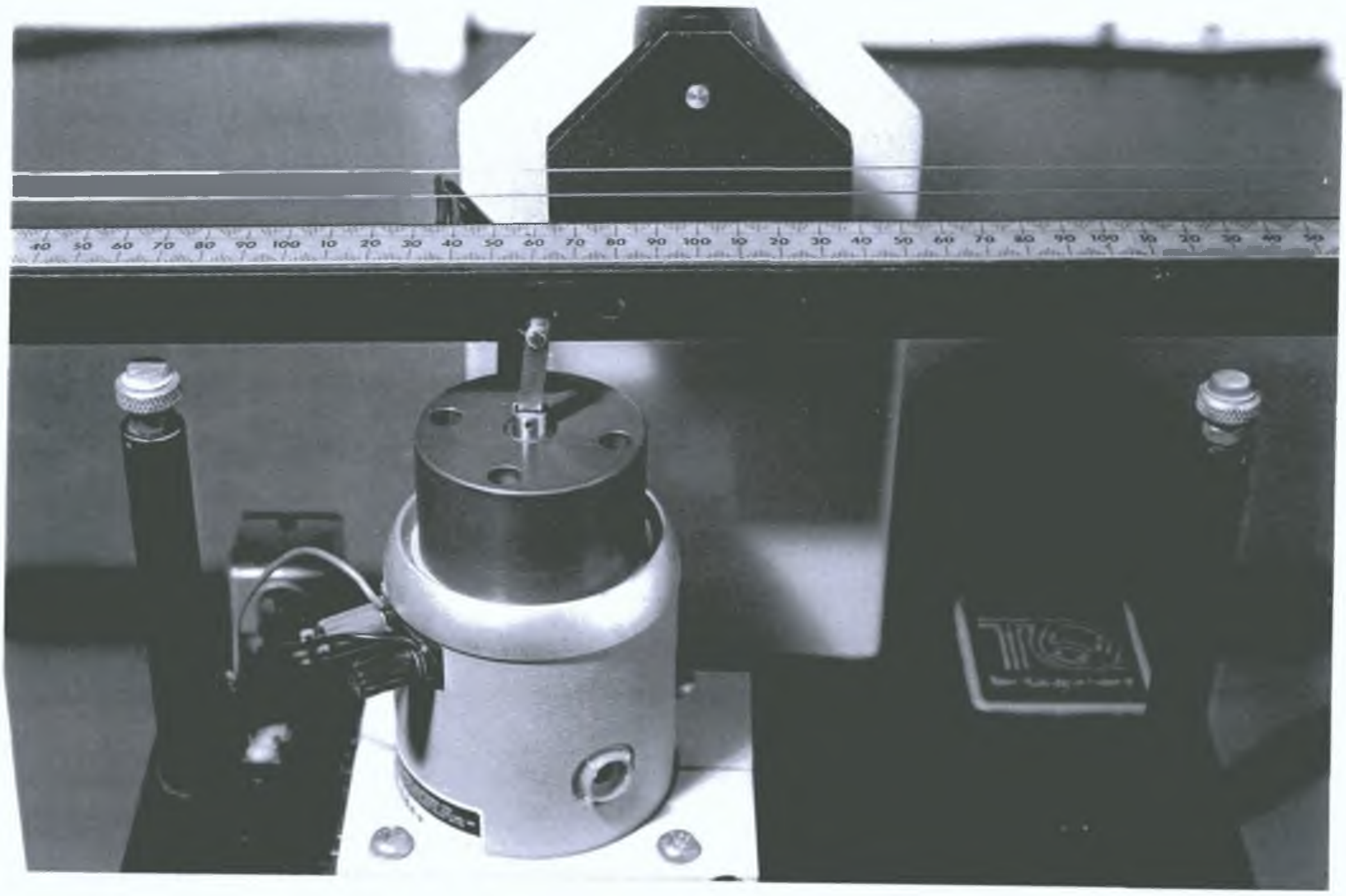


Figure 2.1 The ball-beam apparatus (the actuator)

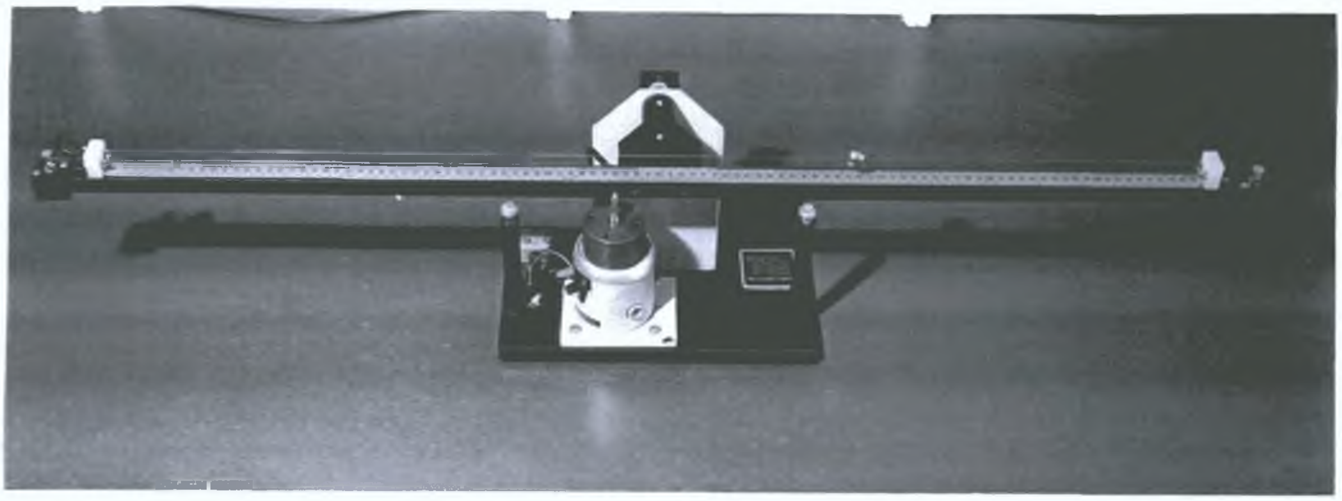


Figure 2.2 Ball-beam apparatus (full view)

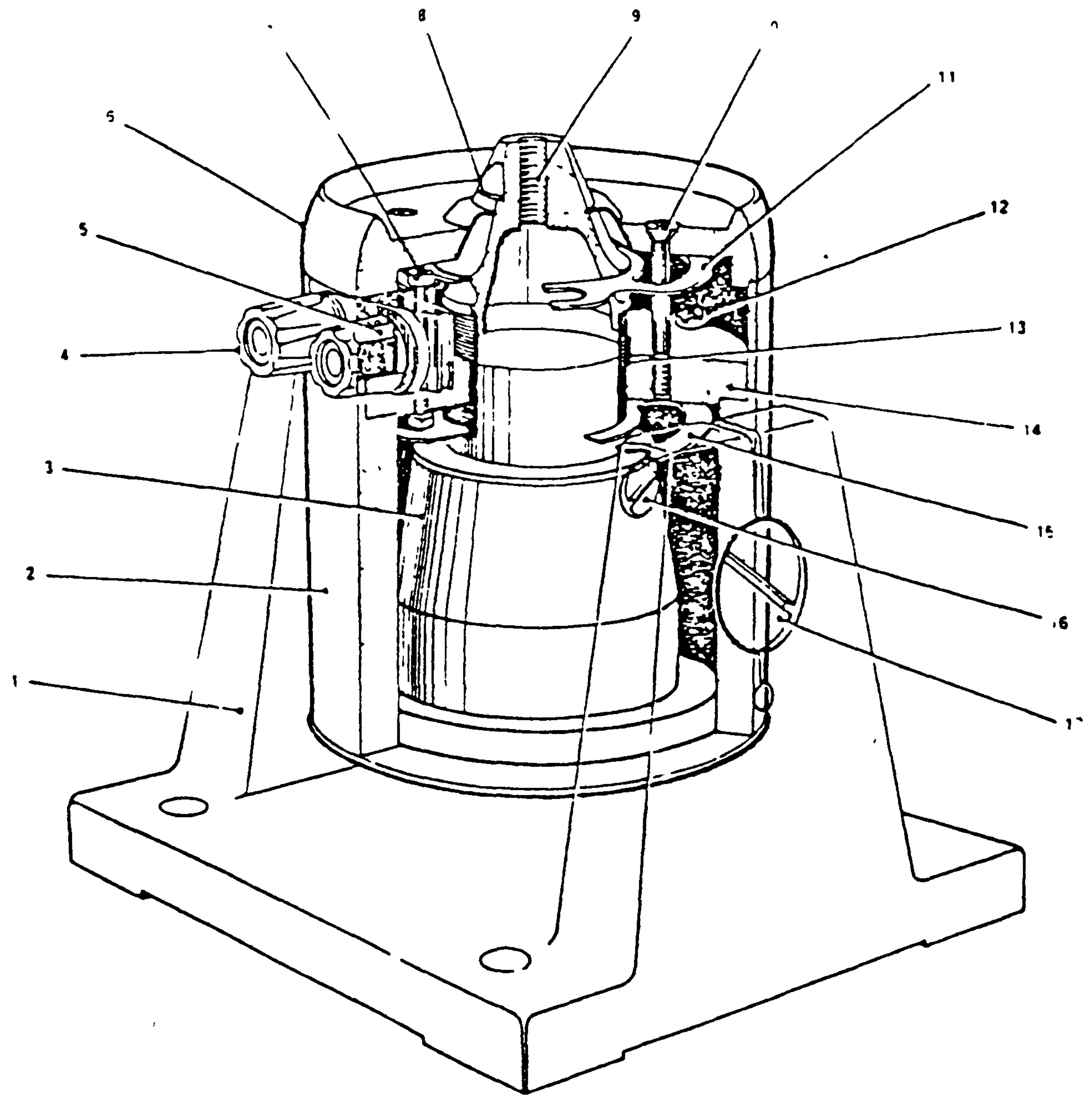


Figure 2.3

1 Trunnion	10 Top cover securing bolt (4 off)
2 Body	11 Top suspension spider (part 7)
3 Centre pole magnet	12 Moving coil suspension support plate securing bolt (3 off)
4 Terminals	13 Moving coil (part 7)
5 Air vent	14 Moving coil suspension support plate
6 Top access cover	15 Bottom suspension spider
7 Top suspension spacer and securing bolt (2 off)	16 Trunnion clamp bolt
8 Moving coil and suspension assembly	17 Support screw
9 Package mounting hole	

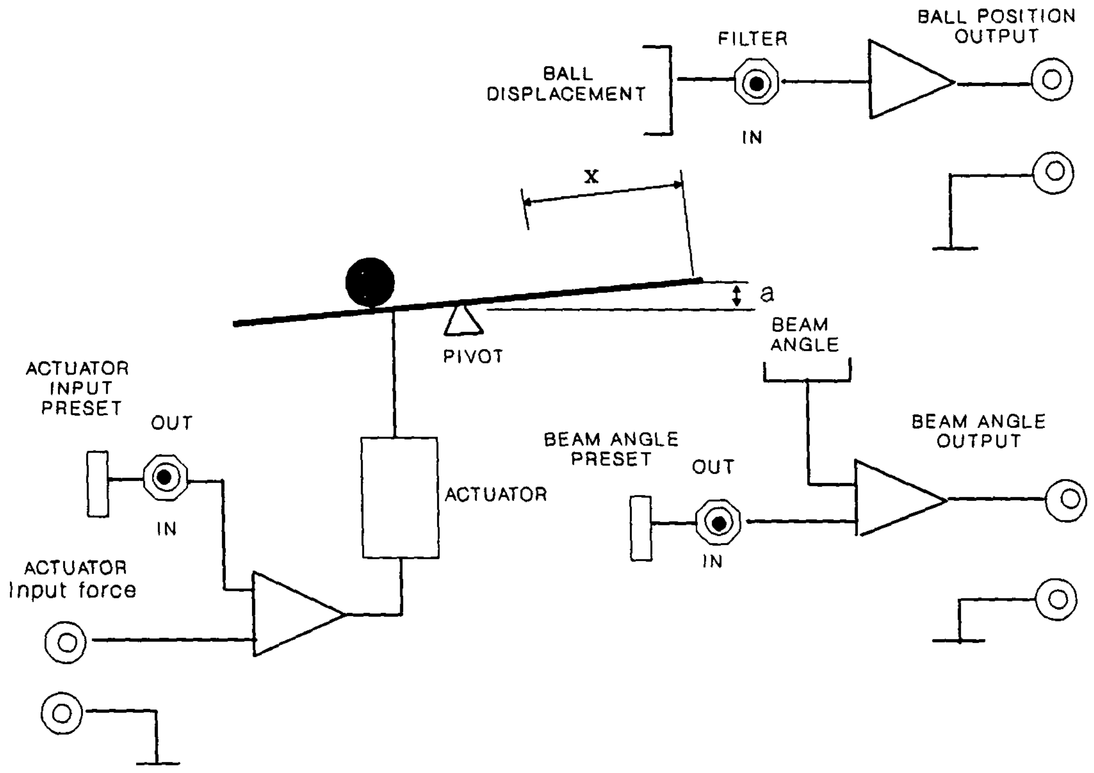


Figure 2.4 The ball and beam system configuration

2.2 The Equations of Motion of the System

To derive the equations of motion in a useable form, it is necessary to make several assumptions. The ball is assumed to move on the beam with pure rotation, disregarding the possible slip between the ball and the beam, the rolling friction between the ball and the beam is considered negligible, and the friction at pivot of the beam is represented by a single linear coefficient b, which is also referred to as the

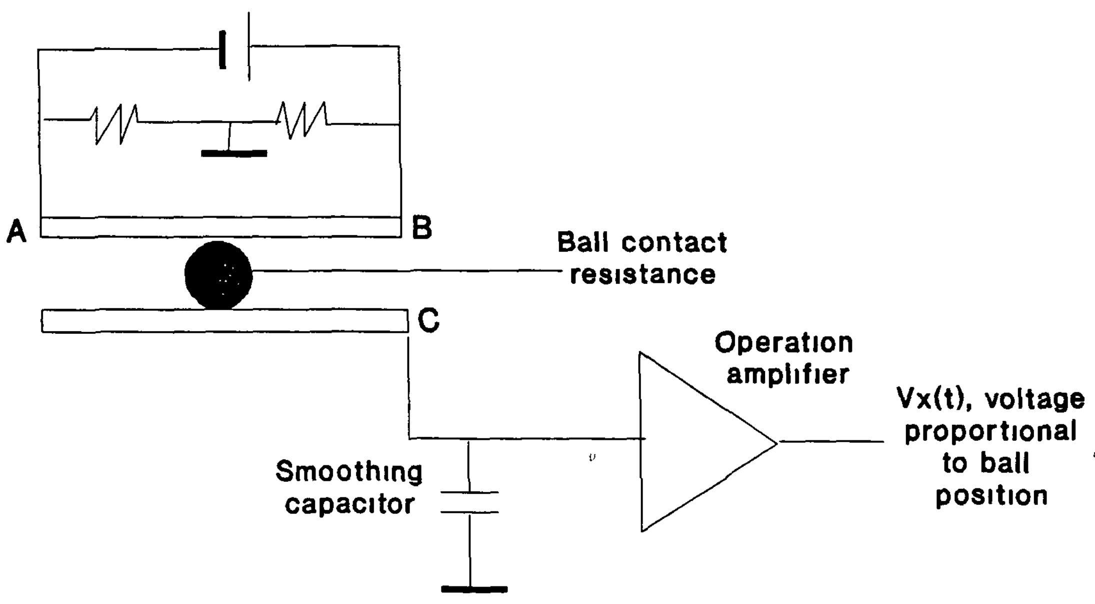


Figure 2.5 The ball position measuring system

damping factor. In addition, the stiffness of the spring mounting of the actuating coil is denoted as k, and the force exerted by the actuator is F(t).

With the above assumptions, the system equations can be conveniently derived using variational methods. Referring to Figure 2.8, the ball position x along the beam and the beam angular position α in relation to the horizontal plane are selected as the independent variables for variation. Thus, the Lagrange equation of the system can be written as

$$L_o = U^* - T \tag{2-1}$$

where L_0 is the Lagrange function, U^* is the kinetic co-energy, and T is the potential energy of the system. The system equations are obtained by applying the Lagrange theorem to Equation (2-1).

The kinetic co-energy U* of the system is the sum of translational and rotational kinetic energy of the ball, and the rotational kinetic energy of the beam. Denote the relative velocity of the ball as $\dot{x}$, and the angular velocity of the beam as $\dot{\alpha}$, then the absolute velocity of the ball can be easily found as (see Figure 2.6)

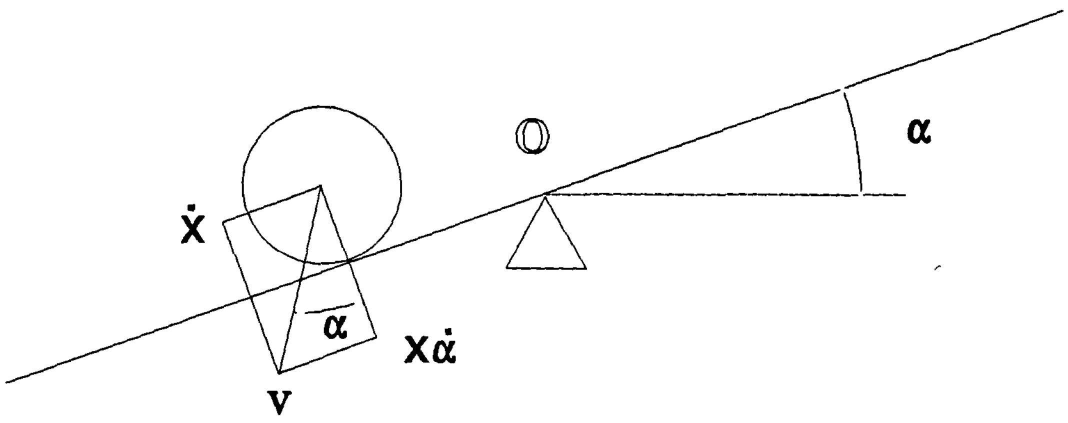


Figure 2.6 The computation of the absolute translational velocity of the ball

$$v = \sqrt{\dot{x}^2 + (x\alpha)^2} \tag{2-2}$$

Thus, the translational kinetic energy of the ball is given by

$$\frac{1}{2} m v^2 \tag{a}$$

Let ω be the absolute angular velocity of the ball and I_b the moment of inertia of the ball around the axis passing through its centre and perpendicular to the plane of the paper (see Figure 2 6), then the rotational kinetic energy of the ball can be written as

$$\frac{1}{2} I_b \omega^2 \tag{b}$$

where ω can be determined from Figure 2.8. The angular displacement of the ball θ

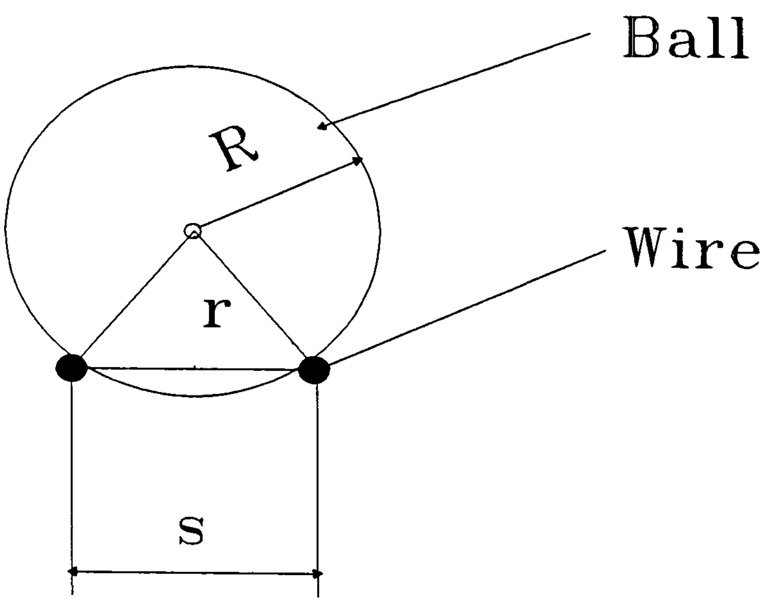


Figure 2.7 The determination of the absolute velocity of the ball

is given by Equation (2-3)

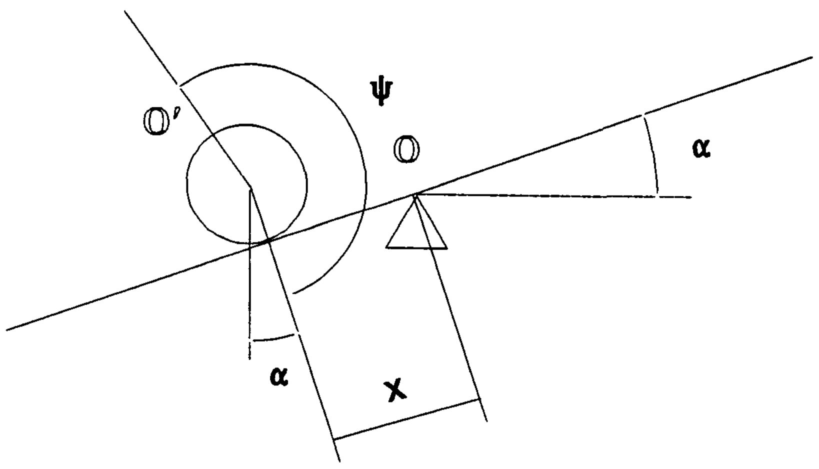


Figure 2.8 The determination of angular velocity of the beam

where ψ is the angle "rolling over" by the ball, r is the rolling radius of the ball,

$$\theta = \psi + \alpha = \frac{x}{r} + \alpha \qquad (2\text{-}3)$$

which is determined by the radius of the ball and the distance between the two parallel wires supporting the ball. From Figure 2.7, r is determined as

$$r = \sqrt{R^2 - s^2/4} = \sqrt{0.012^2 - 0.013^2/4} = 0.0101 \quad m \qquad (2\text{-}4)$$

By differentiating Equation (2-3) with respect to time, we get the angular velocity ω as

$$\omega = \frac{\dot{x}}{r} + \dot{\alpha} \qquad (2\text{-}5)$$

The kinetic energy of the beam can be simply written as

$$\frac{1}{2} I_a \, \dot{\alpha}^2 \qquad (c)$$

where I_a is the moment of inertia of the beam around its centre.

Thus, the co-energy of the system is given by the sum of terms in (a), (b) and (c), or

$$U^* = \frac{1}{2} m \, v^2 + \frac{1}{2} I_b \, \omega^2 + \frac{1}{2} I_a \, \dot{\alpha}^2 \qquad (2\text{-}6)$$

The potential energy of the system is associated with the energy stored in the spring. Assuming small angular excursions, the spring will be a linear one. Thus the potential energy T is given by

$$T = \frac{1}{2} k_1 \, (l \, \alpha)^2 \qquad (2\text{-}7)$$

where l is the distance from the spring to the centre of the beam.

By combining Equation (2-6) and (2-7) and substituting v, ω, we obtain the system Lagrangian as

$$L_o = \frac{1}{2}[m(\dot{x}^2 + \dot{\alpha}^2 x^2) + I_b (\frac{\dot{x}}{r} + \dot{\alpha})^2 + I_a \alpha^2 - k_1 (l \alpha)^2] \qquad (2\text{-}8)$$

The damping effect of the system can be considered by introducing the system co-content J, which is given by

$$J = \frac{1}{2} b (l \dot{\alpha})^2 \qquad (2\text{-}9)$$

By applying Lagrangian theorem to Equations (2-8) and (2-9), we have

$$\frac{d}{dt}[\frac{\partial L_o}{\partial \dot{x}}] - \frac{\partial L_o}{\partial x} + \frac{\partial J}{\partial \dot{x}} = m g \sin\alpha$$

$$\frac{d}{dt}[\frac{\partial L_o}{\partial \dot{\alpha}}] - \frac{\partial L_o}{\partial \alpha} + \frac{\partial J}{\partial \dot{\alpha}} = \cos\alpha (m g x - F(t) l) \qquad (2\text{-}10)$$

$$\frac{\partial L_o}{\partial \dot{x}} = \dot{m} x + I_b (\frac{\dot{x}}{r} + \dot{\alpha}) \frac{1}{r} \qquad (2\text{-}11)$$

Carrying out the differentiation m Equation (2-10) and (2-11), and re-ranging the terms, we obtain the following equations of motion,

$$(m + \frac{I_b}{r^2}) \ddot{x} + \frac{I_b}{r} \ddot{\alpha} - m x \dot{\alpha}^2 = m g \sin\alpha$$

$$\frac{I_b}{r} \ddot{x} + (m x^2 + I_a + +I_b) \ddot{\alpha} + 2 m x \dot{x} \dot{\alpha} \qquad (2\text{-}12)$$

$$+ k l^2 \alpha + b l^2 \dot{\alpha} + \cos\alpha (m g x - lF(t))$$

The equations can be reduced to a more usable from by introducing further assumptions which are valid for the present problem. The moment of inertia of the ball I_a and its mass m are small and can be regarded as having little effect on the behaviour of the beam. Furthermore, α and $\dot{\alpha}$ can also be considered to have negligible effect in the equations.

Using the above approximations and remembering that for small α, $\sin\alpha \approx \alpha$, the equations of motion can be simplified as

$$(m + \frac{I_b}{r^2}) \ddot{x} = m g \alpha \tag{2-13}$$

$$I_a \ddot{\alpha} + b l^2 \dot{\alpha} + k_1 l^2 \alpha = -lF(t) \tag{2-14}$$

2.3 Calculation of Moment of Inertia for the Ball and the Beam

The moment of inertia of the ball I_b can be simply determined as

$$I_b = \frac{2}{5} m R^2 = 0.00162 \quad g\text{-}m^2 \tag{2-15}$$

The moment of inertia I_a for the beam is comprised of the contributions from the attached fixtures on the beam and the beam itself, see Figure 2.9. The calculation of moment for each part is briefly listed below. The total value is found to be

$$I_a = I_{M1} + I_{M2} + I_{M3} + I_{M4} + I_{M5} \tag{2-16}$$

(1) For aluminium,

$\rho_m = 2.7$ g/cm^3 = 2.7×10^{-3} g/mm^3.

(2) The contribution of mass M_1,M_2, M_3, M_4. (See Figure 2.9)

$M_1 = \rho_m V_1 = 0.05052$ kg. $\quad$ $M_2 = \rho_m V_2 = 0.2804$ kg.

$M_3 = \rho_m V_3 = 0\ 25$ kg. $\quad$ $M_4 = \rho_m V_4 = 0.062$ kg.

(3) From point 1 and point 2 in the Figure 2.10 , we have:

Point 1 = (-75.5/2, -14.49) =(-37.75, -14.49).

Point 2 = (-16.1/2, 29.98) = (-8.05, 29.98).

$$\frac{y-29.98}{x+8.05} = \frac{29.98+14.49}{-8.05+37.75} = 1.514. \qquad (2\text{-}17)$$

$$x = 0.67\ (y-29.98) - 8.05$$

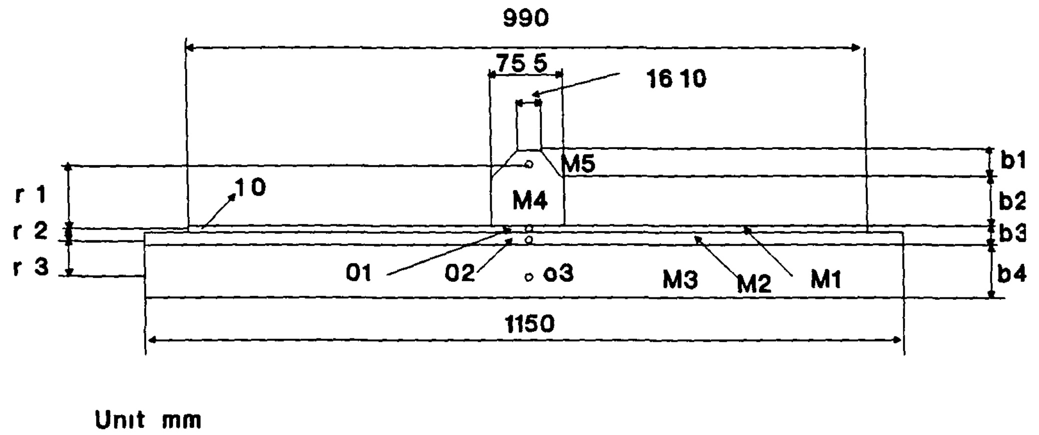


Figure 2.9 The geometrical dimension of the beam. b_1=29.98, b_2=43.87, b_3=4.5, b_4=21.48.

$$I_{m4,5} = I_{m4} + I_{m5} M_4\, r^2$$

$$= 54.856 - 185144\ \rho_m t + M_4(43.87/2 + 14.99)^2$$

$$= 21.25 \text{ kg-mm}^2$$

$$= 0\ 02125 \text{ g-m}^2$$

(4) The moment of inertia about pivot I_a:

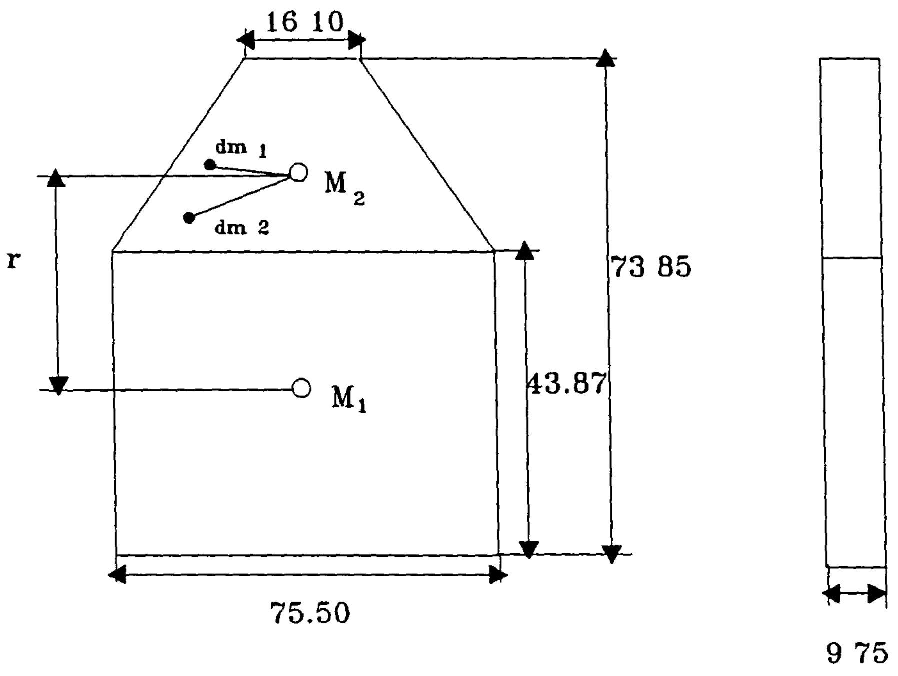


Figure 2.10 The geometry of element M4 and M5, see Figure 2.9

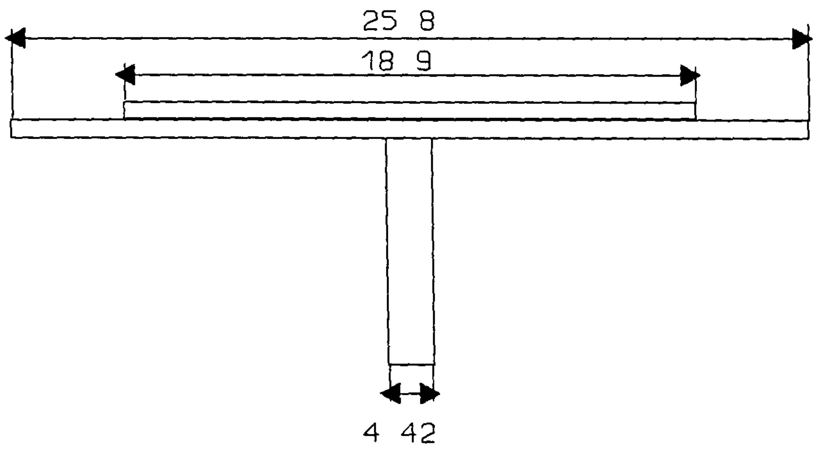


Figure 2.11 The geometry of M1 and M2 shown in Figure 2.4

$$I_{M_1} = I_{01} + r_1^2\, M_1 = 4.1779 \quad g\text{-}m^2 \tag{2-19}$$

$$I_{M_2} = I_{02} + r_2^2\, M_2 = 31.3 \quad g\text{-}m^2 \tag{2-20}$$

$$I_{M_3} = I_{03} + r_3^2\, M_3 = 27.6 \quad g\text{-}m^2 \tag{2-21}$$

$$I_{M_{4,5}} = 0.02125 \quad g\text{-}m^2 \tag{2-22}$$

$$I_a = I_{M_1} + I_{M_2} + I_{M_3} + I_{M_4} + I_{M_5} = 63.04 \quad g\text{-}m^2 \tag{2-23}$$

2.4 Other System Parameters

$$d_{Ia5} = d_m\, r^2 = r^2\, \rho\, dv = r^2 \rho\, t\, ds$$

$$= \rho\, t\, (x^2+y^2)\, dx\, dy.$$

$$I_{a5} = \rho\, t \iint (x^2+y^2)\, dx\, dy \tag{2-18}$$

$$= \rho\, t \int_{-14\,49}^{29\,98} \int_{+0\,67\,(y-29\,98)}^{-0\,67\,(y-29.98)} \times$$

$$(x^2+y^2)\, dx\, dy = -185144\, \rho\, t$$

Same of the other expenmentally determmed system parameters are lısted below,

(1) Ball radıus R=0.012 m.

