NEURO FUZZY CONTROL OF ROBOTIC ARM MOVEMENT

Thesis report submitted in the partial fulfillment of
requirements for the award of the degree of

Master of Engineering

in

Electronics Instrumentation & Control Engineering

Submitted by:

ANSHUL

Roll No. 80751026

Under Guidance of

Ms. GAGANDEEP KAUR
Lecturer (SG), EIED

DEPARTMENT OF ELECTRICAL AND INSTRUMENTATION ENGINEERING

THAPAR UNIVERSITY

Patiala- 147001, Punjab, INDIA

June-2009

THAPAR UNIVERSITY
Patiala- 147001, Punjab, INDIA

CERTIFICATE

This is to certify that my work presented in this thesis entitled "Control of Robotic Arm by Neuro-Fuzzy" in partial fulfilment of the requirements for the award of the degree of Master of Engineering in Electronics Instrumentation & Control Engineering at Thapar University, Patiala is an original record under supervision and guidance of **Ms.Gagandeep Kaur**. The matter embodied in this thesis has not been submitted in any other university/ institute for the award of Master of Engineering.

Anshul
(ANSHUL)
Signature of Student

Date: 3/7/09.....

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Gagandeep
(Ms. Gagandeep Kaur)
Lecturer (SG), EIED
Thapar University, Patiala
.

S. Ghosh 07/07/09
(Dr. Smarajit Ghosh)
Professor & Head, EIED
Thapar University, Patiala

RK Sharma 13/7/09
(Dr. R. K. Sharma)
Dean of Academic Affairs
Thapar University, Patiala

ACKNOWLEDGEMENT

First of all I would like to thank the almighty, who has always guided me to work on the right path of life. This work would not have been possible without the encouragement and able guidance of my supervisor **Ms.Gagandeep Kaur**, Lecturer (SG), Electrical and Instrumentation Department. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. Her timely feedback and editorial comments were also valuable for writing this thesis.

I am thankful to the Head of the Department of Electrical and Instrumentation Engineering, Thapar University, Patiala **Dr. Samarjit Ghosh**, for his encouragement support and for providing the facilities for the completion of this thesis.

I am also thankful to the entire faculty and staff members of Electrical and Instrumentation Department for their direct-indirect help and co-operation. Their enthusiasm and optimism made their experience both rewarding and enjoyable.

I am deeply indebted to my parents and brother Vipul and sisters Vanita, Manu for their inspiration and ever encouraging moral support, which enabled me to pursue my studies. I am also thankful to the authors whose works I have consulted and quoted in this work. Last but not the least I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. Lastly I would like to thank friends for always supporting me and being there on my side whenever I needed them.

ANSHUL

ROLL NO.:80751026

ABSTRACT

This thesis first outlines the theory, historical background, and application of neural networks and fuzzy logic. The review of neural networks and fuzzy logic is followed by a discussion of the combination of the two technologies -- neuro-fuzzy techniques. The two tools have been successfully combined to maximize their individual strengths and compensate for shortcomings. A survey is given of previous work done in applying these technologies to control systems.

The problem of moving a robotic arm in the presence of an obstacle is discussed. In particular, trajectory planning of a planar, redundant manipulator is studied. The primary weakness of previous methods for determining acceptable trajectories is the massive amount of computer time needed to obtain a solution. Neuro-fuzzy systems offer not only the benefit of the parallel nature of its computations, but also the ability to learn the control of an arm by following a human's example.

Several neuro-fuzzy controllers are trained using sample data obtained from a human's control of a robotic arm. Their performance is quantified and compared. It is shown that the definition of the fuzzy membership functions plays a significant role in the ability of the neuro-fuzzy controller to learn and generalize. Possible directions for future work are suggested.

