MODELING AND CONTROL OF NON-LINEAR SYSTEMS USING NEURO-FUZZY APPROACH

A MAJOR THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF ENGINEERING

IN

CONTROL & INSTRUMENTATION

SUBMITTED BY

V.RAVI KISHORE REDDY

(Roll No.13/C&I/06)

UNIVERSITY ROLL NO.10222

UNDER THE ESTEEMED GUIDANCE

OF

Mr. RAM BHAGAT

(LECTURER)

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI
2006-2008

CERTIFICATE

This is to certify that Major thesis titled "MODELING AND CONTROL OF NON-LINEAR SYSTEMS USING NEURO-FUZZY APPROACH" submitted by Mr. V. Ravi Kishore Reddy in partial fulfillment for the degree of Master of Engineering (Control & Instrumentation) of the Electrical Engineering Department, Delhi college of Engineering, Delhi – 110042 is a bonafide record of work, he has carried out under my guidance and supervision.

Mr.RAM BHAGAT

(Lecturer)

Electrical Engineering Department,

Delhi College of Engineering, Delhi.

ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to my guide Mr.RAM BHAGAT (Lecturer, Electrical Engineering Department, Delhi College of Engineering, Delhi) for his assistance and invaluable guidance towards the progress of this thesis.

I would like to thank Prof. PARMOD KUMAR, Head of Department, Electrical Engineering for providing valuable comments.

Mr. V. RAVI KISHORE REDDY

M.E (C& I)

College Roll No: 13/C&I/06

ABSTRACT

In this work a neuro-fuzzy approach is used to model any non-linear data. Fuzzy curve approach is used to know prerequisite parameters to model the system. Back-propagation algorithm is used to properly train the network. The appropriateness of the model is tested with a non-linear data and the model results are compared with actual data.

Neuro-fuzzy controller is designed for LOS stabilization for a two axis gimbal system. Implementation in azimuth axis is presented. A conventional compensator designed in [2] is used as training data for neuro-fuzzy controller. Fuzzy logic based controller is implemented on the system. Neuro-fuzzy model algorithm is used in modelling the controller. Step response of the system using the three controllers is implemented in MATLAB and the results are compared.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 NEURO-FUZZY MODELING

A multilayered neural network can approximate any continuous function on a compact set [6] and a fuzzy system can do the same approximation [7]. A simple fuzzy –neural network for modeling systems from input-output data [1] is introduced. Fuzzy-neural networks can be divided into two main categories. One group of neural networks for fuzzy reasoning uses fuzzy weights in the neural network. In a second group, the input data are fuzzified in the first or second layer, but the neural network weights are not fuzzy. The fuzzy-neural network discussed here comes under second group.

A simple neural network is used to implement a fuzzy rule based model of a real system from input-output data. Fuzzy curves are used for, identification of the significant input variables, estimation of the number of rules needed in the fuzzy model, and determination of initial weights for the neural networks. The number of input variables and the number of rules determine the structure of neural network. Training the network using back-propagation is the heart of the modeling process. The model can be viewed as either a fuzzy system, a neural network, or a fuzzy-neural system.

The algorithm developed is implemented on a non-linear data [7] as explained in chapter 5.

1.2 LINE-OF-SIGHT (LOS) STABILIZATION CONTROL

Line-of-sight stabilization is an essential feature of modern fire control and surveillance systems. As the imaging system in a fire control system undergoes angular vibrations, the resolution within the image decreases. This is primarily due to the spread of incident optical energy on the imaging detector (pixel). The control system must sense the disturbance on the LOS using gyros and generate appropriate control torques to minimize the spread of optical energy on the detector. This is done by integrating the imaging system in a set of gimbals, which are excited by the torquers. These torquers get their drive from the control system. The torque disturbances in the system can be due to bearing and motor friction, unbalanced aerodynamics, vibration forces from onboard mechanisms, and spring torque forces from

wires or flexures. Presence of inherent non-linearities such as stiction friction, saturation of actuators, etc. is also to be taken into account.

1.3 DESIGN AND IMPLEMENTATION OF CONTROLLERS

Conventional controller designs are dependent on the accuracy of the mathematical model of the plant, which usually ignore high order dynamics. Although robust controllers can be designed to overcome uncertainties in plant parameters as well as non-linearities, the resultant controller becomes complex for implementation. In order to overcome above problems, intelligent controlling techniques are used. Artificial neural networks and fuzzy logic are potential tools for intelligent control engineering. Neural networks are best known for their learning capabilities. Fuzzy logic is a method of using human skills and thinking processes in a machine. Neural networks offer the possibility of solving the problem of tuning. A combination of neural networks and fuzzy logic offers the possibility of solving tuning problems and design difficulties of fuzzy logic. The resulting network can be easily recognized in the form of fuzzy logic control rules. This new approach combines the well-established advantages of both the methods and avoids the drawbacks of both. The computation of control value from the given measured input value is seen as a feed forward procedure as in layered networks, where the inputs are forwarded through the network resulting in some output value(s). If the actual output value differs from the desired output value, the resulting error is propagated back through the architecture, which in turn results in modification of certain parameters and reduction in error during the next cycle.

For the line-of-sight stabilization control system considered, conventional controller and fuzzy knowledge based controller (FKBC) are implemented in [2]. LOS stabilization loop is implemented in MATLAB using conventional controller and FKBC and the control laws are stored in an array. These control laws are taken as training data for a neuro-fuzzy controller. The neuro-fuzzy algorithm developed in [1] is used in modeling the controller for Line-of sight stabilization systems which is more robust under non-linearities.

1.4 ORGANIZATION OF THE DISSERTATION

This Dissertation is organized as follows.

Chapter 2 Literature Review

Chapter 3 Fuzzy Neural System Modeling

Chapter 4 Line-of-Sight (LOS) Stabilization

Chapter 5 Implementation of Model to Non-linear system data

Chapter 6 Implementation of LOS Stabilization loop in MATLAB

Chapter 7 Results and Discussions

Chapter 8 Conclusions and Future Scope

CHAPTER 2

LITERATURE REVIEW

Yinghua Lin and George A. Cunningham III developed simple but effective fuzzy rule based neural models of complex systems from input-output data. They introduced fuzzy-neural network for modeling systems that can represent any continuous function over a compact set. [1]

J.A.R. Krishna Moorty, Rajeev Marathe and Hari Babu carried out the controller (conventional and fuzzy logic) design for a two-axis electomechanical gimbal. As an example LOS stabilization in only direction (azimuth) is demonstrated in this paper. Very stringent specifications for disturbance attenuation and command following specifications are met by appropriately designing the various membership functions and rules [2].

Simon Haykin in his book "Neural Networks" [3] provided a comprehensive foundation of neural networks. He explained the role of neural networks in the construction of intelligent machines for pattern recognition, control and signal processing.

Kevin M. Passino and Stephen Yurkovich provided a control-engineering perspective on fuzzy control [4]. Overall a pragmatic engineering approach to the design, analysis, performance evaluation, and implementation of fuzzy control systems is provided.

Han-Xiong Li and H. B. Gatland proposed a general robust rule base for fuzzy two-term control, and leave the optimum tuning to the scaling gains, which greatly reduces the difficulties of design and tuning [5]. More systematic analysis and design are given for the conventional fuzzy control.

M. Sugeno and T. Yasukawa discussed a method of qualitative modeling based on fuzzy logic and input-output data of a non-linear system is presented as an example of identification [7]. This data is verified with the neuro-fuzzy algorithm.

Stelios Papadakis and John Theocharis presented a novel modeling technique based on the fuzzy curve concept [8]. This method exhibits a number of significant attributes, such as

effective input space searching computational simplicity and high accuracy of the resulting fuzzy models

Fernando de Castro Junqueira and Ettore Apolônio de Barros presented the general aspects about Dynamically Tuned Gyroscope (DTG) development, the main difficulties involved, as well as some performance results. Fundamentals and mathematical modeling of a single-gimballed DTG are introduced [9]. An analysis of the constructive requirements, the machining methods and materials employed in the DTG implementation are presented.

Peter J. Kennedy and Rhonda L. Kennedy investigate the impact of LOS disturbances and sensor noise on the performance of each stabilization control loop configuration. They focuses on two methods of implementing the stabilization servo loop design. Mathematical model of gimbal is derived from torque relationships about inner and outer gimbal body axes based on rigid body dynamics [10].

Y. Hayashi, J. Buckley, and E. Czogala discussed the direct fuzzification of a standard layered, feedforward, neural network where the signals and weights are fuzzy sets and a fuzzified delta rule is presented for learning [11].

CHAPTER 3

FUZZY NEURAL SYSTEM MODELING

3.1 INTRODUCTION

Modelling of systems provide several useful information on the expected behaviour of a final system. Great advantage of modeling, which might be considered as a supplement to the simulation of controlling is demonstration, and the examination of the impact of parameters that can be simulated only with difficulty on the system to be developed. Modeling these systems is a very cost effective means of addressing the problem of security from an overall system view. Let us begin by reviewing the modeling process; this should make more apparent the nature of the problem we intend to address. The first stage of the modeling, i.e., identification, is to identify good explanatory variables (i.e., input variables for the model), and to collect sufficient data for the modelling task. This is the hardest and most crucial stage in the modelling process. Analysis of the target is necessary, and for that purposes various tools and methods such as statistical analysis methods already exists.

A simple fuzzy-neural network can be developed for modeling the systems that can represent any continuous function over a compact set [6], [7]. Fuzzy-neural networks can be divided into two main categories. One group of neural networks for fuzzy reasoning uses fuzzy weights in the neural network. [11], [12]. In a second group, the input data are fuzzified in the first or second layer, but the neural network weights are not fuzzy [13],[14]. The fuzzy-neural network model developed comes under this group. There are many methods to obtain the proper network structure and initial weights to reduce training time [15]. Fuzzy curves concept is used to create our system model. Number of input variables and the number of rules to determine the structure of the neural network are found with the help of fuzzy curves. Then network can be trained using back-propagation algorithm.

3.2 NEURAL NETWORKS

In its most general form, a neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer. To achieve good performance, neural networks employ a massive interconnection

of simple computing cells referred to as “neurons” or “processing units”. Neural networks and conventional algorithmic computers are not in competition but complement each other.

In general a neural network viewed as an adaptive machine can be defined as

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two aspects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Neural networks are also referred to in literature as neurocomputers, connectionist networks, parallel distributed processors, etc.

3.2.1 MODEL OF A NEURON

A neuron is an information-processing unit that is fundamental to the operation of a neural network. Three basic elements of the neuron model are

1. A set of synapses or connecting links, each of which is characterized by a weight or strength of its own. Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in the range that includes negative as well as positive values.

2. An adder for summing the input signals, weighted by the respective synapses of the neuron.

3. An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some infinite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed interval [0,1] or alternatively [-1,1].

The neuronal model also includes an externally applied bias, denoted by b. The bias b has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively.

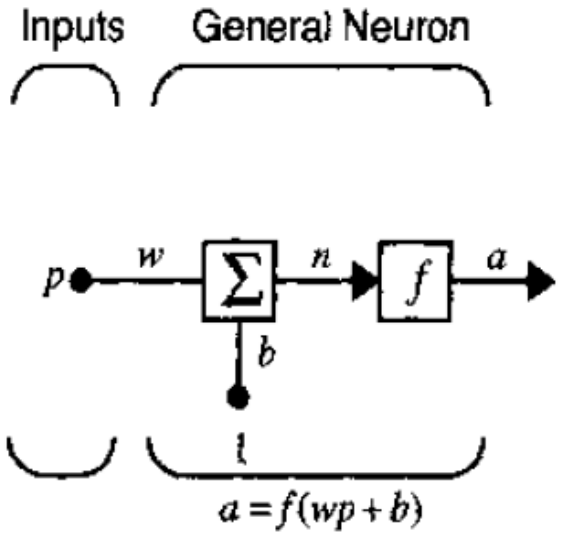


Fig.3.1 : Model of a neuron

Mathematical equations, activation functions, feedback etc, are dealt clearly in [3].

3.2.2 NETWORK ARCHITECTURES

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. In general fundamentally three different classes of network architectures are identified:

1. Single-Layer Feedforward Networks

In a layered neural network the neurons are organized in the form of layers. In the simplest form of a layered network, we have an input layer of source node that projects onto an output layer of layer of neurons (computational nodes), but not vice versa. In other words this network is strictly a feedforward or acyclic type. Such a network is called a single layer network, with the designation "single-layer" referring to the output layer of computation nodes (neurons). We do not count the input layer of source nodes because no computation is performed there.

2. Multilayer Feedforward Networks

The second class of a feedforward neural network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of hidden neurons is to intervene between the external input and the network output in some useful manner. By adding one or more hidden layers, the network

is enabled to extract higher-order statistics. The ability of hidden neurons to extract higher-order statistics is particularly valuable when the size of input layer is large. The neural network may be fully connected or partially connected. It is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communication links (synaptic connections) are missing from the network, the network is said to be partially connected.

3. Recurrent networks

A recurrent neural network distinguishes itself from a feedforward neural network in that it has at least one feedback loop. The presence of feedback loops has a profound impact on the learning capability of the network and on its performance.

3.3 LEARNING

The property that is of primary significance for a neural network is the ability of the network to learn from its environment, and to improve its performance through learning. The improvement in performance takes place over a time in accordance with some prescribed measure. A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels. Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process.

Simon Haykin defined learning in the context of neural network [3] as:

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes takes place.

A prescribed set of well-defined rules for the solution of a learning problem is called a learning algorithm. There is no unique learning algorithm for the design of neural networks. Rather many kit of tools represented by a diverse variety of learning algorithms, each of which offers advantage of its own. Basically, learning algorithms differ from each other in the way in which the adjustment to a synaptic weight of a neuron is formulated. Some of the basic learning rules presented in the literature are error-corrector learning, memory based learning, Hebbian learning, competitive learning, and Boltzmann learning.