(2) Ball mass m=28.11 g.

(3) Ball rollıng radıus r=0.0101 m.

(4) Beam length L_o=1.15 m.

(5) Stıffness of the sprıng k_1=3.26 N/m.

(6) Damp factor b=0.6 N.s/m.

CHAPTER 3 SIMULATION OF CONVENTIONAL AND NEURAL NETWORK CONTROLLERS

The conventional and neural network control algorithms are developed for the ball-beam system and simulation has been performed on a personal computer. In the simulation work, the plant, or the system has been modelled using the set of equations of motion established earlier in equation (2-12). The simulation results have been presented for both the conventional and the neural network control methods.

3.1 State Feedback Control

In the conventional control theory, the state of a system at any time can be described by a set of state variables For the present problem, the state variables are the ball position x_1, velocity x_2 and the beam angular position x_3 and velocity x_4. The goal of the control problem is to decide what the input should be so that the state will behave in a favourable fashion. The most general state space description of a linear system is given by

$$\dot{x}(t) = A(t)\ x(t) + B(t)\ u(t) \tag{3-1}$$

$$y(t) = C(t)\ x(t) + D(t)\ u(t) \tag{3-2}$$

where $x(t)$ is the state vector, $y(t)$ is the output from the system, u(t) is the input to the system and $\mathbf{A}(t)$, $\mathbf{B}(t)$, $\mathbf{C}(t)$ and $\mathbf{D}(t)$ are matrices.

3.1.1 The continuous-time and discrete-time open-loop models

3.1.1.1 System equations

The equations of motion of the system have been established in Section 2.2 earlier. Using the state variables we have

$$(m + \frac{I_b}{r^2}) \dot{x}_2 = m\, g\, x_3 \tag{3-3}$$

$$(I_a + I_b) \dot{x}_4 + b\, l^2\, x_4 + k\, l^2 x_3 = -l\, F(t) \tag{3-4}$$

or

$$\begin{aligned} \dot{x}_2 &= c_1 x_3 \\ \dot{x}_4 &= c_3 x_3 + c_2 x_4 + c_4 F(t) \end{aligned} \tag{3-5}$$

where

$$c_1 = mg/(M+I_b/r^2) \qquad c_2 = -bl^2/(I_b+I_a)$$

$$c_3 = -kl^2/(I_b+I_a) \qquad c_4 = -l/(I_b+I_a)$$

The numerical value of c_1, c_2, c_3 and c_4 are respectively 4.503, -0.003, -0.0171 and -0.009.

3.1.1.2 System model

Re-write Equations (3-1) and (3-2), and insert the identities $x_1 = x_1$ and $x_3 = x_3$, we obtain the continuous-time system model as

$$\begin{aligned} \dot{x}(kt) &= \mathbf{A} x(kt) + \mathbf{B}\, F(kt) \\ y(kt) &= \mathbf{C} x(kt) \end{aligned} \tag{3-6}$$

where x and y are column vectors,

$$x = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} \qquad y = \begin{Bmatrix} x_1 \\ x_3 \end{Bmatrix} \tag{3-7}$$

and **A**, **B** and **C** are

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 5.643 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -0.0171 & -0.003 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.009 \end{bmatrix} \tag{3-8}$$

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

The open-loop discrete-time model can be obtained as

$$\begin{aligned} x[(k+1)\tau] &= \phi(\tau)\, x(k\tau) + \Gamma(\tau)\, F(k\tau) \\ y(k\tau) &= C\, x(k\tau) \end{aligned} \tag{3-9}$$

where $\phi(\tau)$ is a matrix, and $\Gamma(\tau)$ is a column vector. The matrix $\phi(\tau)$ can be determined by first considering the continuous model, then setting time t to the sample time τ. Thus

$$\phi(t) = \mathcal{L}^{-1}\{\, [sI-A]^{-1} \,\} \tag{3-10}$$

Denoting $\Phi(s) = [sI-A]^{-1}$, we have

$$\phi(s) = \begin{bmatrix} \frac{C}{s^2(s-B)} - \frac{1}{s}, & \frac{1}{s^2} - \frac{C}{s^3(s-B)}, & \frac{A}{s^3}, & \frac{A}{s^3(s-B)} \\ 0, & \frac{C}{s^2(s-B)} - \frac{1}{s}, & \frac{A}{s^2}, & \frac{A}{s^2(s-B)} \\ 0, & 0, & \frac{1}{s}, & \frac{1}{s(s-B)} \\ 0, & 0, & \frac{C}{s(s-B)}, & \frac{1}{s-B} \end{bmatrix} \tag{3-11}$$

In the inverse transformation (see Appendix A) let $t=\tau=0.02$ seconds, we obtain the numerical values of transition function as

$$\phi(\tau) = \begin{bmatrix} 0.805 & 0.0201 & 0.001129 & 0 \\ 0 & 0.805 & 0.11286 & -0.312 \\ 0 & 0 & 1 & -0.333 \\ 0 & 0 & 0.0067 & 0.999 \end{bmatrix} \tag{3-12}$$

The matrix $\Gamma(\tau)$ can be obtained as

$$\Gamma(\tau) = \int_0^\tau e^{At}\, B\, dt = \begin{bmatrix} 0 \\ -0.000562 \\ -0.000598 \\ -0.0199 \end{bmatrix} \tag{3-13}$$

Finally, we obtain the open-loop discrete-time model as

$$x[(k+1)\tau] = \begin{bmatrix} 0.805 & 0.0201 & 0.001129 & 0 \\ 0 & 0.805 & 0.11286 & -0.312 \\ 0 & 0 & 1 & -0.333 \\ 0 & 0 & 0.0067 & 0.999 \end{bmatrix} x(k\tau) + \begin{bmatrix} 0 \\ -0.000562 \\ -0.000598 \\ -0.0199 \end{bmatrix} F(k\tau) \qquad (3\text{-}14)$$

$$y(k\tau) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(k\tau)$$

3.1.2 The closed-loop discrete-time model

To implement the state feedback controller, the closed-loop discrete-time model

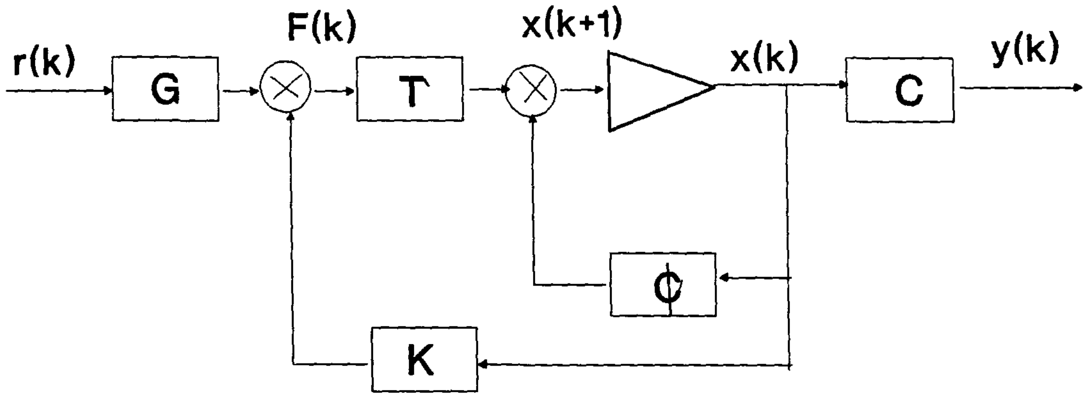


Figure 3.1 System closed-loop state model

of the plant is needed. Referred to Figure 3.1, we can write the system model as

$$\begin{aligned} x[(k+1)\tau] &= (\phi+\Gamma\, K)\; x(k\tau) + \Gamma\; G\; F(k\tau) \\ y(k\tau) &= C\; x(k\tau) \end{aligned} \qquad (3\text{-}15)$$

where ϕ and Γ are as shown in Equations , respectively, $\underline{K}$ is feedback vector and G is the gain.

The vector K and the scalar G can be determined using the Bass-Gura method. The characteristic polynomial of the system can be written as

$$a(z) = \det[zI-\phi] = z^4 + a_1 z^3 + a_2 z^2 + a_3 z + a_4 \tag{3-16}$$

where a_1, a_2, a_3 and a_4 are factors to be determined. By substituting Equation (3-12) into the above we get

$$a_1 = -3.700 \qquad a_2 = 5.049$$

$$a_3 = -2.998 \qquad a_4 = 0.647$$

According to the Bass-Gura formula the feedback vector K can be determined by

$$K = [\alpha - a]\ \{a_-'\}^{-1}\ \bar{c}^{-1} \tag{3-17}$$

where z is the close-loop eigenvector of the following (see Appendix B)

$$z = [\ z_1,\ z_2,\ z_3,\ z_4\ ] = [-0.9 \pm j2.85,\ -0.075,\ -1.412\ ]$$

and

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \tag{3-18}$$

a_- $[1\ a_1\ a_2\ a_3\ ]'$ is the lower triangular Toeplitz with the first column as

$$[1\ a_1\ a_2\ a_3]'$$

and

$$c = [\Gamma, \phi\Gamma, \phi^2\Gamma, \phi^3\Gamma] =$$

$$\begin{bmatrix} 0 & 0.0000192 & -0.0000555 & -0.000129 \\ -0.000484 & -0.00185 & -0.00397 & -0.00526 \\ -0.0084 & -0.00156 & -0.0132 & -0.0109 \\ -0.3885 & -0.326 & -0.266 & -0.210 \end{bmatrix} \quad (3\text{-}19)$$

Thus, the feedback vector is obtained as,

$$K = [\ 11.06,\ -0.098,\ 8.078,\ -0.098] \quad (3\text{-}20)$$

The gain G can be determined by the fact that

$$H(z) = C(z)-(\phi + \tau k)\ BG = 1 \quad (3\text{-}21)$$

which gives

$$G = -0.248 \quad (3\text{-}22)$$

The close-loop discrete-time system model as

$$x[(k+1)\tau] = \begin{bmatrix} 0.8951, & -2.514, & -0.000177, & 0 \\ 0.129, & 0.651, & 0.000144 & 0 \\ -0.0759, & 0, & 0.843, & 0 \\ -4.634, & 0.0685, & -3.396, & 0.896 \end{bmatrix} x(k\tau) + \begin{bmatrix} 0.226 \\ 0.0284 \\ 0.000124 \\ 0.104 \end{bmatrix} F(k\tau) \quad (3\text{-}23)$$

$$y(k\tau) = C\ x(k\tau)$$

3.1.3 Results

The computer implementation of the conventional simulation are described in this section. The input of the system is a force acting on the beam. The output of the system includes the position and velocity of the ball and the angular position and velocity of the beam. The Equations (3-23) were used to program the controller in the simulation. The sample time is τ=0.02 seconds.

The system has been simulated for different combinations of initial state variables. The position of the ball along the beam ranges from -0.5 to 0.5 metres and the angular position of the beam varies between -20° and 20°, or between -0.3491 and 0.3491 in radius. The results from the simulation are presented in Figures (3-2) through (3-7). From these results it is evident that in simulation the ball-beam system can be balanced under any initial conditions of interest.

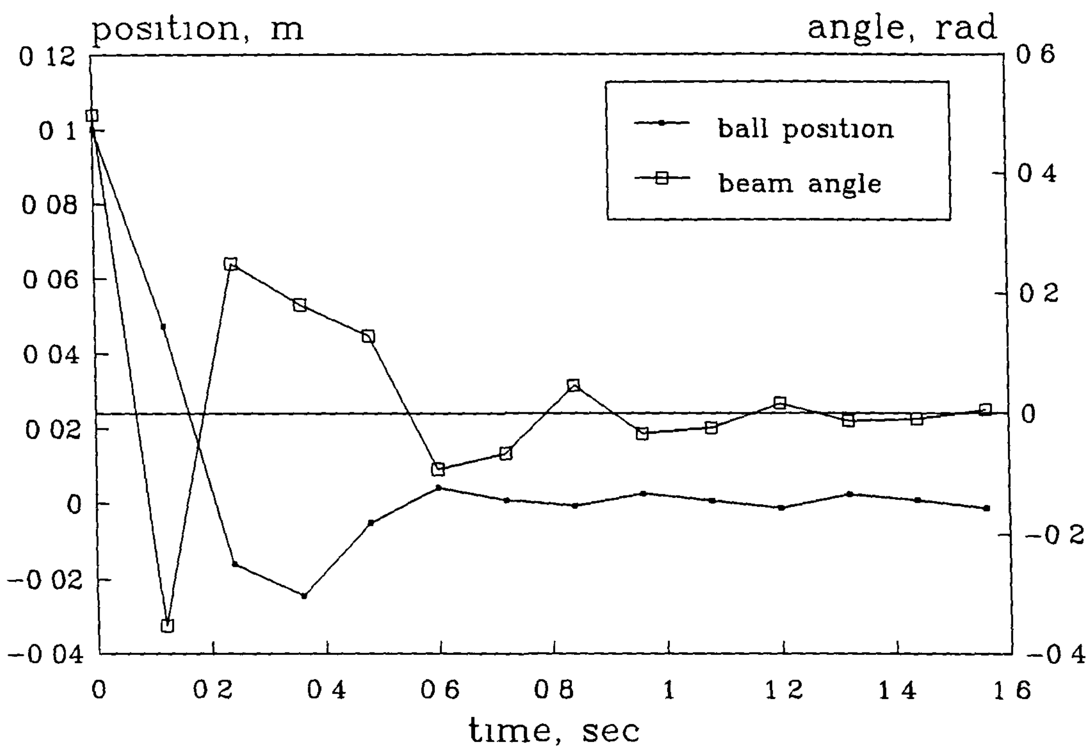


Figure 3.2 The Simulation using conventional method. Initial position: x=0.1 m, α=0.5 rad.

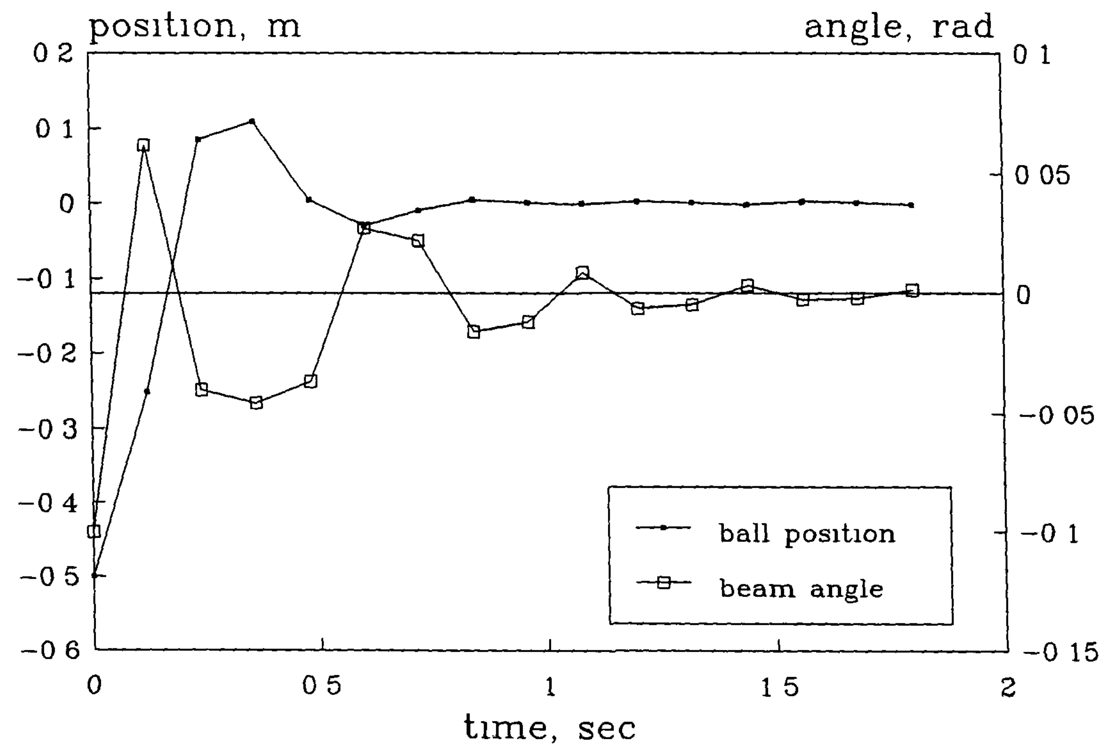


Figure 3.3 Simulation using conventional method. Initial position: x=-0.5 m, α=-0.1 rad.

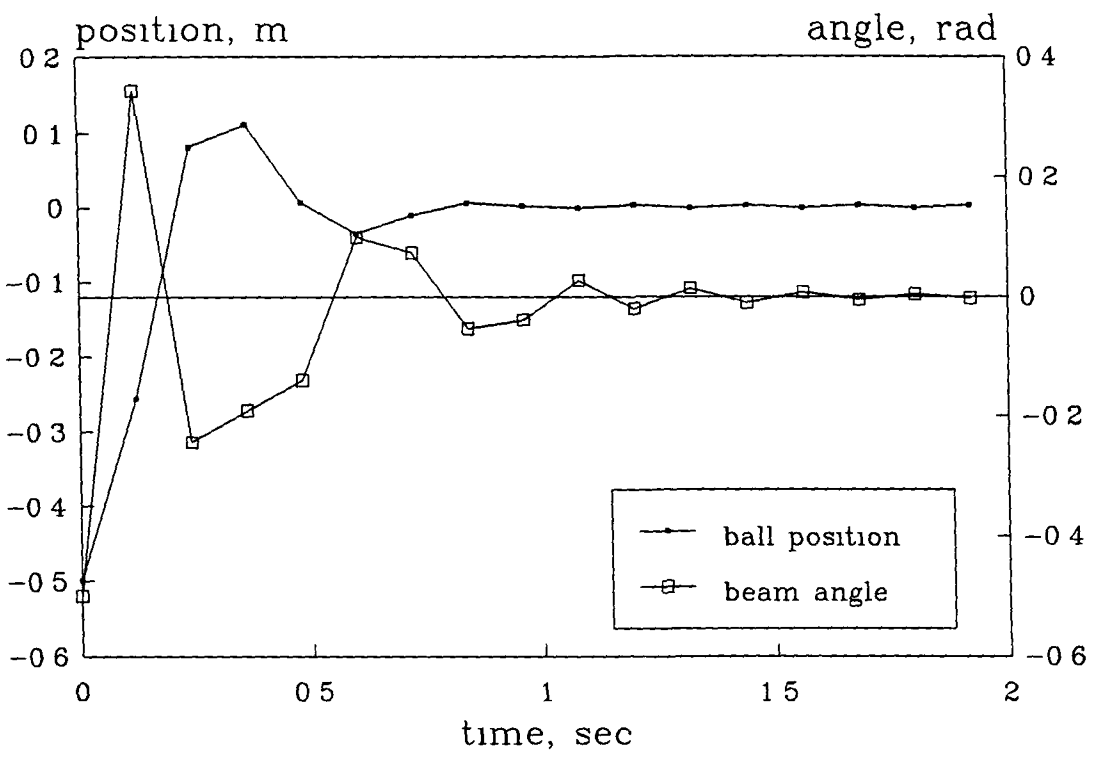


Figure 3.4 Simulation using conventional method. Initial position: x=-0 5 m, α=-0.5 rad.

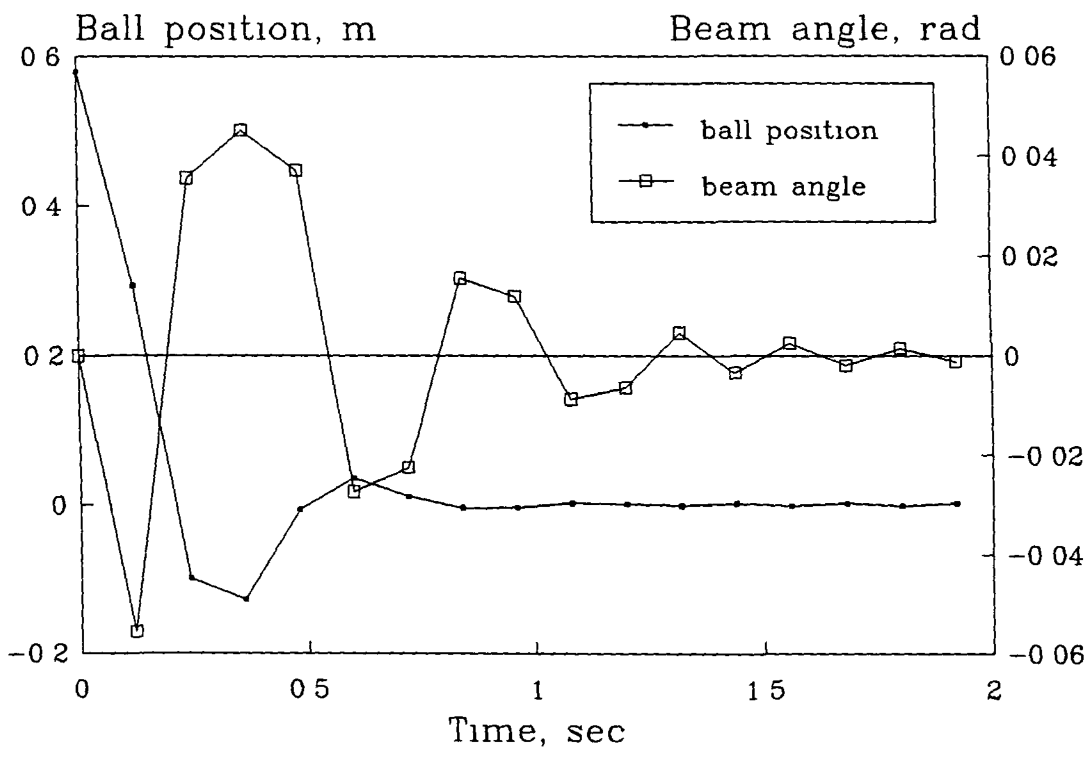


Figure 3.5 Simulation using conventional method. Initial position: x=0 m, α=0.5 rad.

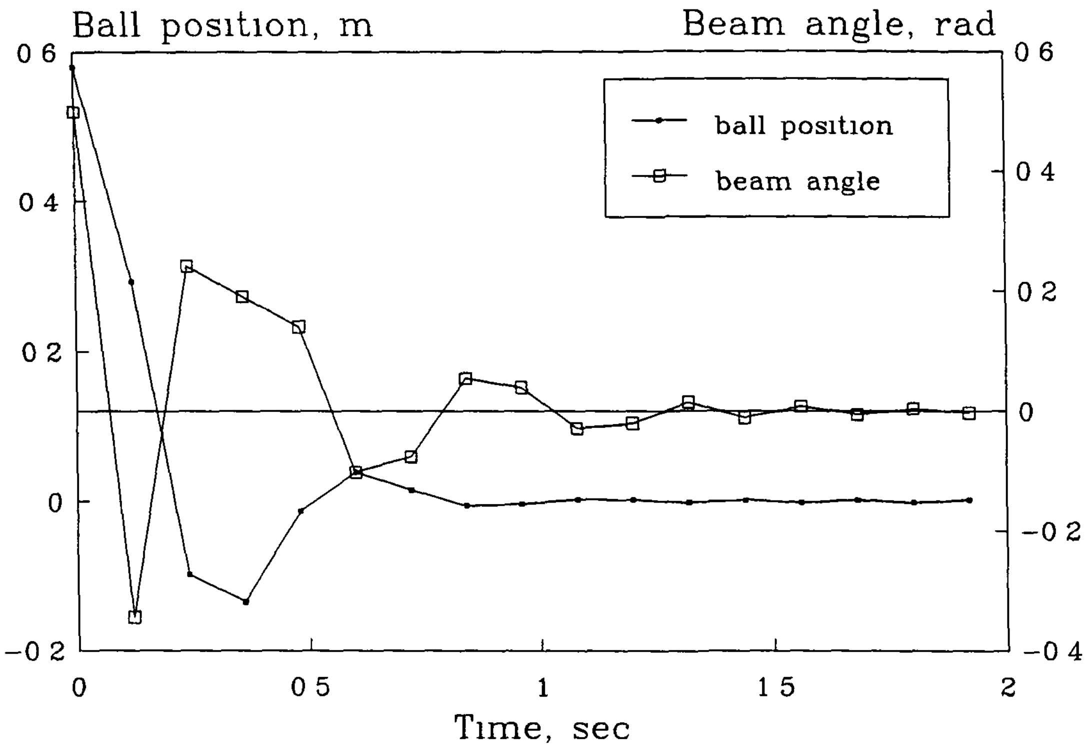


Figure 3.6 Simulation using conventional method. Initial position: x=0.58 m, α=0.5 rad.

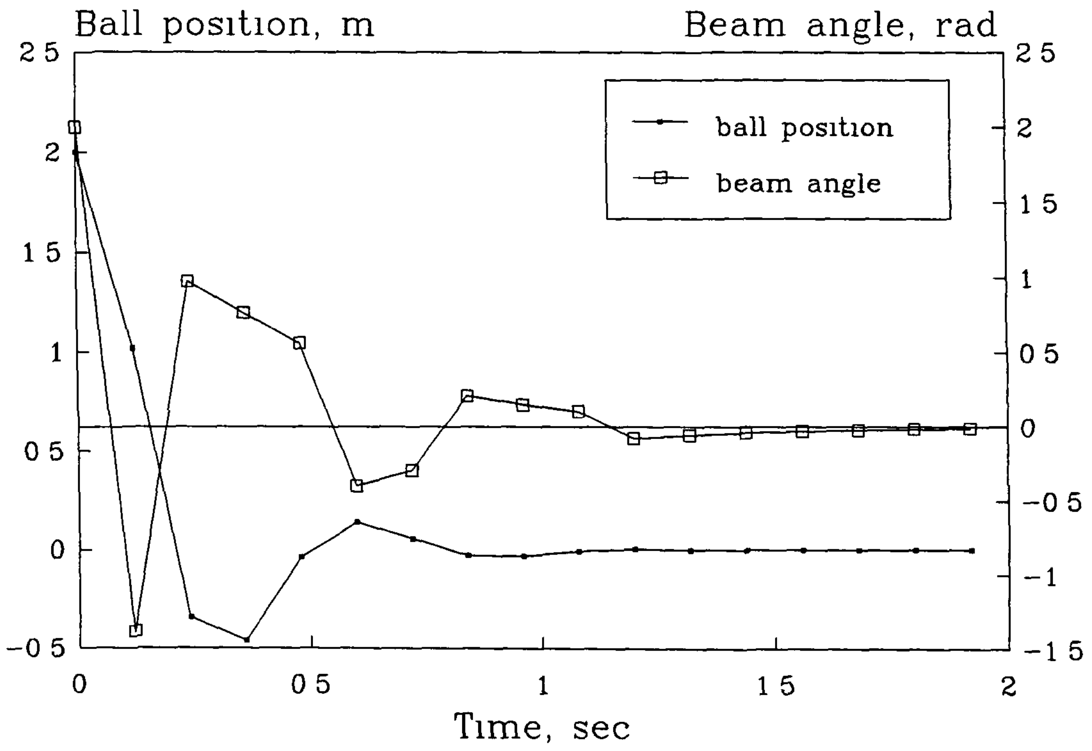


Figure 3.7 Simulation using conventional method. Initial position: x=2 m, α=2 rad.

3.2 Neural Network Feedback Control

The present task is to balance the ball-beam system using the neural network technique. As shown in Figure 1.9 a ball moves on two parallel wires spanned on a beam. The beam itself is pivoted at the centre to a mount. The movement of both the ball and the beam is constrained in the vertical plane. Thus, the state of this system is given by the position and velocity of the ball and the angular position and velocity of the beam.

Starting from any initial state, the ball tends to move away from the initial position thus causing the system un-balanced. Hence the system is inherently a unstable one.

To balance the system, the only control actions available are to exert forces of fixed amplitude on the beam is such a way that the beam can be kept at the horizontal position and the ball at a predetermined position. The way the force is applied is dependent on the control method used. In this section, the neural network technique will be used to evaluate the system performance and applied the action force accordingly.

The neural network system used, as shown in Figure 3.8, consists of two networks termed the evaluation network and the action network. The evaluation network learns an evaluation function of the system state, so that it can predict the future action the system needs to take in order to avoid certain states. The action network generates the system behaviour. It decides which action to apply for a given state of the ball-beam system.

When an undesired state of the system is reached, it is called a failure In the current problem, a failure is defined as the beam's falling past a certain angle relative to the horizontal plane, or the ball's running into the bounds of its track. The purpose of the controller is to apply a sequence of forces so that the failure is avoided as much as possible, or the balance is maintained as long as possible

Three different learning algorithms have been used in the neural networks. In the hidden layers of both the evaluation and the action network, the error back-propagation algorithm has been used. The temporal-difference algorithm was used in the output layer of the evaluation network. Reinforcement learning algorithm was used in the output layer of the action network.

3.2.1 The simulation of the plant

As stated in Section 1.2, in the control system using neural networks the system dynamics is not required. This is certainly a great advantage in real time control, but in simulation it is necessary to simulate the system dynamics using a set of differential equations In the present study, the ball-beam system is modelled using the equations of motion, Equation (2-12). In the simulation, the equations of motion are solved using the Euler's method. By denoting x_1 as the horizontal position of the ball in relation to the centre of the beam, x_2 as the velocity of the ball, x_3 as the angular position of the beam and x_4 the angular velocity of the beam, the Euler's method gives the solution to the equations of motion as (see Appendix C)

$$x_1[(k+1)\tau]=x_1[k\tau]+\tau\ x_2[k\tau].$$

$$x_2[(k+1)\tau] = x_2[k\tau]+\tau\frac{(g\ x_3[k\tau]+x_2[k\tau]\ x_4[k\tau]^2)}{(1+\frac{I_b}{m\ r^2})}.$$

$$x_3[(k+1)\tau]=x_3[k\tau]+\tau\ x_4[k\tau].$$

$$x_4[(k+1)\tau]=x_4[k\tau]+\tau\frac{(m\ g\ x_0[k\tau]-F(k\tau)\ l-\frac{I_b}{r\zeta}-2\ m\ x_1[k\tau]x_2[k\tau]x_4[k\tau]-k\ l^2x_2[k\tau}{(m\ x_1[k\tau]^2+I_b+I_a)}$$

(3-24)

where $\zeta=\frac{(gx_3+x_2x_4^2)}{1+Ib/mr^2}$

The objective of this study is to avoid failure. The beam was balanced within a very narrow angle about the horizontal position, and the ball was balanced at any position on the beam. This objective can be formalized by defining a failure signal.

$$r'[t] = \begin{cases} -1, & if\ |\theta(t)|>6^o\Lambda|x(t)|>0.5m \\ 0, & otherwise. \end{cases} \qquad (3\text{-}25)$$

Other limits imposed on system parameters are

An additional input, x_5, with a constant value of 0.5 is provided.

$$|x_1| < 0.01\ m$$

$$|x_2| < 0.05\ m/s$$

$$|x_3| < 6°$$

$$|x_4| < 1.0°/s$$

3.2.2 Structure of the neural networks and the learning algorithms

The architecture of the neural networks control system is shown in Figure 3-8.