LIST OF CONTENTS

Certificate…….....……...i
Acknowledgement…….....……...ii
Abstract…….....……iii
List of contents...…….....……...iv
List of figures……...…….....……...vi
List of tables……..…….....……viii
List of abbreviations……....ix

CHAPTER 1

INTRODUCTION…..... 1
1.1Introduction about robotics.....……....1
1.1.1 Basic components.....1
1.1.2 Three laws of robotics.....2
1.2 Control of robotic arm.....……..……...……...2
1.3 Fuzzy Logic.....……..…….4
1.4 Neural networks.....9
1.5 Neuro-Fuzzy systems.....……..……..17
1.5.1 Adaptive neuro-fuzzy inference system……..20

CHAPTER 2

REVIEW OF LITERATURE……..…….....25
2.1 Fuzzy Logic……...25
2.2 Neural Network……..……...27
2.3 Neuro-Fuzzy Technology……..……29
2.4 Trajectory Planning……...32

CHAPTER 3

ROBOTIC ARM PROBLEM……..……..34
3.1 INTRODUCTION……....34
3.1 Problem formulation…….....34

CHAPTER 4

NEURO FUZZY SOLUTION……...…….....39

4.1 Introduction……..…….....39

4.2 State Representation……...……..41

4.3 Robotic Arm Simulation……...44

4.4 Neuro Fuzzy Programming……...46

4.5 Neuro Fuzzy Controller……48

CHAPTER 5

EXPERIMENTAL RESULTS…….. ……...52

5.1 INTRODUCTION.....52

5.2 SIMULATION RESULTS.....61

CHAPTER 6

CONCLUSION & FUTURE WORK……..…….69

6.1Conclusion…….69

6.2 Future work……..70

REFERENCES……..71

APPENDIX-A……...76

Linear Function……...…….76

Sinusoidal function…….....77

Two-dimensional input function…….....77

The learning rate constant…….....79

The momentum constant…….....79

The fuzzy membership functions…….....80

LIST OF FIGURES

Figure1.1 A typical fuzzy system.....5
Figure1.2 A sample fuzzy system.....6
Figure1.3 An artificial neuron.....10
Figure1.4 A feed-forward neural network.....10
Figure1.5 A forward neural network.....11
Figure1.6 the Graph of sample activation function.....14
Figure1.7 A fuzzy system whose membership functions are adjusted by a neural network.....17
Figure1.8 A fuzzy system defined by a neural network.....17
Figure1.9 A neural network of fuzzy neurons.....18
Figure1.10 A fuzzy system with neural network rule base.....19
Figure 1.11. ANFIS architecture with two fuzzy if-then rules.....21
Figure3.1 A diagram of robotic arm in presence of an obstacle.....33
Figure 3.2 A sample configuration for the problem application.....34
Figure3.3 The Robotic arm definition variables.....35
Figure3.4 Normalized dimensions for remote arm and reachable region.....37
Figure 4.1 A sample neuro fuzzy system 39
Figure4.2 The relative position of the objects is same.....41
Figure4.3 The obstacle and goal switch places.....41
Figure4.4 The state representation for the obstacle problem.....42
Figure4.5 The robotic arm link data structure.....43
Figure4.6 A sample training trajectory obtained from the simulator.....44
Figure4.7 The neuro-Fuzzy data structure.....45
Figure4.8 A sample FNN data file.....46
Figure4.9 Determination of a collision between the obstacle and the link.....48
Figure4.10 Source code for collision detection function.....49
Figure5.1 The Fuzzy membership function definition FNN1.....50
Figure5.2 The Fuzzy membership function definition FNN2.....51

Figure5.3 The Fuzzy membership function definition FNN3……...51
Figure5.4 The RMS training error of FNN1, FNN2, and FNN3……...52

Figure5.5 Histogram of the neuro-fuzzy controller for
FNN1, FNN2, FNN3……...52
Figure5.6 Samples of collisions while under neuro-fuzzy control…….....53
Figure5.7 Samples of neuro-fuzzy control in which constraint limits
were exceeded……...54