3.3.1 Error-Correction Learning

Neuron k is driven by a signal vector x (n) produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applies to the source node (i.e., input layer) of the neural network. The argument n denotes discrete time, or precisely, the time step of an iterative process involved in adjusting the synaptic weights of neuron k. The output signal of neuron k is denoted by y_k (n). This output signal, representing the only output of the neural network, is compared to a desired response or target output, denoted by d_k (n). Consequently, an error signal, denoted by e_k (n), is produced. By definition, we thus have e_k (n) = d_k (n) - y_k (n). The error signal e_k (n) actuates a control mechanism, the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron k. The corrective adjustments are designed to make the output signal y_k (n) come closer to the desired response d_k (n) in a step-by-step manner. This objective is achieved by minimizing a cost function or index of performance, ξ (n), defined in terms of the error signal e_k (n) as:

$$\xi(n) = e_k^2(n)/2$$

That is, ξ (n) is the instantaneous value of the error energy. The step-by-step adjustments to the synaptic weights of neuron k are continued until the system reaches a steady state. At that point the learning process is terminated. This learning process is referred to as error-correcting learning. In particular, minimization of the cost function ξ(n) leads to a learning rule commonly referred to as the delta rule or Widrow-Hoff rule, named in honor of its originators. Let w_{kj} (n) denote the value of synaptic weight w_{kj} of neuron k excited by element x_j (n) of the signal vector **x** (n) at time step n. According to delta rule, the adjustment Δw_{kj} (n) applied to the synaptic weight w_{kj} at time step n is defined by

$$\Delta w_{kj}(n) = \eta\, e_k(n)\, x_j(n)$$

where η is a learning parameter, which determines the rate of learning as moved from one step in the learning process to another.

The delta rule discussed herein, presumes that the error signal is directly measurable. For this measurement to be feasible, clearly desired response from some external source should be supplied which is directly accessible to neuron k. In other words, neuron k is visible to the outside world, as shown in fig .

Having computed the synaptic weight adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight w_{kj} is determined by

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

In effect, $w_{kj}(n)$ and $w_{kj}(n+1)$ may be viewed as the old and new values of synaptic weights $w_{kj}(n)$, respectively. In computational terms $w_{kj}(n) = z^{-1}[w_{kj}(n+1)]$ where z^{-1} is the unit – delay operator. That is, z^{-1} represents a storage element.

3.3.2 LEAST MEAN SQUARE ALGORITHM

The least-mean-square (LMS) algorithm is based on the use of instantaneous values for the cost function, namely

$$\xi(w) = \frac{1}{2} e^2(n) \qquad \text{---------- (3.1)}$$

where e (n) is the error signal measured at time n. Differentiating ξ (w) with respect to the weight vector w yields

$$\frac{\partial \xi(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w} \qquad \text{---------- (3.2)}$$

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

As with the linear least-square filter, the LMS algorithm operates with a linear neuron so the error signal is

$$e(n) = d(n) - x^T(n) w(n) \qquad \text{---------- (3.3)}$$

Hence,

$$\frac{\partial e(n)}{\partial w(n)} = -x(n) \qquad \text{---------- (3.4)}$$

and

$$\frac{\partial \xi(w)}{\partial w(n)} = -x(n) e(n) \qquad \text{---------- (3.5)}$$

Using (3.5) as an estimate for the gradient vector, g (n) = -x (n) e (n)

From the steepest gradient method $\Delta w(n) = -\eta\, g(n)$

Finally LMS algorithm is formulated as

$$w(n+1) = w(n) + \eta\, x(n) e(n)$$

where η is the learning parameter.

3.4 BACK PROPAGATION ALGORITHM

A multilayer feedforward networks consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptrons (MLPs).

Multilayer perceptrons solve difficult and diverse problems by training them in a supervised manner with a high popular algorithm known as *error back-propagation algorithm*. This algorithm is based on the error-correction learning rule. It may be viewed as a generalization of an equally popular adaptive filtering algorithm: the ubiquitous least-mean-square (LMS) algorithm.

Basically, error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network, against the direction of synaptic connections-hence the name “error back-propagation.” The synaptic weights are adjusted to make the actual response of the network move closer to the desired response in a statistical sense.

The development of the back-propagation algorithm represents a landmark in neural networks in that it provides a computationally efficient method for training of multilayer perceptrons.

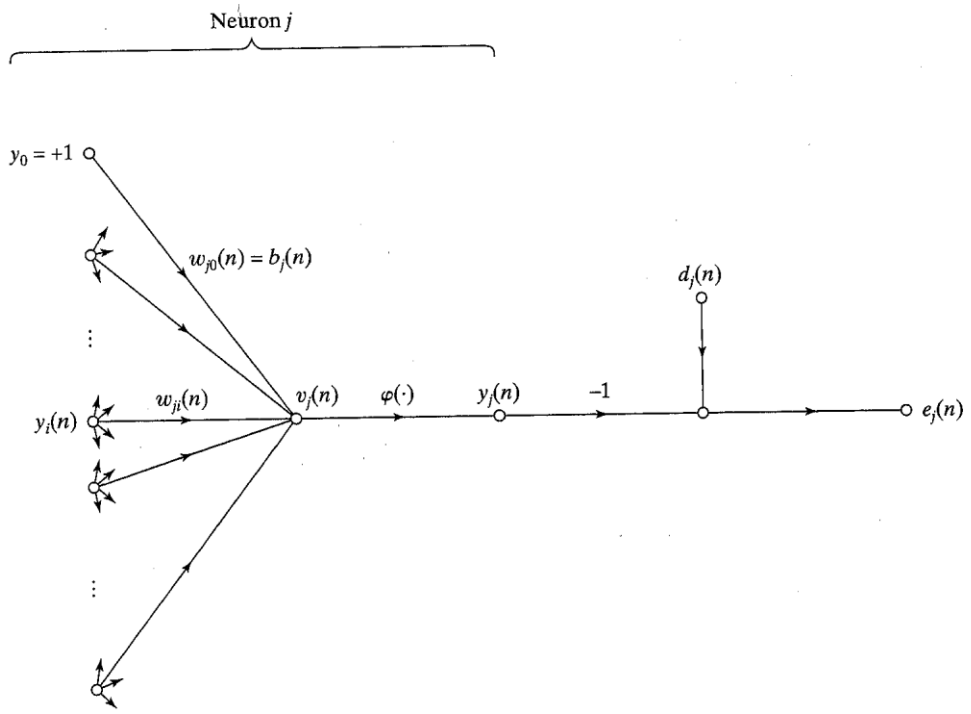


Fig.3.2 Details of output neuron

The error signal at the output of neuron j at iteration n as shown in fig 3.2 (i.e., presentation of the nth training example) is defined by

$$e_j(n) = d_j(n) - y_j(n), \qquad \text{---------- (3.6)}$$

neuron j is an output node

Instantaneous value of the error energy for neuron j is $\frac{1}{2}e_j^2(n)$. Correspondingly, the instantaneous value of the total energy is obtained by summing $\frac{1}{2}e_j^2(n)$ over all neurons in the output layer; these are only "visible" neurons for which error signals can be calculated directly. Therefore

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e^2(n) \qquad \text{---------- (3.7)}$$

where the set C includes all the neurons in the output layer of the network. Let N denote the total number of patterns contained in the training set. The average squared error energy is obtained by summing $\xi(n)$ over all n and then normalizing with respect to the set size N.

$$\xi_{av} = \sum_{n=1}^{N} \xi(n) \qquad \text{---------- (3.8)}$$

For a given training set, ξ_{av} represents the cost function as a measure of learning performance. The objective of the learning process is to adjust the free parameters of the network to minimize ξ_{av}. To do this minimization an approximation similar in rationale to that used in LMS algorithm is used.

The arithmetic average of these individual weight changes over the training set is therefore an estimate of the true change that would result from modifying the weights based on minimizing the cost function ξ_{av} over the entire training set.

Consider fig.2 which depicts neuron j being fed by a set of function signal produced by a layer of neurons to its left. The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron j is therefore

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \qquad \text{---------- (3.9)}$$

where m is the total number of inputs (excluding the bias) applied to neuron j. The synaptic weight w_{jo} (corresponding to the fixed input $y_o = +1$) equals to bias b_j applied to neuron j. Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi(v_j(n)) \qquad \text{---------- (3.10)}$$

In a manner similar to the LMS algorithm, the back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the partial derivative $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$. According to the chain rule of calculus,

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad \text{---------- (3.11)}$$

The partial derivative $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight $w_{ji.}$

Differentiating both sides of (3.7) with respect to e_j (n),

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j\ (n) \qquad \text{---------- (3.12)}$$

Differentiating both sides of (3.6) with respect to y_j (n),

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \qquad \text{---------- (3.13)}$$

Next, differentiating (3.10) with respect to v_j (n),

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi^1{}_j\,(v_j\ (n)) \qquad \text{---------- (3.14)}$$

Finally, differentiating (3.9) with respect to w_{ji} (n) yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i\ (n) \qquad \text{---------- (3.15)}$$

Using Eqs (3.12), (3.13), (3.14), and (3.15) in (3.11) yields

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j\ (n)\ \varphi^1 j\ (v_j\ (n))\ yi\ (n) \qquad \text{---------- (3.16)}$$

The correction Δw_{ji} (n) applied to w_{ji} (n) is defined by the delta rule:

$$\Delta w_{ji}\ (n) = -\eta\ \frac{\partial \xi(n)}{\partial w_{ji}(n)} \qquad \text{---------- (3.17)}$$

where η is the learning-rate parameter of the back-propagation algorithm. The use of the minus sign in (3.17) accounts for the gradient descent in weight space (i.e., seeking the direction for weight change that reduces the value of ξ (n)). Accordingly, the use of (3.16) in (3.17) yields

$$\Delta w_{ji}\ (n) = \eta\ \delta_j(n) y_i(n) \qquad \text{---------- (3.18)}$$

where the local gradient $\delta_j(n)$ is defined by

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial v_j(n)}$$

$$= -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$= e_j(n)\, \varphi'_j(v_j(n)) \qquad \text{---------- (3.19)}$$

The local gradients points to required changes in synaptic weights. According to Eq. (3.19) the local gradient $\delta_j(n)$ for output neuron j is equal to the product of the corresponding error signal $e_j(n)$ for that neuron and the derivative $\varphi'_j(v_j(n))$ of the associated activation function.

From Eq. (3.18) and (3.19), the key factor involved in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j. Two distinct cases are identified in this context, depending on where in the network neuron j is located. In case 1, neuron j is an output node. This is simple to handle because each output node of the network is supplied with a desired response of its own, making it a straight forward matter to calculate the associated error signal. In case 2, neuron j is a hidden node. Even though hidden neurons are not directly accessible, they share responsibility for any error made at the output of the network. The question is to know how to penalize or reward hidden neurons for their share of the responsibility. This problem is the credit-assignment problem which can be solved in an elegant fashion by back-propagating the error signals through the network.

Case 1 Neuron j is an Output Node

When neuron j is located in the output layer of the network, it is supplied with a desired response of its own. Eq. (3.6) is used to compute the error signal $e_j(n)$ associated with this neuron. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$ using Eq. (3.19).

Case 2 Neuron j is a Hidden Node

When neuron j is located in a hidden layer of the network, as shown in fig.3.3 there is no specified desired response for that neuron. Accordingly, the error signal for a hidden layer would have to be determined recursively in terms of the error signals of all neurons to which that hidden neuron is directly connected.

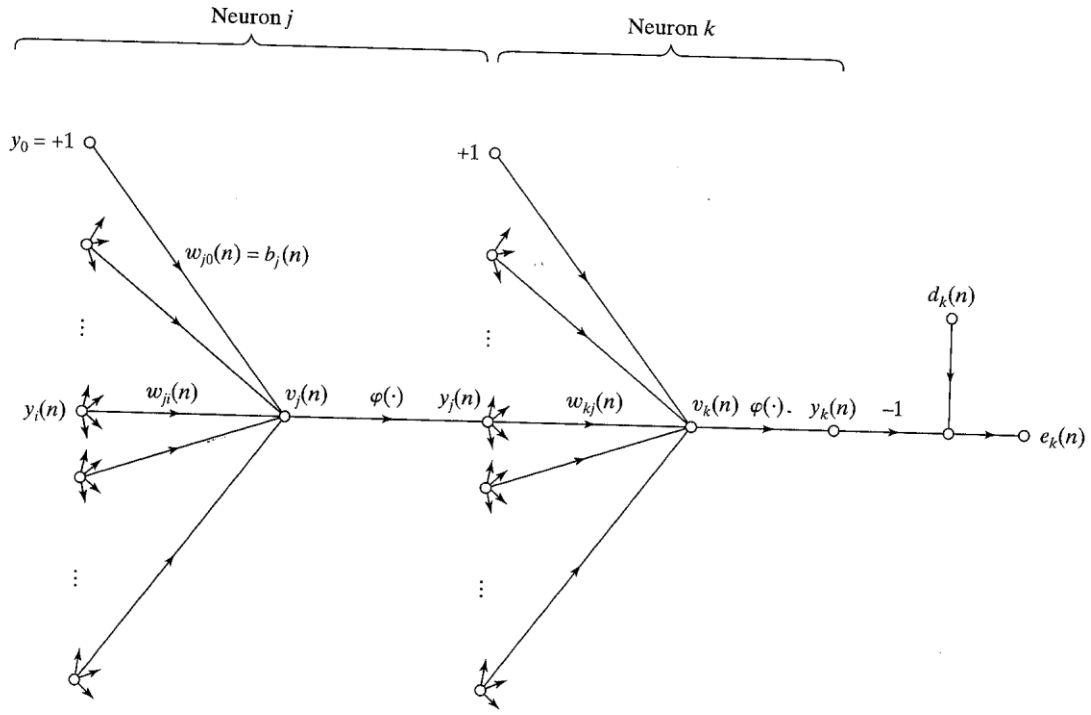


Fig.3.3: Details of output neuron k connected to hidden neuron j

According to Eq. (3.19), local gradient may be redefined for hidden neuron as

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \qquad \text{----------(3.20)}$$

$$= -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi'_j(v_j(n)), \text{ neuron j is hidden} \qquad \text{---------- (3.21)}$$

Eq. (3.14) is used in the second line. To calculate the partial derivative ($\partial \xi(n)/\partial y_j(n)$), Consider

$$\xi(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n), \text{ neuron k is an hidden node} \qquad \text{---------- (3.22)}$$

Differentiating Eq. (3.22) with respect to the function signal $y_j(n)$,

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \qquad \text{---------- (3.23)}$$

Using chain rule for the partial derivative $\frac{\partial e_k(n)}{\partial y_j(n)}$, and rewriting Eq. (3.23)

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \qquad \text{---------- (3.24)}$$

However, $e_k(n) = d_k(n) - y_k(n)$

$= d_k(n) - \varphi(v_k(n))$, neuron k is an output node

Hence

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n)) \qquad \text{---------- (3.25)}$$

For neuron k, the induced field is

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \qquad \text{---------- (3.26)}$$

where m is the total number of inputs (excluding the bias) applied to neuron k. Differentiating Eq. (3.26) with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \qquad \text{---------- (3.27)}$$

By using Eq. (3.25) and (3.27) in (3.24)

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n)$$

$$= -\sum_k \delta_k(n) w_{kj}(n) \qquad \text{---------- (3.28)}$$

in the second line definition of local gradient is used

Finally using Eq. (3.28) in (3.21), back-propagation formula for the local gradient is:

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \qquad \text{---------- (3.29)}$$

where neuron j is hidden node.