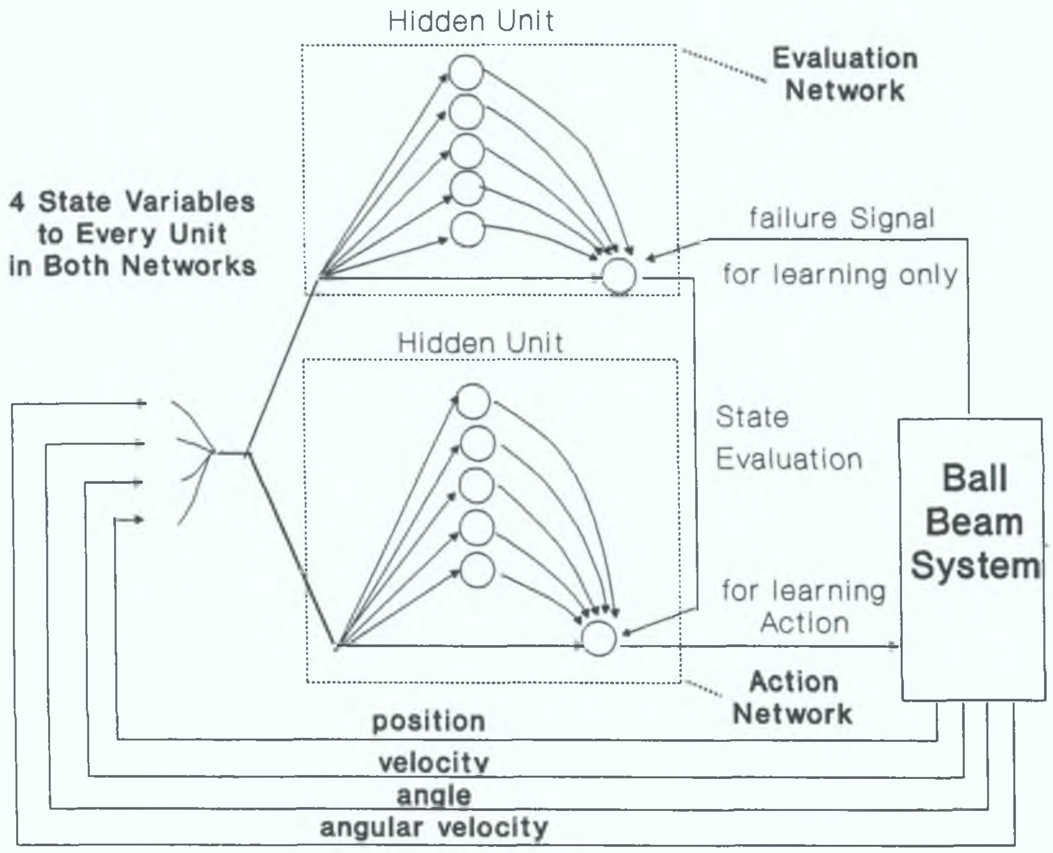


Figure 3.8 Two-layer neural networks used in the control system.

The system is composed of two networks, one evaluation network and one action network. Each of the two networks has two layers of nodes, a hidden layer and an output layer. There are 5 nodes in the hidden layers and one node in the output layers.

The node connectivities of the action and the evaluation networks are shown in Figure 3-9 and Figure 3-10 respectively. In Figure 3-9, $\boldsymbol{A}$ is the matrix of connection weights for the hidden layer with components a_{ij}, and $\boldsymbol{B}$ and $\boldsymbol{C}$ are the vectors of connection weights with components b_{ij}, and c_{ij}, for the input and output layers respectively. $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{F}$ in Figure 3-10 have similar meanings as $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ in Figure 3-9, and their components are respectively d_{ij}, e_i and f_i.

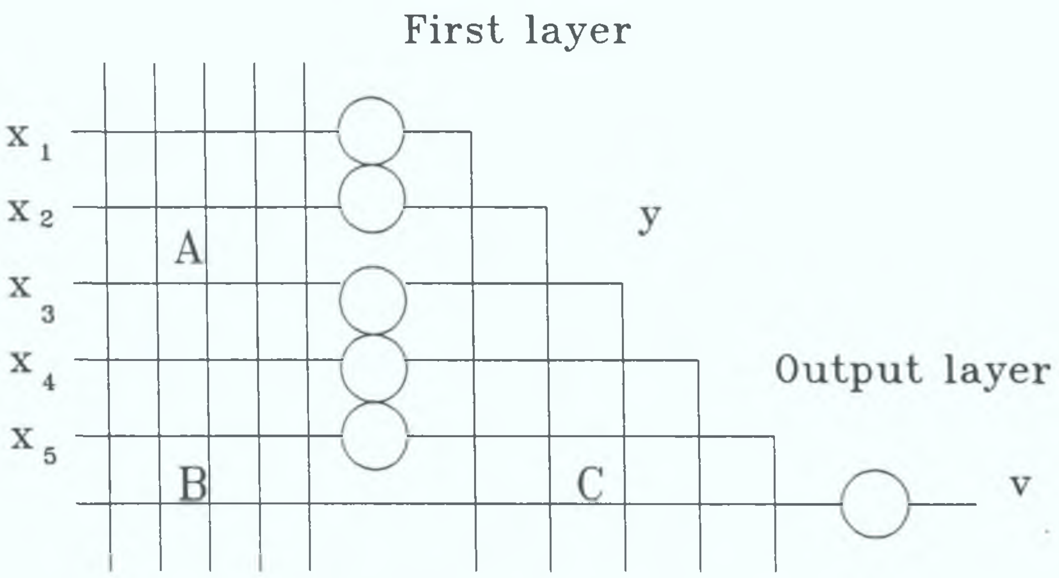


Figure 3.9 Evaluation network connectivity

Three types of learning algorithms have been used in the neural network system. The reinforcement learning method has been used in the output layer of the action network, and the temporal difference learning algorithm was used in the output layer of the evaluation network to adjust the connection weights. In the hidden layers of both the action net and the evaluation net, the back-propagation algorithm was used. In the output layer of the action network, the difference between the actual value r_1 and

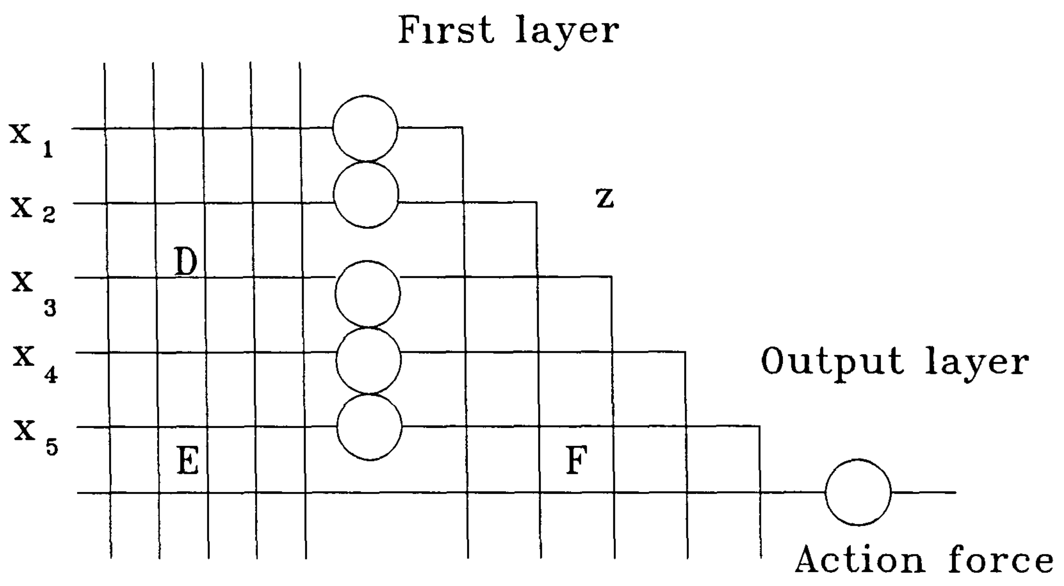


Figure 3.10 Action network connectivity

expected value r_1 of the action are fed-back to adjust the connection weights of the action network. The weights in the output layer of the evaluation network was adjusted by feeding the difference between the successive values of v, which is the vector of outputs as given by Equation (3-29).

The parallel algorithm of the neural network has been simulated seriously using an IBM PC 80386. The learning algorithms of the neural networks are outlined below. Here, k+1 refers to the current time step and k the previous step. τ is the sample time.

1. The outputs y_i from the first layer in the evaluation network are calculated according to the error back-propagation algorithm,

$$y_i[t,t] = g(\Sigma a_{ij}[t] x_j[t])$$
$$y_i[t,t+1] = g(\Sigma a_{ij}[t] x_j[t+1]) \qquad (3\text{-}26)$$

2. The output v_i from the output layer in the evaluation network is determined by the temporal-difference learning algorithm as

$$v[t,t] = \Sigma b_i[t] x_i[t] + \Sigma c_i[t] y_i[t,t]$$
$$v[t,t+1] = \Sigma b_i[t] x_i[t] + \Sigma c_i[t] y_i[t,t+1] \qquad (3\text{-}27)$$

3. The failure signal from the evaluation network is given by Equation (3-28)

$$r_1[t+1] = \begin{cases} 0; & \textit{if state at time } t+1 \textit{ is a start state;} \\ r[t+1] - v[t,t]; & \textit{if state at time } t+1 \textit{ is a failure state;} \\ r[t+1] + \gamma\, v[t,t+1] - v[t,t]; & \textit{otherwise;} \end{cases} \qquad (3\text{-}28)$$

4. The modification on the connection weights in the evaluation network is performed according to Equation (3-29).

$$b_i[t+1] = b_i[t] + \beta\ r_1[t+1]\ \xi[t]$$
$$c_i[t+1] = c_i[t] + \beta\ r_1[t+1]\ y_i[t,t] \qquad (3\text{-}29)$$
$$a_{ij}[t+1] = a_{ij}[t] + \beta_h\ r_1[t+1]\ y_i[t,t]\ (1 - y_i[t,t])\ sgn(ci[t]) x_j[t]$$

where sgn is the sign function defined by

$$\mathrm{sgn}(c_i[t]) = \begin{cases} +1, & c_i[t] \geq 0 \\ -1, & c_i[t] < 0 \end{cases} \tag{3-30}$$

5. The first layer output in the action network is given by

$$z_i[t]=g(\Sigma d_{ij}[t]x_i[t].$$
$$p[t]=g(\Sigma e_i[t]x_i[t]+\Sigma f_i[t]z_i[t]) \tag{3-31}$$

$$q[t] = \begin{cases} 1, & \textit{with probability } p[t]; \\ 0, & \textit{with probability } 1-p[t]; \end{cases} \tag{4-1}$$

6. And the action force is determined by

$$F[t] = \begin{cases} +3.5, & \textit{if } q[t]=1 \\ -3.5, & \textit{if } q[t]=0 \end{cases} \tag{4-2}$$

7. The connection weights in the action network are adjusted according to the following,

$$e_i[t+1]=e_i[t]+\rho\ r_1[t+1](q[t]-p[t])z_i[t].$$
$$f_i[t+1]=e_i[t]+\rho\ r_1[t+1](q[t]-p[t])z_i[t]. \tag{4-3}$$
$$d_{ij}[t+1]=d_{ij}[t]+\rho_h\ r_1[t+1]\ z_i[t]\ (1-z_i[t])\ sgn(f_i[t])\ (q[t]-p[t])x_i[t]$$

The Values of the parameters used in the above equations are as following:

$\beta=0.045$ $\beta_h=0.045$ $\rho_e=0.2$

$\rho=0.95$ $\rho_h=0.2$ $\gamma=0.9$

3.2.3 Flow chart of the control program

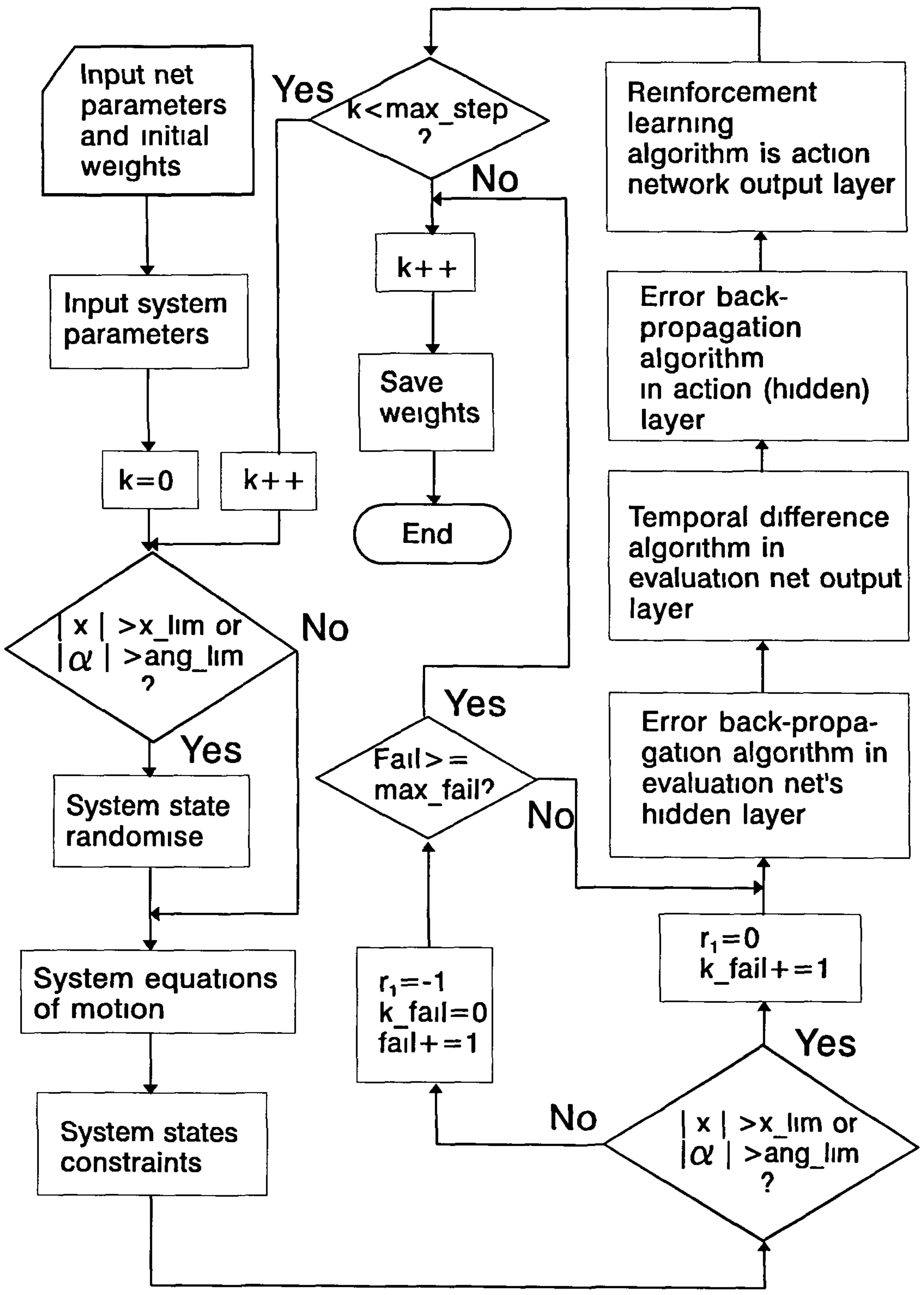

Input net parameters and initial weights
Input system parameters
k=0
k++
|x| >x_lim or |α| >ang_lim ?
No
Yes
System state randomise
System equations of motion
System states constraints
Yes
k<max_step ?
No
k++
Save weights
End
Reinforcement learning algorithm is action network output layer
Error back-propagation algorithm in action (hidden) layer
Temporal difference algorithm in evaluation net output layer
Error back-propagation algorithm in evaluation net's hidden layer
Yes
Fail>= max_fail?
No
r1=-1 k_fail=0 fail+=1
r1=0 k_fail+=1
Yes
No
|x| >x_lim or |α| >ang_lim ?

Figure 3.11 The flow chart for programming the neural network simulation.

3.2.4 Results

The simulation work has been performed on a personal computer equipped with an Intel 80386 processor, by training the neural networks with different initial conditions. At the start of each training run, the ball-beam system was "initialized", which included the assignment of random values to state variables, within the corresponding limits, and the assignment of small random values to all the connection weights During each training run, the weights were adjusted according to the learning rules and the system performance. These adjusted weights were consequently stored and became the initial weights for the subsequent training run.

In the first training run, the system starts from the initialized state. In all the simulation it took 30 to 40 seconds for the network to learn to balance the system in the first training run. The curve of the number of failure versus the number of time steps before failure is shown in Figure 3.12 for the first run. The network failed about 1500 times before it could balance the system.

The time needed to balance the system decreases sharply in the subsequent two training runs, and approached a constant of 0.3 to 0.4 seconds in and after the fourth training run. While it took 20-25 seconds to balance the ball and beam in the second run, it only took 0.6-0.7 seconds in the third run. The simulation results from first to four runs are presented in Figures 3.12 to 3.38 inclusive.

It has been noted that all the weights tends to approach constant values after a number of training runs. The set of weights after the fifth run is presented here.

$$D = \begin{bmatrix} -0.00255 & 0.03012 & -0.08009 & -0.66811 & 0.12363 \\ -0.07688 & 0.02718 & 0.03043 & -0.64686 & 0.20023 \\ 0.07473 & -0.0199 & -0.00338 & -0.58907 & 0.03698 \\ -0.08559 & -0.00247 & -0.02312 & -0.52872 & 0.05066 \\ 0.02908 & -0.00247 & -0.00112 & -0.54343 & 0.10600 \end{bmatrix} \qquad (4\text{-}4)$$

$$A = \begin{bmatrix} -0.63912 & -0.06556 & 0.08299 & 0.05223 & -0.96392 \\ -0.09169 & 0.06309 & 0.52880 & 0.91910 & -0.01464 \\ 0.06791 & 0.06218 & 0.02095 & 0.01991 & 0.04400 \\ -0.0187 & 0.03425 & -0.00178 & -0.03110 & -0.04711 \\ -0.0153 & -0.06726 & 0.02909 & -0.09939 & -0.04620 \end{bmatrix} \qquad (4\text{-}5)$$

$$E = \{-0.05484 \quad -0.08159 \quad -0.05249 \quad 0.33470 \quad 0.09490\} \qquad (4\text{-}6)$$

$$F = \{-2.7037 \quad -2.65897 \quad -1.85765 \quad 10.1612 \quad -2.49669\} \qquad (4\text{-}7)$$

$$B = \{0.02070 \quad 0.06100 \quad -0.017200 \quad -0.01803 \quad -0.01898\} \qquad (4\text{-}8)$$

$$C = \{-3.98423 \quad 0.06100 \quad -0.01720 \quad -0.01803 \quad -0.02898\} \qquad (4\text{-}9)$$

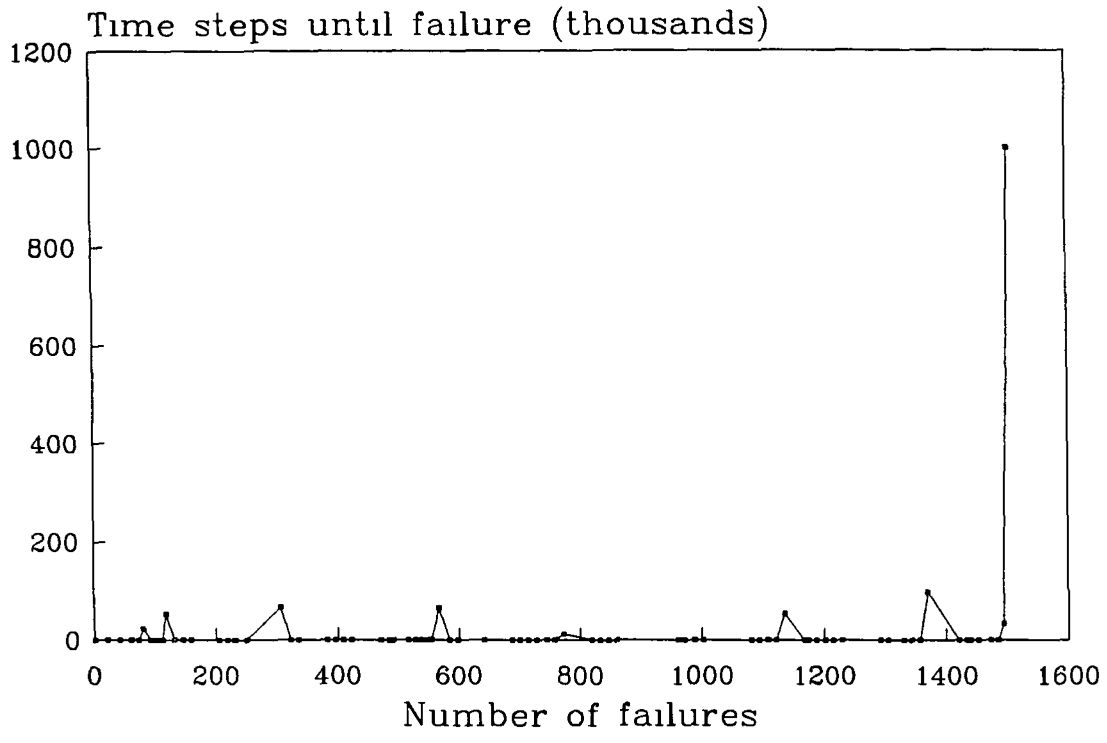


Figure 3.12 The first traınıng run.

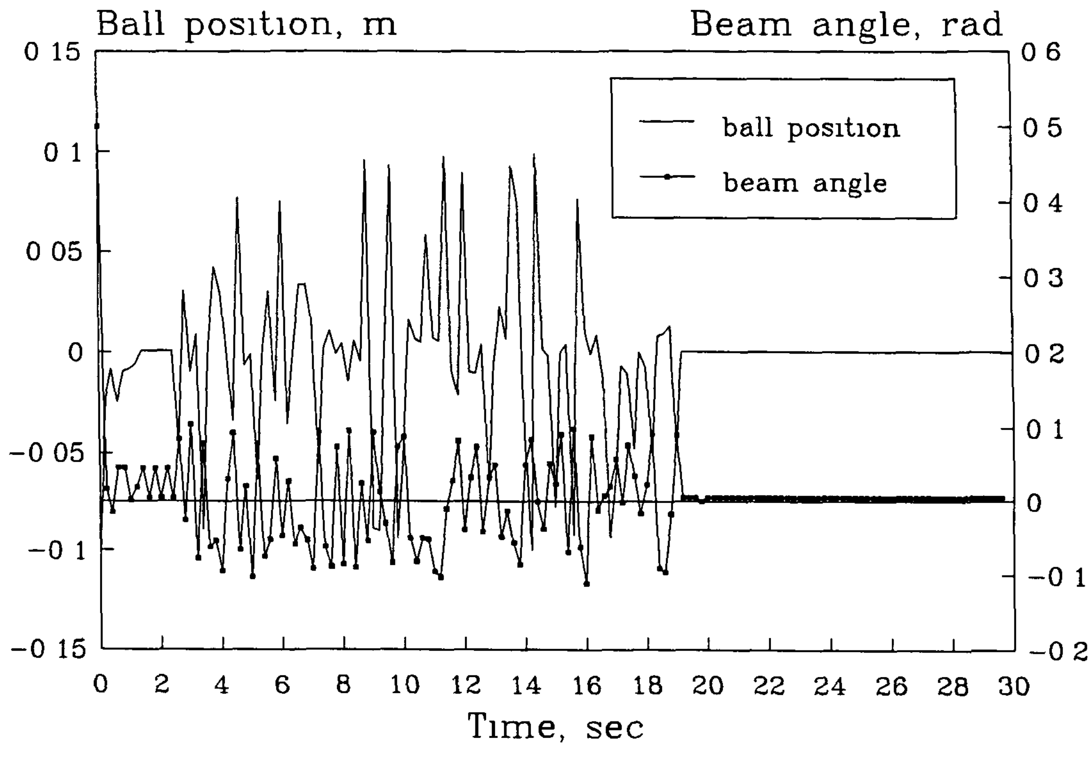


Figure 3.13 The second traınıng run. Inıtıal posıtıon: x=-0.1 m, α=0.5 rad.

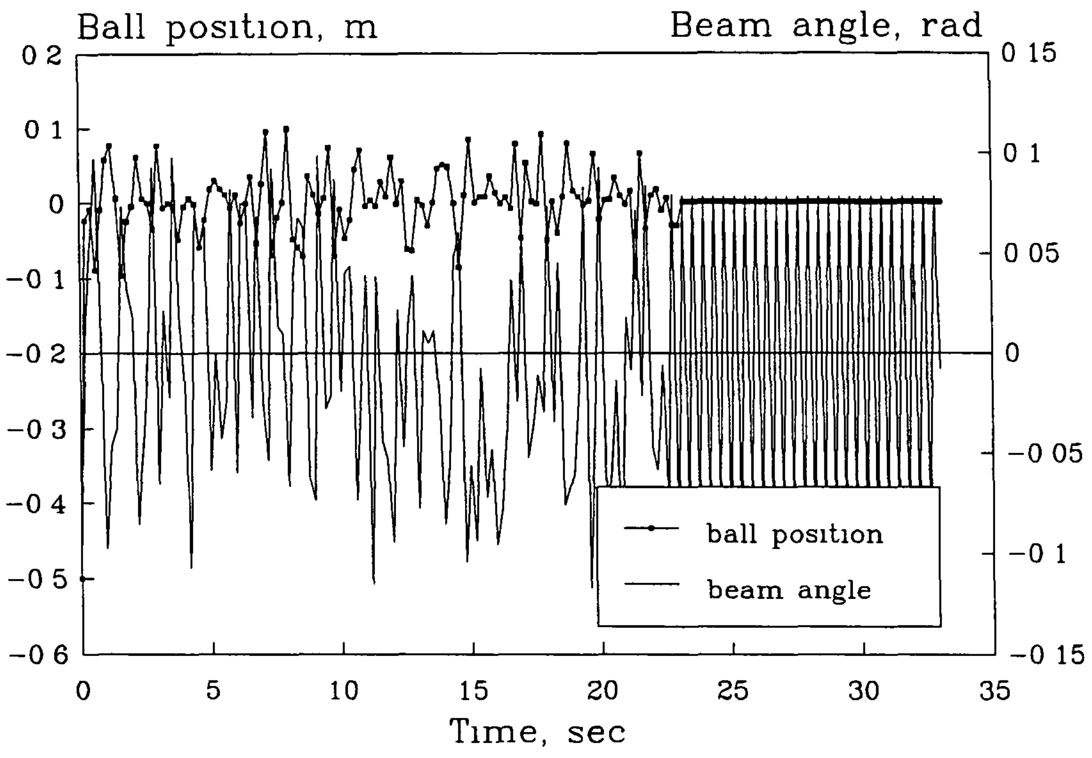


Figure 3.14 The second training run. Initial position x=-0.5 m, α=-0.1 rad.

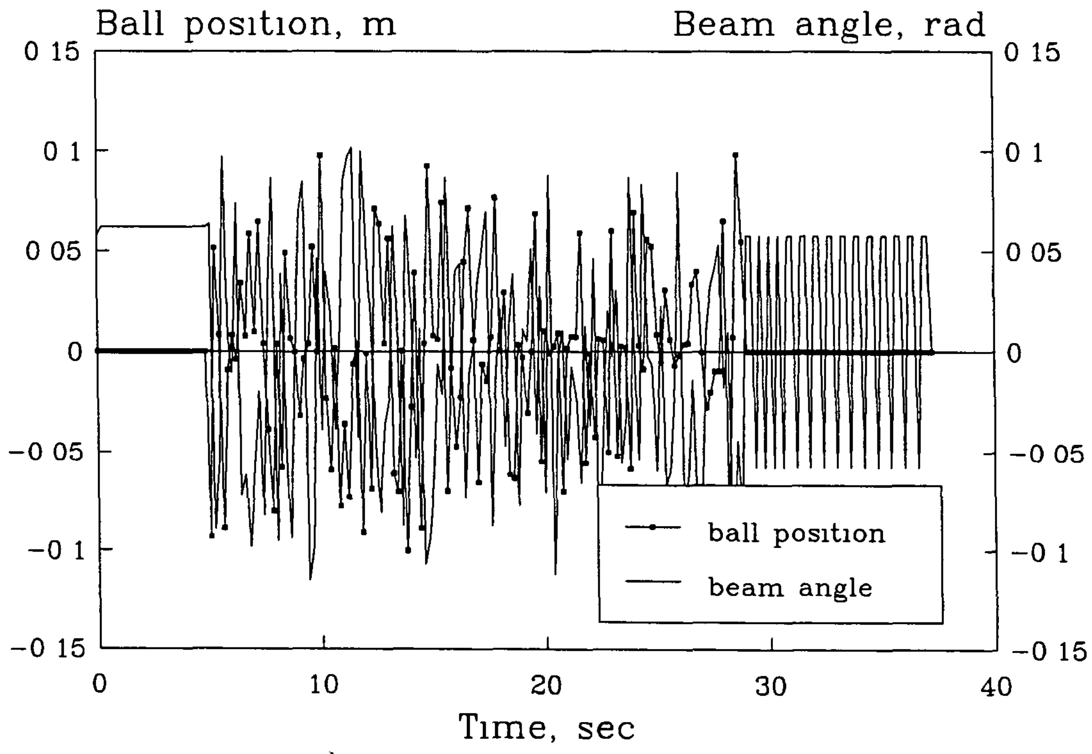


Figure 3.15 The second training run. Initial position: x=0 m, α=0.058 rad.

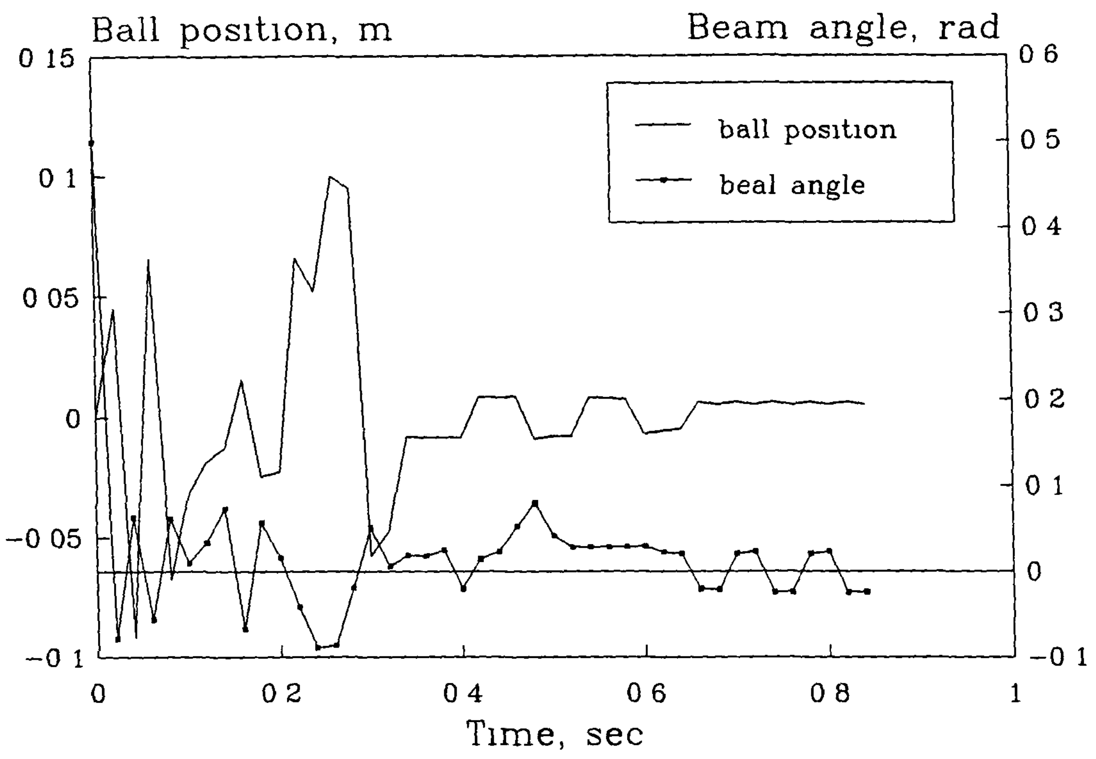


Figure 3.16 The third training run. Initial position: x=0 1 m, α=-0.1 rad.

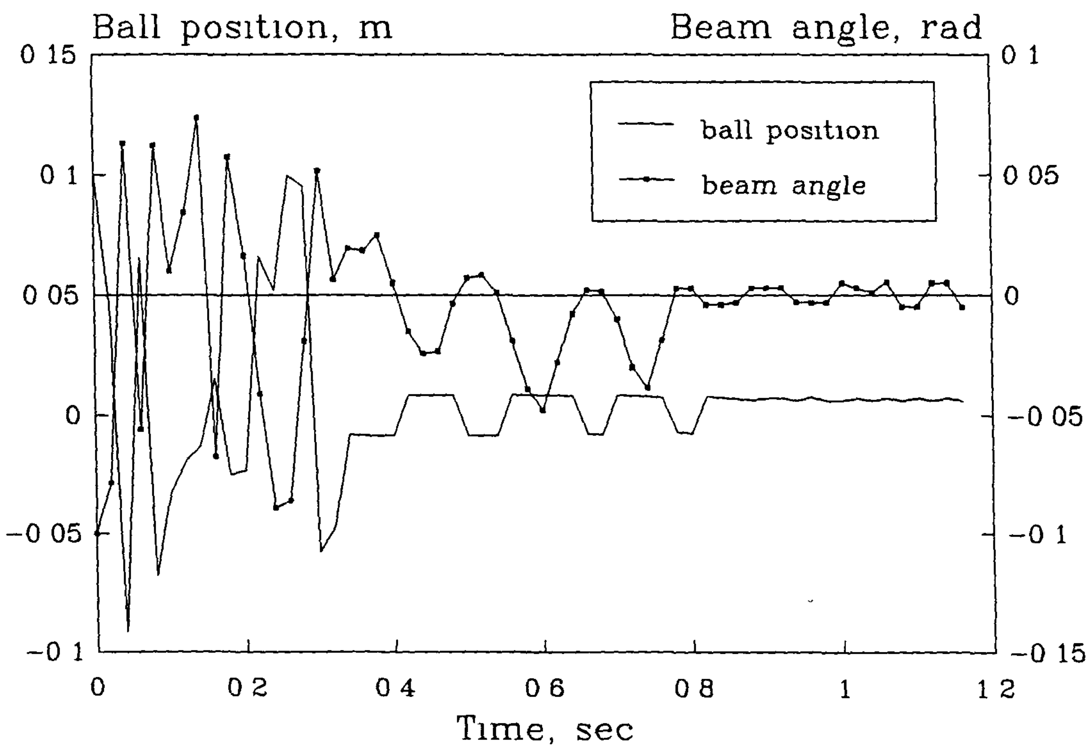


Figure 3.17 The third training run. Initial position: x=0.3 m, α=-0.3 rad.