Summarizing the relations of the derivation of back-propagation algorithm,

First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

(Weight correction $\Delta w_{ji}(n)$) = (η) ($\delta_j(n)$) (input signal to neuron j $y_i(n)$) ---------- (3.30)

Second, the local gradient δ_j (n) depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, δ_j (n) equals the product of the derivative φ^1_j (v_j (n)) and the error signal e_j (n), both of which are associated with neuron j.

2. If neuron j is a hidden node, δ_j (n) equals the product of the associated derivative φ^1_j (v_j (n)) and the weighted sum of the δs computed for the neurons in the next hidden or output layer that are connected to neuron j.

The two passes of computation

In the application of the back-propagation algorithm, two distinct passes of computation are distinguished. The first pass is referred to as the forward pass, and the second is referred to as the backward pass.

In the forward pass the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. The function signal appearing at the output to neuron j is computed as

$$y_j (n) = \varphi (v_j (n))$$

where v_j (n) is the induced local field neuron j, defined by

$$v_j (n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

where m is the local number of inputs (excluding the bias) applied to neuron j, and w_{ji} is the synaptic weight connecting neuron i to neuron j, and y_i (n) is the input signal of neuron j or equivalently, the function signal appearing at the output of neuron i. If neuron j is in the first hidden layer of the network, $m=m_o$ and the index i refers to the i^{th} input terminal of the network, for which

$$y_i (n) = x_i (n)$$

where x_i (n) is the i^{th} element of the input vector (pattern). If, on the other hand, neuron j is the output layer of the network, $m=m_L$ and the index j refers to the j^{th} output terminal of the network, for which

$$y_j(n) = o_j(n)$$

where $o_j(n)$is the j^{th} element of the output vector (pattern). This output is compared with the desired response $d_j(n)$, obtaining the error signal $e_j(n)$ for the jth output neuron. Thus the forward phase of computation begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of output layer.

The backward pass, on the other hand, starts at the output layer by passing the error signal leftward through the network, layer by layer, and recursively computing the δ (i.e. the local gradient) for each neuron. This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule of Eq. (3.30). For a neuron located in the output layer, the δ is simply equal to the error signal of that neuron multiplied by the first derivative of its nonlinearity. Hence Eq. (3.30) is used to compute the changes to the weights of all the connections feeding into the output layer. Given the δs for the neurons of the output layer, Eq. (3.29) is used to compute the δs for all the neuron in the penultimate layer and therefore the changes to the weights of all connections feeding into it. The recursive computation is continued layer by layer, by propagating the changes to all synaptic weights in the network.

For the presentation of each training example, the input pattern is fixed ("clamped") throughout the round-trip process, encompassing the forward pass followed by the backward pass.

3.5 FUZZY CURVES

Fuzzy curves are used for i) identification of the significant variables, ii) estimation of the number of rules needed in the fuzzy model, and iii) determination of initial weights for the neural network.

Consider a multiple-input, single-output system for which we have input-output data with possible extraneous inputs. To determine the initial membership functions, first step is to determine the significant inputs, the number of rules R, and initial values for the weights. Let the input candidates be x_i (i=1,2,…,*n*), and the output variable y. Let us assume that *m* training data points are available and that x_{ik} (k-1,2,…,*m*) are the i^{th} coordinates of each of the *m* training points. For each input variable x_i, plot the *m* data points in x_i - y space. For every

point (x_{ik}, y_k) in the x_i – y space, draw a fuzzy membership function for the input variable defined by

$$\phi_{ik}(x_i) = \exp\left(-\left(\frac{x_{ik} - x_i}{b}\right)^2\right), \quad k = 1, 2, 3, \cdots, m. \qquad \text{---------- (3.31)}$$

Each pair of ϕ_{ik} and the corresponding y_k provides a fuzzy rule for y with respect to x_i**,** The rule is represented as "if x_i is ϕ_{ik} (x_i), then y is y_k**".** ϕ_{ik} is the input variable fuzzy membership function for x_i, corresponding to the data point k. ϕ_{ik} can be any fuzzy membership function, including triangle, trapezoidal, Gaussian, and others. Here Gaussian is used. Typically take b as about 20% of the length of the input interval of x_i. For *m* training data points, we have *m* fuzzy rules for each input variable.

Centroid defuzzification is used to produce a fuzzy curve c_i for each input variable x_i by

$$c_i(x_i) = \frac{\sum_{k=1}^{m} \phi_{ik}(x_i) \cdot y_k}{\sum_{k=1}^{m} \phi_{ik}(x_i)}. \qquad \text{---------- (3.32)}$$

If the fuzzy curve for given input is flat, then this input has little influence in the output data and it is not a significant input. If the range of a fuzzy curve c_i is about the range of the output data y**,** then the input variable x_i is important to the output variable. The fuzzy curve tells us that the output is changing when x_i is changing. We rank the importance of the input variables x_i according to the range covered by their fuzzy curves c_i. For input identification we drop those input with fuzzy curve Range < 1/4th of Max Range and the rest are the significant inputs.

The number of rules, R_i, needed to approximate each fuzzy curve c_i, are estimated by the maximum and minimum points on the curve. This is a heuristic based on the idea that the fuzzy model will interpolate between the maximum and minimum points. If the maximum and minimum points are far apart, or the curve is not smooth between the maximum and minimum points, we may add rules. If we have N fuzzy curves, then we will have N different numbers R_1, R_2, . . . , R_N corresponding to the N fuzzy curves. To determine the number of rules R needed in the fuzzy neural network, we let R = max (R_1, R_2, . . . , R_N).

3.6 THE ARCHITECTURE OF THE FUZZY-NEURAL NETWORK

Architecture of four layer (input, fuzzification, inference, and defuzzification) fuzzy-neural network (Fig. 1).There are N inputs with N neurons in the input layer, and R rules, with R neurons in the inference layer. There are N x R neurons in the fuzzification layer. Hence, once we determine the number of inputs ***N*** and the number of rules R, the structure of the network can be determined. The first ***N*** neurons (one per input variable) in the fuzzification layer incorporate the first rule, the second ***N*** neurons incorporate the second rule, and so on.

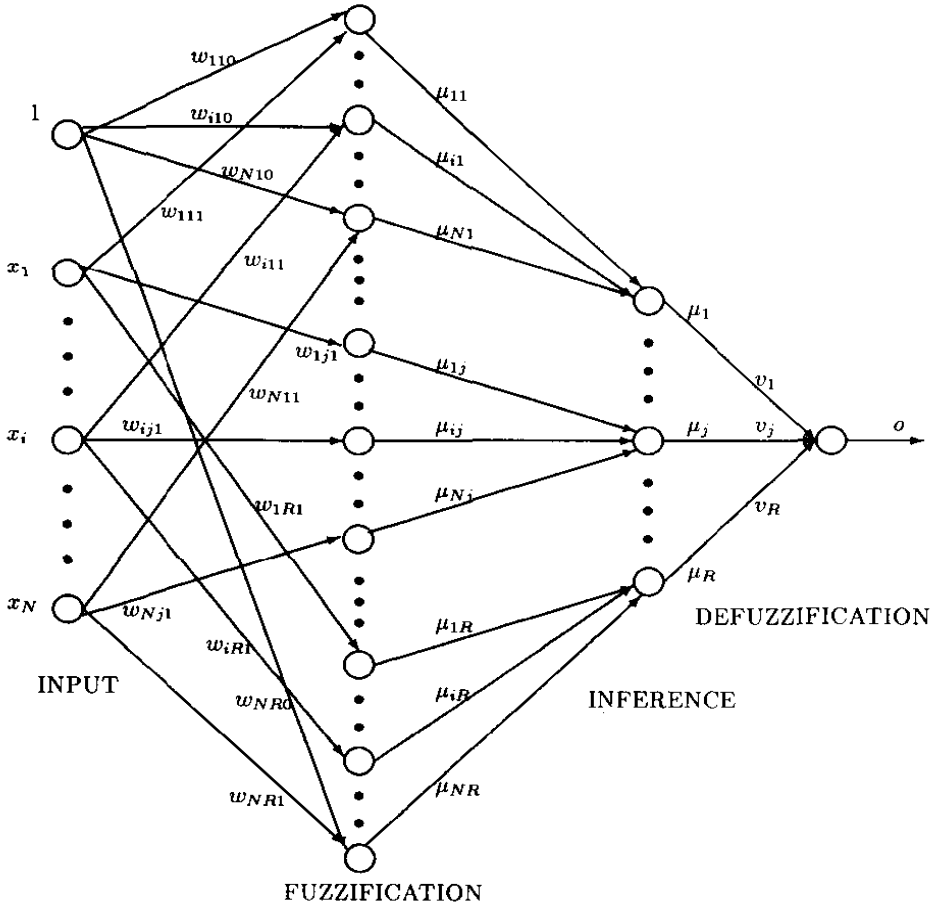


Fig.1 The architecture of fuzzy-neural network

Every neuron in the second or fuzzification layer represents a fuzzy membership function for one of the input variables. The activation function used in the fuzzification layer is

$$f(net_{ij})=\exp(-\left|net_{ij}\right|^{l_{ij}})$$

where l_{ij} is typically in the range $0.5 \le l_{ij} \le 5$ and initially equals one or two, and

$$net_{ij} = w_{ij1}x_i + w_{ij0}$$

Hence, the output of the fuzzification layer is

$$\mu_{ij}(x_i) = \exp\left(-|w_{ij1}x_i + w_{ij0}|^{l_{ij}}\right) \quad \text{---------- (3.33)}$$

where μ_{ij} is the value of fuzzy membership function of the i^{th} input variable corresponding to the j^{th} rule. We label the set of weights between the input and the fuzzification layer by

$$W = \{\{w_{ij0}, w_{ij1}\}: i = 1, \cdots, N; j = 1, \cdots, R\} \quad \text{---------- (3.34)}$$

The activation function used in the third or inference layer is *f(netj) = netj.*

We use multiplicative inference, so the output of the inference layer is

$$\mu_j(x_1, x_2, \cdots, x_N) = \prod_i^N \mu_{ij}(x_i) \quad \text{---------- (3.35)}$$

where the μ_{ij} are from (3.33).

The connecting weights between the third layer and the fourth layer are the central values, v_j, of the fuzzy membership functions of the output variable. We label the set of weights $\{v_j\}$ by $V = \{ v_j : j = 1, \ldots, R \}$. Each fuzzy rule r_j (j = 1, **2,** . . . , R) is of the form: if x_1 is μ_{1j} and x_2 is μ_{2j} and …. if x_N is μ_{Nj} then y is v_j. Note that the neural network weights in V and W determine the fuzzy rules. We use the weighted sum defuzzification, and the equation for the output is

$$\begin{aligned} o(x_1, x_2, \cdots, x_N) &= \sum_j^R \mu_j(x_1, x_2, \cdots, x_N) v_j \\ &= \sum_j^R v_j \prod_i^N \exp\left(-|w_{ij1}x_i + w_{ij0}|^{l_{ij}}\right) \end{aligned} \quad \text{- (3.36)}$$

The initial weights in V are set to the centers of output variable fuzzy membership functions. To do this divide the range of the desired output data into R intervals, and set the initial v_j (*j* = **1, 2,** . . , R) to be the central value of these R intervals, and assign these central values to the weights v_j in ascending order.

Fuzzy curves are used to set the initial weights in W. Divide the domain for each fuzzy curve c_i, into R intervals corresponding to the R intervals in the output space. For the fuzzy curve c_i, label the centers of the intervals x_{ij} ***(j***= 1, **2 , . - . ,**R). Order x_{ij} ***(j***= 1, **2 , . - . ,**R) by the value of c_i, at the center of each interval. x_{iR} corresponds to the interval containing the largest value of c,. The interval containing the point x_{iR} is associated with the output interval whose center is at v_r In a similar fashion x_{iR-1} is the center of the interval which contains the next largest

central point on the curve c_i and x_{iR-1} is associated with v_{R-1}, and so on for j = R - 2 , R - 3 , . . , 1. The length of the interval over which a rule applies in the domain of c_i, is denoted as Δx_i. The initial fuzzy membership function of x_i for rule j is defined as $\exp(-\left|(x_{ij}-x_i)/a\cdot\Delta x_i\right|^{lij})$ where *a* is typically in the range of [0.5,2]. Hence, referring to (l), the initial weights w_{ij0} and w_{ij1} are $w_{ij0}=1/(a\cdot\Delta x_i)$ and $w_{ij1}= -x_{ij}/(a\cdot\Delta x_i)$.

The training of the network is done using back-propagation algorithm .

The performance index for the model is

$$PI = \frac{\sqrt{\sum_{k=1}^{m}(o_k^d - o_k)^2}}{\sum_{k=1}^{m}|o_k^d|} \qquad \text{---------- (3.37)}$$

where $o^d_{k,}$ (k=1,2,…,m), are the actual or desired output values and o_k, (k=1,2,…,m), are the outputs from the model.

Choosing a maximum number of iterations I_{max}, and some small number $\varepsilon>0$, the training is continued until, for some i, $\sum_{j=1}^{i+100} PI_j - \sum_{j=i+100}^{i+200} PI_j \leq \epsilon$ or the number of iterations reaches $I_{max.}$

CHAPTER 4

LINE OF SIGHT STABILIZATION CONTROL

4.1 INTRODUCTION

Line-of-sight (LOS) stabilization form part of modern surveillance and fire control systems (FCS). Gimbals are precision electro-mechanical assemblies designed primarily to steer the telescope and isolate the optical system from disturbances induced by the operating environment. The two gimbal model has an azimuth gimbal and an elevation gimbal to which the payload (telescope) is attached. The gimbals together with the control system are responsible for tracking commands and LOS (stabilization).

The rate sensors are bore sight inertial sensors used for angular rate feedback in the stabilization loop of the control system. Hence they play a very important role in the stabilization of the gimbal-mounted payload against disturbances.

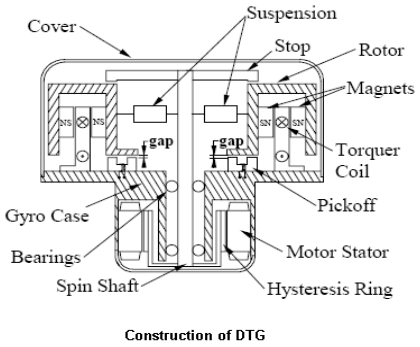
Following are some of the options available for rate sensors:

1. Rate gyro (RG)
2. Rate integrating gyro (RIG)
3. Dynamically tuned gyro (DTG)
4. Ring laser gyro (RLG)
5. Fiber optic gyro (FOG)

A dynamic tuned gyroscope, the so-called DTG, is one of the options nowadays that can provide the required performance for many applications.

4.2. DYNAMICALLY TUNED GYROSCOPE (DTG)

The DTG is two-axis, spinning mass, inertial rate sensor whose rotor is suspended by a universal hinge of zero stiffness at the tuned speed. Due to their low cost, fast reaction time, small size and ruggedness, DTGs have become very popular in navigation and gimballed systems. As they are dry, they provide good performance over wider range of temperatures than conventional rate gyros. The main disadvantage of DTGs is that the gyro electronics are more complicated. Figures 4.1 show the construction of a DTG.



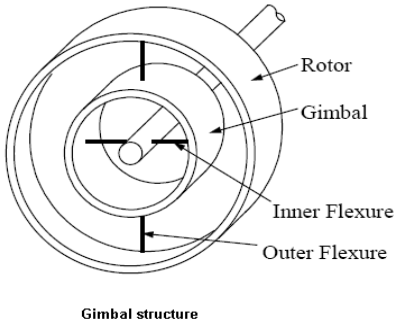


Fig. 4.1 (a): Construction of DTG **Fig. 4.1 (b): Gimbal Structure**

4.2.1 Modeling of DTG rotor:

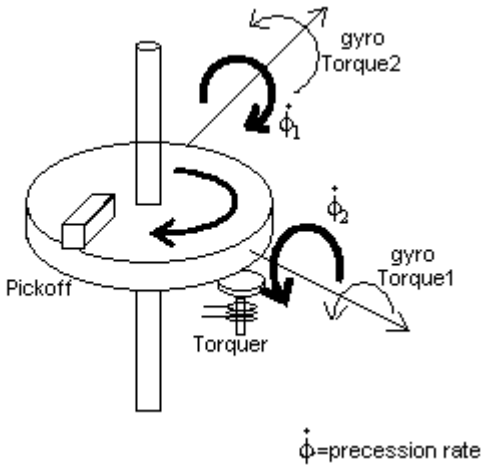


Fig. 4.2 DTG rotor

The sum of the external torques on the rotor both presses the rotor at an angular velocity proportional to the angular momentum of the rotor (H) and accelerate the rotor proportionately to its transverse inertia (A),

$$\sum T_{1,ext} = A\frac{d^2\phi_1}{\partial t^2} + H\frac{d\phi_2}{dt} \qquad \text{---------- (4.1)}$$

$$\sum T_{2,ext} = A\frac{d^2\phi_2}{\partial t^2} + H\frac{d\phi_1}{dt} \qquad \text{---------- (4.2)}$$

The inertial rotor rates $\frac{d\phi_1}{dt}$ (or $\frac{d\phi_2}{dt}$) is the sum of the corresponding inertial case rate and the rotor rate relative to the case. The primary external torques on the rotor is those generated by the torquers. There are two torquers, one in each axis, which generate torque proportional to the amount of current in their coils. Figure shows the torquer in the first axis generating a precession rate $\frac{d\phi_2}{dt}$ in the second axis. The pickoff measures the rotor position relative to the case (θ_2). The role of the subsequent compensators and electronics is to null this pickoff, so that the current in the torquer is proportional to the inertial case rate (precession rate). Having complete control of the rotor's precession rate is the key to the DTG's ability to measure rate. Note that the external torqueses include not only the controlled torquer inputs K_TI_1 and K_TI_2, but also viscous damping, mistuning elastic restraints and windage torques.

The dynamic equations for the DTG model are:

$$K_T I_1 - C_d\frac{d\theta}{dt} - H\frac{\delta N}{F_m}\theta_1 - T_d\theta_2 = A\frac{d^2\phi_1}{dt^2} + H\frac{d\phi_2}{dt} \qquad \text{---------- (4.3)}$$

$$K_T I_1 - C_d\frac{d\theta_2}{dt} - H\frac{\delta N}{F_m}\theta_1 - T_d\theta_2 = A\frac{d^2\phi_2}{dt^2} + H\frac{d\phi_1}{dt} \qquad \text{---------- (4.4)}$$

where

I_1, I_2 = currents in torquer coils

θ_1, θ_2 = pickoff angles (rotor position relative to the case)

$\frac{d\phi_1}{dt}, \frac{d\phi 2}{dt}$ = precession rates of rotor (inertial)

K_T = torquer constant

C_d = viscous damping coefficient

H = angular momentum

δN = Difference between actual rotor rate and tuned rotor rate

F_m = Figure of merit of the rotor system

T_d = Drag torque due to windage effects from gas around rotor

A = Transverse inertia

A simple second order transfer function with f Hz bandwidth of the form $G_{gyro}(s) = \frac{(scale.factor)\cdot(2\cdot\pi\cdot f)^2}{s^2 + 2\cdot\delta\cdot(2\cdot\pi\cdot f)\cdot s + (2\cdot\pi\cdot f)^2}$ may be used to model the gyro response in each of the rate feedback channels (azimuth and elevation) [2].

4.3 GIMBAL DYNAMICS

A gimbal is a pivoted support that allows the rotation of an object about a single axis. A set of two gimbals, one mounted on the other with pivot axes orthogonal, may be used to allow an object mounted on the innermost gimbal to remain vertical regardless of the motion of its support. The gimbal dynamics model can be derived from the torque relationships about the inner and outer gimbal body axes based on rigid body dynamics. A two-axis gimbal rigid model dynamics is formulated in APPENDIX.

4.4 MODELING OF DC MOTOR

Motors play the role of actuators that drive the gimbal assembly. Two motors will be used, one for driving the azimuth gimbal assembly and one for elevation. Consider the mathematical modeling of DC motor,

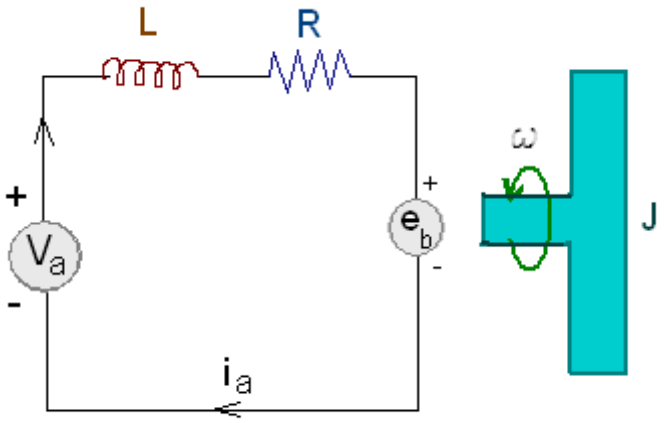


Fig. 4.3 A Simple DC Motor

Nomenclature:

L=armature inductance, R=armature resistance, V_a =armature voltage,

i_a= armature current, e_b=back emf, ω=angular speed, T=motor torque

J=load inertia, K_a=motor torque constant, k_b=back emf constant

Figure 4.3 shows a simple electromechanical model of a DC motor. The armature voltage is the input. The mathematical equations representing the model (excluding drive electronics) are:

$$e_b = k_b \omega \qquad \text{---------- (4.5)}$$

$$T = K_a i_a \qquad \text{---------- (4.6)}$$

Using Newton's law combined with Kirchhoff's law

$$J\frac{d\omega}{dt} + b\omega = k_a i_a \qquad \text{---------- (4.7)}$$

$$V_a - L\frac{di_a}{dt} - i_a R - e_b = 0 \qquad \text{---------- (4.8)}$$

In Laplace domain using Eq. (4.5) and (4.6) in (4.8) and (4.7),

$$T(s) = \frac{K_a}{Ls + R} \cdot \left(V_a(s) - k_b \omega(s)\right) \qquad \text{---------- (4.9)}$$

$$(Js + b)\omega(s) = T(s) \qquad \textbf{---------- (4.10)}$$

Eliminating $\omega(s)$ from Eq. (4.9) and (4.10) deriving the transfer function between T(s) and V_a(s),

$$\frac{T(s)}{V_a(s)} = \left(\frac{k_a(Js + b)}{LJs \wedge 2 + (Lb + JR)s + (Rb + k_a k_b)}\right) \qquad \textbf{------ (4.11)}$$

4.5 OVERALL SYSTEM (TRACKER) OPERATION

The laser and track sensor (i.e., telescope, camera, etc.) are mounted on the inner axis of a multi-axis mechanical gimbal; or below the gimbal, coupled to the line of sight (LOS) via a stabilized pointing mirror. Pointing control is implemented via two servo loops, the outer track or pointing loop and an inner stabilization or rate loop. The track sensor detects the laser returns from the target location. The track processor uses this information to generate rate commands that direct the gimbal bore-sight toward the target LOS. The stabilization loop

isolates the laser and sensor from platform motion and disturbances that would otherwise perturb the aim-point. The track loop must have sufficient bandwidth to track the LOS kinematics. The stabilization loop bandwidth must be high enough to reject the platform disturbance spectrum. A typical configuration is shown in fig.4.5.1

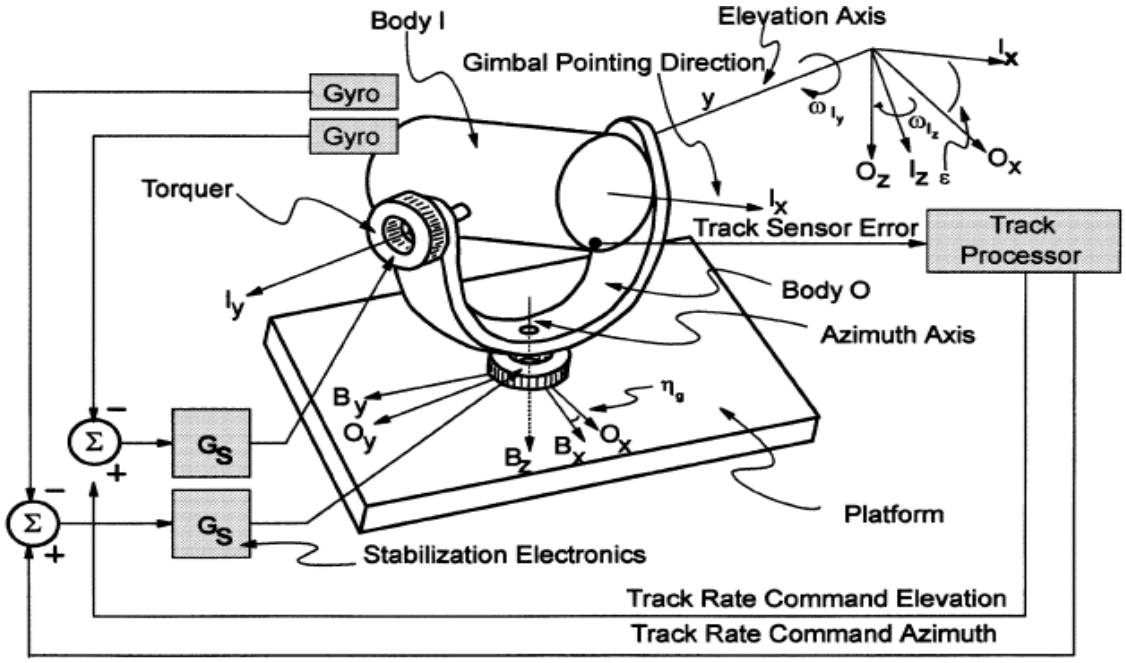


Fig 4.4 Two-axis tracker configuration

In this work we concentrate on the inner stabilization loop (rate loop) of the azimuthal axis.

4.6 CONTROLLER DESIGN FOR LOS STABILIZATION

The plant under consideration consists of a gimbaled payload that is driven by DC motor. A servo power amplifier amplifies the controller output before being fed to the DC torque motor. A high performance dual axis dynamically tuned gyro (DTG) is used to sense the inertial angular rate of the gimbal azimuth and elevation axis.