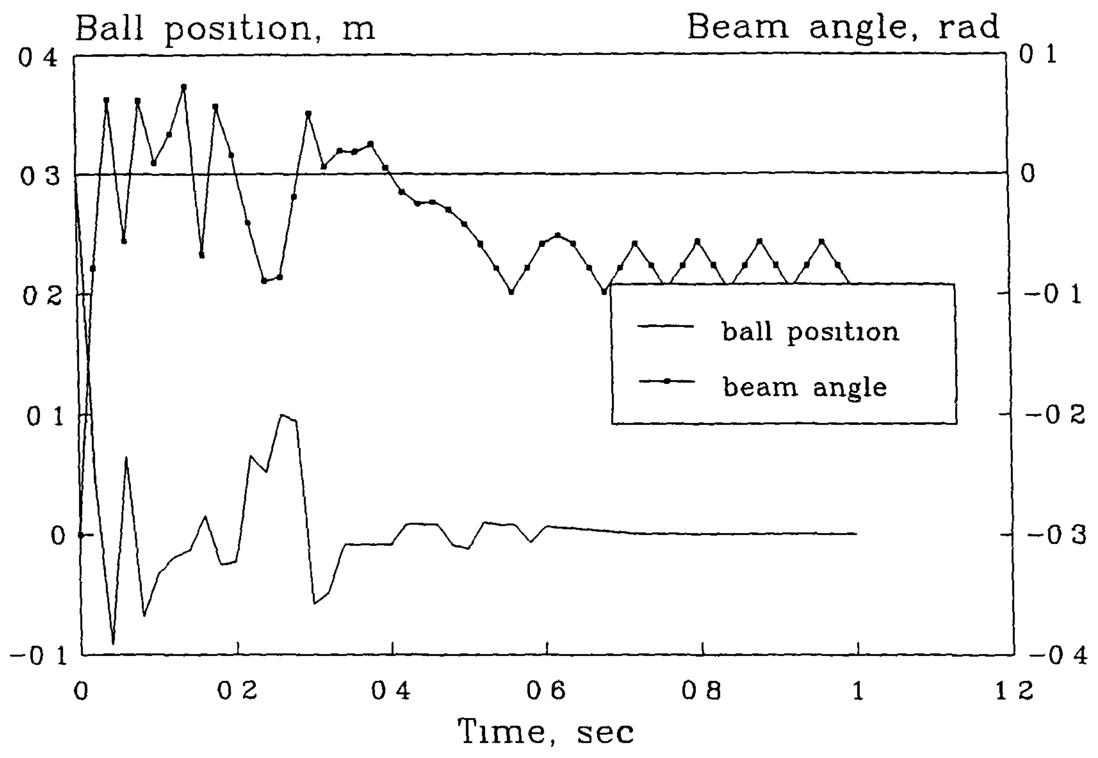


Figure 3.18 The third training run. Initial position: x=0.3 m, α=0.3 rad.

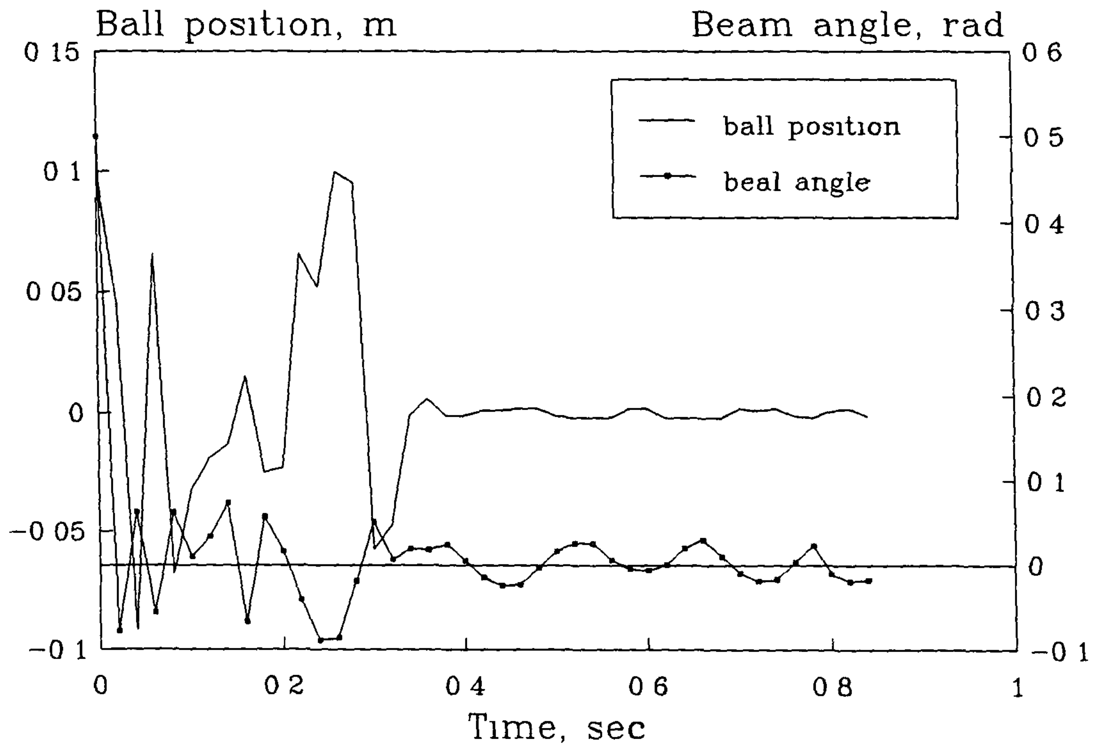


Figure 3.19 The fourth training run. Initial position: x=0.1 m, α=0.5 rad.

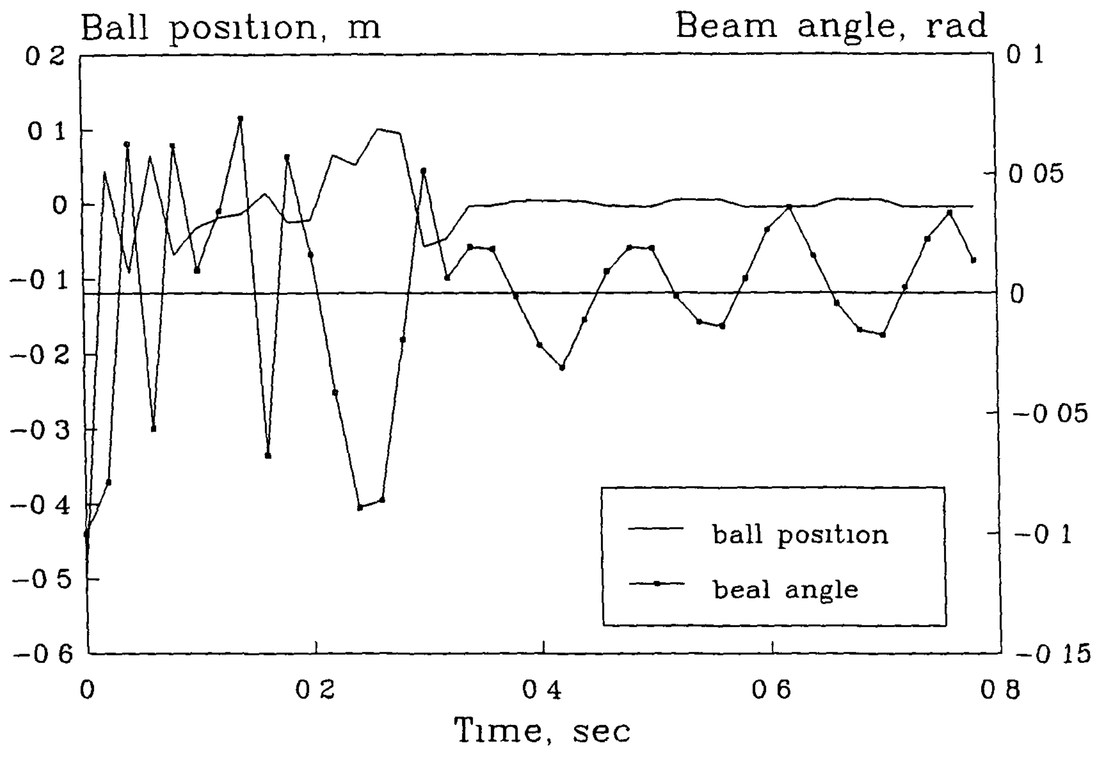


Figure 3.20 The fourth training run. Initial position: x=-0.5 m, α=-0.1 rad.

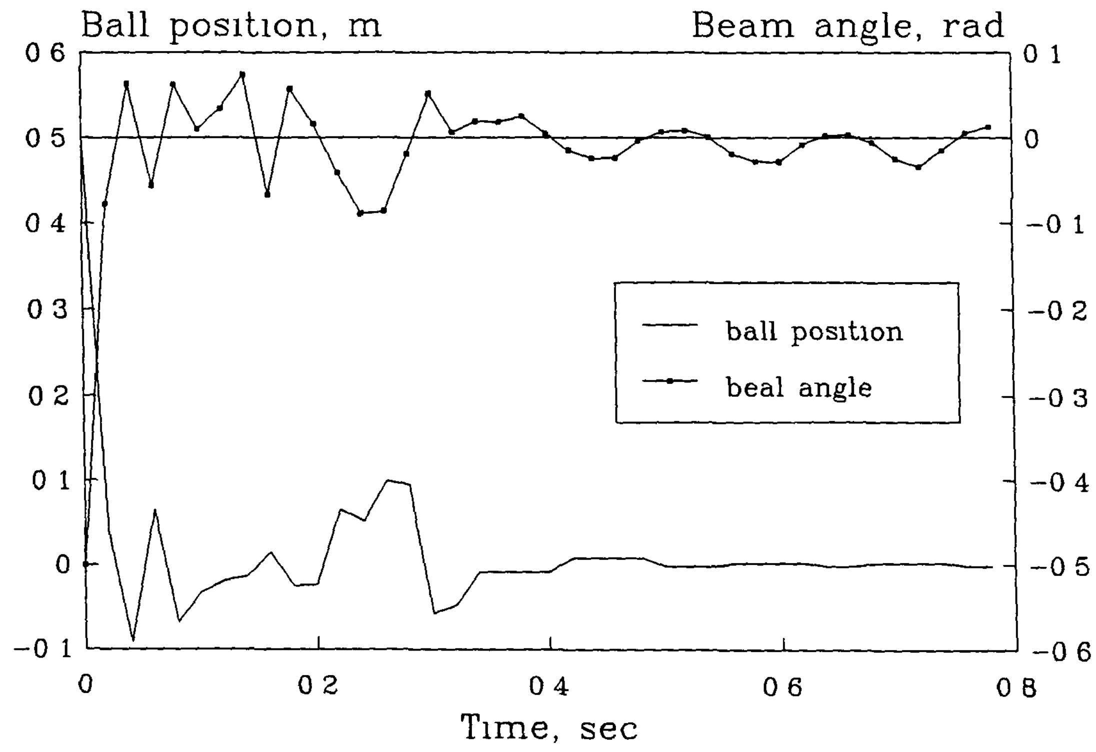


Figure 3.21 The fourth training run. Initial position: x=0.5 m, α=-0.5 rad.

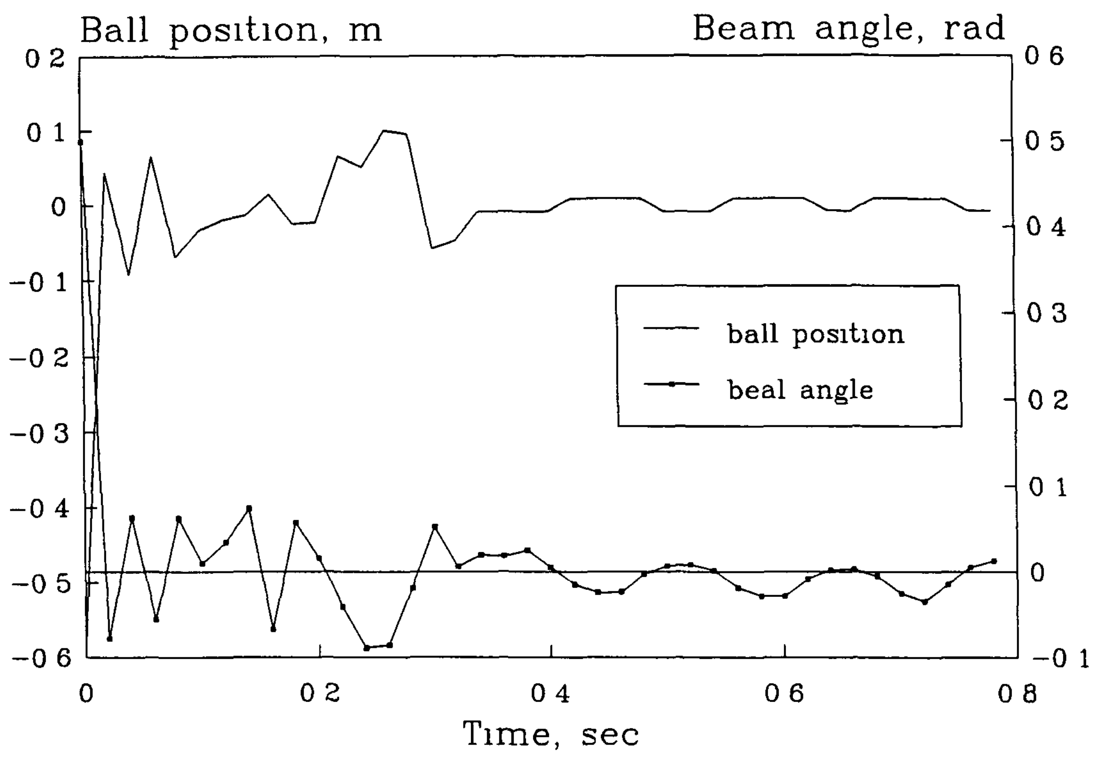


Figure 3.22 The fourth training run. Initial position: x=-0.58 m, α=0.5 rad.

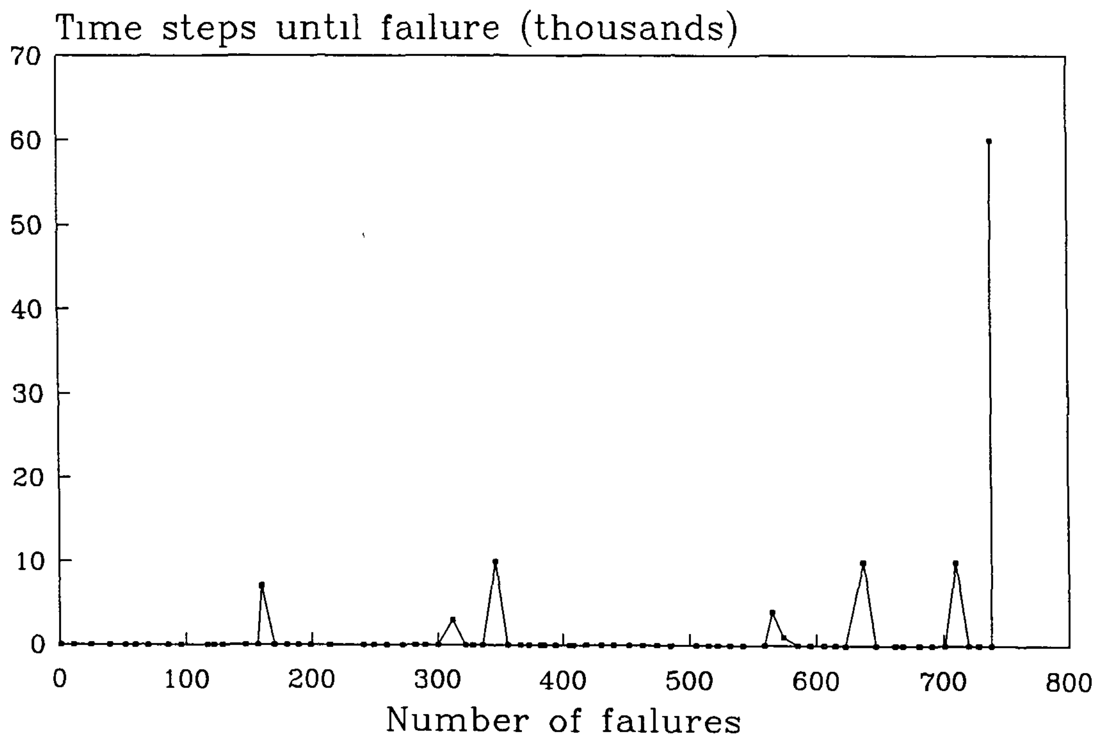


Figure 3.23 The second training run Initial position: x=-0.5 m, α=-0.5 rad

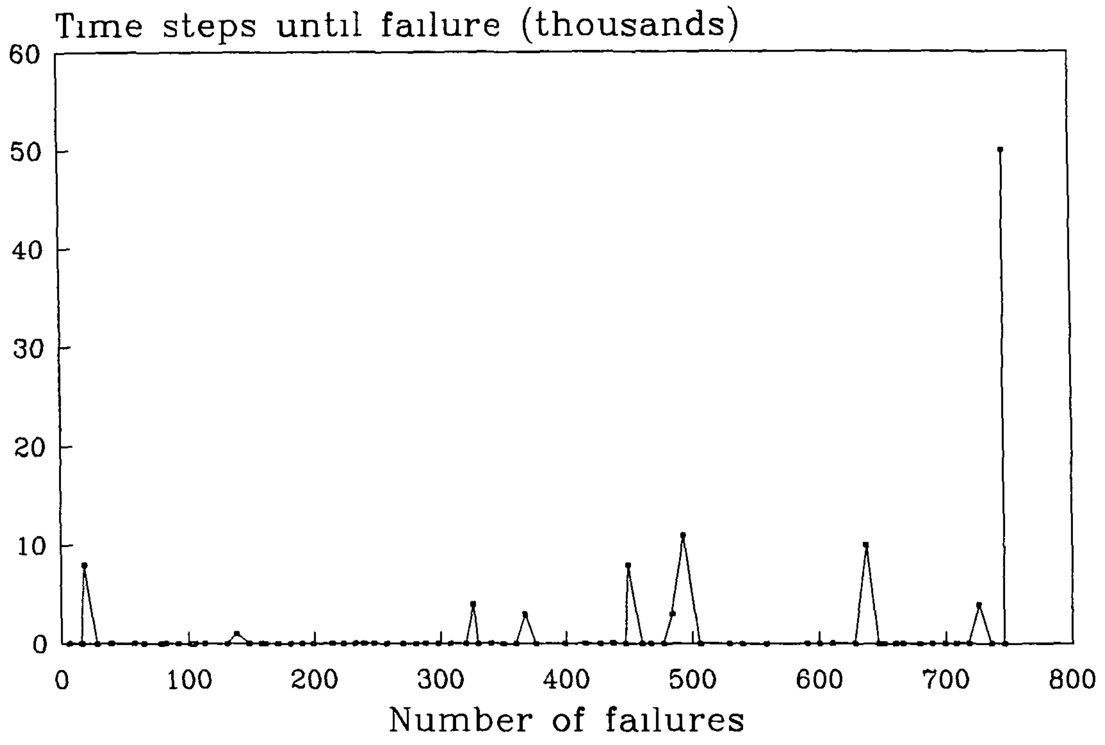


Figure 3.24 The second training run. Initial position: x=0 m, α=0 rad.

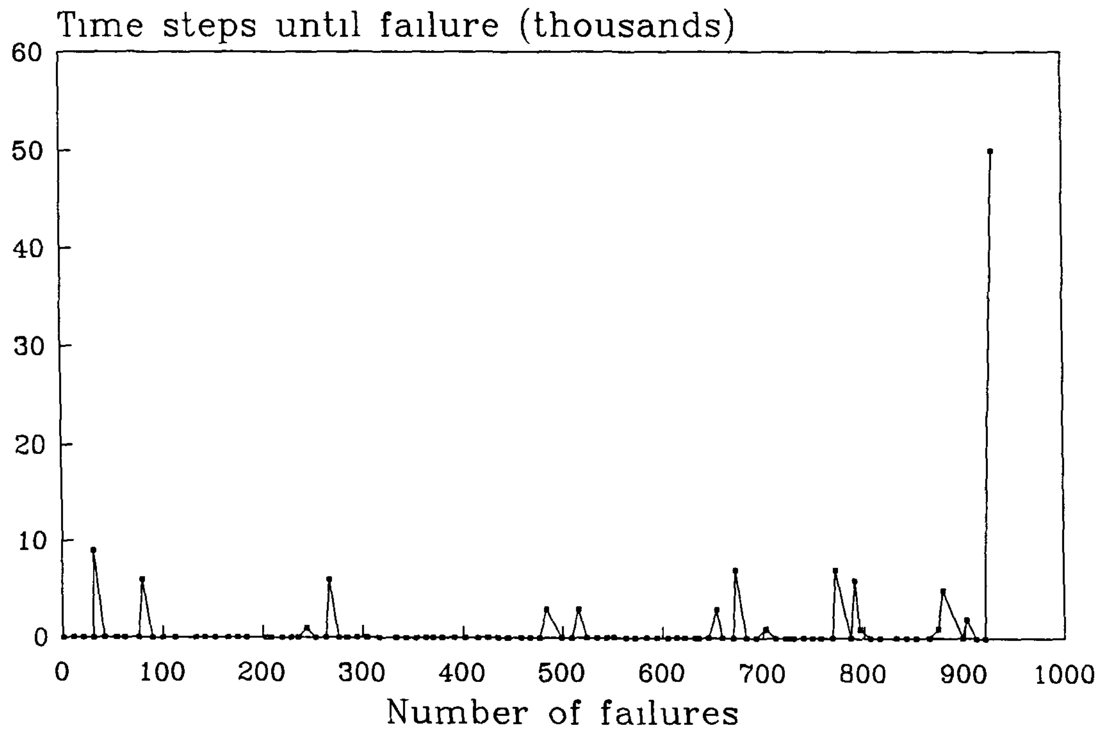


Figure 3.25 The second training run. Initial position: x=0.58 m, α=0.5 rad.

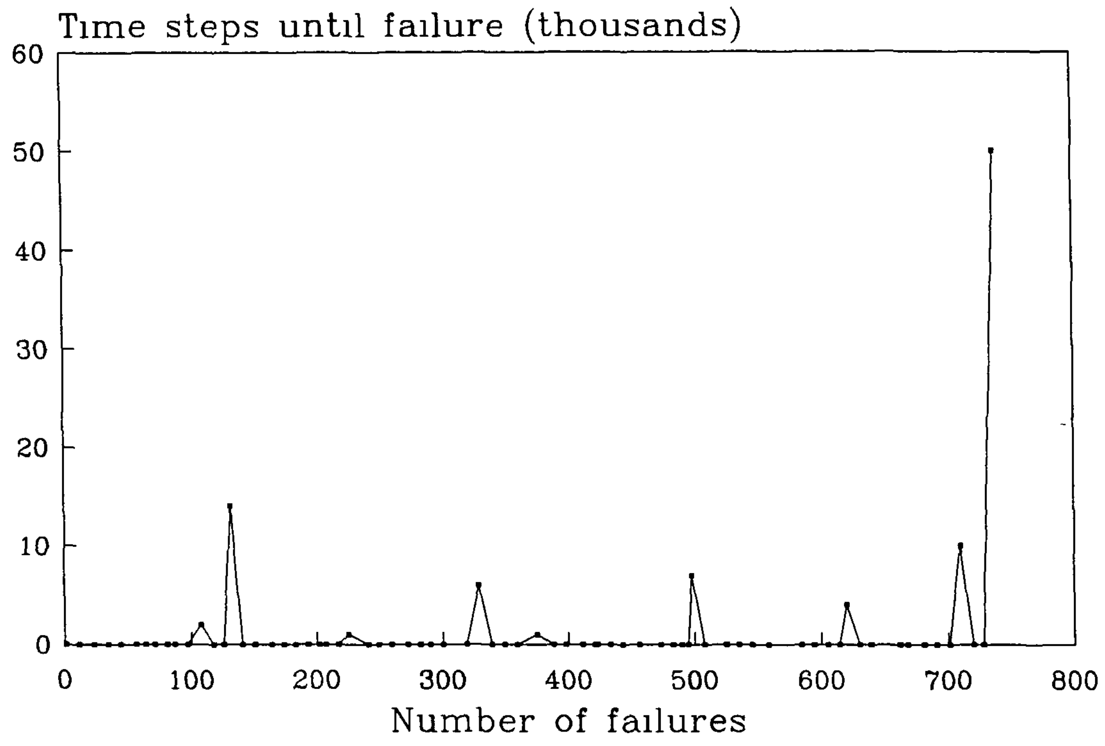


Figure 3.26 The second training run. Initial position: x=0.58 m, α=0.5 rad.

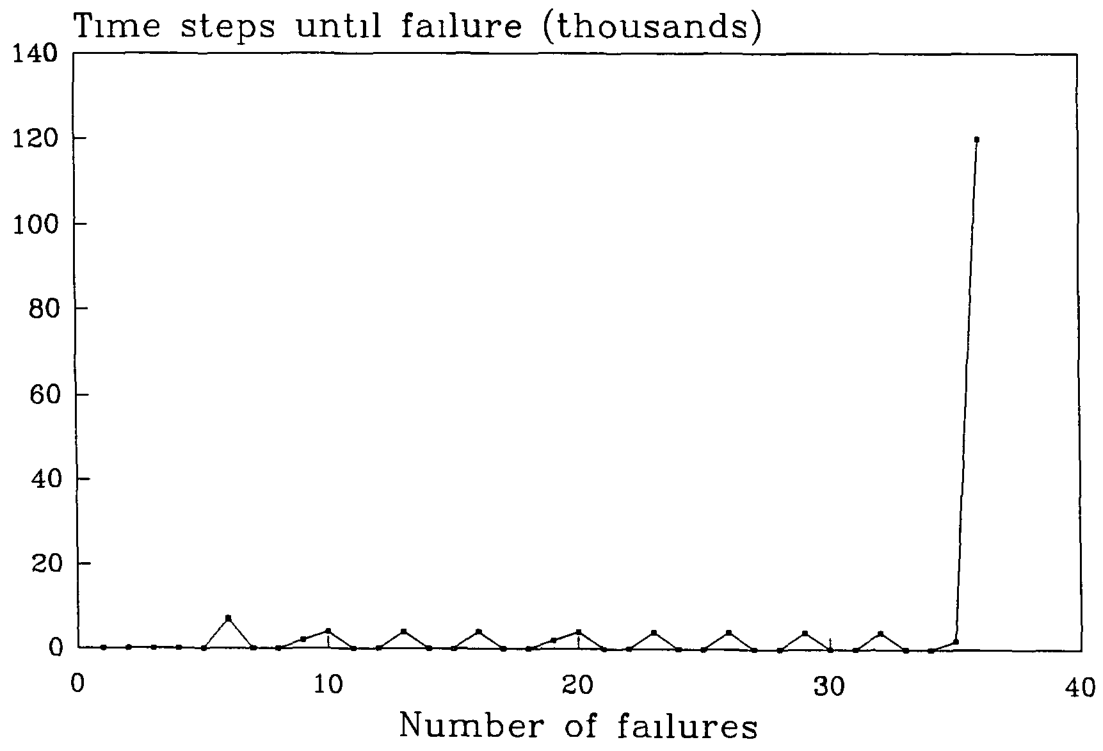


Figure 3.27 The third training run. Initial position: x=0.1 m, α=0.5 rad.

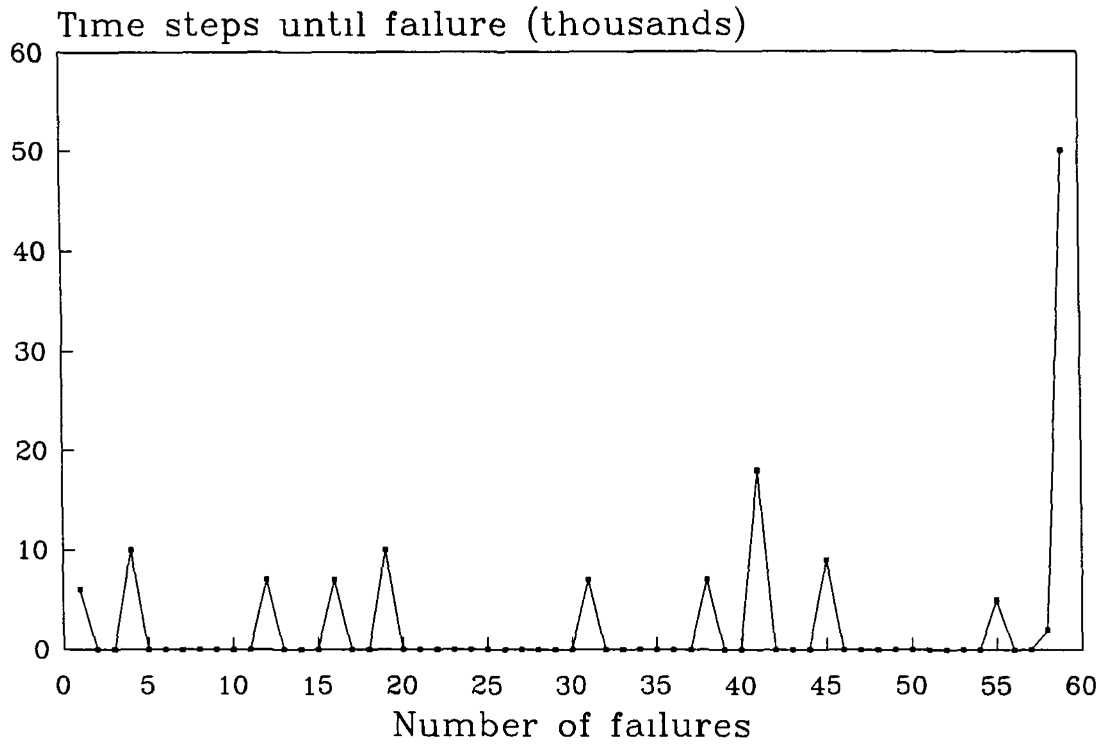


Figure 3.28 The third training run Initial position: x=0 m, α=0 rad.

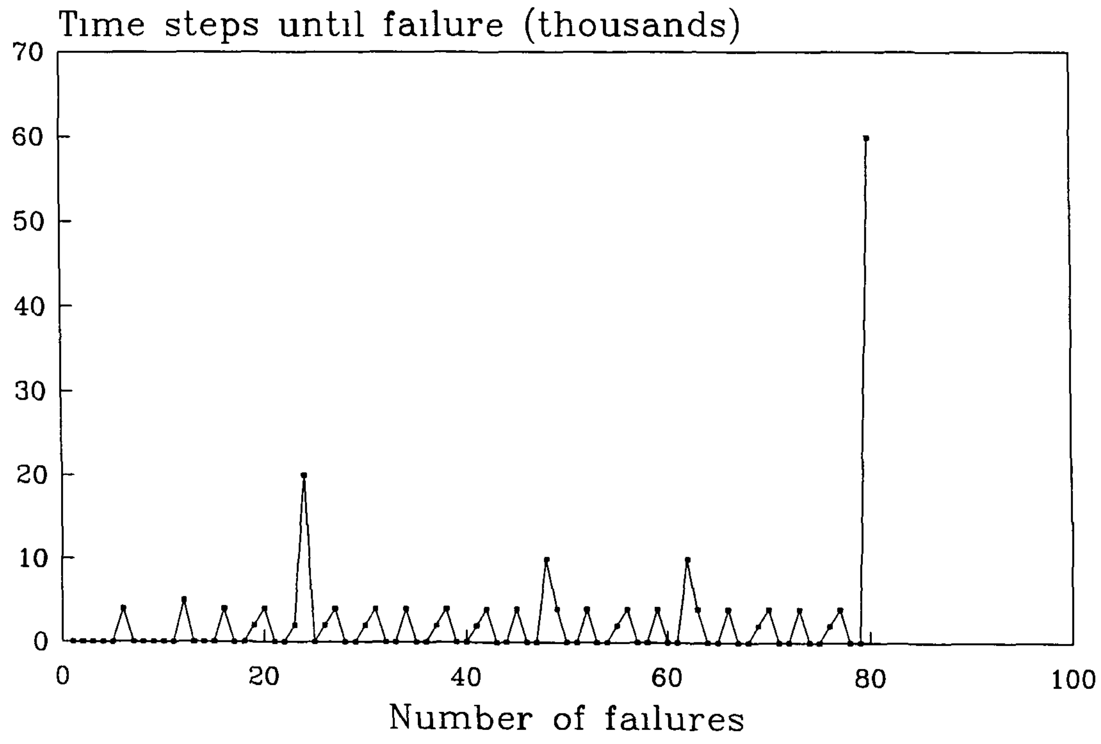


Figure 3.29 The third training run. Initial position: x=0.58 m, α=0.5 rad.