The relevant parameters of gimbal/electronic system are as follows:

1. Gimbal inertia, 0.5 kg m^2

2. Weight of pay load, 35 kg

3. Load pole, 1 Hz

4. Gimbal resonance, 140 Hz

5. Torquer rating, 3.5 nm (peak)

6. Torque sensitivity (Kt), 0.786 nm/A

7. Back emf constant (Kb), 0.786 V/rad s^{-1}

8. Gyro scale factor, 5.73 V/rad

9. Gyro dynamics, single pole at 100 Hz

10. Data acquisition resolution, 16 bits (max. input= $\pm$ 10 V)

12. dead band due to stiction friction, 10% of the peak torque

13. digital-to-analog converter resolution, 16 bits (max output= $\pm$ 10 V)

and the design is carried out for the following design specifications:

1. Steady state error for step response, $\leq 0.1\%$

2. Percent overshoot, $\leq 40\%$

3. Rise time, ≤ 50 msec

4.6.1 Conventional Controller

The Bode plot technique is used for the design. A linear model of the plant is used for this purpose. Lead and Lag compensator design procedures are used in the design process [16]. The transfer function of the controller designed is as follows:

$$\frac{(s/80+1)}{(s/1.5+1)}\frac{(s/91+1)}{(s/400+1)}\frac{(s/20+1)}{(s/5+1)}\frac{(9200)}{(s/400+1)} \qquad \text{---------- (4.12)}$$

The analog controller is transformed to digital domain using "Tustin" method. The synthesis of the control law in the digital domain is carried out with a 4-kHz sampling frequency. The four stages of the z transform of the controller is given as follows:

$$\frac{(0.018934z-0.018559)}{(z-0.9963)}\frac{(4.2239z-4.1387)}{(z-0.90476)}\frac{(0.25047z-0.249922)}{(z-0.99875)}\frac{(438.1z+438.1)}{(z-0.90476)} \text{ --(4.13)}$$

4.6.2 Fuzzy Controller

For the system under study, seven linguistic variables for each of the input and output variables with one normalized universe of discourse (-1, +1) are used to describe them. These are NB (negative big), NM (negative medium), NS (negative small), ZO (zero), PS (positive small), PM (positive medium), PB (positive big). Each fuzzy variable is a member of the subsets with a degree of membership μ. After specifying the fuzzy sets, it is required to determine the membership functions for these sets. For inputs and output Gaussian membership functions have been used. For input variables of error and change of error the output of the fuzzy controller is the incrementel control force. The membership functions were defined using the standard Gaussian function.

$$f(x,\sigma,c)= \exp(-(x-c))^2/2*\sigma^2)$$

Table 4.1 Parameters of fuzzy membership fiunctions

variables	e		*de*		u	
function parameters	c	σ	c	σ	c	σ
fuzzy sets						
nb	-1.0	0.35	-1.0	0.141	-1.0	0.141
nm	-0.25	0.1	-0.66	0.141	-0.57	0.142
ns	-0.1	0.04	-0.2	0.12	-0.15	0.1
z	0.0	0.013	0.0	0.05	0.0	0.007
ps	0.1	0.04	0.2	0.12	0.15	0.1
pm	0.25	0.1	0.66	0.141	0.57	0.142
pb	1.0	0.35	1.0	0.141	1.0	0.141

Having specified the inputs from the simulation, a set of rules have to be defined using the linguistic variables. For a system with two inputs and with each input universe defined with seven linguistic variables, 49 rules can be formed considering all the combinations of inputs. A proper way to show these rules is given in Table 4.2 where all the symbols are defined in the basic of fuzzy logic terminology.

Δe/e	NB	NM	NS	ZR	PS	PM	PB
NB	NB	NB	NB	NB	NM	NS	ZR
NM	NB	NB	NB	NM	NS	ZR	PS
NS	NB	NB	NM	NS	ZR	PS	PM
ZR	NB	NM	NS	ZR	PS	PM	PB
PS	NM	NS	ZR	PS	PM	PM	PB
PM	NS	ZR	PS	PM	PB	PB	PB
PB	ZR	PS	PM	PB	PB	PB	PB

Table 4.2: Fuzzy rules

The fuzzy controller can be programmed in C, FORTRAN, MATLAB, or virtually any other programming language. There may be some advantage to programming it in C since it is then sometimes easier to transfer the code directly to an experimental setting for use in real-time control. At other times it may be advantageous to program it in MATLAB since plotting capabilities and other control computations may be easier to perform there.

Pseudo-code:

The pseudo-code for a simple fuzzy controller [4] that is used to compute the fuzzy controller output given its two inputs:

1. Obtain x1 and x2 values. (Get inputs to fuzzy controller)

2. Compute mf1[i] and mf2[j] for all i,j. (Find the values of all membership functions given the values for u1 and u2)

3. Compute prem [i,j]=min[mf1[i],mf2[j]] for all i, j (Find the values for the premise membership functions for a given x1 and x2 using the AND(minimum) operation)

4. Implication method of min is used, implies the minimum values of the AND operation in the previous step is carried forward.

5. Compute agg [i,j] = agg[rule[i,j],prem[i,j]] for all i, j (Find the aggregate of each output linguistic variable by evaluating all rules by using ‘max’ operator)

6. Let Num=0, Den=0 (Initialize the COG numerator and denominator values)

7. For all i , j (Cycle through all areas to determine COG)

Num=Num+agg[i,j]*center[rule[i,j]] (Compute numerator for COG)

Den=Den+agg[i,j] (Compute denominator for COG)

8. Output Crisp=Num/Den (Output the value computed by the fuzzy controller)

9. Go to Step 1.

4.6.3 Neuro-fuzzy controller:

Using Eq 4.12 or 4.13 the conventional controller is implemented in MATLAB and the control law is saved in workspace and used as training data for the neuro-fuzzy algorithm and the model is properly trained.

CHAPTER 5

IMPLEMENTATION IN MATLAB

5.1 IMPLEMENTATION OF MODEL TO NON-LINEAR SYSTEM DATA

Consider the input-output data of a non-linear system [7] as shown in table 5.1

Table 5.1 Input output data of a non-linear system

Group A						Group B					
No	x_1	x_2	x_3	x_4	y	No	x_1	x_2	x_3	x_4	y
1	1.40	1.80	3.00	3.80	3.70	26	2.00	2.06	2.25	2.37	2.52
2	4.28	4.96	3.02	4.39	1.31	27	2.71	4.13	4.38	3.21	1.58
3	1.18	4.29	1.60	3.80	3.35	28	1.78	1.11	3.13	1.80	4.71
4	1.96	1.90	1.71	1.59	2.70	29	3.61	2.27	2.27	3.61	1.87
5	1.85	1.43	4.15	3.30	3.52	30	2.24	3.74	4.25	3.26	1.79
6	3.66	1.60	3.44	3.33	2.46	31	1.81	3.18	3.31	2.07	2.20
7	3.64	2.14	1.64	2.64	1.95	32	4.85	4.66	4.11	3.74	1.30
8	4.51	1.52	4.53	2.54	2.51	33	3.41	3.88	1.27	2.21	1.48
9	3.77	1.45	2.50	1.86	2.70	34	1.38	2.55	2.07	4.42	3.14
10	4.84	4.32	2.75	1.70	1.33	35	2.46	2.12	1.11	4.44	2.22
11	1.05	2.55	3.03	2.02	4.63	36	2.66	4.42	1.71	1.23	1.56
12	4.51	1.37	3.97	1.70	2.80	37	4.44	4.71	1.53	2.08	1.32
13	1.84	4.43	4.20	1.38	1.97	38	3.11	1.06	2.91	2.80	4.08
14	1.67	2.81	2.23	4.51	2.47	39	4.47	3.66	1.23	3.62	1.42
15	2.03	1.88	1.41	1.10	2.66	40	1.35	1.76	3.00	3.82	3.91
16	3.62	1.95	4.93	1.58	2.08	41	1.24	1.41	1.92	2.25	5.05
17	1.67	2.23	3.93	1.06	2.75	42	2.81	1.35	4.96	4.04	1.97
18	3.38	3.70	4.65	1.28	1.51	43	1.92	4.25	3.24	3.89	1.92

19	2.83	1.77	2.61	4.50	2.40	44	4.61	2.68	4.89	1.03	1.63
20	1.48	4.44	1.33	3.25	2.44	45	3.04	4.97	2.77	2.63	1.44
21	3.37	2.13	2.42	3.95	1.99	46	4.82	3.80	4.73	2.69	1.39
22	2.84	1.24	4.42	1.21	3.42	47	2.58	1.97	4.16	2.95	2.29
23	1.19	1.53	2.54	3.22	4.99	48	4.14	4.76	2.63	3.88	1.33
24	4.10	1.71	2.54	1.76	2.27	49	4.35	3.90	2.55	1.65	1.40
25	1.65	1.38	4.57	4.03	3.94	50	2.22	1.35	2.75	1.01	3.39

The data of x_3 and x_4 are put as dummy inputs to check the appropriateness of the model algorithm.

Model algorithm is evaluated on the data and the results are shown in the figures below.

Fig. 5.1 shows the actual output value for each data point.

Fig. 5.2 shows the fuzzy curves drawn for the data.

Fig. 5.3 shows the modeled output for each data point.

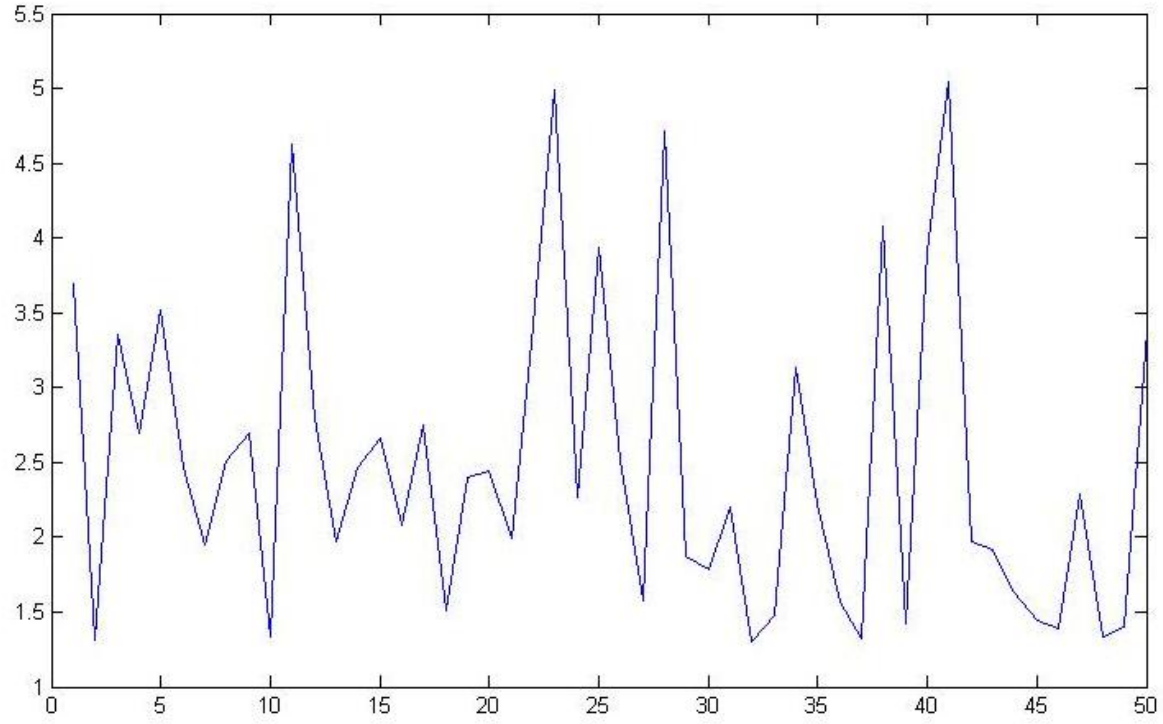


Fig 5.1: Output data for data points (50)

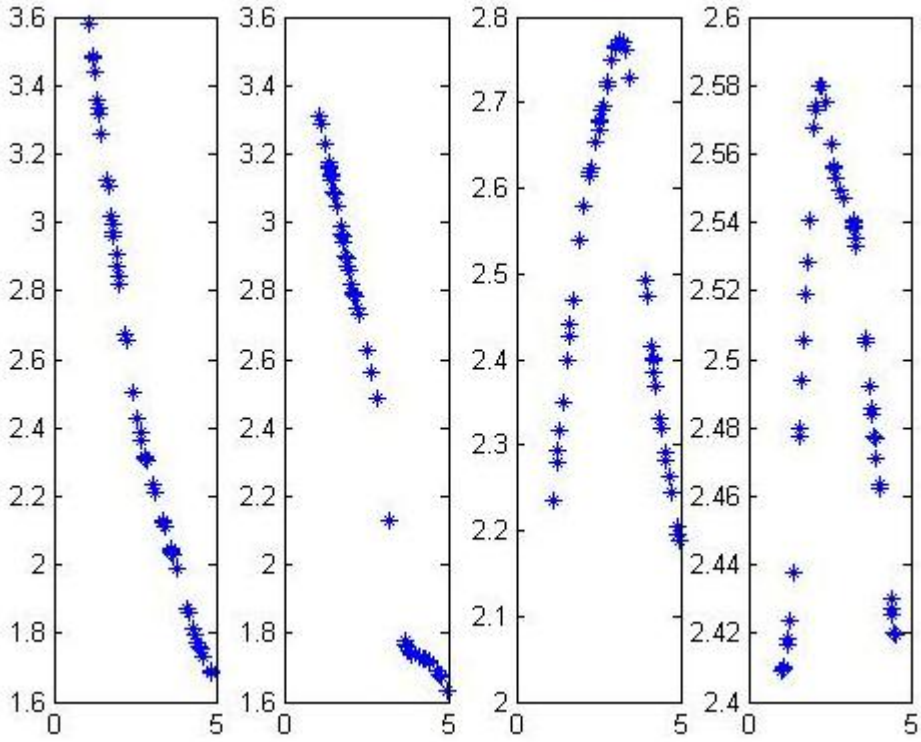


Fig. 5.2: Fuzzy Curves for the non-linear system data

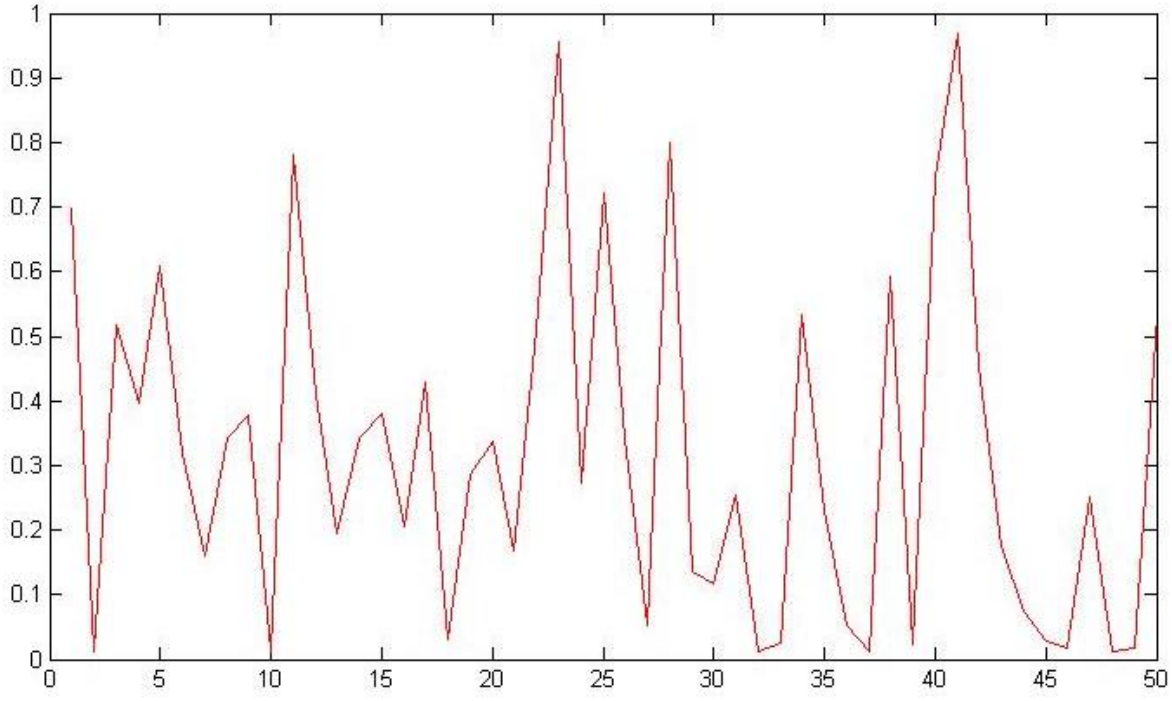


Fig.5.3: Modeled output for each data point

5.2 IMPLEMENTATION OF LOS STABILIZATION LOOP CONTROL

Implementation of LOS stabilization loop control (in azimuthal axis) is explained in chapter 4. Stabilization loop is controlled using conventional controllers and intelligent controllers.