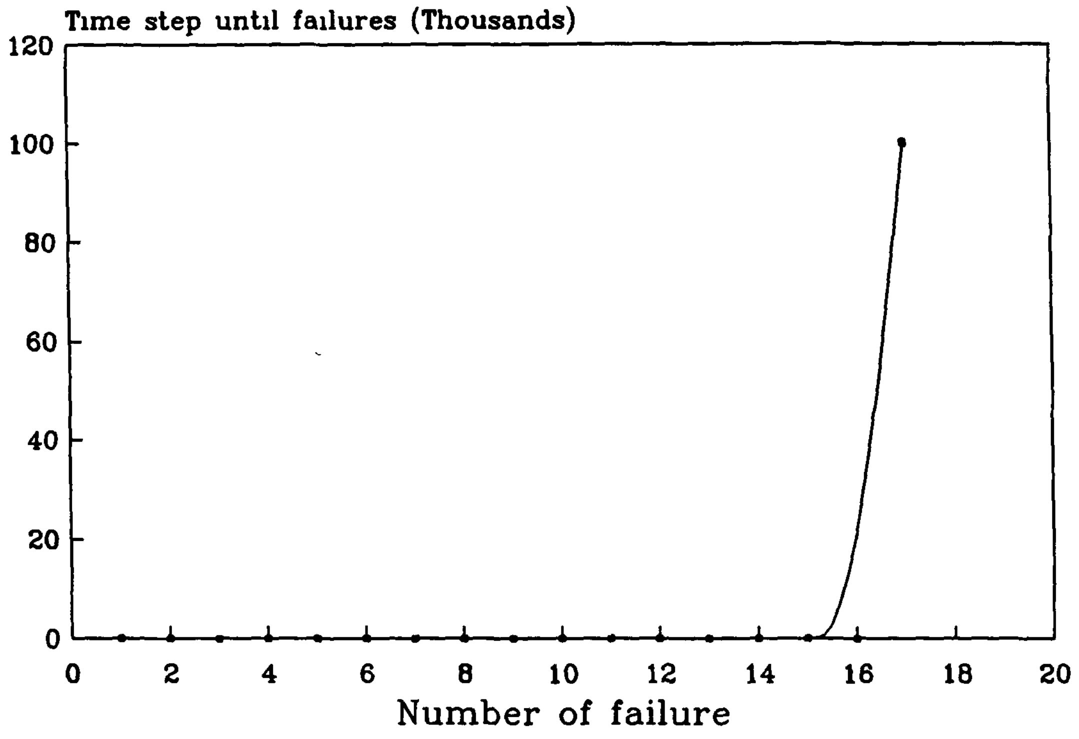


Figure 3.30 The fourth training run.

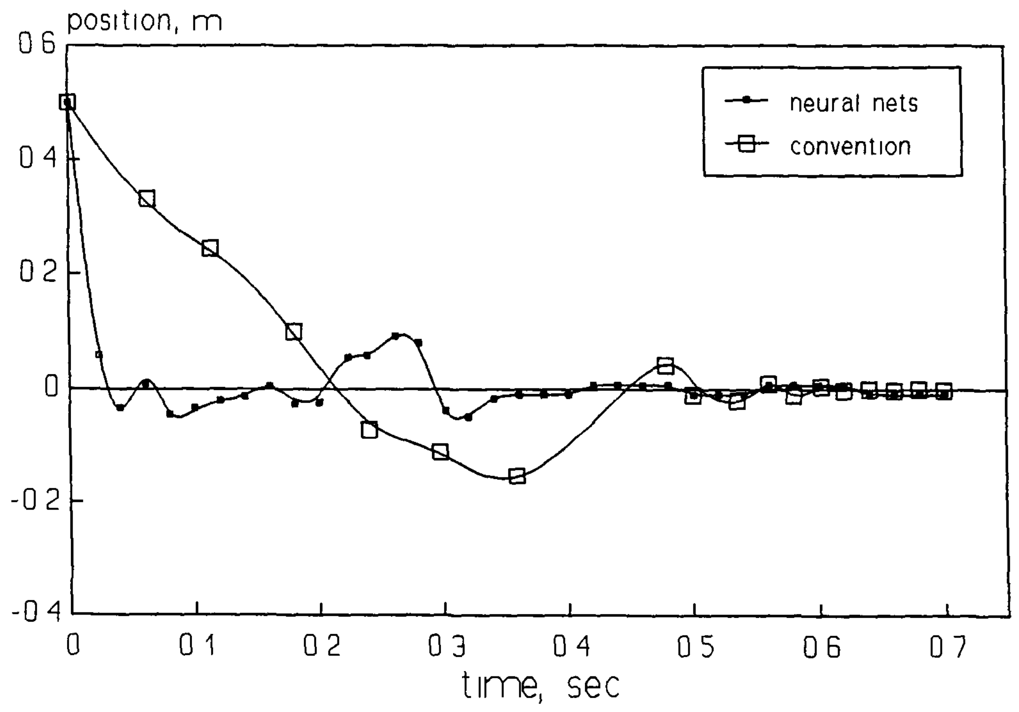


Figure 3.31 The neural network and conventional simulation comparison. Initial position: x=0 5 m, α=-0.5 rad.

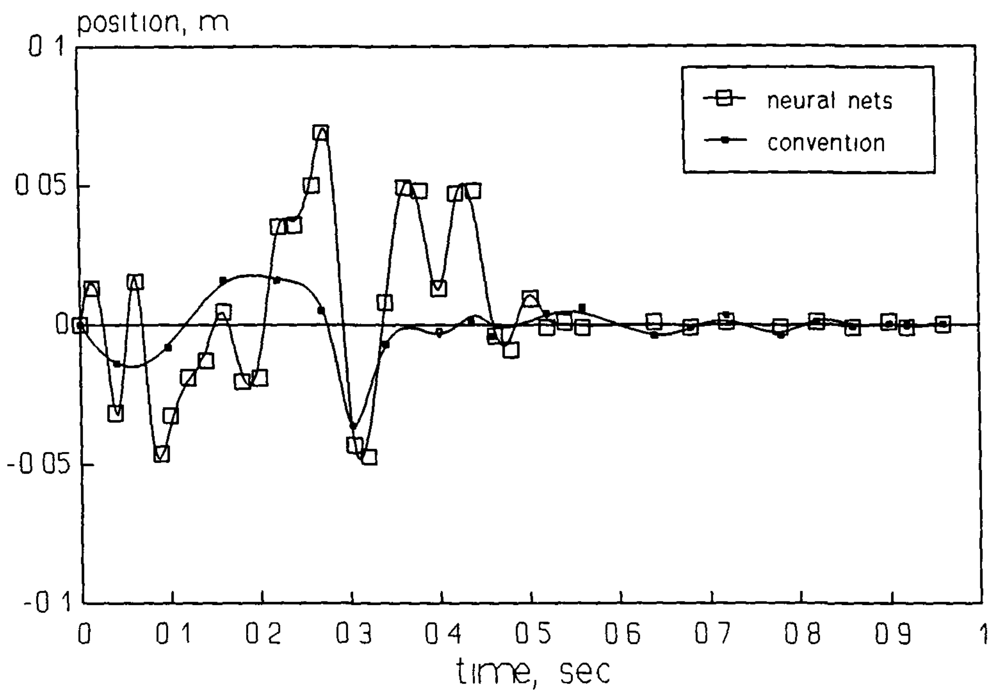


Figure 3-32 The neural network and conventional simulation comparison. Initial position: x=0 m, α=0.5 rad.

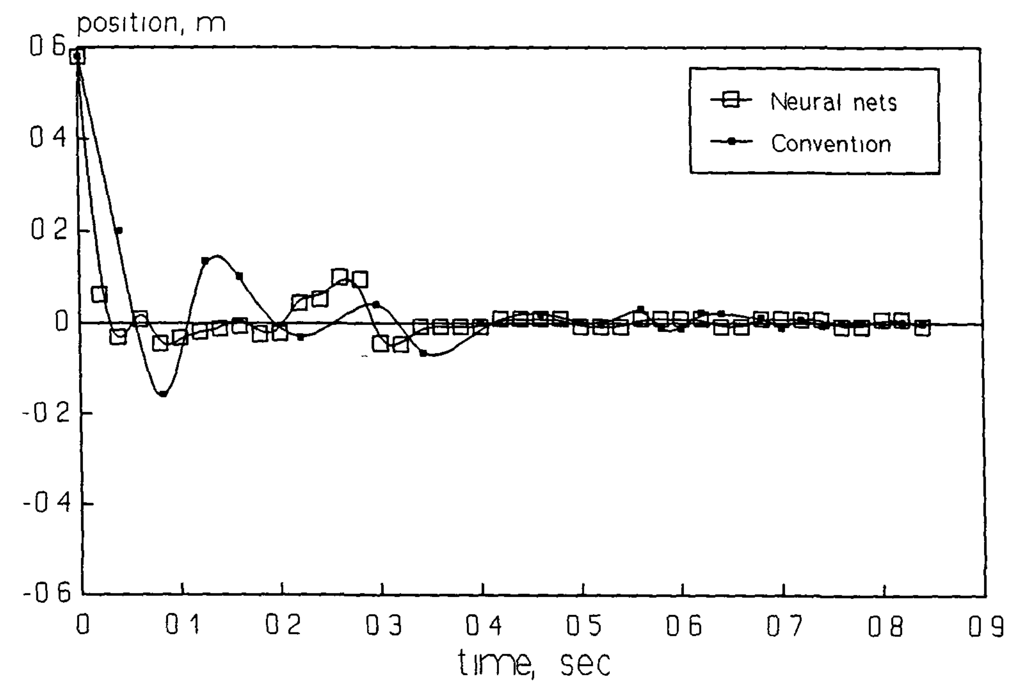


Figure 3-33 The neural network and conventional simulation comparison Initial position: x=0.58 m, α=0 rad.

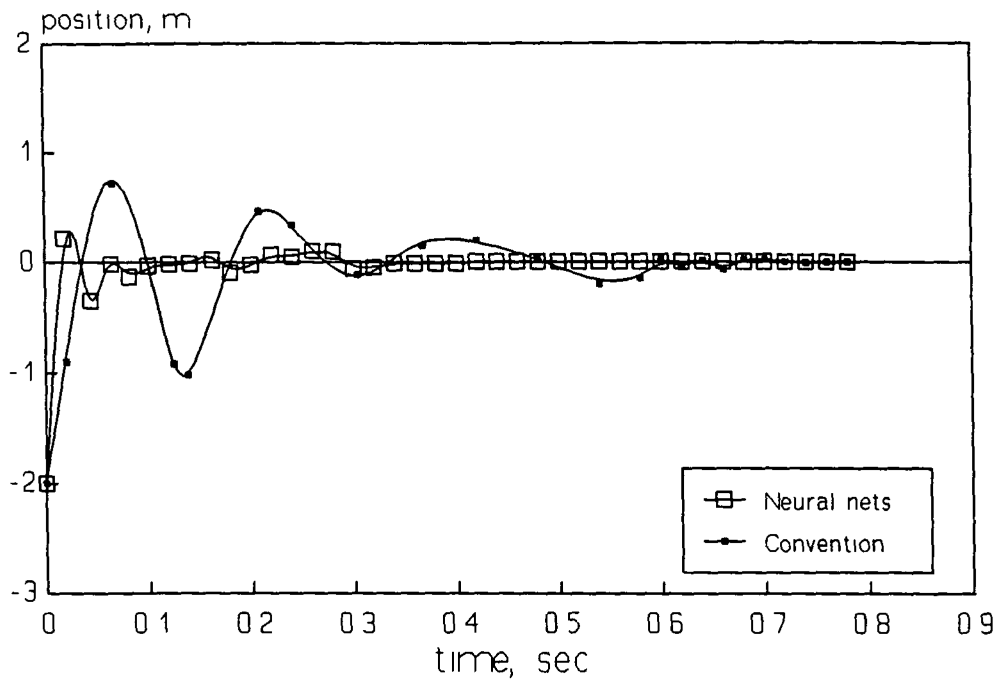


Figure 3-34 The neural network and conventional simulation comparison. Initial position: x=-2 m, α=2 rad.

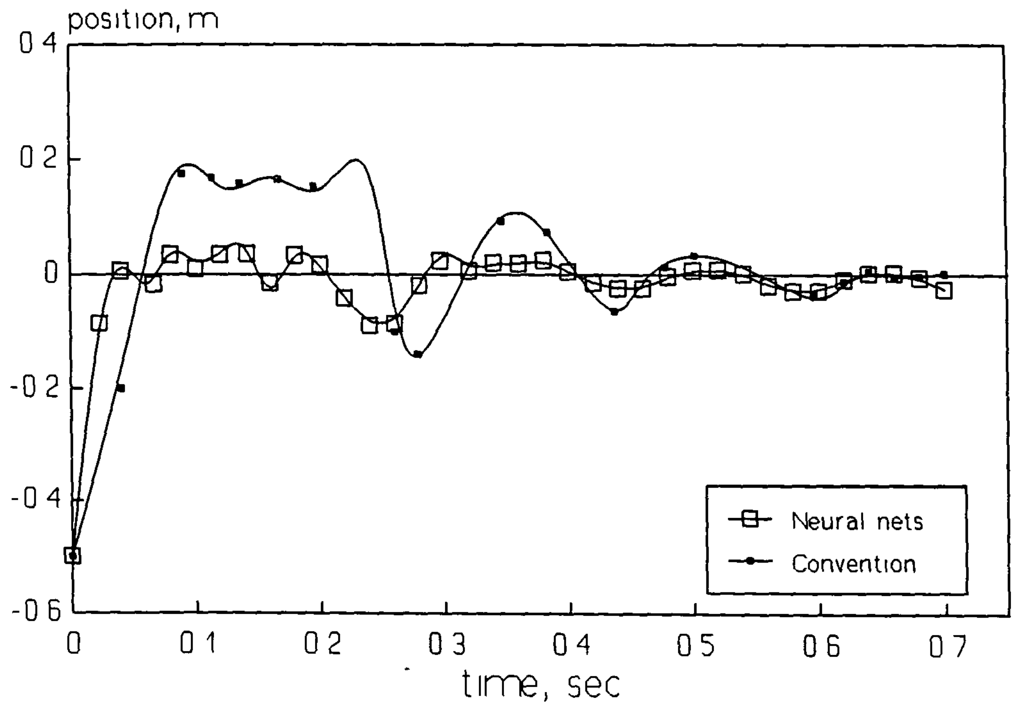


Figure 3-35 The neural network and conventional simulation comparison. Initial position: x=0.5 m, α=-0.5 rad.

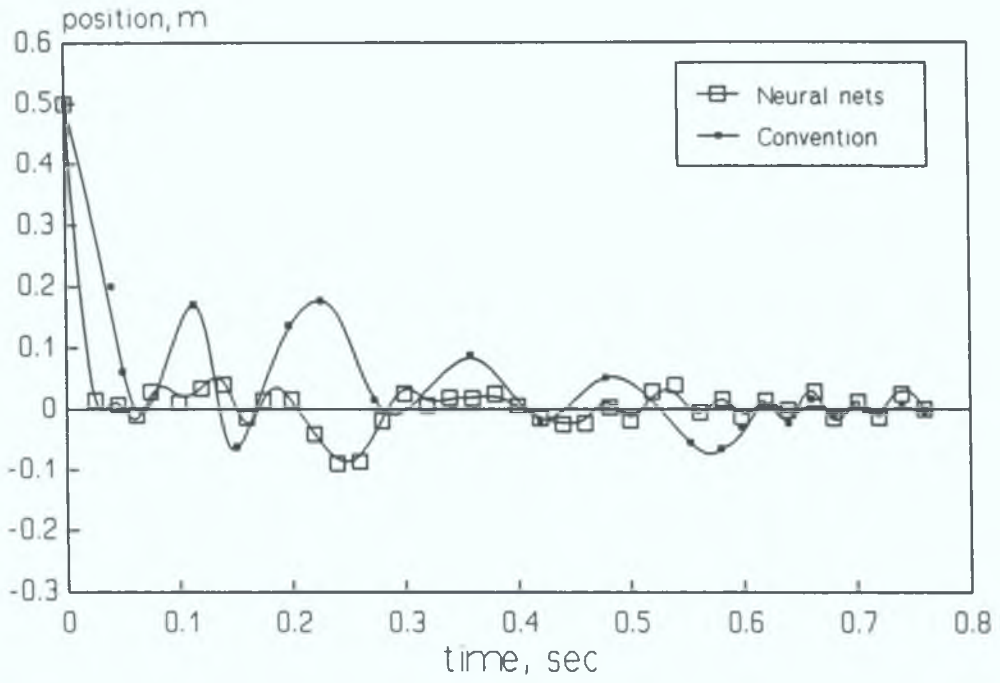


Figure 3-36 The neural network and conventional simulation comparison. Initial condition: x=0 m, α=0.5 rad.

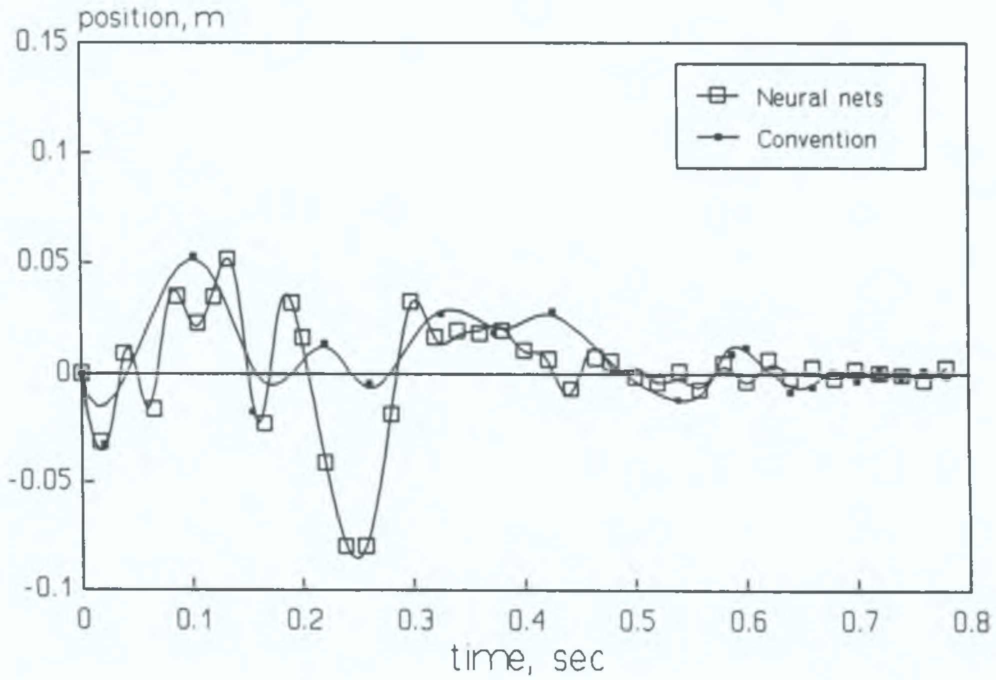


Figure 3-37 The neural network and conventional simulation comparison. Initial position: x=0.58 m, α=0 rad.

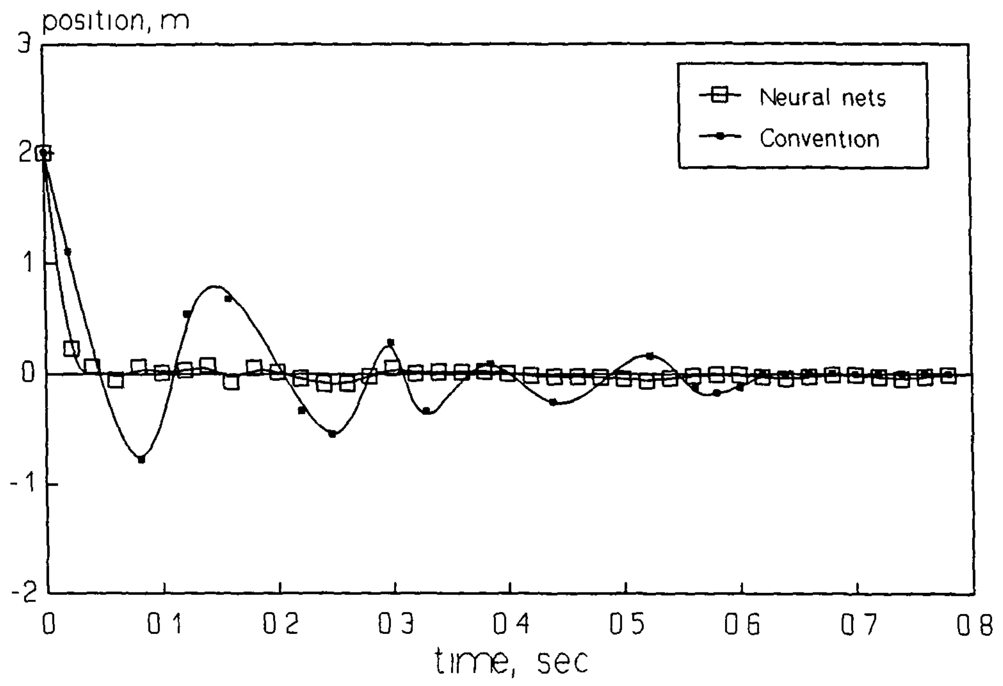


Figure 3-38 The neural network and conventional simulation comparison. Initial position: x=-2 m, α=2 rad.

3.3 Comparisons and Discussions

The histories of the ball position x and beam angular position α are given in Figure (3-31) to (3-38) for different initial positions. From these curves, it is easy to find out the time needed to balance the system for different control method. For example, Figure (3-19) shows that it took 0.3 seconds for the neural network controller to balance the system while it took 0.7 seconds to do the same. This shows that the neural network is competitive even when implemented serially.

The learning process is crucial for the neural network controller. For a novice controller, as shown in Figure (3-12), it took about 20-30 seconds to balance the system, while after several training runs, it only needed 0.3-0 4 seconds.

It took about 30-40 seconds for the network to learn to balance from the first training run. The balanced weights are used as initial weights in the second training run. They took about 1500 times running program making ball balanced on the beam. It took about 20-25 seconds for the network to learn to balance from the second training run. The balanced weights are used as initial weights in the third training run. They took about 700-900 times running program making ball balanced on the beam. It took about 0.6-0.7 seconds for the network to learn to balance from the third training run. The balanced weights are used as initial weights in the third training run. They took about 30-60 times running the program to make the ball balance on the beam. It took about 0.3-0.4 seconds for the network to learn to balance from fourth to tenth training runs. The balanced weights are used as initial weights in the fourth to tenth training runs. It took 17 times running the program to make the ball balance on the beam.

In the conventional controller, no learning is needed, and the "best" performance is obtained from the beginning. In neural network control, the system performance is improved through learning. In essence, dynamics by adjusting the connection weights of the nodes. This is why the system dynamics is not needed in neural network control. Through the learning runs, the connection weights gradually approach constant values. It can be predicted that since the system dynamics is certain for a given problem, so should the connection weights, given the structure of the networks is predetermined. This has been shown in the simulation that after the fourth training run, the set of weights is nearly constant. If this set of connection of weights are applied to the network, then the ideal performance is reached. It has also been noted that the ideal set of connection can be reached starting from any set of initial conditions for the training runs. In another word, the ideal set of weights is characteristic of the system, including the controller and plant, and not to be altered by different time histories.

It should be noted that in the neural network control simulation, only the results of the first four runs are given, since after the fourth training, the results are essentially the same.

The amplitude of the actuating force F(t) should be properly chosen. If the force is too great, the ball will be tossed out of the wire track and if it is too small, it will not be able to balance the system.

CHAPTER 4 IMPLEMENTATION OF CONVENTIONAL AND NEURAL NETWORK METHODS

The control of a ball moving on a beam has been simulated in the previous chapter using conventional and neural network control methods. The simulation results show that the neural network controller can achieve similar or better performance than the conventional controller. In this chapter, both the conventional and the neural network method will be implemented on a laboratory apparatus using a personal computer.

4.1 Instrumentation and procedures

The control task involves the balancing of a ball on a beam which is pivoted in the middle, as shown in Figure 1.9 earlier. The control system consists of the ball and beam apparatus (type CE6) itself, a 80386 based IBM compatible personal computer, analogue to digital and digital to analogue converters and certain other general purpose meters.

Figure 4.1 shows a system sketch. It should be noted that although the neural network technique favours parallel processing, it is implemented here serially due to lack of resource.

4.2 Conventional Control System

4.2.1 Control process

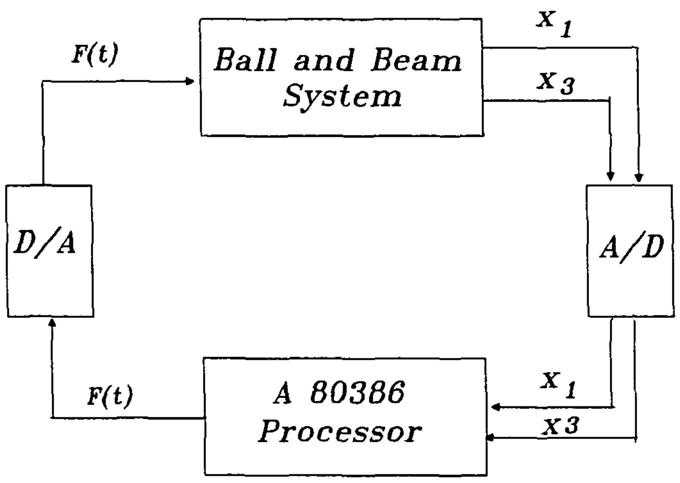


Figure 4.1 The control system structure.

The control cycle begins by evaluating the system status, which is represented by the linear displacement and the velocity of the ball moving on the beam and the angular displacement and the velocity of the beam revolving around the pivot. These variables are measured using a linear and an angular transducers, and they are converted into digital signals by an A/D converter. The linear and angular velocities of the ball and beam can be computed from the displacement history and this has been done by the programs of conventional control method and neural network control method, respectively. The decision making process was performed by the computer, according to conventional theory and neural network algorithms. The computer then outputs a digital signal which represents a positive force +F(t) (upward) or a negative force -F(t) (downward). This signal is converted into an analog signal and sent to the plant to control its behaviour. At this point the cycle is complete and a new cycle begins.

According to the laboratory experiment, the following equation is obtained to provide low pass filtering. The calculated ball position is:

$$x_1 = 0.75\, x_1' + 0.25\, x_{A/D} \tag{4-1}$$

The velocity of the ball is:

$$x_2 = (x_1' - x_1)\,/\tau \tag{4-2}$$

where x_1 is ball position obtained from the previous sampling, x_1' is ball position obtained from the last sampling, $x_{A/D}$ is ball position obtained from the A/D, and τ is the sample time.

The continuous time model of the plant is given by

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \boldsymbol{A}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \boldsymbol{B}\,F(t)$$
$$y(t) = \boldsymbol{C}\begin{bmatrix}x_1\\ x_2\end{bmatrix} \tag{4-3}$$

where x_1 is the ball position and x_2 is the ball velocity. The output $y(t) = x_1$ is the ball position. The matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are given as

Since velocity is the first derivative with respect to time of position, we get

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} b \\ 0 \end{bmatrix} \qquad (4\text{-}4)$$

$$C = [1 \quad 0]$$

$$\dot{x}_1 = x_2 \qquad (4\text{-}5)$$

System stability is determined by the eigenvalues of the system A matrix:

$| \lambda I - A | = 0$. $\lambda_1 = 0$, $\lambda_2 = 0$. system unstable.

To stabilize the system, a controller has shown in Figure (4-1) has been used. According to the laboratory practice, the feedback matrix is determined as

$$K = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0\ 6 \end{bmatrix}$$

The action force is given by

$$F(t) = G\, x(t) = [k_1, k_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = k_1\, x_1 + k_2\, x_2. \qquad (4\text{-}6)$$

The controller can thus be programmed by combining Equations (4-3) and (4-6).

4.2.2 Results

The experimental results obtained from the conventional control are presented below. Results corresponding to different initial states are shown in Figures (4-2) to (4-5). It is evident that the conventional controller was able to balance the system and the average time needed was 15-20 seconds, depending on the initial conditions and the system noise such as the influence on the measurement accuracy of the ball's intermittent contact, etc.

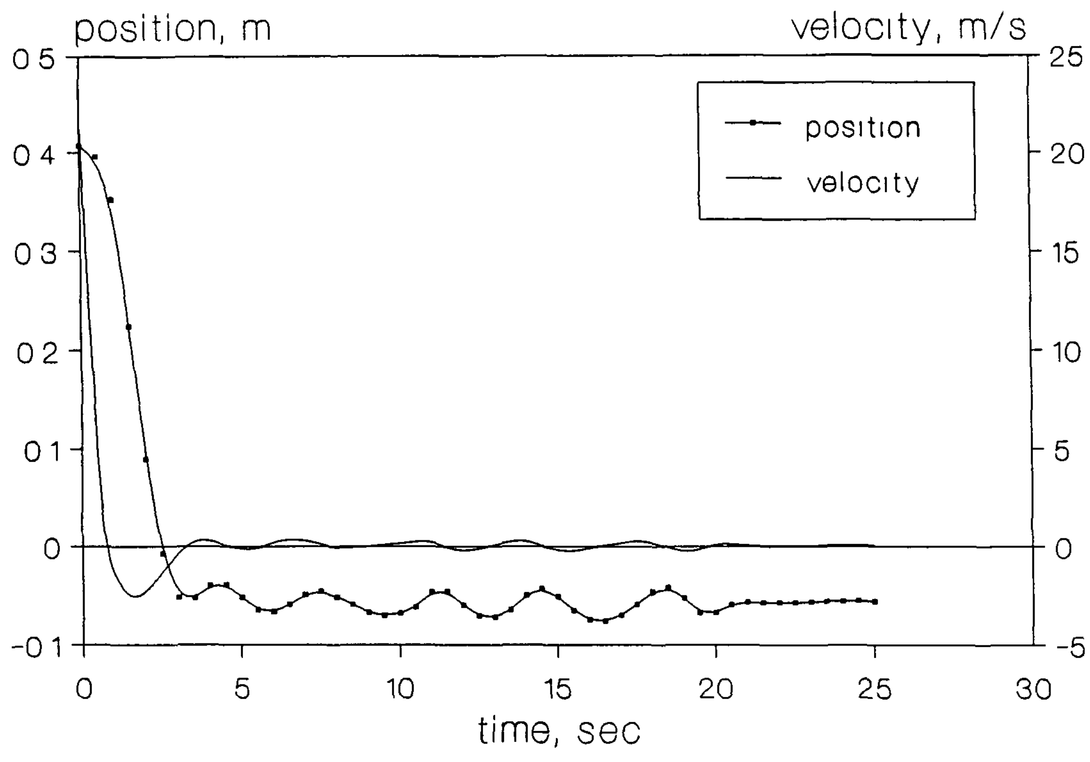


Figure 4.2 Initial position: x=0.407 m.

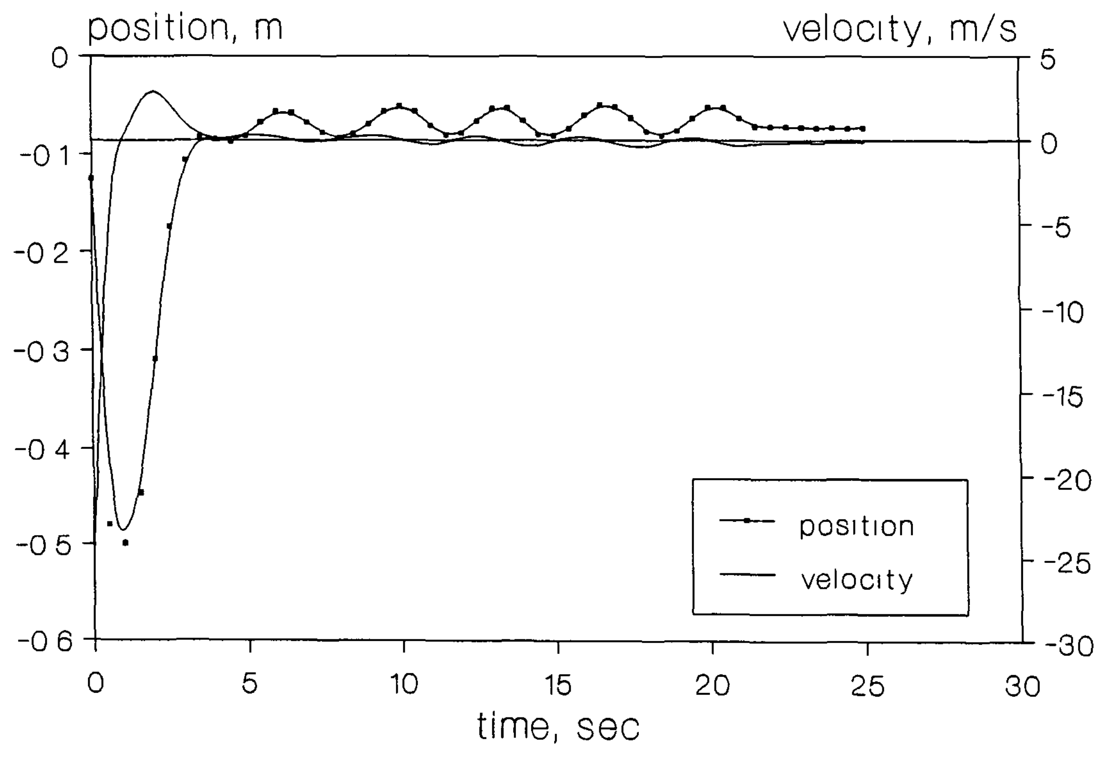


Figure 4.3 Initial position: x=-0.125 m.

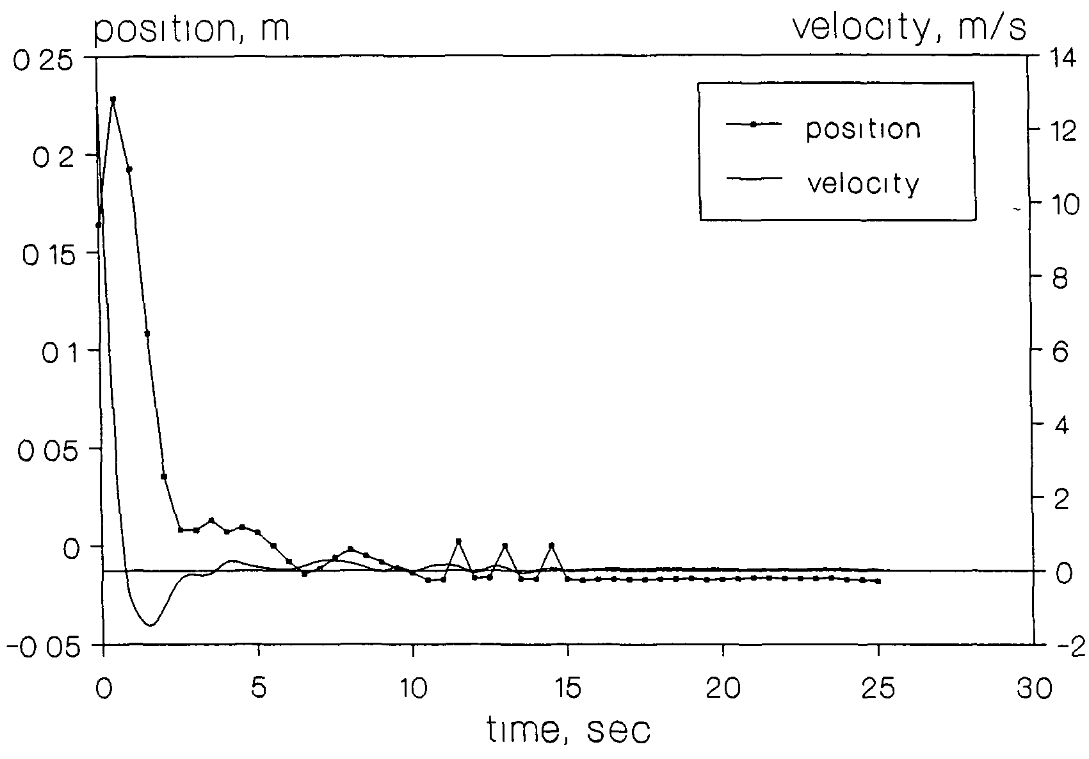


Figure 4.4 Initial position: x=0 164 m.

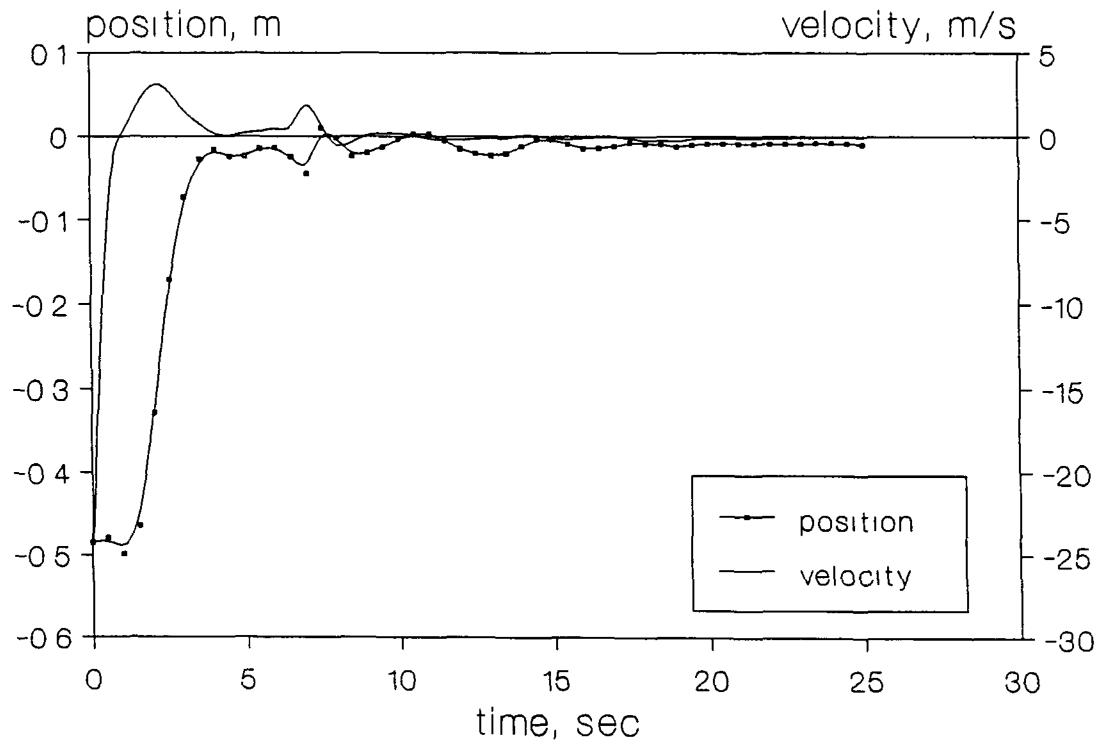


Figure 4.5 Initial position: x=-0.485 m.

4.3 Application of Neural Network Feedback

4.3.1 Neural networks learning algorithms

The same learning algorithms as used in the simulation work are used in the experiments. The rules for updating the connection weights in the evaluation network and the action network are given in Equations (3-29) and (3-36).

The value of F(t) is different in simulation and real time control.

$F(t)=q(t)\ (x_1+x_2)$.

4.3.2 Flow chart

See Figure 4-6.

4.3.3 Results

4.3.3.1 First and second training runs

For the first two training runs, it took about 10-11 seconds for the network to learn to balance the ball-beam system. When the ball is balanced on the beam, the network weight matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{F}$ are saved and these weights are used as initial weights in the subsequent training run. The results of experiments are shown in Figure 4-7, 4-8, 4-9 and 4-10. The corresponding failure curves are shown in Figure 4-15 and 4-16.

4.3.3.2 Third to tenth training runs

The neural networks were able to adapt themselves rapidly to control the system. The time need to balance the ball reduced sharply for the third training run, to roughly 5-7 seconds. From the fourth run on, up to tenth run, the system stabilized and the balancing time remained approximately the same. The experimental results are shown in Figures 4-11 to 4-21

4.4 Comparison of State Feedback and Neural Network Control Results

Figures 4-18 to 4-21 present the experimental results for both the conventional control method and the neural network method, with the same initial conditions and same sample time. It is evident that results from the two control methods show different characteristics. It is evident that the neural network controller performed better than the conventional controller in the experiments.

It took about 20-30 seconds to balance the system, while after several training runs, it only needed 0.3-0.4 seconds.

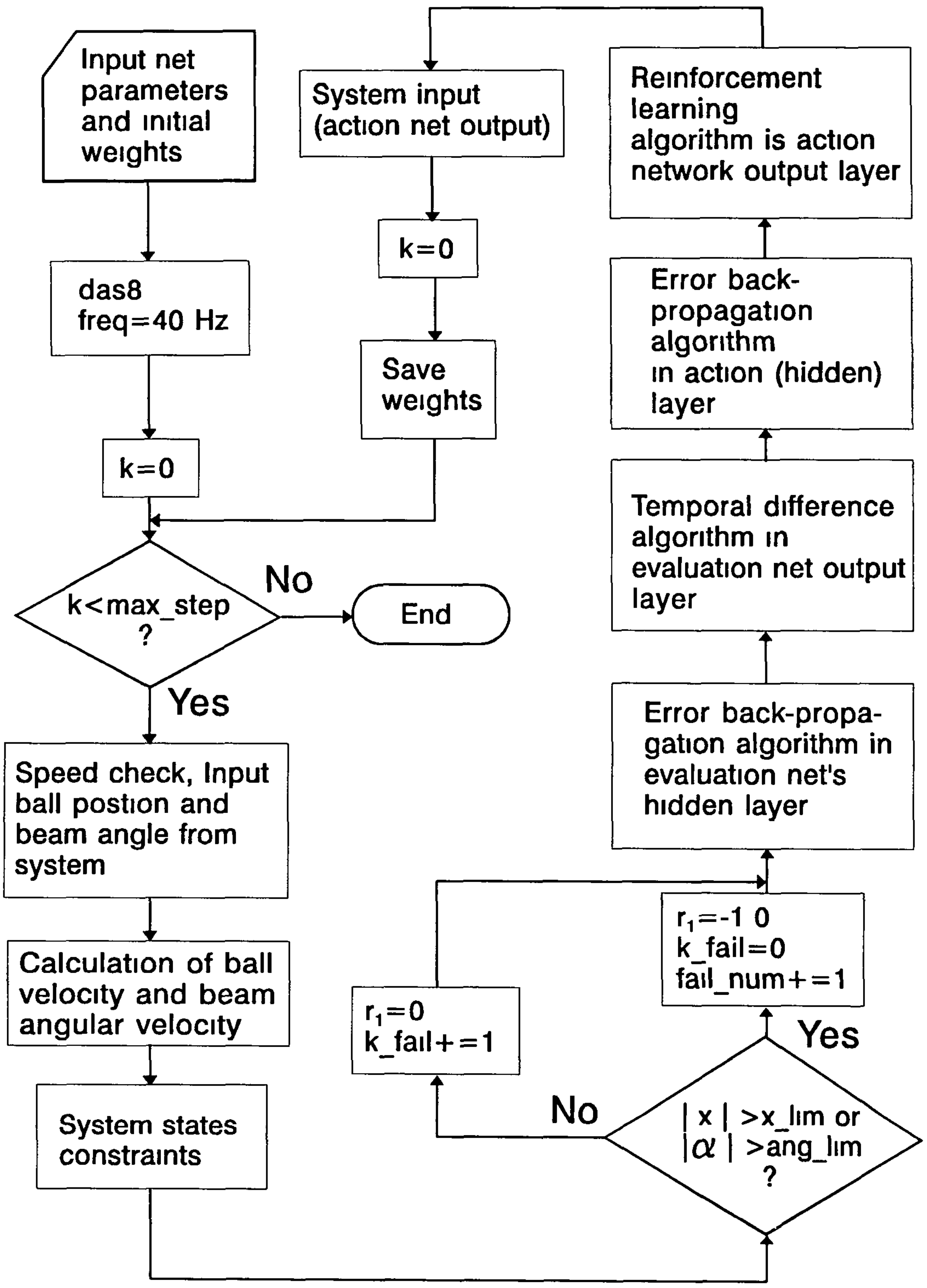

Input net parameters and initial weights
das8 freq=40 Hz
k=0
k<max_step ?
No
End
Yes
Speed check, Input ball postion and beam angle from system
Calculation of ball velocity and beam angular velocity
System states constraints
System input (action net output)
k=0
Save weights
Reinforcement learning algorithm is action network output layer
Error back-propagation algorithm in action (hidden) layer
Temporal difference algorithm in evaluation net output layer
Error back-propagation algorithm in evaluation net's hidden layer
r1=-1 0
k_fail=0
fail_num+=1
Yes
r1=0
k_fail+=1
No
\| x \| >x_lim or
\|α \| >ang_lim
?

Figure 4.6 The flow chart for programming the neural network controller.

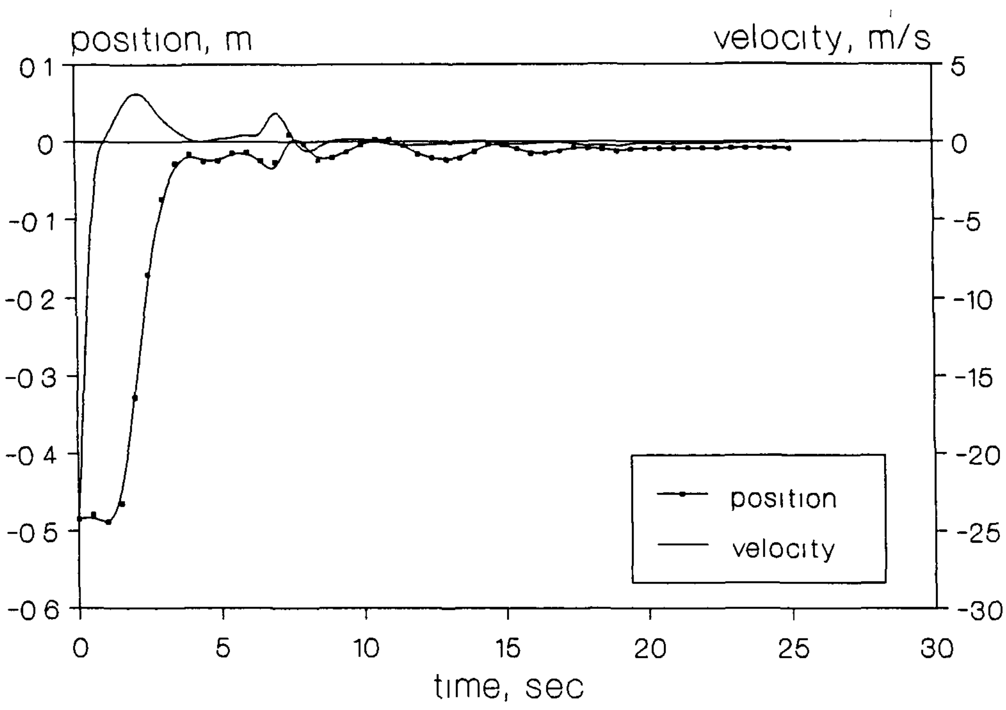


Figure 4.7 The first and second training run using neural network control. Initial position: x=0.453 m, α=0.017 rad.

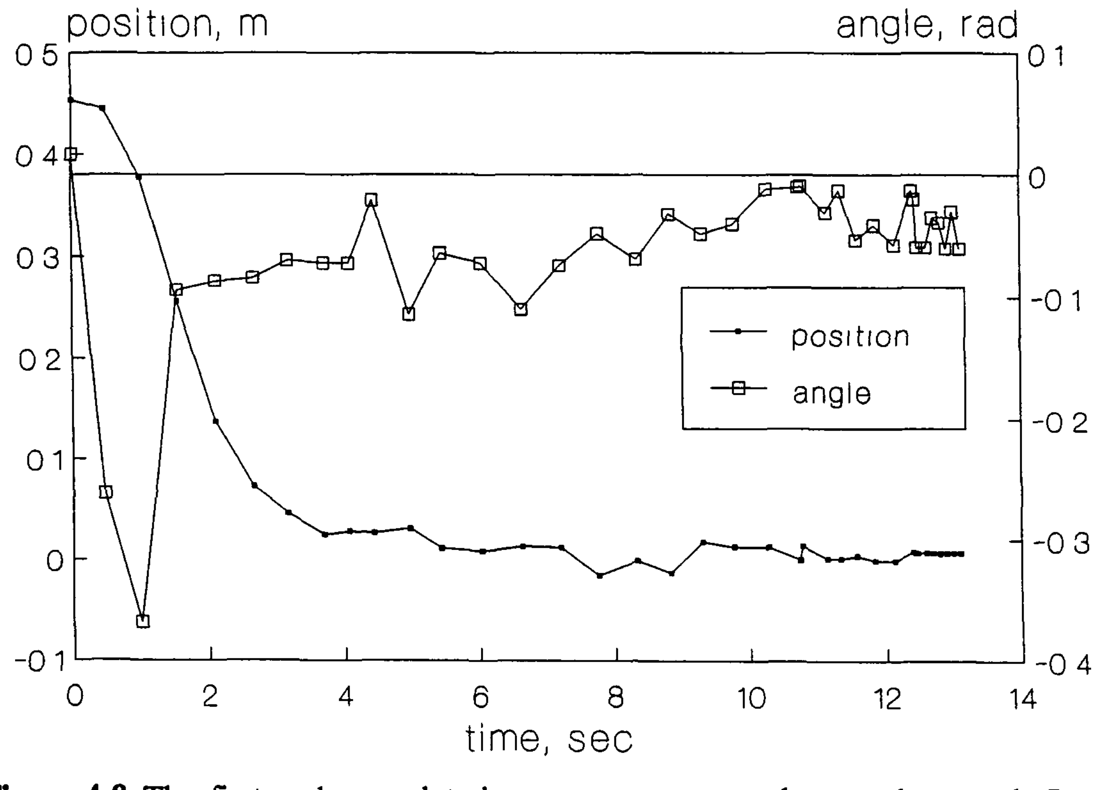


Figure 4.8 The first and second training run using neural network control. Initial position: x=-0 269 m, α=-0.349 rad.

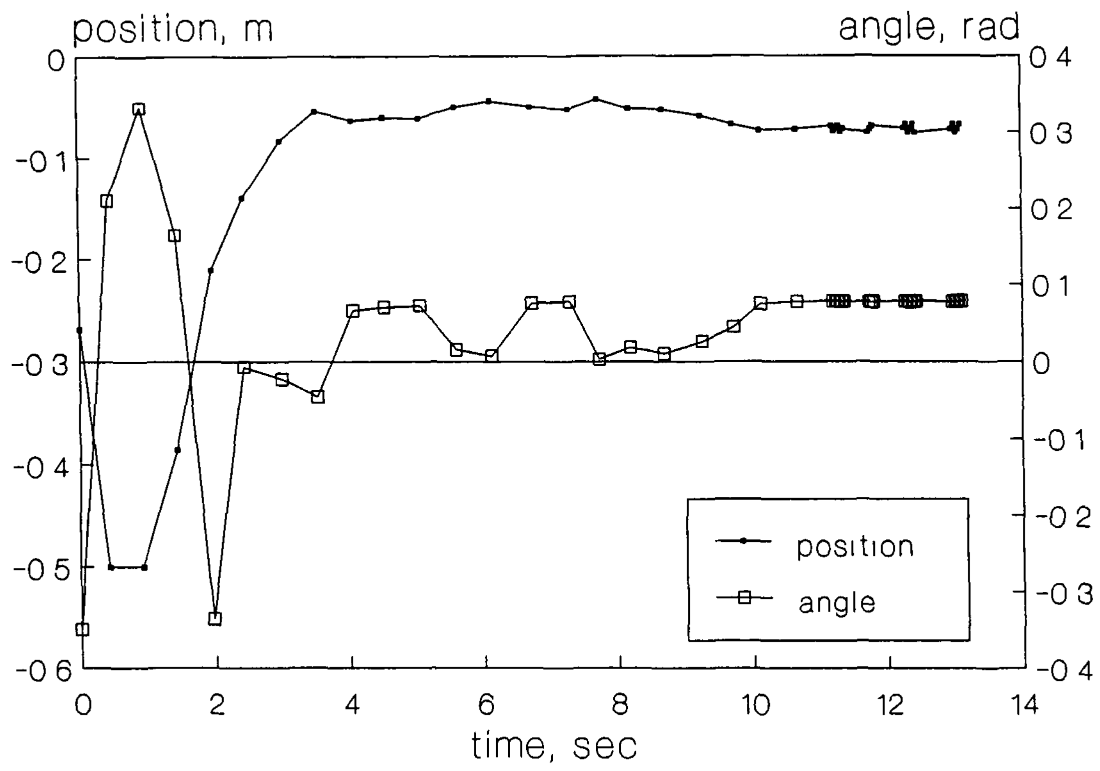


Figure 4.9 The first and second trainıng run usıng neural network control. Inıtıal posıtıon: x=0.294 m, α=0.079 rad.

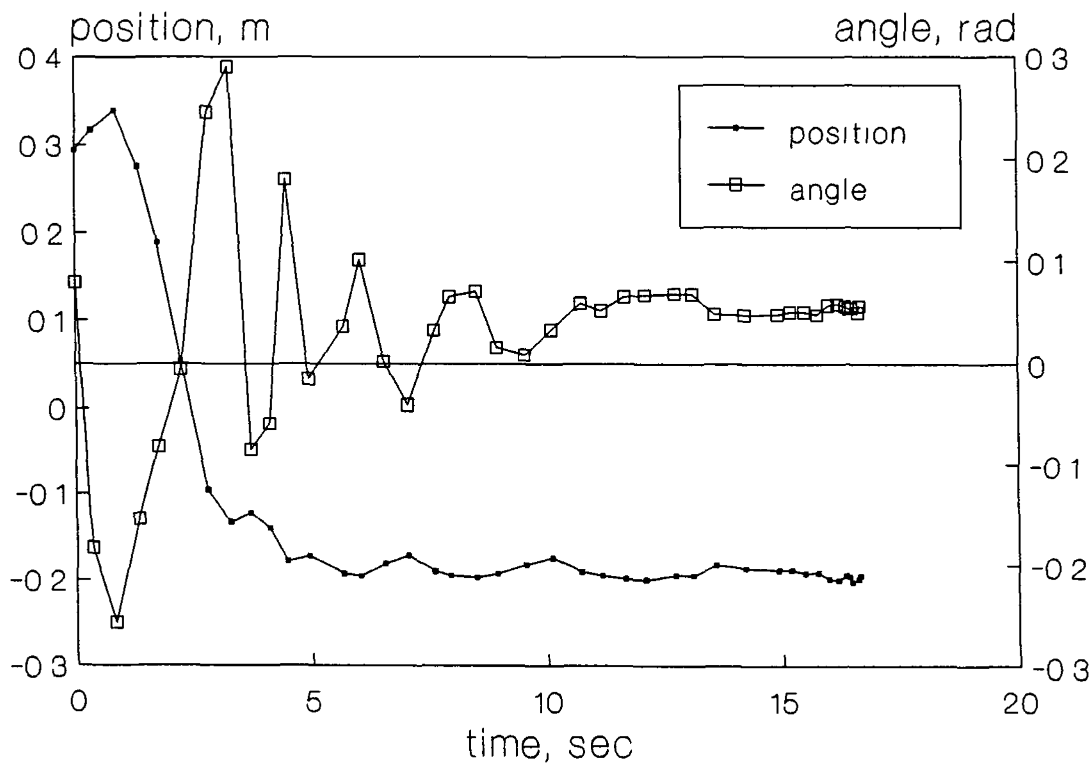


Figure 4.10 The first and second traınıng usıng neural network control. Inıtıal posıtıon: x=-0.368 m, α=-0 001 rad.

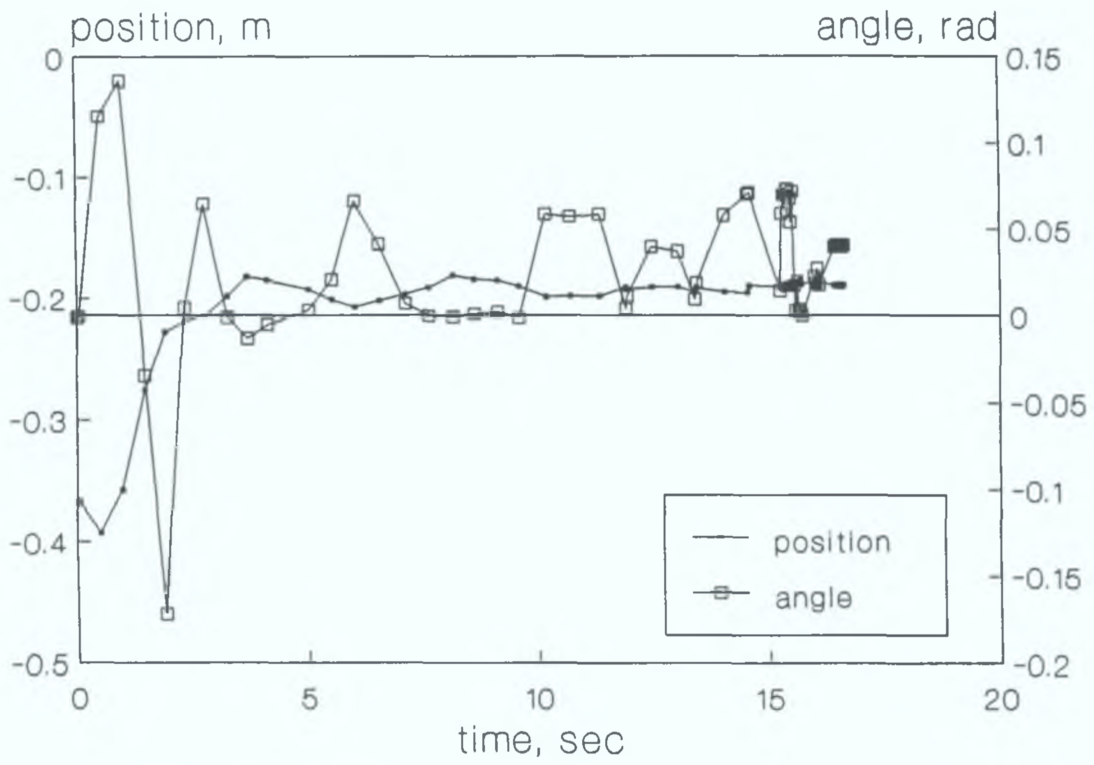


Figure 4.11 The third training run using neural network control. Initial position: x=0.444 m, α=-0.052 rad.

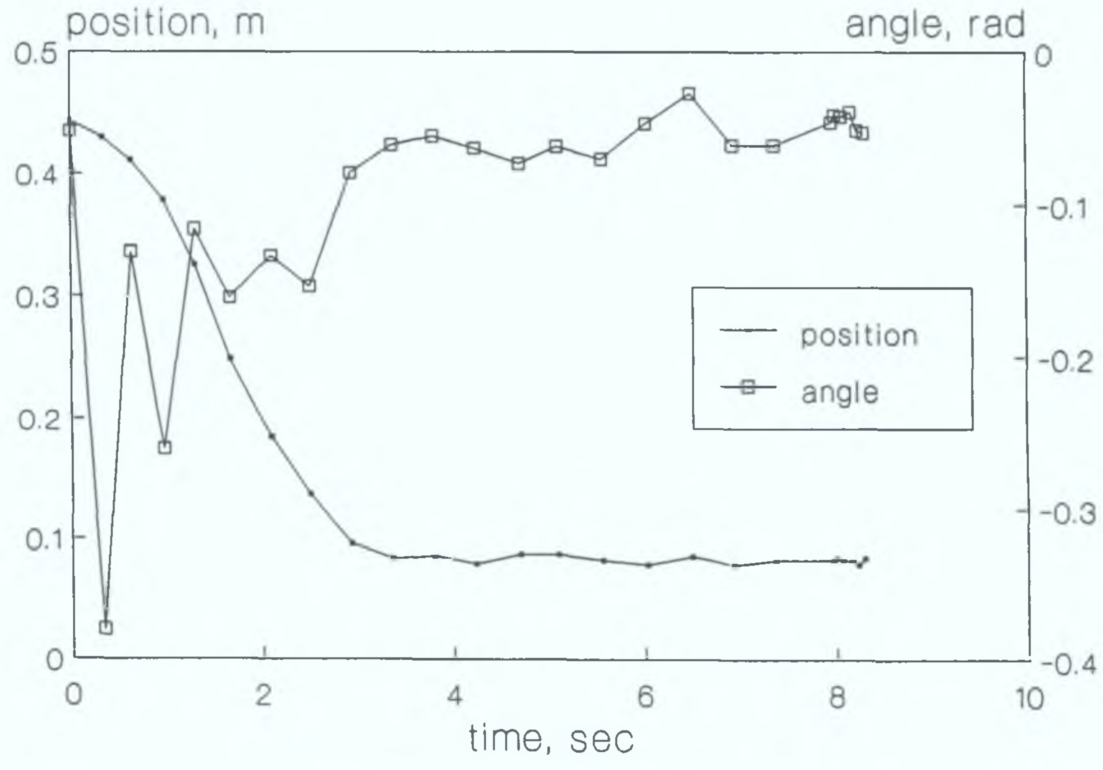


Figure 4.12 The third training run using neural network control. Initial position: x=-0.5 m, α=-0.204 rad.

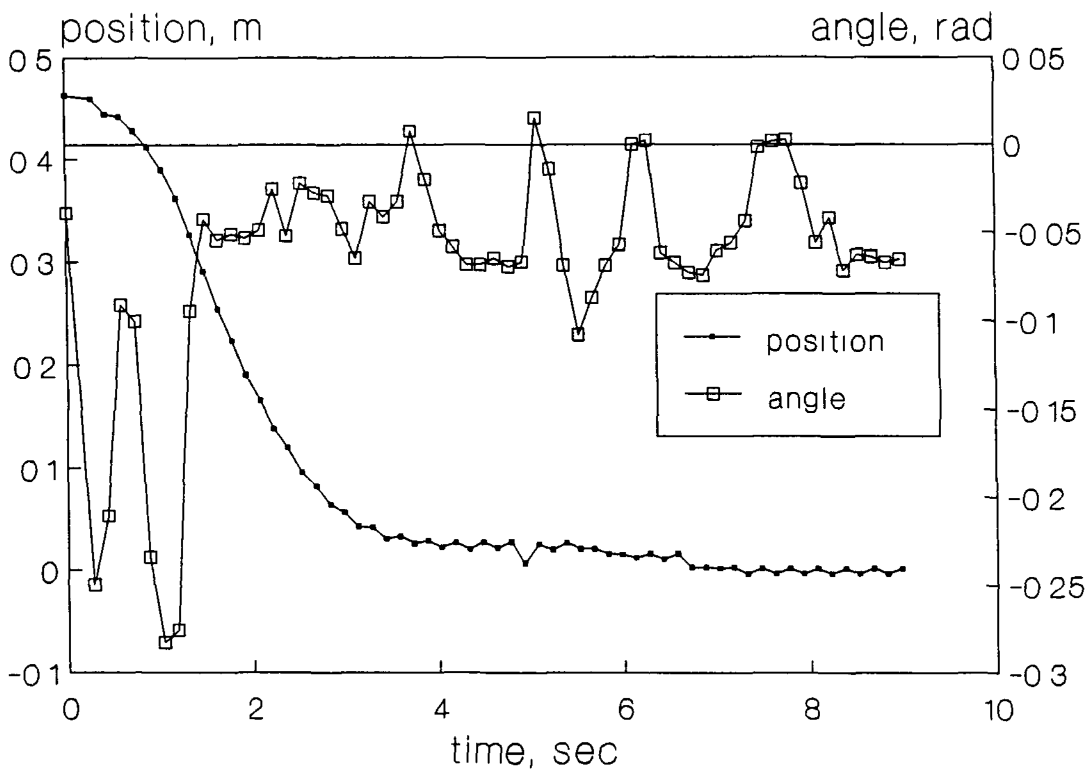


Figure 4.13 The third training run using neural network control. Initial position: x=0.463 m, α=-0.039 rad.

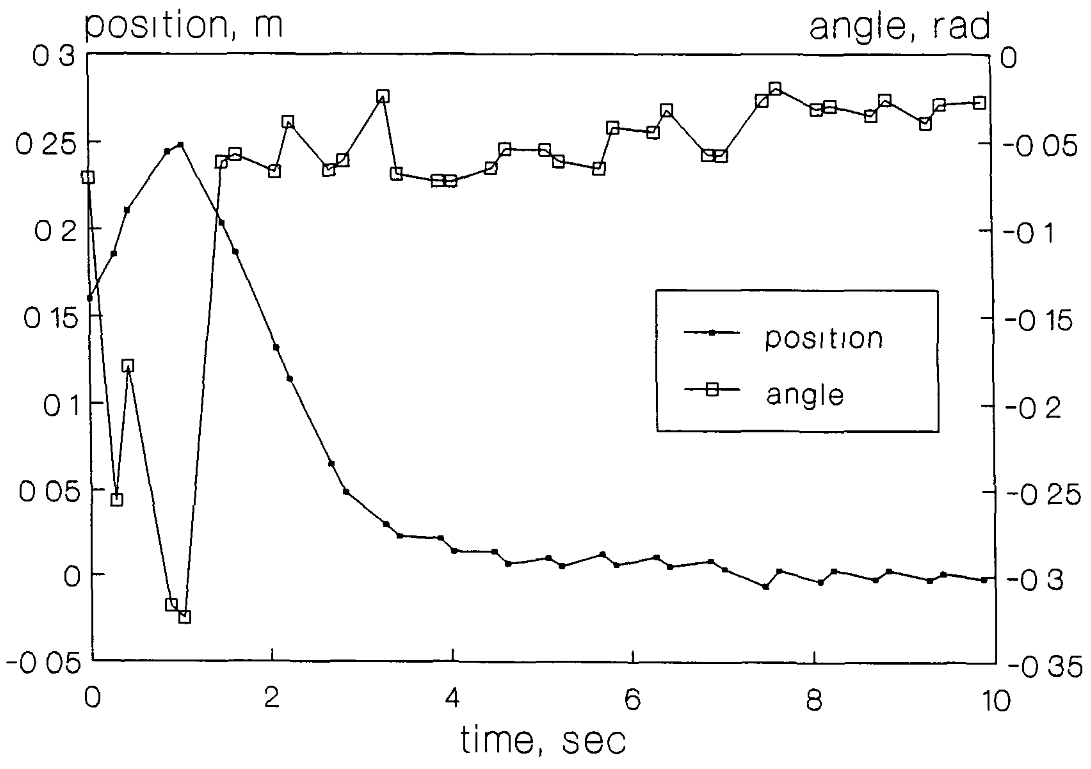


Figure 4.14 The third training run using neural network control. Initial position: x=0.160 m, α=-0.070 rad.