5.2.1 CONVENTIONAL CONTROLLER

Conventional controller designed for LOS stabilization loop [2] in discrete domain is given by Eq. 4.13. Complete simulink diagram as given by [2] is shown in Fig.5.4

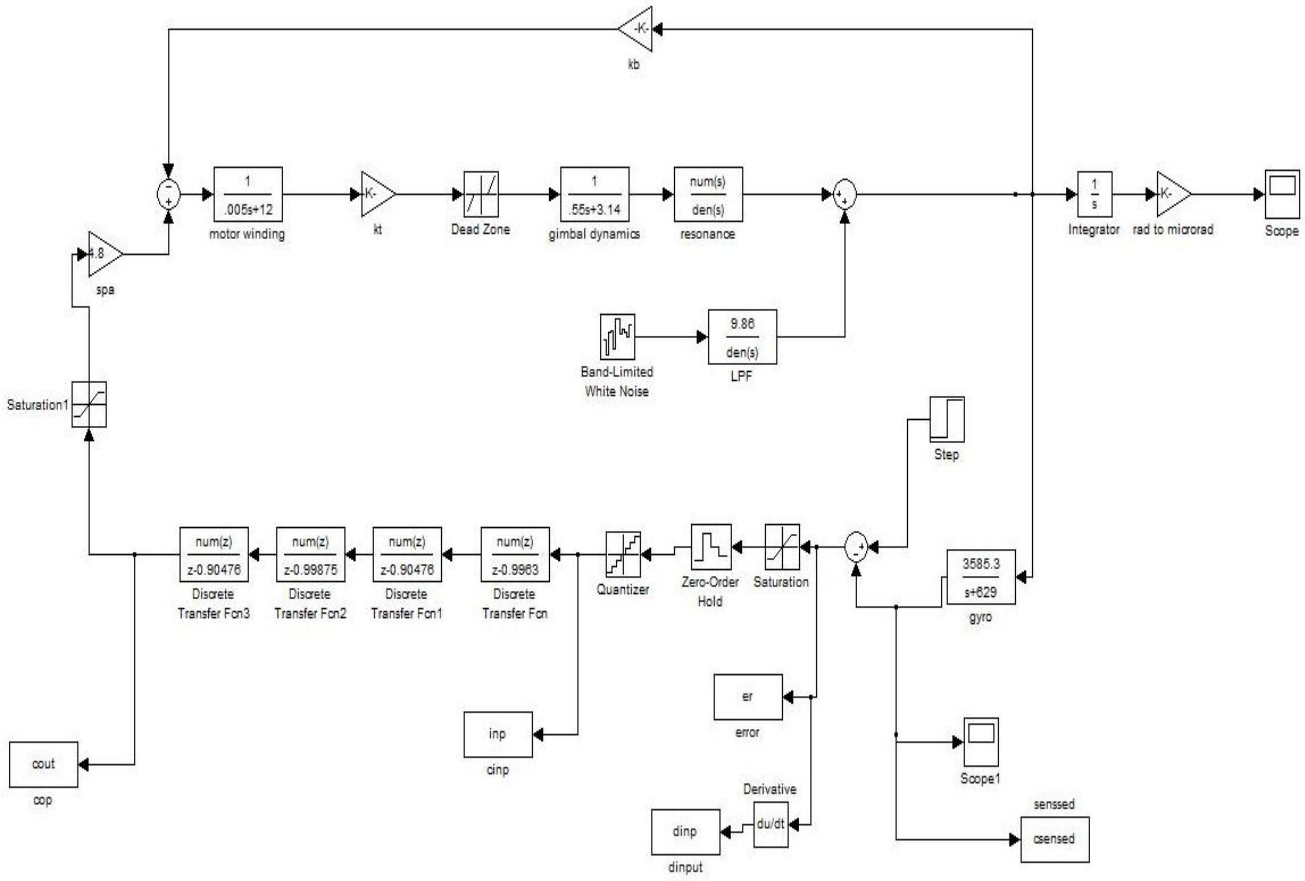


Fig. 5.4: LOS stabilization loop using a conventional controller

5.2.2 FUZZY CONTROLLER

Fuzzy controller designed for LOS stabilization loop is explained in Chapter 4. As explained in pseudo-code fuzzy controller was coded in MATLAB m-file and linked to simulation environment. Complete simulink diagram is shown in Fig.5.5

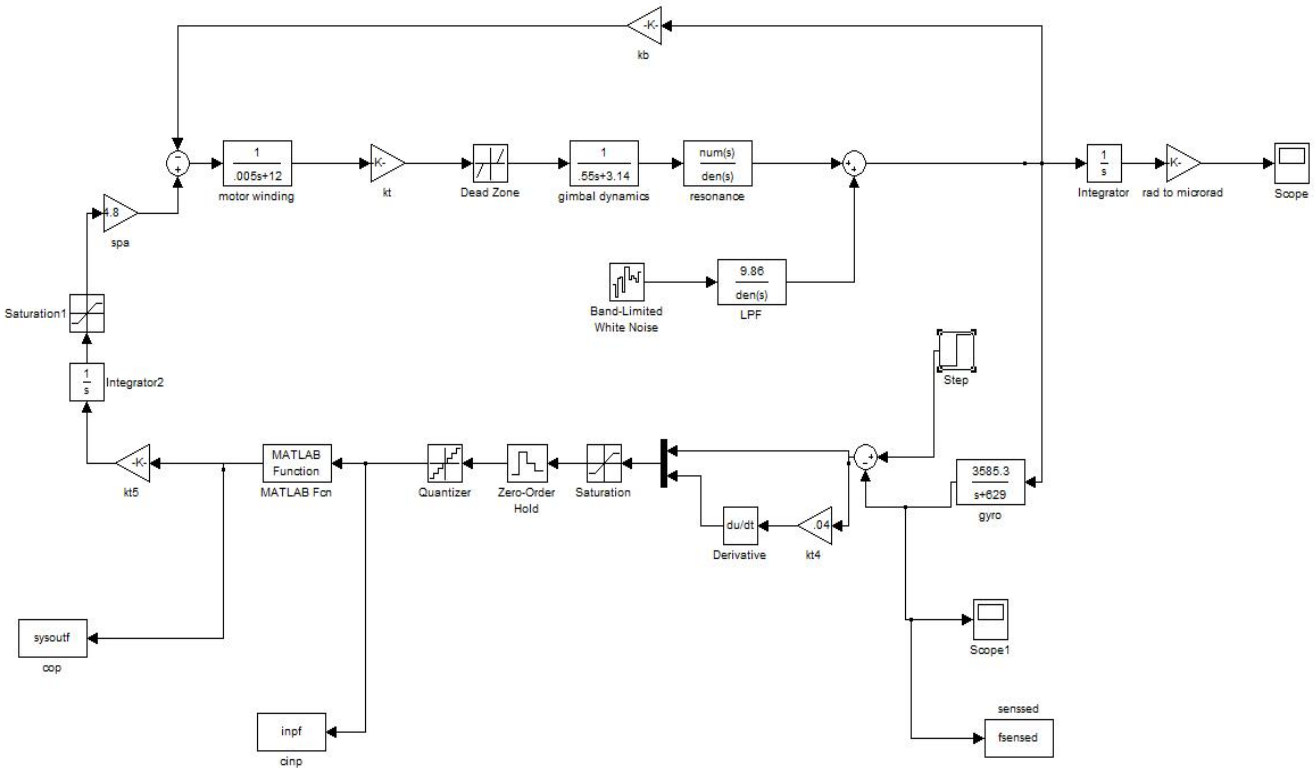


Fig. 5.5: LOS stabilization loop using a fuzzy logic controller

5.2.3 NEURO-FUZZY CONTROLLER

Neuro-fuzzy controller design process for LOS stabilization loop is explained in Chapter 4. As explained, the control law of conventional controller is taken as training data for the neuro-fuzzy model and the model is properly trained. Finally the computation process is coded in MATLAB m-file and linked to the simulation environment. Complete simulink diagram is shown in Fig.5.6.

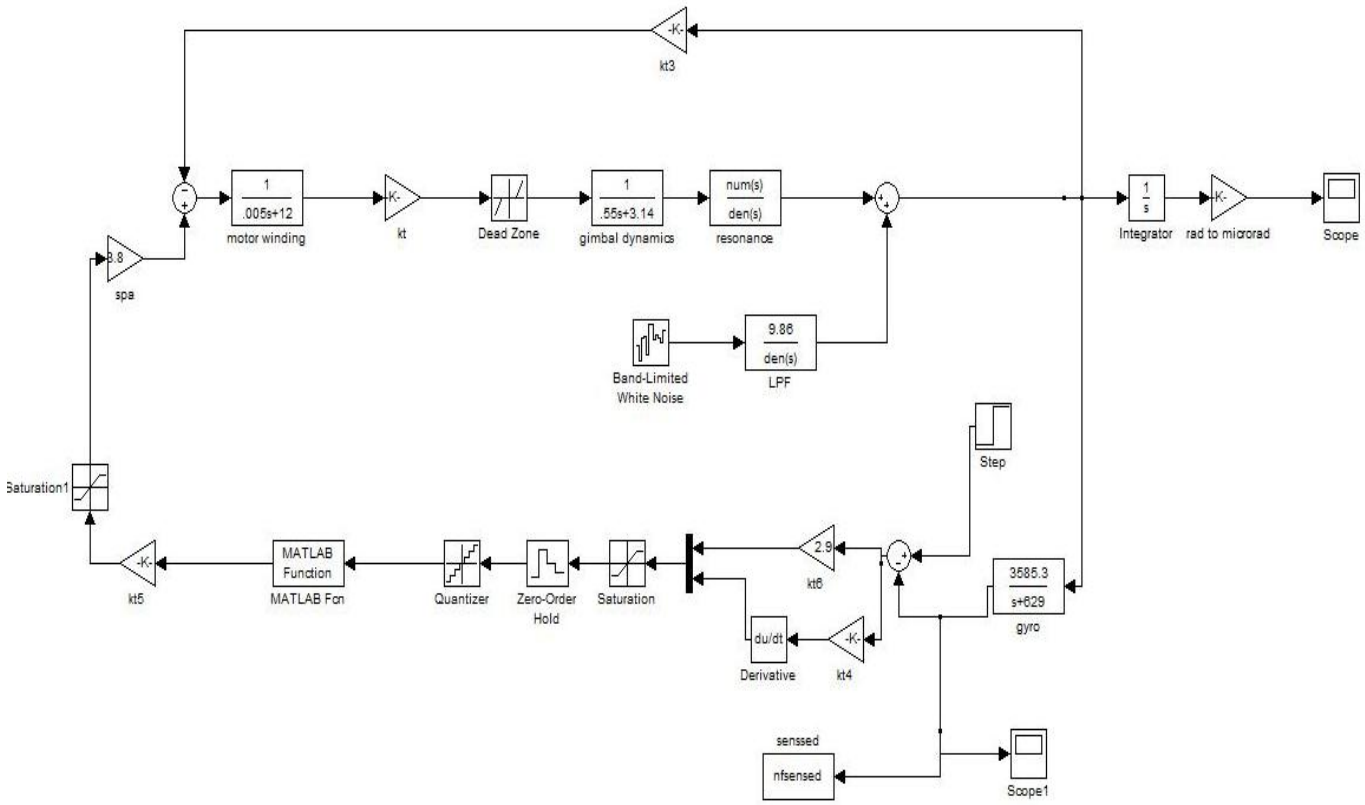


Fig. 5.6: LOS stabilization loop using a neuro-fuzzy logic controller

CHAPTER 6

RESULTS

Neuro-fuzzy model algorithm is evaluated on the non-linear data presented in table 5.1 and the plot of actual data and modeled data is shown in Fig. 6.1

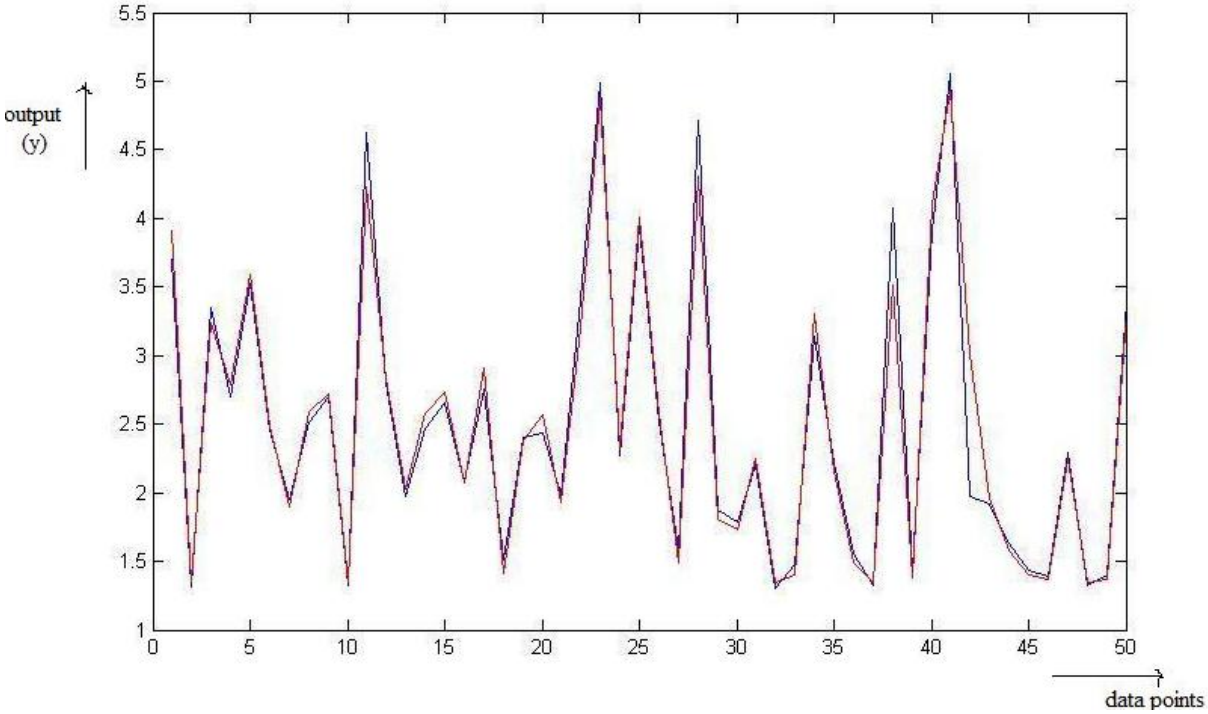


Fig. 6.1 : Comparison of Actual and Modeled output data of non-linear system

Implementation of LOS stabilization loop using conventional, fuzzy and neuro-fuzzy controllers is presented in chapter 5. The simulation results for step command are shown in the following figures:

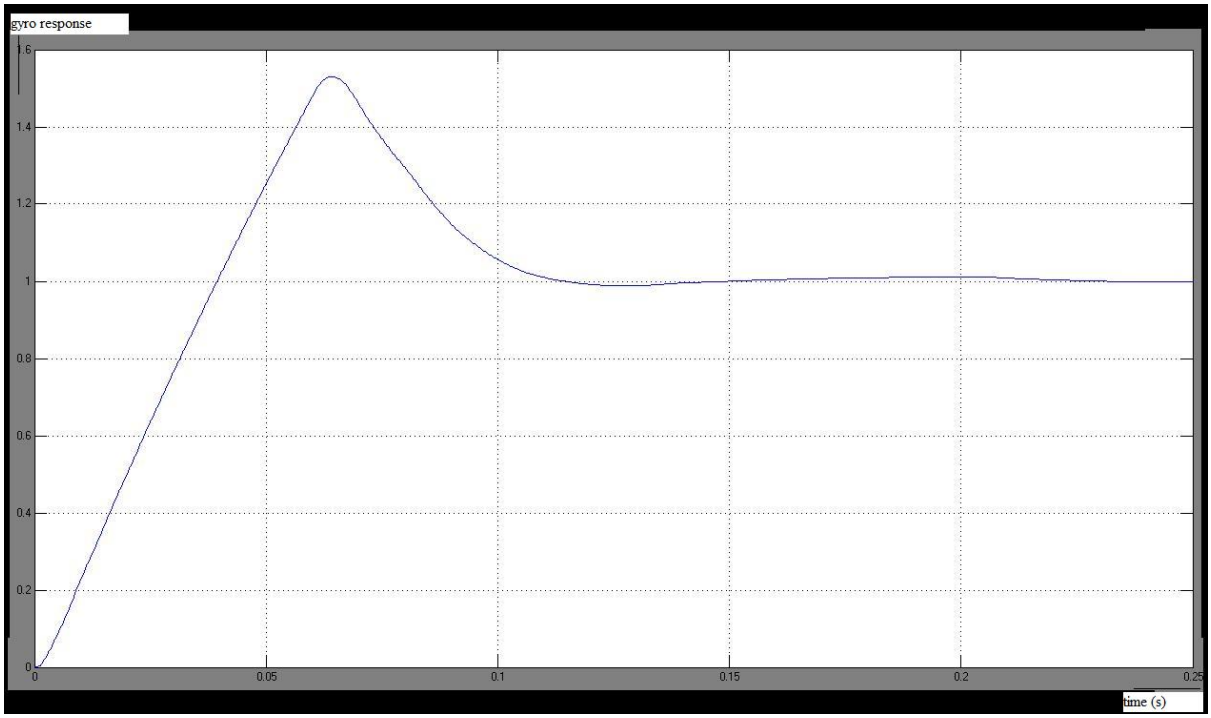


Fig. 6.2: Step response using conventional controller

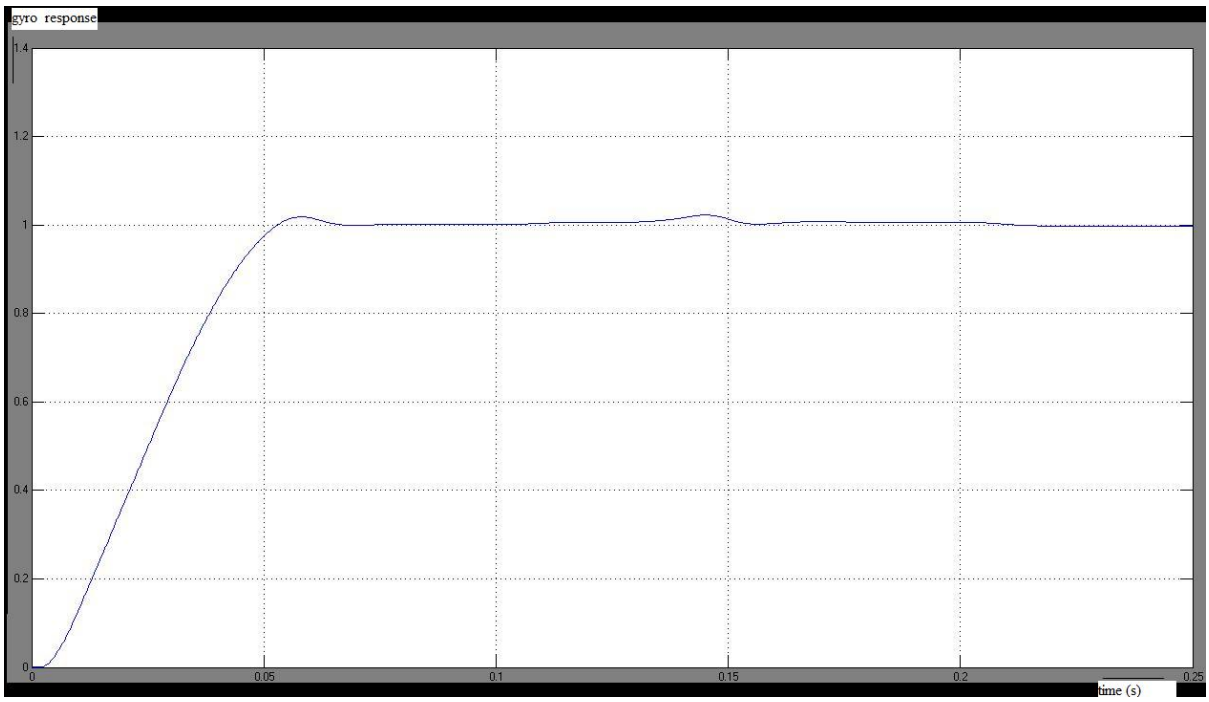


Fig. 6.3: Step response using fuzzy controller

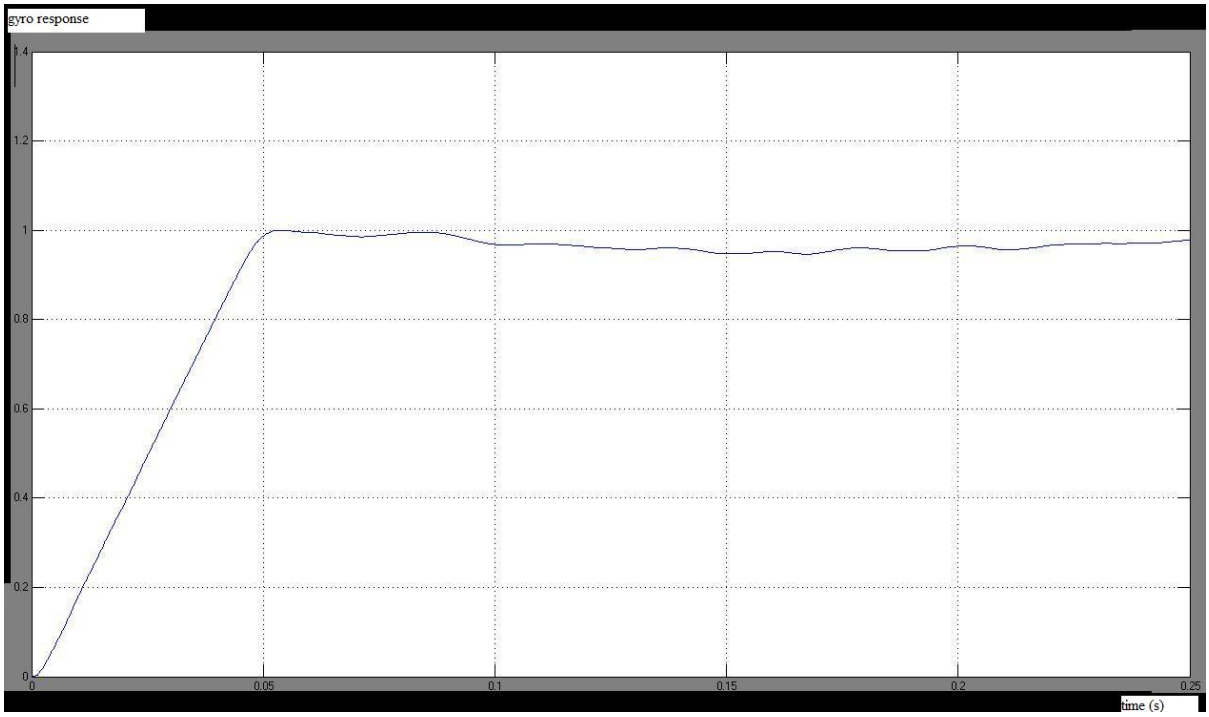


Fig. 6.4: Step response using neuro-fuzzy controller

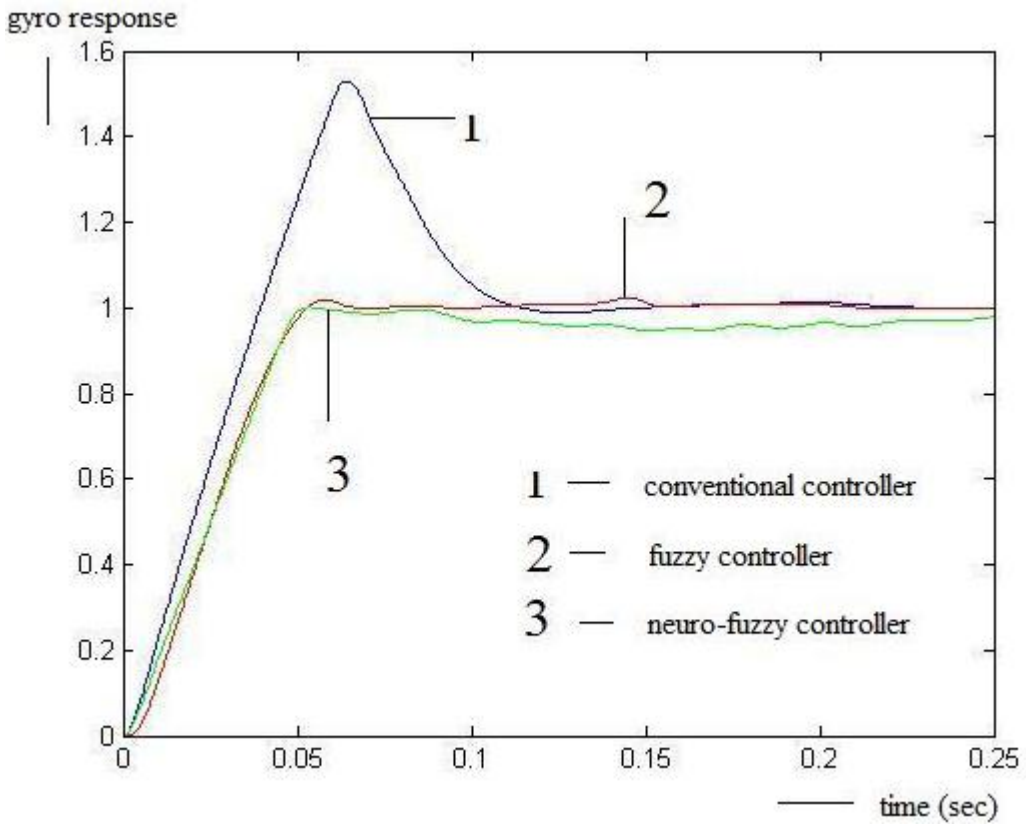


Fig. 6.5: Comparison of step response using three controllers

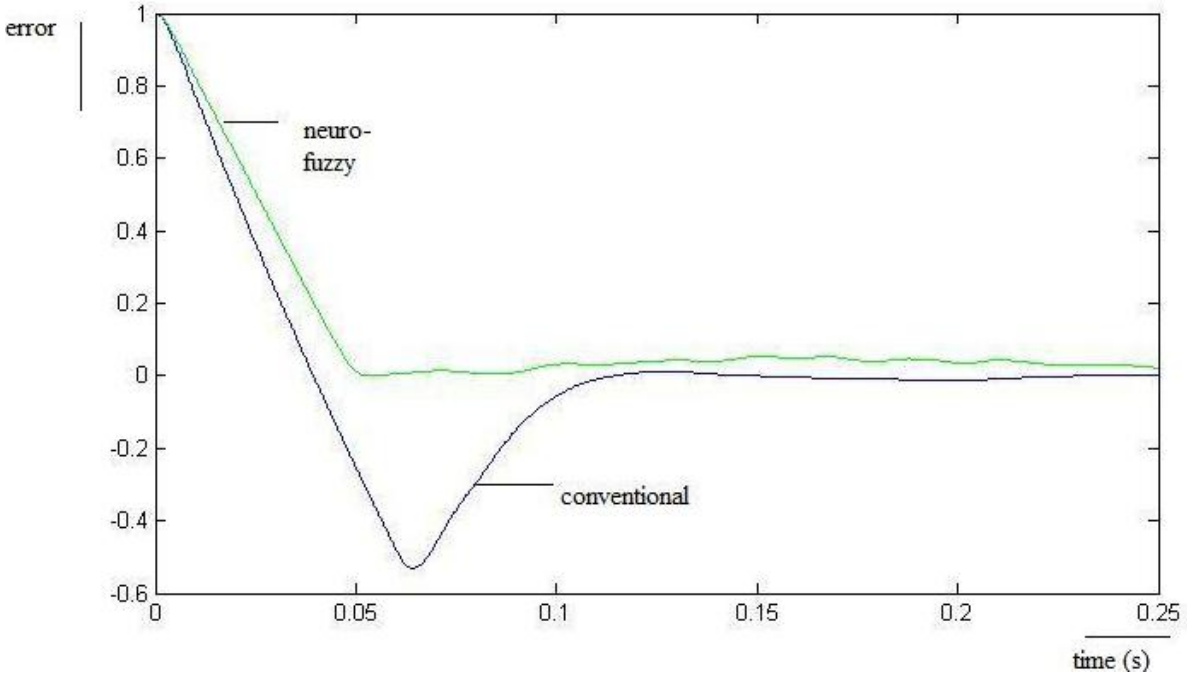


Fig. 6.6 Error comparison of conventional and neuro-fuzzy controller

CHAPTER 7

CONCLUSIONS & FUTURE SCOPE

7.1 CONCLUSIONS

From the results obtained in chapter 6

- The neuro-fuzzy model implemented in MATLAB can be used for system identification and it can model non-linear data appropriately by properly choosing the number of neurons in each layer.

- Neuro-fuzzy controller implemented has given satisfactory results and it is a simple non-linear controller that performs well even in the presence of non-linearities. Since the training data is from conventional controller designed for the linear model of the system, design specifications are also considered in the neuro-fuzzy controller design process.

- Intelligent (Fuzzy and neuro-fuzzy) controllers have performed well in the presence of non-linearities and provide more robust control and have less complex design process than the conventional non-linear controllers.

7.2 FUTURE SCOPE

- Off-line training is used in training the neuro-fuzzy model, a better model can be built with on-line training.

- The controller modeled can be extended to control the overall stabilization loop of two-axis gimbal and also for position control.

REFERENCES

1. Yinghua Lin, and George A. Cunningham III, “A New Approach to Fuzzy-Neural system Modeling,” IEEE Transactions on Fuzzy systems, vol. 3, no. 2, May 1995.

2. J. A. R. Krishna Moorty, Rajeev Marathe and Hari Babu, “Fuzzy controller for line-of sight stabilization systems,” Opt. Eng. 43(6) 1-0 (June 2004). © 2004 Society of Photo-Optical Instrumentation Engineers.

3. Simon Haykins, “Neural Networks A Comprehensive Foundation,” Second Edition. © 1999 by Prentice-hall, Inc.

4. Fuzzy Control, “Kevin M. Passino and Stephen Yurkovich,” © 1998 Addison Wesley Longman Inc.

5. Han-Xiong Li and H. B. Gatland, “Conventional Fuzzy Control and Its Enhancement**,”** IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 26, No. ***5,*** October 1996.

6. T. Poggio and F. Girosi, “Networks for approximation and learning,” in Proc. IEEE, vol. 78, no. 9, Sept.1990, pp. 1481-1497.

7. M. Sugeno and T. Yasukawa, “A fuzzy-logic-based approach to qualitative modeling,” IEEE Trans. Fuzzy Systems**,** vol. 1, no. 1, pp. 7-31, 1993.

8. Stelios Papadakis and John Theocharis, “An Efficient Fuzzy Neural Modeling Approach Using the Fuzzy Curve Concept,” ICES 96.pp. 279-282.

9. Fernando de Castro Junqueira and Ettore Apolônio de Barros, “Development of a Dynamically Tuned Gyroscope (DTG),” ABCM Symposium Series in Mechatronics- Vol. pp. 470-478. © 2004 by ACM.

10. Peter J. Kennedy and Rhonda L. Kennedy, “Direct versus Indirect Line of Sight (LOS) Stabilization,” IEEE Transactions on Control Systems Technology, Vol. 11, No. 1, January 2003.

11. Y. Hayashi, J. Buckley, and E. Czogala, “Fuzzy neural network with fuzzy signals and weights,” International Journal of Intelligent Systems. Vol. 8, pp. 527-537, 1993.

12. H. Ishibuchi, R. Fujioka, and H. Tanaka, “Neural networks that learn from fuzzy if-then rules,” IEEE Trans. Fuzzy System, vol. 1, no. 2, pp. 85-97, 1993.

APPENDIX

MATLAB PROGRAM TO IMPLEMENT NEURO-FUZZY MODEL

```
Inputs = size(x,1);                              % determination of inputs
datasets = size(x,2);                            % determination of samples
a=1;                                        % Setting the parameter 'a'
lij=1.5;                                         % Setting the parameter 'l'
sumN=0;
sumD=0;
point_old = 0;

%------ Fuzzy Curve Generation ---------------------c--------------

for inp=1:Inputs
    r = range(x(inp,:));
    for i=1:datasets
        for k=1:datasets
            phi(inp,i,k)= exp(-(((x(inp,i)-x(inp,k))/(0.2*r))^2));
            sumN = sumN + (phi(inp,i,k))*y(k);
            sumD = sumD + (phi(inp,i,k));
        end
        c(inp,i)= sumN/sumD;
        sumN=0;
        sumD=0;
    end
    clear r;
rc(inp) = range(c(inp,:));
    fprintf('\n Range of Curve %d is %d',inp,rc(inp))
    % Range display on console
end
fprintf('\n');
maxrc = max(rc);
leftinp=0;
reminp=0;
for inp=1:Inputs
    if(rc(inp)<(0.5*maxrc))       % Change the dropping criterion here
        fprintf('\n Curve %d is dropped',inp)
    else
        leftinp = leftinp + 1;
        reminp(leftinp)= inp;
    end
end
fprintf('\n \n \n');
rules = input('Total No. of Rules: ');

fprintf('\n Total number of neurons in the Input layer = %d',leftinp)
fprintf('\n Total number of neurons in the Inference layer = %d',rules)
fprintf('\n Total number of neurons in the Fuzzification Layer layer =
%d',(leftinp*rules))
fprintf('\n \n ');

minY = min(y);
centY=0;
initcent=0;
intY=0;
intvalY= range(y)/rules;
initcent= min(y) + ( (range(y)/(2*rules)) );
```

```
 fprintf('\n The centers of the the output \n --------------------------')
for j=1:rules
    centY(j)= initcent + ((j-1)*intvalY);
        fprintf('\n Interval %d = %f',(j),centY(j))
    intY(j) = min(y)+ (j*intvalY);
end

 fprintf('\n \n ');


%-----------------Centers of Input Intervals----------c----------
centX=0;
for i=1:leftinp

    rangeX(i)= range(ins(i,:));
    minX(i) = min(ins(i,:));

    initcent = 0;

    %deltaXi(i) = rc(reminp(i));

 deltaXi(i) = 0.72; %rangeX(i);          % DELTA Xi
%deltaXi(i) = rc(reminp(i));
    % deltaXi(i) = rangeX(i)/rules;
    initcentX(i) = minX(i) + (rangeX(i)/(2*rules));
    fprintf('\n The centers of the the Input %d \n ------------------------
---',i)
    for j=1:rules
        centX(i,j) = initcentX(i) + ((j-1)*deltaXi(i));

        fprintf('\n  interval %d = %f',(j),centX(i,j))
    end
    fprintf('\n \n ');
end

%--------------------------------------------------------------------



    end
%-------------------------- Setting up initial weights

    for i=1:leftinp
        for j=1:rules
            w1(i,j) = -( centXn(i,cvC(i,j,2))./(a*deltaXi(i)) );
        end
    end

for j=1:rules
v(j) = centY(j);
end


% ------------ FORWARD PROPAGATION OF SIGNAL -----------------------------
-------
% Output and weights calculation

hold on
Nsum = 0;
```

```
Dsum = 0;
maxError = 0.001;

 eta =0.1; % Learning Rate

epochs =1000;
% PI = zeros(1,epochs);
%-------------------------------------------------------------

for t=1:epochs
    mape = zeros(epochs)';

    for k=1:datasets

        inplay = 0;
        fuzlay = 0;
        inflay = 0;
% ----------------------Input Layer

        for i=1:leftinp
            for j=1:rules
                inplay(i,j)= (w0(i,j)+(ins(i,k)*w1(i,j)));
            end
        end
  % ---------------------Inference Layer


        for j=1:rules
            mult = 1;
            for i=1:leftinp
                mult = mult * fuzlay(i,j);
            end
            inflay(j) = mult;
        end

        %-----------------------Output Layer

        sum =0;
        for j=1:rules
            sum = sum + (inflay(j)* v(j));
        end
        out(k)=sum;

        err=0;
err = (y(k)-out(k));
        errorsig(k) = err;
%          err2  = err * out(k) * (1-out(k));
        % --------------------- ERROR CHECK -----------------------------
        many = 0;

        if( abs(err) < maxError)
            many = many+1;
            % fprintf('\n the process converged successfully')

        else
ierr = 0;
            for j=1:rules
%                 ierr(j) =  (err*v(j));
                ierr(j) = err;
```

```
            end
ferrsum=0;
            for j=1:rules
                for i=1:leftinp
                end
            end
   % ----- Input layer Error
            inerrsum1 = 0;
            for i = 1:leftinp
                for j =1:rules

                    inerrsum1(i,j) = err * inflay(j)* v(j) * inplay(i,j)*
abs(inplay(i,j))^(l(i,j)-2);

                end
            end
%------- Updating First Layer weights -----------------------------

            for i = 1:leftinp
                for j=1:rules
                    w1(i,j) = w1(i,j) - (eta* inerrsum1(i,j) * ins(i,k)) ;
                    if(w1(i,j) > 50)
                        w1(i,j) = 50;
                    end
                    if(w1(i,j) < -50)
                        w1(i,j) = -50;
                    end
                end
            end

            for i = 1:leftinp
                for j=1:rules
                    w0(i,j) = w0(i,j) - (eta* inerrsum0(i,j)) ;
                    if(w0(i,j) > 50)
                        w0(i,j) = 50;
                    end
                    if(w0(i,j) < -50)
                        w0(i,j) = -50;
                    end
                end
            end
for j=1:rules
%                 v(j) = v(j) + (0.01 * err * inflay(j));
                  v(j) = v(j) + (0.01 * err * inflay(j)*inflay(j));
            end
end
if(y(k) > 0)
         mape(t)  = mape(t) + abs((y(k) - out(k))/y(k));
        end
        mape(t) = (mape(t)*100)/datasets;
    end
    %PI(t) = sqrt(Nsum)/Dsum;
    plot(t,mape(t));
end


for k=1:datasets

        fuzlay = 0;
        inflay = 0;
```

```
        % ---------------------Fuzzification Layer

        for i=1:leftinp
            for j=1:rules
                fuzlay(i,j)= (exp(-((abs( w0(i,j)+(ins(i,k)*w1(i,j))
))^l(i,j))));
            end
        end

        % ---------------------Inference Layer

        for j=1:rules
            mult = 1;
            for i=1:leftinp
                mult = mult * fuzlay(i,j);
            end
            inflay(j) = mult;
        end

        %-----------------------Output Layer

        sum =0;
        for j=1:rules
            sum = sum + (inflay(j)* v(j));
        end
        out(k)=sum;
        % ----------------------- ERROR CALCULATION -----------------------
-
        %         Nsum =  Nsum +(y(k) - out(k))^2;
        %         Dsum = Dsum + abs(y(k)) ;
        %         err = ((y(k)-out(k)))/y(k);

        %             PI(k) = sqrt(Nsum)/Dsum;   Performance Index
        %         hold on
        %             plot(k,PI(k),'.');
        err=0;
        err = (y(k)-out(k)); % Error propagation   *out(k)*(1-out(k))
        errorsig(k) = err;

end

plot(y,'b');
    hold on
plot(out,'r');

    mape = 0;
    for k=1:datasets
        if(y(k) > 0)
         mape  = mape + abs((y(k) - out(k))/y(k));
        end
    end

    fprintf(' mape = %f \n\n',(100*mape)/datasets);
```

MATLAB PROGRAM TO IMPLEMENT FUZZY-LOGIC CONTROLLER

```
liv=7;

% STANDARD DEVIATION OF INPUT CURVES
sigma=[0.35 0.1 0.04 0.013 0.04 0.1 0.35;
       0.141 0.141 0.12 0.05 0.12 0.141 0.141];
sigmau=[0.141 0.142 0.1 0.007 0.1 0.142 0.141];

% CENTERS OF INPUT CURVES
% ctrs=[-100 -25 -10 0.0 10 25 100;
%       -100 -66 -20 0.0 20 66 100];
% ctru = [-200 -114 -30 0.0 30 114 200];

ctrs=[-1.0 -0.25 -0.1 0.0 0.1 0.25 1.0;
      -1.0 -0.66 -0.2 0.0 0.2 0.66 1.0];
ctru = [-1.0 -0.57 -0.15 0.0 0.15 0.57 1.0];

% GENERATING GAUSSIAN MEMBERSHIP FUNCTIONS

for i=1:2
    for j=1:liv
         mi(i,j)= exp((-((u(i)-ctrs(i,j))^2))/(2*((sigma(i,j)^2))));
    end

end

% AND METHOD OF EVALUATION
for j=1:liv
        for k=1:liv
            miinf(j,k) = min(mi(1,j),mi(2,k));
          %  mu(j,k,i)= exp((-((u(i)-ctrs(i,k))^2))/(2*((sigma(i,k)^2))));
        end
end
%RULE BASE
rij=[1 1 1 1 2 3 4;
    1 1 1 2 3 4 5;
    1 1 2 3 4 5 6;
    1 2 3 4 5 6 7;
    2 3 4 5 6 7 7;
    3 4 5 6 7 7 7;
    4 5 6 7 7 7 7];

% AGGREGATION
agg=zeros(1,7);
i=0;
hold on
for j=1:liv
    maxm=0;
    for k=1:liv
        for l=1:liv
            if(rij(k,l)==j)
                maxm = max(maxm,miinf(k,l));
               i=i+1;
               plot(i,miinf(k,l),'.');
            end
        end
    end
agg(j) = maxm;
```

```
end

% DEFUZZIFICATION
num=0;
den=0;
for j=1:liv
        num = num + (agg(j)*ctru(j));
        den = den + agg(j);
end
deout = num/den;
```