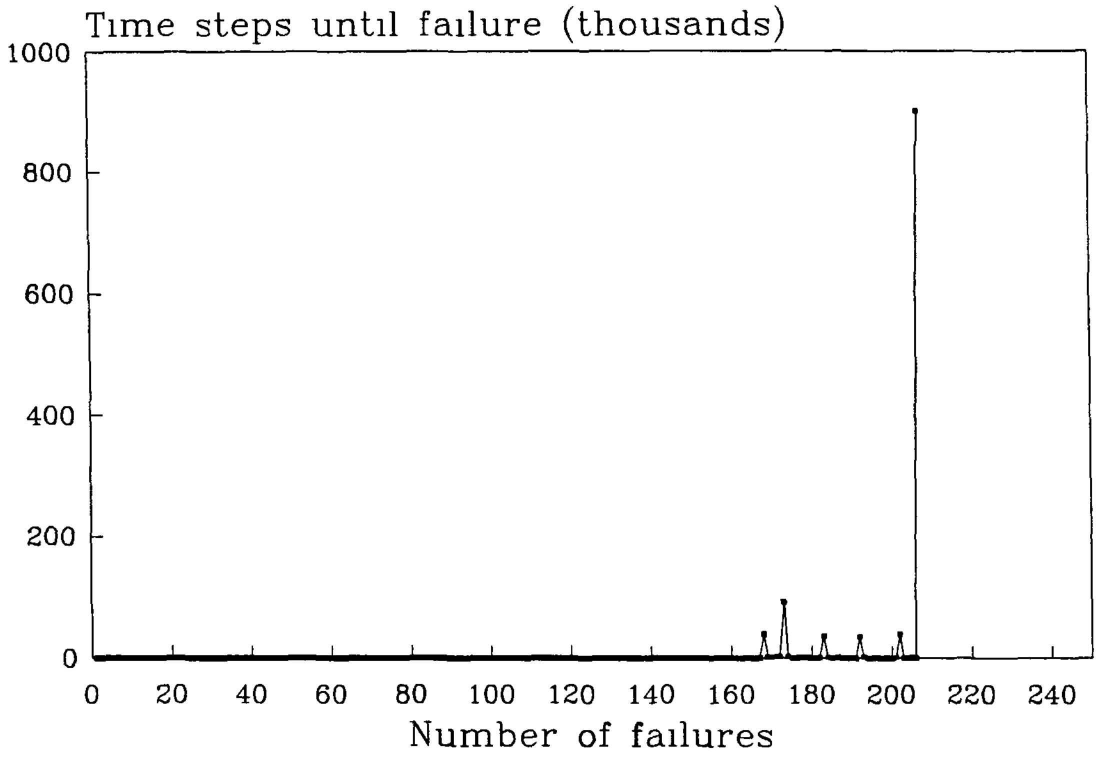


Figure 4.15 The first and second training run in neural network control.

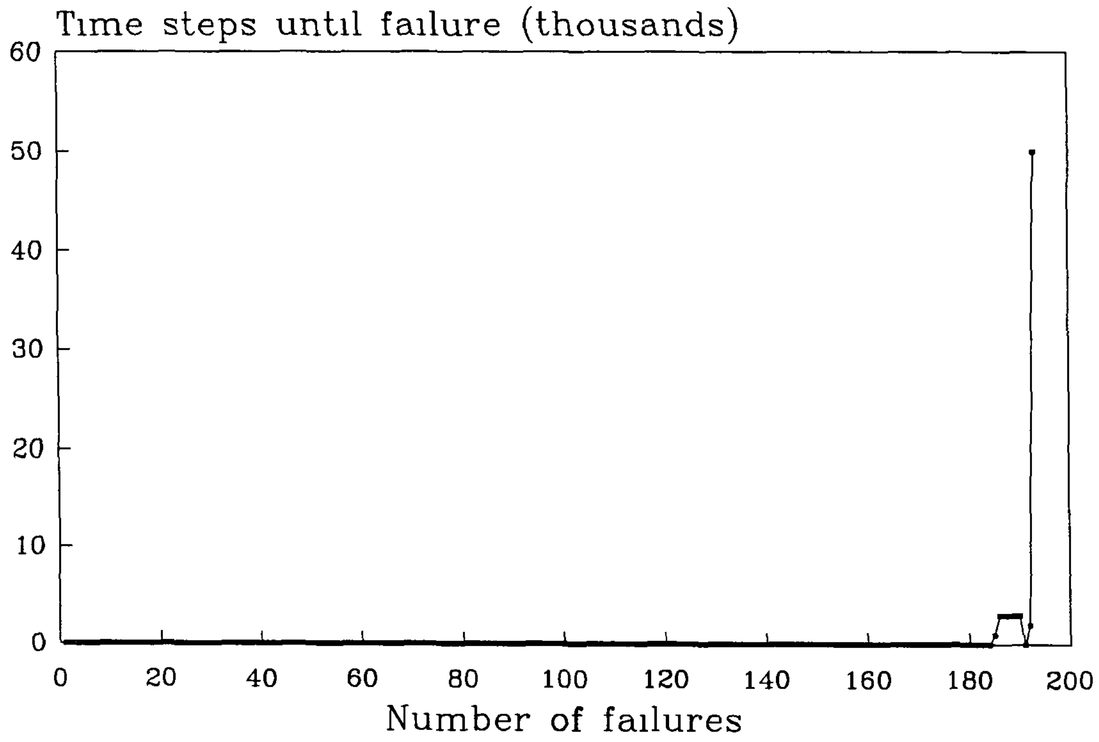


Figure 4.16 The first and second training run in neural network control.

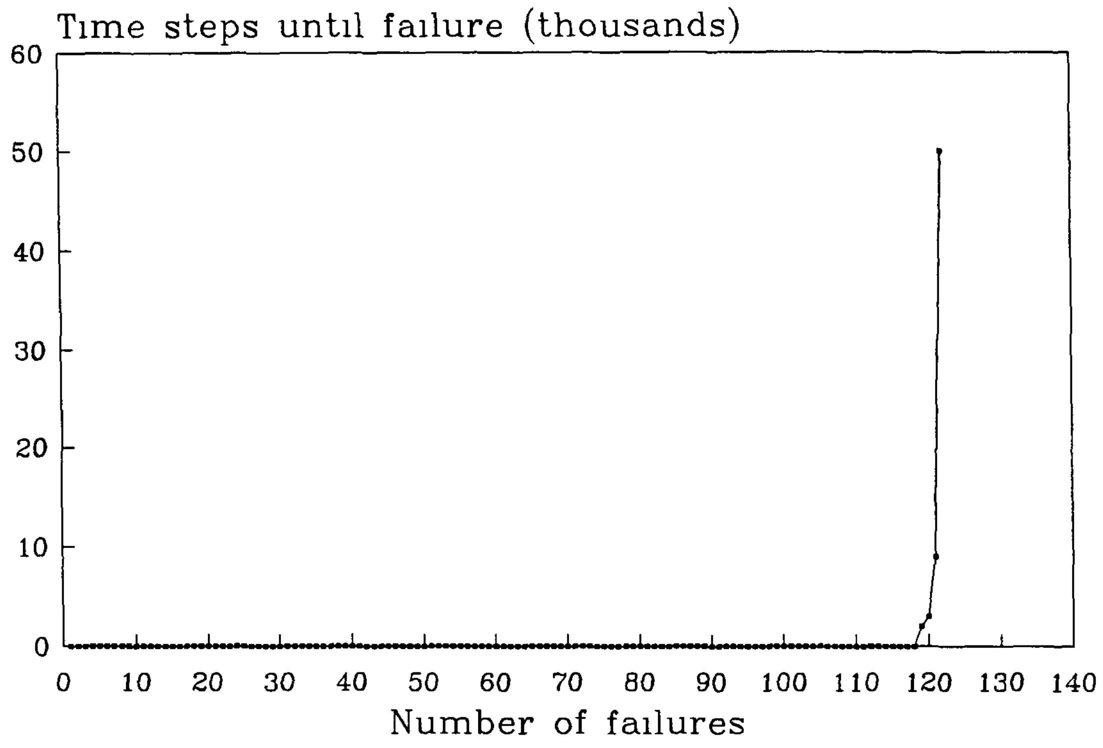


Figure 4.17 The third training run in neural network control.

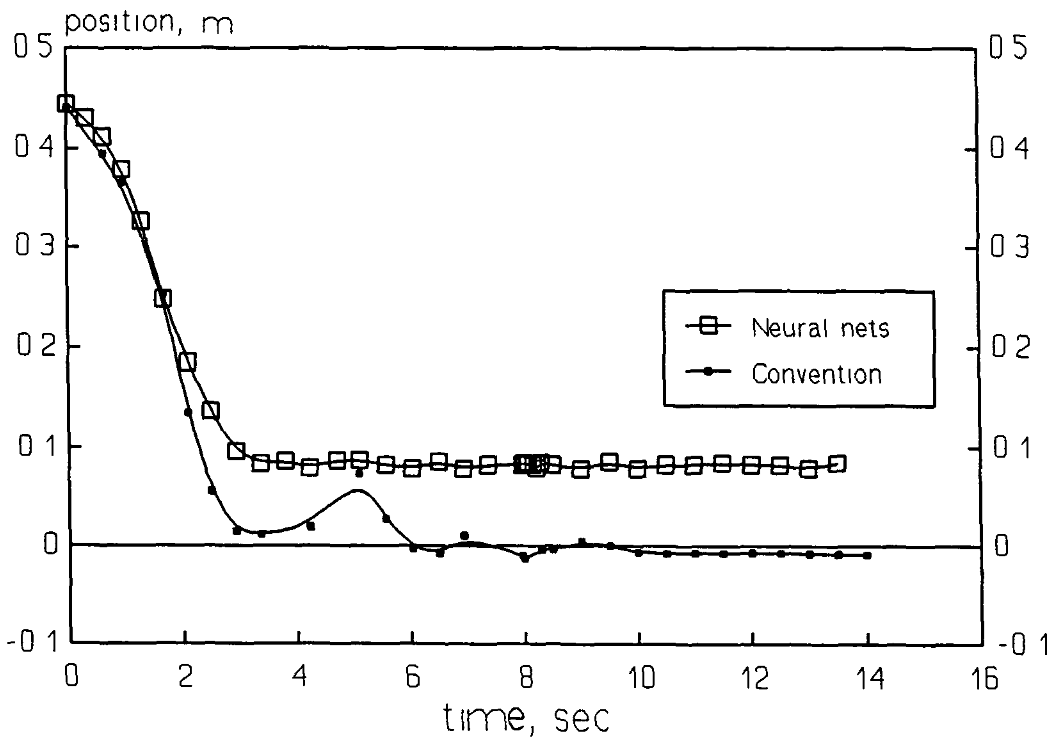


Figure 4.18 The neural network and conventional control comparison. Initial position: x=0.4 m.

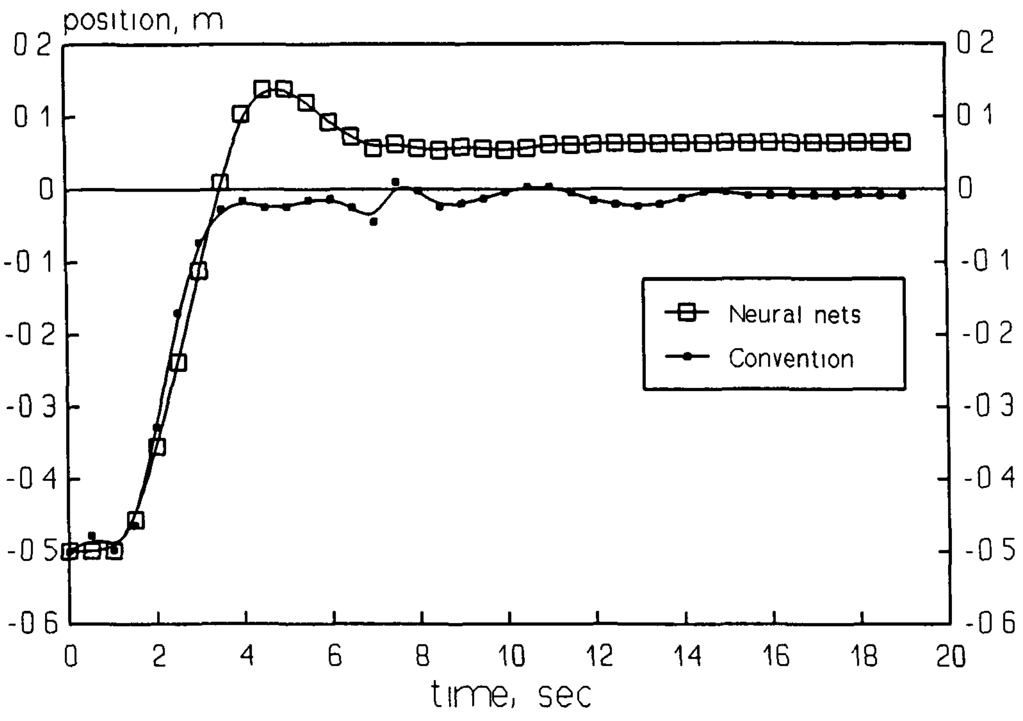


Figure 4.19 The neural network and conventional control comparison. Initial position: x=-0.12 m.

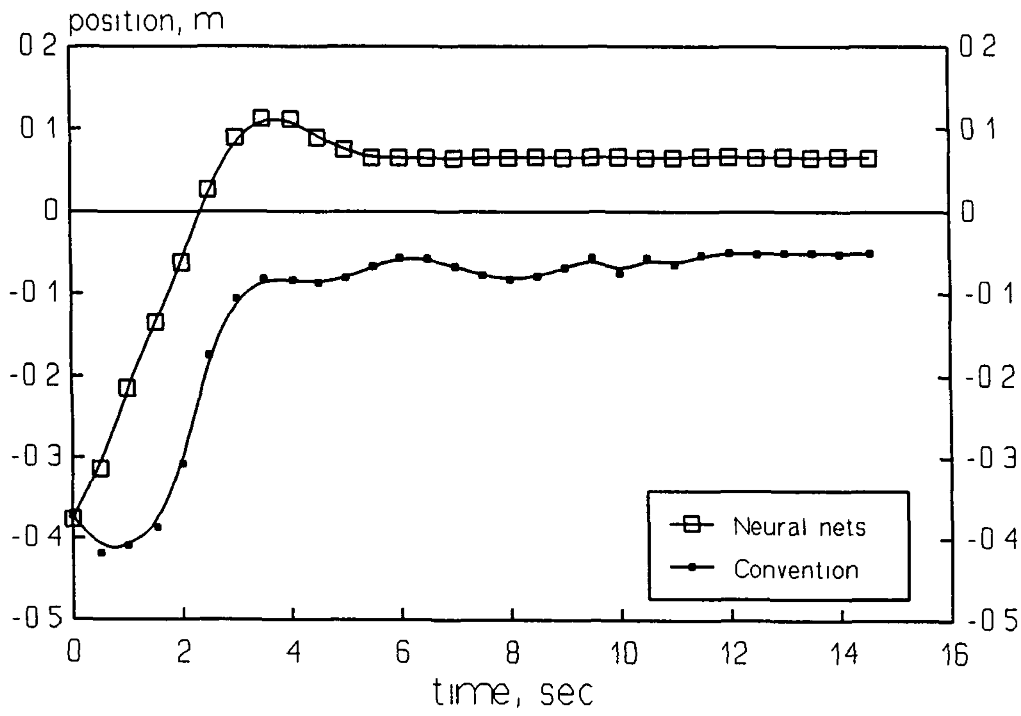


Figure 4.20 The neural network and conventional control comparison. Initial position. x=0.16 m.

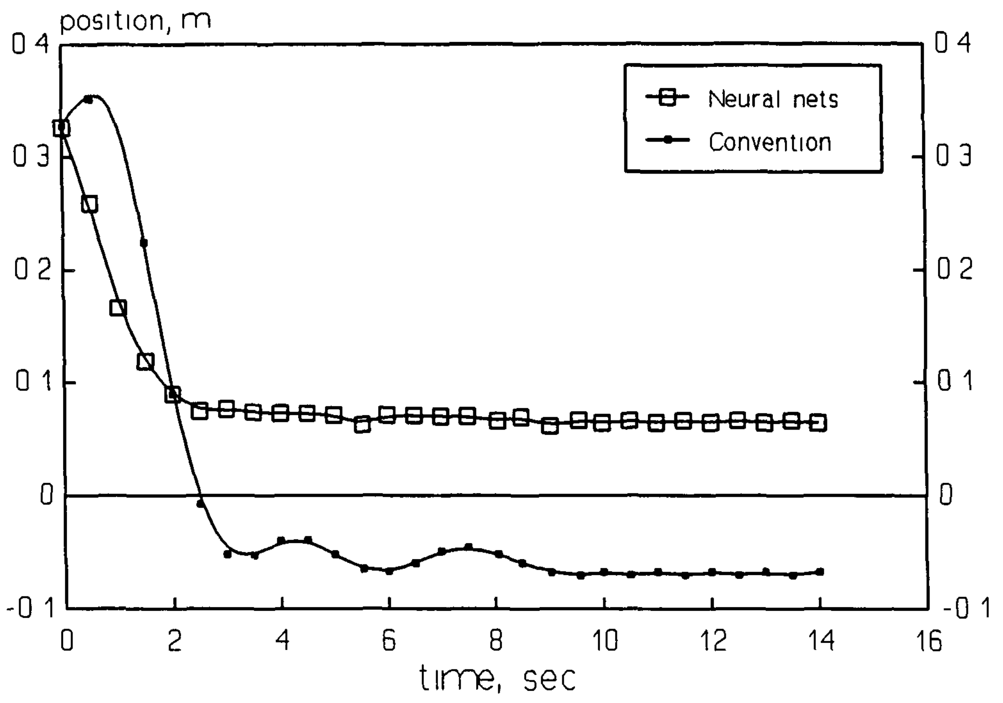


Figure 4.21 The neural network and conventional control comparison. Initial position: x=-0.4 rad.

4.5 Discussions

In this chapter, the neural networks and conventional control are used in real time control. In the conventional control. It took about 15-20 seconds to balance the ball and beam system. In the neural networks control, two-layered neural networks are used in the control experimental rig. Input states are ball position, ball velocity, beam angle, beam angular velocity. The output is an action force applied on the beam , action force $F[t]=p[t]|xo+x1|$. Ball position ranges from -0 5 to 0.5 meter, and the beam angle is between -20° - 20°. The system sample period is $\tau=0.02$ seconds. An action force is applied on the beam, which is controlled by the output from the action network. This force is determined by calculating the output of the action networks once for each action. The weights values are adjusted after each learning training, and these weights are used to retrain themselves. When the system is balanced, the weights values become constant values. It took about 11-12 seconds for the networks to learn to balance for the first and second training runs, and gradually reduced to 5-7 seconds for third to tenth training runs. It took about 11-12 seconds to balance the system for first training runs, while after several training runs, it only needed 5-7 seconds

The experimental results show that the neural networks control method was able to balance the ball beam system under all the initial conditions tested. The ball started moving from the initial position towards the balancing position, then when it was in the vicinity of it, the ball oscillated around it and as the amplitude decreases, balance was achieved. The balancing position can be changed by adjusting the potentiometer on the ball beam system.

It has been observed in the experiments that the amplitude of the actuating force should be properly chosen. If the force is too great, the ball will be tossed out of the wire track and if it is two small, it will not be able to balance the system. The amplitude of the actuating force was determined experimentally in this work. (see section 4.3.1)

CHAPTER 5 CONCLUSIONS

A neural networks techmque has been successfully apphed to the control of a balancmg system. A detaıled study of neural network control and conventıonal pole placement control apphed to the ball-beam system has been completed. System modellıng, sımulatıon, and controller ımplementatıon usıng a personal computer for control have been presented ın thıs thesıs.

A great advantage of neural network control system ıs that no pnor knowledge of system dynamıcs of plant ıs needed. The neural network determıned the actıon from the prevıous performance of the system, whıch ıs very much the same as the human neural system. To sımulate the neural network control system solely on computer, wıthout the ınvolvement of an " external" plant, ıt ıs necessary to sımulate the plant ıtself. Here we need to sımulate the ball-beam system. Thıs ıs done by ıntegratıng the system equatıons denved m chapter 3 (equation 3-15) for any gıven ınıtıal condıtions from whıch we can obtaın the ball lınear posıtıon and the beam angle. Thıs ıs sımılar to the process of measunng the real value. It should be noted however, the closeness of these values to real ones not only depends on the numencal method used, but also depends on the accuracy on the ongınal model.

5.1 System Modelling and Coefficient Measurements

The system equations have been established by using Lagrangian variational principle. The input to the system is an action-force on the beam, and the outputs from the system are the ball position, and velocity, and beam angle, and angular velocity. The relevant coefficients of the system have been determined experimentally.

5.2 <u>Simulation and Implementation of Conventional Method</u>

This model was simulated on a digital computer using Bass-gura feedback control method. It took about 1 second to balance the ball beam system using conventional simulation. It took about 15-20 second to balance the ball beam system in the experiment control.

5.3 <u>Simulation and Implementation of Neural Network Method</u>

The simulation of neural networks method means that a simulated neural network system is used to control a simulated plant. In the present work, the plant, which is the ball-beam system, has been simulated by the equations of motion (Equation from 3-3, 3-4). Euler method has been used to integrate the equations of motion to obtain the current positions of the ball and the beam. The neural network was simulated usmg software.

The results from simulation show that by adjusting the connection weight, the neural network was able to learn to balance. The time needed to balance reduced from 40 seconds for the first training run to 0.3-0 4 seconds for tenth training run.

The neural network control method has been used to control the actual rig. Neural networks are, again, simulated serially using a personal computer, while the plant is related by the apparatus. The experimental results show that in the first several training runs, the neural network method takes a longer time to balance the system than the conventional method. But after 4-5 training runs, the neural networks could balance the system within 6 seconds, while it took 15-20 seconds for the conventional method to do the same. In addition, since no system dynamics are involved in neural network method, it would be relatively easy to control similar systems with different parameters. For instance, it has been demonstrated successfully that balls with different diameters, hence different weights, can be balanced with effectually no change (the actuating force may need to be adjusted).

5.4 Further work

In the present work, the parallel processing neural network system was simulated using a serial digital computer. This no doubt has slowed down the system speed and degraded the performance. It is hoped that the neural network method presented here would be implemented using parallel hardware to exploit the full power of the neural network's ability to perform parallel processing.

One of the main subjects of future research on neural networks will be the improvement of the speed of learning. At present there are ways which can be used to speed up learning. One is to up grade the computer.

References

Almeida, L.B., A learnıng Rule for Asynchronous Perceptrons wıth Feedback ın a Combınatorıal Envıronment, *Proc. of the 1987 Intl, conf. on Neural Networks*, vol 2, pp. 609-618. 1987.

Anderson, C.W., Strategy Learnıng wıth Multılayer Connectıonıst Representatıon, *Tech. Rept.* TR87-509. 3, GTE Laboratorıes,Waltham, MA, 1087. (Thıs ıs a corrected versıon of the report publıshed ın Proc. Fourth Internatıonal Workshop on Machıne Learnıng, Irvıne, CA, pp.103-114, June 1987.)

Anderson, W., Learnıng to Control an Inverted Pendulum Usıng Neural Networks, *IEEE Control System Mag*, pp 31-37, Aprıl. 1989.

Anderson, C.W., Learnıng and problem solvıng wıth multılayer connectıonıst systems; *Doctoral Dıssertatıon*; Dept.of Computer and Informatıon scıence, Unıv.of Massachusetts (Amherst, MA).1986.

Antsaklis, P.J., Passino, K.M. and Wang, S.J., Towards Intellıgent Autonomous Control System: Archıtecture and Fundamental Issues, *J.Intell.Robotıc Syst.*, vol. 1, pp. 315-342, 1989; a shorter versıon appeared ın the *Proceedıngs of the Amerıcan Control Conference*, pp. 602-607, Atlanta, GA, June 15-17, 1988.

Antsaklis, J., Neural Networks ın Control System, *IEEE Control System Mag*. pp.3-5, Apnl 1990.

Arbib, M.A., Brains, Machines, and Mathematics, Springer, 1987.

Ash, T. Dynamic Node Creation in Backpropagation Networks; *ICS Report No. 8901*, Institute for Cognitive Science, University of California, San Diego.1989.

Astrom, K.J., Anton, J.J. and Arzen, K.E., *Expert Control, Automatica*, vol.22, no.3, pp.277-286, 1986.

Astrom, K.J., Adaptive Feedback Control, *Proc. IEEE*, vol.75, no.2, pp. 185-217, 1987.

Autsaklis, P.J., Passino, K.M. and Wang, S.J., Towards Intelligent Autonomous Control Systems, Architecture and Fundumental Issues. *Intel, Robotic Syst.* vol 1, pp. 315-342, 1989.

Barto, A.G. and Sutton, R.S., Goal Seeking Components for Adaptive Intelligence, *Air Force Wright Aeronautical Lab. AFWAL-TR-81-1070*, 1981.

Barto, A.G. and Anderson, C.W., Structural Learning in Connectionist Systems, *Proc. 7th Annual Conf. Cognitive Science Society*, pp. 43-53, 1985.

Barto, A.G., An Approach to Learning Control Surfaces by Connectionist Systems, *Vision, Brain and Cooperative Computation*, MIT Press (Cambridge), pp. 665-703, 1987.

Barto, A.G., Sutton, R.S. and Anderson C.W., Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 834-836, Sept.-Oct. 1983.

Barto, A.G., Sutton, R.S. and Brouwer,P.S., Associative Search Network: A Reinforcement Learning Associative Memory, *Biol. Cybern.*, Vol.40, pp.201-211, 1981.

Barto, A.G. and Sutton, R.S., Simulation of Anticipatory Respouses in Classical Conditioning by a Neuron-like Adaptive Element, *Behavioral Brain Research*, vol.4, pp. 221-235, 1982.

Baum, E.B. and Hanssler, D., What Size Net Gives Valid Generalization?, *Neural Computation*, vol.1, pp.151-160, 1989.

Bavarian,B., Guest Editor of Special Section on Neural Networks for Systems and Control, *IEEE Contr. Syst. Mag* , vol.8, no.2, pp.3-31, Apr. 1988.

Carpenter, G. A. and Grossberg, S, Self-organization of Stable Category Recognition Code for Analog Input Pattern, *Proc Int. Conf. Neural Networks*, Vol II, pp. 727-735, 1987.

Cheok, K.C. and Loh, N.K., A Ball-Balancıng Demonstratıon of Optımal and Dısturbance-Accommodating Control, *IEEE Contr. Syst. Mag.*, vol. 7, no. 1, pp. 54-57, Feb. 1987.

Chu, R., Shoureshi, R. and Tenorio, M., Neural Networks for System Identıficatıon. *IEEE Control System Mag*, Aprıl 1990.

Cohen, M.A. and Grossberg, S., Absolute Stabılıty of Global Patten Formatıon and Parallel Memory Storage by Competıtıve Neural Networks, *IEEE Trans. Syst., Man, Cyber.*, vol.13, no.5, pp.815-826, 1983.

Ei_Leithy, N. and Newcomb, R.N., Guest Edıtors of Specıal Issue on Neural Networks, *IEEE Trans Cırc. Syst* , vol.36, no.5, May 1989

Fu, K.S., Learnıng control systems-Revıew and outlook. *IEEE Transactıons on Automatıc Control*, pages 210-221, 1970.

Fukushima, K. Cognıtron: A Self-Organızıng Multılayered Neural Network, *Bıol. Cyber* , Vol.20, pp.121-136, 1975.

Grossberg, S., Nonlınear Dıfference-Dıfferentıal Equatıons ın Predıctıon and Learnıng Theory, *Proc. Nat. Acad. Scı.*, 58, pp 1329-1334, 1967.

Grossberg, S. A Theory of Human Memory: Self-Organızatıon and Performance of

Sensory-Motor Codes, Maps, and Plans, *Progress ın Theoretıcal Biology* (Edıtors: Resen R. and Snell F.), Vol 5, Academıc press, New York, pp. 233-374, 1978.

Grossberg, S., Studıes of Mınd and Braın, Boston: Reıdel, 1982.

Grossberg, S. Adaptıve Pattern Classıficatıon and Unıversal Recodıng II: Feedback, Expectatıon, Olfactıon and Illusıons, *Bıologıcal Cybernetıcs*, Vol. 23, pp. 187-202, 1976.

Hebb, D.O., The Organızatıon of the Behavıour, Wıley (NEW YORK), 1949

Hecht-Nielsen, R., Performance Lımıtes of Optıcal, Electro-Optıcal and Electronıc Artıficıal Neural System Processors, *Proc. SPIE*, vol.624, pp.277-306, 1986.

Hinton, G.E., Connectıonıst learnıng procedures; *Technıcal Report CMU-CS-87-115* (versıon 2); Computer Scıence Department, Carnegıemellon Unıversıty. 1987.

Hopfield, J.J. Neural Networks and Physıcal Systems wıth Emergent Collectıve Computatıonal Abılıtıes, *Proc.nat.Acad. Scı.*, vol. 79, pp.1554-1558, Apr. 1982.

Hopfield, J.J., Neurons wıth Graded Response Have Collectıve Computatıonal Propertıes Lıke Those of Two-State Neurons, *Proc. of the Natl. Acad. Sci, USA*, vol. 81, pp. 3088-3092. 1984.

Hopfield, J.J. and Tank, D.W., Computıng wıth Neural Cırcuıts: a Model, *Scıence*, Vol 233, pp. 625-533, 1986.

Hopfield, J.J. and Tank, D.W., Neural Computatıon of Decısıons ın Optımızatıon Problems, *Bıol Cyber.*, vol.52, no.3, pp.1-25, July 1985.

Hornik, K., Stinchombe, M. and White, H., Multı-layer Feedforward Networks are Unıversal Approxımators, 1989.

Kleene, S.C., Representatıons of Events m Nerve Nets and Fınıte Automata, *Automata Studıes*, Prınceton Unıversıty Press (Prınceton); pp. 3-40, 1956.

Kohonen, T., A Sımple Paradıgm for the Self-Orgnızed for Motıon of Structured Feature Maps, *Amarıs and Arbıb M.A.*, pp. 248-266, 1982.

Kohonen, T., Self-Orgnızatıon and Assocıatıve Memory, Sprınger, 1988.

Kohonen, T. Assocıatıve Memory, Berlın: Sprınger-Verlag, 1977.

Kosko, B.A., Adaptıve Bıdırectıonal Assocıatıve Memorıes, *Appl. Opt.*, vol.26, no.23, pp.4947-4960, Dec, 1987.

Kraft, L.G. and Campagna, D.P., A Comparıson Between CAMC Neural Network Control and Two Tradıtıonal Adaptıve Control Systems, *IEEE Control System Mag.*

Apnl 1990.

Linsker, R., From basic Network Pnnciples to Neural Archıtecture: Emergence of Spatial-opponent Cells, *Proc. Natl. Acad. Sci, USA,* Vol.83, pp7508-7512, 1988.

Linsker, R., Self-organızation ın a Perceptual Network, *Computer,* Vol. 21, pp. 105-117, 1988.

Lippmann, R.L., An Introduction to Computıng with Neural Nets, *IEEE Acoustıcs, Speech, Sıgnal Proc. Mag.*, pp.4-22, Apr. 1987.

Mayr, O., The Ongıns of Feedback Control, Cambndge, MA. MAT Press,1970.

McClelland, J.L., Rumelhert D.E. and Hinton G.E., The Appeal of Parallel Dıstnbuted Processıng, *Rumelhert D.E. and McClelland, J.L.*, pp.3-44.1986.

Mendel, J.M. and Fu, K.S. Adaptıve, Learnıng, and Pattern Recognıtıon Systems: Theory and Applıcatıons. Academıc Press, New York,1970.

Minsky, M. and Papert, S., Perceptrons. An Introductıon to Computatıonal Geometry; MIT Press, 1969.

Nagata, S., Sekiguchi, M., and Asakawa, K., Mobıle Robot Control by a Structured Hıerarchıcal Neural, *IEEE Control System Mag*, Apnl 1990.

Nilsson, N.J., Learnıng Machınes: Foundatıons of Traınable Pattern Classıfyıng Systems. McGraw-Hıll New York.1965.

Parker, D.B., Learnıng -Logıc; *Inventıon Report*, S81-64, File 1; Office of Technology Lıcensıng, Stanford Unıversıty. 1982.

Pineda, F.J., Generalızatıon of back-propagatıon to recurrent networks, *Physical Revıew Letters*, vol.59, no.19, pp.2229-2232. 1987.

Psaltis, D., Sideris, A., and Yamamura, A.A., A Multılayered Neural Network Controller. *IEEE control system mag.* pp.17-21. Aprıl, 1988.

Rohwer, R. and Forrest, B. Traınıng Tıme-dependence ın Neural Networks, *Proc. of the Intl.Conf. on neural networks*. 1987.

Rosenblatt, R., Prıncıples of Neurodynamıcs. New York: Spartan Books, 1959.

Rosenblatt, F., Prıncıples of Neurodynamıcs and the Theory of Braın Mechanısms, Spartan Books (Washıngton D.C.), 1962.

Rumelhart, D.E. and McClelland, J.L., Parallel Dıstrıbuted Processıng: exploratıons ın the Mıcrostructure of Cognıtıon. vols.I and II,'MAT Press, 1986.

Samuel, G.N., Saridis and Gillbert, H.D., Self-Organizing approach to the stochastic fuel regulator problem. *IEEE Transactions on System, Man, and Cybernetics*, 6: 186-191, 1970.

Sanner, R.M. and Akin, D.L., Neu-romorphic Pitch Attitude reghlation of an Underwater Telerobot, *IEEE Control System Mag*, April 1990.

Shriver, B.D., Guest Editor of Special Issue on Artificial Neural Systems, *IEEE Computer*, vol.21, no.3, Mar. 1988.

Special Section on Neural Networks for Control System, *IEEE Contr. Syst. Mag.*, vol. 9, no. 3, pp. 25-59, Apr. 1989.

Stotzka, R. and Maenner, R., Self-organization of A Linear Multilayered Feedforward Neural Network, *Physics Institut,* University of Heidelberg, FRG, 1989.

Sutton, R.S. and Barto, A.G., Toward a Modern Theory of Adaptive Networks: Expectation and Prediction, *Psychol. Rev.*, vol. 88, no. 2, pp.135-170, 1981.

Sutton, R.S., Learning to Predict by the Methods of Temporal differences, *Machine Learning*, vol. 3, pp. 9-44, 1988.

Sutton, R.S., Temporal Credit Assignment in Reinforcement Learning, *Doctoral Dissertation*, COINS Tech. Rept. 84-02,Univ.of Massachusetts, Amherst, 1984.

Waltz, M.D. and Fu, K.S., A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10:390-398, 1965.

Werbos P.J., Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, *Thesis in Applied Methematics*, Harvard University, 1974

Widrow, B. and Hoff, M.E. Adaptive Switching Circuits, *IRE WESCON Convention Record*, pp.96-104. 1960.

Williams R.J., Reinforcement Learning in Connectionist Networks, A Mathematical Analysis, *ICS Report 8605*, Institute for Cognitive Science, University of California, San Diego, 1986.

APPENDIX A

The dynamics of the ball beam system are given by the following equations of motion. All angular measurements are given in radians.

$$\dot{x}_2 = \frac{g\,x_3 + x_2\,x_4}{1+\dfrac{I_b}{m\,r^2}}$$

$$\dot{x}_4 = \frac{m\,g\,x_1 - F\,l - \dfrac{I_b}{r\,\zeta} - 2\,m\,x_1\,x_2\,x_4}{m\,x_1^2 + I_b + I_a}$$

where

$$\zeta = \frac{g\,x_3 + x_2\,x_4^2}{1 + \dfrac{I_b}{m\,r^2}}$$

This system was simulated on a digital computer by numerically approximating the equations of motion using Euler's method with a time step τ =0.02 seconds and the following discrete-time state equations:

$$x_1[(k+1)\,\tau] = x_1[k\tau] + \tau\,x_2[k\tau]$$

$$x_2[(k+1)\,\tau] = x_2[k\tau] + \tau\,\dot{x}_2[k\tau]$$

$$x_3[(k+1)\,\tau] = x_3[k\tau] + \tau x_4[k\tau]$$

$$x_4[(k+1)\,\tau] = x_4[k\tau] + \tau\dot{x}_4[k\tau]$$

The sampling rate of the ball beam system's state and the rate at which control forces are applied are the same as the basic simulation rate, i.e., 50 Hz.

APPENDIX B

The eigenvalue-eigenvector method is usually used in the solution of higher order systems.

The open-loop discrete-time model is given by

$$x[(k+1)\tau] = \mathbf{A}x[k\tau] + \mathbf{B}F[k\tau]$$

$$= \begin{bmatrix} 0.805 & 0.0201 & 0.001129 & 0 \\ 0 & 0\,805 & 0\,11286 & -0.312 \\ 0 & 0 & 1 & -0.333 \\ 0 & 0 & 0.0067 & 0.999 \end{bmatrix} x[k\tau] + \begin{bmatrix} 0 \\ -0\,000562 \\ -0.000598 \\ -0.0199 \end{bmatrix} F[k\tau]$$

$$y[k\tau] = C\, x[k\tau]$$

where $\phi(\tau) = e^{A\tau}$, $\quad \theta(\tau) = \int_0^{\tau} \phi\,(\tau)\,B\,d\tau$

By applying the eigenvalue-eigenvector method, we obtain

where

$$\hat{\theta}(\tau) = \int_{k\tau}^{(k+1)\tau} \phi^{T}(t-k\tau)\ \theta(k\tau)\ \phi(t-k\tau)\ dt$$

$$\mathbf{M}(\tau) = \int_{k\tau}^{(k+1)\tau} \phi^{T}(t-k\tau)\ \theta(k\tau)\ \theta(t-k\tau)\ dt$$

$$\hat{\mathbf{R}}(\tau) = \int_{k\tau}^{(k+1)\tau} \theta^{T}(t-k\tau)\ \theta(k\tau)\,\theta(t-k\tau)\ dt$$

$$\Omega = \phi - \theta\hat{\mathbf{R}}^{-1}\mathbf{M}^{T}$$

$$\Gamma = \theta - \mathbf{M}\,\hat{\mathbf{R}} + \mathbf{M}^{T}$$

Thus the eigenvalue can be determined by

$$[z\mathbf{I} - \mathbf{V}]=0$$

where the matrix $\mathbf{V}$ is given by

$$\mathbf{V} = \begin{bmatrix} \mathbf{\Omega}^{-1} & \mathbf{\Omega}^{-1}\,\boldsymbol{\theta}\,\hat{\mathbf{R}}^{-1}\,\boldsymbol{\theta}^{T} \\ \mathbf{\Gamma}\,\mathbf{\Omega}^{-1} & \mathbf{\Omega}^{T} + \mathbf{\Gamma}\,\mathbf{\Omega}^{-1}\,\boldsymbol{\theta}\,\hat{\mathbf{R}}^{-1}\,\boldsymbol{\theta}^{T} \end{bmatrix}$$

The Equations above were used in programming, and the eigenvalues are

$$z = [\, z_1,\, z_2,\, z_3,\, z_4] = [\, -0.075,\ -1.4125,\ -0.9+j2.85, -0.9-j2.85] \ .$$

APPENDIX C

PROGRAMS LISTS

```
   ****************************************************
           Simulation Using Conventional Method
   ****************************************************

 #include <math.h>
 #include <stdio.h>
 #include <dos.h>

main()
{
 char strin[50], strout[50];
 FILE *iop, *iop1;
 double fmod(double x, double y);
 float x[4],v[4],Force,l[4],Q,k,sample,result[4],i,j,time;


          /* Open input/output file */

 printf("Input file name:");
 gets(strin);
 printf("Output file name:");
 gets(strout);
 iop=fopen(strin,"r");
 iop1=fopen(strout,"w");

             /* Read in data */

 (void)    fscanf(iop,"    %f    %f    %f    %f    \    n",
&x[0],&x[1],&x[2],&x[3]);
  fprintf(iop1," t    x[t] \n");
  Q=0.0,time=0.0,Force=0.0, sample=0.02;
  v[0]=0.226, v[1]=0.0284,v[2]=0.000174,v[3]=0.104;
    fprintf ( iop1,"%.6f   %.6f    %.6f \n", Q,    x[0],
x[2]);

loop1:
   if (time<=100.0)
     {
      x[0]=0.8951*x[0]-2.514*x[1]-0.000177*x[2];
      x[1]=0.129*x[0]+0.651*x[1]+0.000144*x[2];
      x[2]=-0.0759*x[0]+0.843*x[2];
      x[3]=-4.634*x[0]+0.0685*x[1]-3.396*x[2]+0.896*x[3];

     x[0]=x[0]+v[0]*Force;
     x[1]=x[1]+v[1]*Force;
     x[2]=x[2]+v[2]*Force;
     x[3]=x[3]+v[3]*Force;
     if(fabs(x[0])>0.001 || fabs(x[2])>0.01)
       {
         Force=-2.0;
```

```
          }
          else
          {
            Force=2.0;
          }
        time++;
        Q=time*o;
        if(fmod(time,2)==0)
        {
          fprintf ( iop1,"%.3f  %.6f  %.6f \n", Q,  x[0], x[2]);
        }
        goto loop1;
        }
        else
        {
        fclose(iop);
        fclose(iop1);
        }
}
```

```
          ******************************************
               Simulation Using Neural Networks
          *******************************************

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
main()
{
char      strin[100],   strout1[50],   s t r o u t 2 [ 5 0 ] ,
          strout3[50],  strout4[50],   strout5[50];
 FILE     *iop, *iop1,*iop2,*iop3,*iop4,*iop5;
 extern double exp();
 double   fabs(double x);
 unsigned long k,maxk_step,k_fail;
 int      i,j,times,q,min_step,fail_num,max_fail;
 float     x[5],    z1[5],    z[5],     y2[5],    y1[5],
          y22[5],   x1[5],    c[5],     b[5],     e[5],
          f[5],     ff[5],    d1[5][5], d[5][5];
 float     p,       beida,    push,     limit,    Ib,  Ia,
          mm,       rr,       l,        bb,       kk,  r1,
          satime,   temp2,    low,      high,     g,   xx,
          yy,       yy1,      zz,       zz1,      zz2,
          templl,   vv,       gama,     r,        lou,
          loue,     1         o        u         f        ,
          louh,     v,        s,       m11,      beidah,
          sgnc,     sgnf,     pp,       x_lim,
          angle_lim,    temp22,   t;

 float a[5][5]={{-0.63912,-0.06556,0.082999,0.05223,-0.96392
 },
             {-0.09169, 0.06309,0.05288, 0.09191, -0.01464},
             { 0.06791, 0.06218, 0.020951, 0.01994, 0.044 },
             {-0.01876, 0.03425, -0.00178,-0.0311,-0.047118
 },
             {-0.01539,-0.06726,    0.02909,    -0.09939,
 -0.0462}};

  extern int rand();

       /*------------ Open Input/Output File ------------ */

   printf("Input file name:");
   gets(strin);
   printf("Output fail_num and k_fail number file name:");
   gets(strout1);
   printf("Output d[i][j] and a[i][j] weight file name:");
   gets(strout2);
   printf("Output state e[i],f[i]...:");
   gets(strout3);
   printf("Output state f[i]:");
   gets(strout4);
   printf("Output state variables:");
   gets(strout5);
   iop=fopen(strin,"r");
  iop1=fopen(strout1,"w");
  iop2=fopen(strout2,"w");
  iop3=fopen(strout3,"w");
  iop4=fopen(strout4,"w");
  iop5=fopen(strout5,"w");

       /* ---------- Read In Data ---------- */

  (void)   fscanf(iop,"%f,%f,%f,%f,%f,%f
\n",&lou,&loue,&louh,&beida,&beidah,&gama);
 (void) fscanf(iop,"%f,%f,%d \n",&x_lim,&angle_lim,&limit);
  (void)   fscanf(iop,"%d,%d,%d   \n",
&times,&min_step,&max_fail);
 (void)   fscanf(iop,"%f,%f,%f,%f,%f,%f,%f,%f         \n",
&Ib,&Ia,&mm,&l,&rr,&kk,&bb,&g);
 (void)   fscanf(iop,"%f,%f,%f,%f          \n",
&x[0],&x[1],&x[2],&x[3]);

       /*  --- Input random limit and initial weights-- */

          low=-0.1;  high=0.1;    push=0.0;

           for(i=0;i<5;i++)
           {
           z1[i]=(rand()/32767.0)*(high-low)+low;
           }

      /* ----- zero  state -----*/

k=0.0, t=0.0,  fail_num=0, k_fail=0, temp2=0.0, temp11=0.0,
satime=0.02,vv=0.0,     s=0.0,    p=0.0,     maxk_step=300;
r1=0.0,  x[4]=0.5,        x1[0]=0.58,  x1[1]=0.1,x1[2]=0.5,
x1[3]=0.1; x1[4]=0.5;
e[0]=-0.05484,e[1]=-0.08159,   e[2]=-0.05249,e[3]=0.2247,
e[4]=0.0949;
f[0]=-2.7037,f[1]=-2.65893,f[2]=-1.85765,    f[3]=10.16119,
f[4]=-2.49699;
b[0]=0.0207,b[1]=-0.03322,b[2]=-0.00118,    b[3]=-0.0852,
b[4]=0.0606;
c[0]=-3.98423,c[1]=0.0610,c[2]=-0.0172,   c[3]=-0.01803,
c[4]=-0.02898;

d[0][0]=-0.00255,d[0][1]=0.03012,d[0][2]=-0.08009,d[0][3]=-
0.66811,d[0][4]=0.12363;
d[1][0]=-0.07688,   d[1][1]=0.02718,   d[1][2]=0.030437,
d[1][3]=-0.64686,d[1][4]=0.20023;
d[2][0]=0.074733,   d[2][1]=-0.0199,   d[2][2]=-0.003387,
d[2][3]=-0.58907,d[2][4]=0.03698;
d[3][0]=-0.08559,   d[3][1]=-0.01518,   d[3][2]=-0.023125,
d[3][3]=-0.528729,d[3][4]=0.05066;
d[4][0]=0.02908,   d[4][1]=-0.00247,   d[4][2]=-0.00112,
d[4][3]=-0.54343,d[4][4]=0.106;

  /*--------  Using Euler's method  soluting system function
          and determine output state limit
```

```
        ---------- */

loop1:
                if(f[i]>=0.0){
                      sgnf=1.0;}
                   else{
                      sgnf=-1.0;}
                if(c[i]>=0.0){
                      sgnc=1.0;}
                   else{
                      sgnc=-1.0;}
   if(k!=0)
     {
       if(fabs(x[2])>=angle_lim || fabs(x[0])>=x_lim)   /*
have failure */
         {
            for(i=0;i<5;i++)
           {
           x[i]=(rand()/32767.0)*(high-low)+low;
              }
      }
     }
     xx=(g*x[2]+x[1]*x[3]*x[3])/(1.0+Ib/(mm*rr*rr));
     yy=mm*x[0]*x[0]+Ib+Ia;

zz2=mm*g*x[0]-Ib*xx/rr-2*mm*x[0]*x[1]*x[3]-kk*l*l*x[2]-bb*l
*l*x[3];
     zz1=push*l;
     zz=zz2-zz1;
     yy1=zz/yy;
     if(x[2]>0.0){
        x[0]=-fabs(x[0]);}
      else{
        x[0]=fabs(x[0]);}
     fprintf(iop1,"%0.3f   %0.6f   %0.6f \n ",t, x[0],x[2]);

     x[0]+=satime*x[1];  x[1]+=satime*xx;
     x[2]+=satime*x[3];  x[3]+=satime*yy1;


       if(x[0]>x_lim){
           x[0]=x_lim;}
         else{
           if(x[0]<-x_lim){
            x[0]=-x_lim;}
          else{
            x[0]=x[0];}}
       if(x[2]>angle_lim)   {
           x[2]=angle_lim;}
         else {
           if(x[2]<-angle_lim){
            x[2]=-angle_lim;}
          else{
          {
            x[2]=x[2];}}
       if(x[1]>0.05){
           x[1]=0.05;
         else{
           if(x[1]<-0.05){
            x[1]=-0.05;}
          else{
            x[1]=x[1];}}
       if(x[3]>1.){
           x[3]=1.;}
         else{
           if(x[3]<-1.){
            x[3]=-1.;}
          else{
            x[3]=x[3];}}

        x[4]=0.5;


       /*   -------  Output  sate  ealuation  in  evaluation
network -----*/

   for (i=0;i<5;i++){
        temp11=0.0;   temp22=0.0;
         for (j=0;j<5;j++){
              temp11 += a[i][j]*x[j];
              temp22 += a[i][j]*x1[j];}
                if(temp11>6.0){
                     y2[i]=1.0;}
                  else{
                     if (temp11<-6.0){
                          y2[i]=0.0;}
                       else{
                       y2[i]=1.0/(1.0+exp(-1.0*temp11));}}

             if(temp22>6.0){
            y22[i]=1.0;}
            else{
            if (temp22<-6.0){
               y22[i]=0.0;}
             else{
                y22[i]=1.0/(1.0+exp(-1.0*temp22));}}
                 v=c[i]*y2[i]+b[i]*x[i];
                 vv=c[i]*y22[i]+b[i]*x1[i];}

/* ------   Action  network:  Failure  signal  plus  chang  in
evaluation net---*/

  if(fabs(x[2])>=angle_lim || fabs(x[0])>=x_lim)   /* have
failure */
    {
      fail_num +=1;
     printf(" %d %d \n", fail_num,k_fail);
      k_fail=0;
       if (fail_num>max_fail){
           goto loop2;}
```

```
            r=-1.0;
            r1=r-vv;}
      else{
         if(k!=0){
              r=0.0;
              r1=r+gama*v-vv;        /* vv=v[t,t],  v=v[t,t+1]
*/
              k_fail+=1;}}

   /* ------------ Modification in evaluation ----------- */


      for(i=0;i<9;i++){
          if(i<5){
               c[i]+=beida*r1*y22[i];}
            else{
               b[i]+=beida*r1*x1[i-5];}}

      for (i=0;i<5;i++){
          for(j=0;j<5;j++){

a[i][j]+=beidah*r1*y22[i]*(1.0-y22[i])*sgnc*x1[j];}}

    /*------------ Output action---------- */

                 for (i=0;i<5;i++){
                     temp2=0.0;
                     for (j=0;j<5;j++){
                         temp2 += d[i][j]*x1[j];}
                       if(temp2>6.0){
                         z[i]=1.0;}
                         else{
                         if(temp2<=-6.0){
                              z[i]=0.0;}
                           else{
                              z[i]=1.0/(1.0+exp(-1.0*temp2));}}}
         s=0.0;
         for(i=0;i<5;i++){
               s+=f[i]*z[i]+e[i]*x1[i];}
            if(s>6.0){
             p=1.0;}
              else{
             if(s<-6.0){
                 p=0.0;}
               else{
                p=1.0/(1.0+exp(-s));}}
             if(p>=0.5){
                q=1;}
            else{
                q=0;}
             if(q==1){
               push=5.0;}
            else{
               push=-5.0;}
```

```
     /* -------------Output action modificatıon-------------
*/

     for(i=0;i<9;i++){
         if(i<5){
           f[i]+=lou*r1*(q-p)*z[i];}
           else{
           e[i]+=lou*r1*(q-p)*x1[i-5];}}
      for(i=0;i<5;i++){
          for(j=0;j<5;j++){

d[i][j]+=louh*r1*z[i]*(1.0-z[i])*sgnf*(q-p)*x1[j];}}
      x1[0]=x[0],x1[1]=x[1],x1[2]=x[2],x1[3]=x[3],x1[4]=x[4];
      d1[i][j]=d[i][j];

      /*-----------------  Save Weights ----------- */

   if(k<maxk_step){
      if(k_fail>1){
          for (i=0;i<4;i++){
            for(j=0;j<5;j++){
               fprintf(iop4," %f %f ", d[i][j],a[i][j]);
               fprintf(iop4,"\n");}}
          for(i=0;i<9;i++){
            fprintf(iop3,"%f   %f   %f   %f   %f   %f   %f",
b[i],e[i],c[i],f[i],z[i],x[i],y1[i]);
            fprintf(iop3,"\n");}}
        k++;
        t+=0.02;
        goto loop1;}
       else{
        k++;
        goto loop2;}

loop2:         fclose(iop);
          fclose(iop1);
          fclose(iop2);
          fclose(iop3);
          fclose(iop4);
          fclose(iop5);
 }
```

```
     **************************************************
               Real Time Conventional Control
     **************************************************

#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include "dash8.h"

float   ball_pos,   beam_angle,   control_input,   old_pos,
ball_vel,term1,term2,sample;
int k=0;
main()
{
   char  strout[50];
   FILE  *iop1;
/* 'error' is a variable declared globally that is set to 1
i.e. TRUE if
   an error occurs in any function. */

setfreq(30);                  /*function  to  set  interrupt
freq*/

getset_dat();                 /* function to get setup info
file */

   printf("Output file name:");
   gets(strout);
   iop1=fopen(strout,"w");

printf("\n Enter the ball position multiplying term\n\n");
scanf("%f",&term1);

printf("\nBall position multiplying term is %f",term1);
printf("\n Enter the ball velocity multiplying term\n\n");

scanf("%f",&term2);
printf("\nBall velocity multiplying term is %f",term2);


printf("\n\n\nProgram  is  running..Hit  any  key  to  stop
program..Not Ctrl/Break");

install();                    /*  function  to  install  isr
'getdata'
                             at appropriate interrupt vector
*/
das8set();                    /* function to 'let her rip'*/
old_pos = 0.0;                /*initialize  before  entering
loop*/
sample=0.0;
loop:if(k<=500)
{
 speedchk();                  /*function  to  ensure  user
program ru
```

```
                                      between interrupts    */
/* USER APPLICATION CODE IS PUT HERE */

ball_pos = int_volt(int_in[0]);
ball_pos = old_pos*.75+ball_pos*.25;    /*low pass filter*/

ball_vel = (ball_pos - old_pos)*samp_freq;

control_input = term1*ball_pos + term2*ball_vel;
fprintf(iop1,"   %.4f              %0.4f              %0.4f
\n",sample,ball_pos/10.0,ball_vel);
if (control_input > 2.0) control_input = 2.0;     /*clamp
output to avoid error*/
if (control_input < -2.0) control_input = -2.0;
old_pos = ball_pos;
volt_dac(control_input,0);              /*send  voltage  to
DAC*/
sample+=0.05;
k++;
goto loop;
}
volt_dac(0,0);
stop();
fclose(iop1);
}
/* END */
```

```
    ************************************************
            Real Time Neural Network Control
    ************************************************

#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include "dash8.h"
main()
{
  char strout[100];
  FILE *iop;
  extern double exp();
  double fabs(double x);
  int     i,j,times,k,q,fail_num,n,intval,DAC_channel,chann
          el;
  float   z1[5],z[5],y2[5],y1[5],y22[5],x1[5],c[5],b[5],
          e[5],f[5],ff[5],d1[5][5],x[5],sgnc[5],sgnf[5],
          xx[2],Xk[4],d[5][5],p,beida,limit, Ib, Ia, mm, rr,
          l,bb ,kk,r1, temp2, low, high, g, yy,yy1, zz,zz1,
          zz2,temp11,   vv,    gama,    r,    loue,    louf,
          louh,v,s,s1,s2,m11,beidah,pp,x_lim,angle_lim,
          temp22,k_fail,push,sample,ball_pos,old_pos,
          ball_vel;
 float    a[5][5]={{-0.6084,-0.0649,0.1005,0.0475,-0.776},
           { -0.0663, 0.0618,  0.0557, 0.0654,  0.1498 },
           { 0.1087,  0.0634,  0.0414, -0.0030, 0.2455 },
           { 0.0223,  0.0355,  0.0187, -0.0548, 0.1544 },
           { 0.0254, -0.0660,  0.0495, -0.1226, 0.1544 } };
 extern int rand();
/*Parameter assigning*/
 loue=1.0,    louf=1.0, louh=0.2, beida=0.05,
beidah=0.05,  gama=0.9, x_lim=0.3,
angle_lim=0.069,   n    =     5     0     0    ,
low=-0.1, high=0.1, push=0.0, x  [  4  ]  =  0  .  5  ;
ball_pos=ball_vel=old_pos=0.0;

/* ---------- Initialise Weights ------------ */

           for(i=0;i<5;i++)
           {
           z1[i]=(rand()/32767.0)*(high-low)+low;
           }
          for(i=0;i<5;i++)
            {
            y1[i]=(rand()/32767.0)*(high-low)+low;
            }

       /* ----- zero  state -----*/

k=0.0,          fail_num=0,     k_fail=0,      temp2=0.0,
temp11=0.0,     vv=0.0,         s=0.0,p=0.0;   r1=0.0,
x[4]=0.5,       x1[0]=0.58,     x1[1]=0.1,     x1[2]=0.5,
x1[3]=0.1,      x   1   [    4    ]    =    0    .    5    ;
xx[0]=0.0,      xx[2]=0.0,      Xk[0]=0.0,     Xk[1]=0.0,
Xk[2]=0.0,      Xk[3]=0.0,      push=0.0;

e[0]=-0.2093,  e[1]=-0.0845,  e[2]=-0.1322,  e[3]=0.2856,
e[4]=0.0949;
f[0]=-1.4708,  f[1]=-2.646,   f[2]=-1.5804,  f[3]=8.043,
f[4]=-2.59338;
b[0]=0.0207,   b[1]=-0.03322, b[2]=-0.00118, b[3]=-0.0852,
b[4]=0.0606;
c[0]=-3.98423, c[1]=0.0610,   c[2]=-0.0172,  c[3]=-0.01803,
c[4]=-0.02898;
d[0][0]=0.0625,     d[0][1]=0.0254,     d[0][2]=-0.0946,
d[0][3]=-0.5782,    d[0][4]=0.1396;     d[1][0]=-0.1375,
d[1][1]=0.0224,     d[1][2]=0.0141,     d[1][3]=-0.5549,
d[1][4]=0.2168;     d[2][0]=0.0142,     d[2][1]=-0.0246,
d[2][2]=-0.0197,    d[2][3]=-0.4987,    d[2][4]=0.0529;
d[3][0]=-0.0245,    d[3][1]=-0.0103,    d[3][2]=-0.0067,
d[3][3]=-0.6227,    d[3][4]=0.0346;     d[4][0]=-0.0322,
d[4][1]=-0.0072,    d[4][2]=-0.0175,    d[4][3]=-0.4508,
d[4][4]=0.1227;

       getset_dat();

       printf("Output  state x file name:");
       gets(strout);
       iop=fopen(strout,"w");
       k=0;
       sample=0.0;
      setfreq(20);

       getset_dat();
          install();
       das8set();
loop:
      if(k<n)
       {
       speedchk();
       Xk[0]=int_volt(int_in[0]);
       Xk[2]=int_volt(int_in[1]);
       x[0]=Xk[0]/10.0;          /* Position of beam is meter
(m) */
       x[2]=Xk[2]/10.0;            /* Angle of beam is degree
(C) */

       x[1]=(x[0]-xx[0])*samp_freq;
       x[3]=(x[2]-xx[2])*samp_freq;
      fprintf(iop,"%0.4f      %0.4f      %0.4f          %0.4f
\n",sample,x[0],x[1],x[2]);
       for (i=0;i<5;i++)
          {
              if (f[i]>=0.0){
                 sgnf[i]=1.0;
                 }
                 else
                 {
                 sgnf[i]=-1.0;
              }

            if(c[i]>=0.0)
            {
               sgnc[i]=1.0;
               }
               else
               {
               sgnc[i]=-1.0;
               }
       }

  if(x[1]>0.0)
   {
  x[0]=-fabs(x[0]);
   }
   else
   {
  x[0]=fabs(x[0]);
   }
    if(x[0]>x_lim)
      {
        x[0]=x_lim;
      }
      else
      {
        if(x[0]<-x_lim)
       {
         x[0]=-x_lim;
       }
       else
       {
         x[0]=x[0];
       }
      }
    if(x[2]>angle_lim)
      {
        x[2]=angle_lim;
      }
      else
      {
        if(x[2]<-angle_lim)
       {
         x[2]=-angle_lim;
       }
       else
       {
         x[2]=x[2];
       }
      }

     if(x[1]>0.1)
       {
         x[1]=0.1;
       }
```

```
          else
          {
            if(x[1]<-0.1)
           {
             x[1]=-0.1;
           }
           else
           {
             x[1]=x[1];
           }
          }
        if(x[3]>1.0)
          {
            x[3]=1.0;
          }
          else
          {
            if(x[3]<-1.0)
           {
             x[3]=-1.0;
           }
           else
           {
             x[3]=x[3];
           }
          }
         x[4]=0.5;
/*-- Output State Evaluation In Evaluation Network---- */

        for(i=0;i<5;i++)
       {
             temp11=0.0;
             temp22=0.0;

             for (j=0;j<5;j++)
             {
                temp11 += a[i][j]*x[j];
                temp22 += a[i][j]*x1[j];
             }
                if(temp11>6.0)
                  {
                       y2[i]=1.0;
                  }
                  else
                  {
                       if  (temp11<-6.0)
                         {
                            y2[i]=0.0;
                         }
                         else
                         {
                            y2[i]=1.0/(1.0+exp(-1.0*temp11));
                     }
               }
             if(temp22>6.0)
          {
          y22[i]=1.0;
          }
          else
          {
          if (temp22<-6.0)
           {
              y22[i]=0.0;
           }
           else
           {
              y22[i]=1.0/(1.0+exp(-1.0*temp22));
           }
          }
               v=c[i]*y2[i]+b[i]*x[i];
               vv=c[i]*y22[i]+b[i]*x1[i];

      }

/* -----  Action network: Failure Signal Plus Change In
          Evaluation Network
-------- */

  if(fabs(Xk[2])>=0.1 || fabs(x[0])>=0.01)
    {
      fail_num +=1;
      k_fail=0;
      r=-1.0;
      r1=r-vv;
    }
    else
    {
       if(k!=0)
      {
            r=0.0;
            r1=r+gama*v-vv;        /* vv=v[t,t],  v=v[t,t+1]
*/
            k_fail+=1;
      }
     }

   /* --------  Modification In Evaluation -----------  */


     for(i=0;i<9;i++)
       {
         if(i<5)
           {
               c[i]+=beida*r1*y22[i];
           }
           else
           {
               b[i]+=beida*r1*x1[i-5];
           }
       }
      for (i=0;i<5;i++)
        {
          for(j=0;j<5;j++)
            {

a[i][j]+=beidah*r1*y22[i]*(1.0-y22[i])*sgnc[i]*x1[j];

            }
        }

    /*------------ Output Action ------------- */

              for (i=0;i<5;i++)
              {
                  temp2=0.0;
                  for (j=0;j<5;j++)
                    {
                      temp2 += d[i][j]*x1[j];
                    }
                   if(temp2>6.0)
                  {
                     z[i]=1.0;
                      }
                      else
                      {
                     if(temp2<=-6.0)
                       {
                       z[i]=0.0;
                       }
                       else
                       {
                       z[i]=1.0/(1.0+exp(-1.0*temp2));
                       }
                      }
              }
         s=0.0;
         for(i=0;i<5;i++)
            {
              s1=f[i]*z[i];
              s2=e[i]*x1[i];
               s+=s1+s2;
            }
            if(s>6.0)
              {
             p=1.0;
              }
              else
              {
             if(s<-6.0)
               {
                 p=0.0;
               }
               else
               {
                p=1.0/(1.0+exp(-s));
                }
              }
              if(p>=0.5)
             {
                 q=1;
             }
             else
             {
                 q=0;
             }
ball_vel = (ball_pos - old_pos)*samp_freq;

push =p*(ball_pos + ball_vel);
if(push>2.0) push=2.0;
if(push<-2.0) push=-2.0;

          volt_dac(push,0);
          xx[0]=x[0],xx[2]=x[2];
          old_pos=ball_pos;
        /* ------------Output Action Modification-------------
*/

        for(i=0;i<9;i++)
          {
            if(i<5)
              {
              f[i]+=louf*r1*(q-p)*z[i];
              }
              else
              {
              e[i]+=loue*r1*(q-p)*x1[i-5];
              }
          }
         for(i=0;i<5;i++)
           {
             for(j=0;j<5;j++)
               {

d[i][j]+=louh*r1*z[i]*(1.0-z[i])*sgnf[i]*(q-p)*x1[j];
               }
           }

x1[0]=x[0],x1[1]=x[1],x1[2]=x[2],x1[3]=x[3],x1[4]=x[4],d1[i
][j]=d[i][j];
    k++;
    sample+=0.05;
      goto loop;
    }
    volt_dac(0,0);
    stop();
    fclose(iop);
  }
```