Figure5.8 Samples of successful control by the neuro-fuzzy controller…….....55
Figure5.9 The fuzzy membership function definition FNN4……55
Figure5.10 The fuzzy membership function definition FNN5……..56
Figure5.11 The RMS training error of FNN4 and FNN5……..56
Figure5.12 Performance histogram of the neuro-fuzzy controller for
FNN4 and FNN5…….....57

Figure5.13 The fuzzy membership function definition FNN6……..57

Figure5.14The fuzzy membership function definition FNN7……....58

Figure5.15 The RMS training error of FNN6 and FNN7…….....58
Figure5.16 Performance histogram of the neuro-fuzzy controller
for FNN6 and FNN7……...59

Figure5.17 A comparison of the final RMS training error…….....60
Figure5.18 A comparison of performance of all the controllers……....60
FigureA.1 The performance of the neuro-fuzzy system on a
linear function…….....74
FigureA.2 The performance of the neuro-fuzzy system on a
sinusoidal function……....75

FigureA.3 The desired output and the actual output for the 2D
test data……76

FigureA.4 The effect of the learning rate constant…….....77

FigureA.5 The effect of momentum term…….....78

FigureA.6 The effect of the fuzzy membership function definition…….....79

FigureA.7 The fuzzy membership function, test definition…….....80

LIST OF TABLES

Table 3.1 The physical characteristics of a remote robotic arm....……...36
Table 5.1 Data when obstacle is at (0.5, 2.3), goal at (-0.82, 2.31)....……..61
Table 5.2 Data when obstacle is at (1.3, 2.2), goal at (-0.64, 2.31)....……..61
Table 5.3 Data when obstacle is at (0, 2.44), goal at (-2.03, 1.01)....……...62
Table 5.4 Data when obstacle is at (1, 2.44), goal at (-1.97, 0.59)....……...62
Table 5.5 Data when obstacle is at (1, 2.44), goal at (2.14, 1.40)....……63
Table 5.6 Data when obstacle is at (2.25, 1.2), goal at (1.96, 1.65)....…….63
Table 5.7 Data when obstacle is at (0, 0.23), goal at (-1.52, 0.75)...……64
Table 5.8 Data when obstacle is at (0.2, 2.3), goal at (-0.44, 2.52)...……...64
Table 5.9: Data when obstacle is at (1.5, 2.1), goal at (0.85, 2.31).....65
Table 5.10: Data when obstacle is at (-1.5, 1.3), goal at (-1.92,-0.12).....65
Table 5.11: Data when obstacle is at (0.1, 1.5), goal at (-1.41,-1.50).....66
Table 5.12: Data when obstacle is at (-1.5, 1.5), goal at (-1.98,-.0.99).....66

LIST OF ABBREVATIONS

FMM	Fuzzy neural network
MRAC	Model reference adaptive control
FAM	Fuzzy associative memory
AFAM	Adaptive Fuzzy associative memory
DCL	Differential competitive learning
RSFC	Rule Self-regulating Fuzzy Controller
FMM	Fuzzy Membership Model
GARIC	Generalized approximate-reasoning-based intelligent Control
DPP	Dynamic path planning
DCET	Doubly connected edge table
RMS	Root mean square
NEG	Negative
ZR	zero
POS	Positive
NL	Negative large
NM	Negative medium
NS	Negative small
PS	Positive small
PM	Positive medium
PL	Positive large
AI	Artificial intelligence

CHAPTER 1
INTRODUCTION

1.1INTRODUCTION TO ROBOTICS:

Robot technology is an applied science that is referred to as a combination of machine tools and computer applications. It includes such diverse fields as machine design, control theory, microelectronics, computer programming, artificial intelligence, human factors, and production theory [1].

1.1.1 Basic components

The basic components of an industrial robot are:

- The manipulator
- The end effector (which is the part of the manipulator).
- The power supply
- The controller

The manipulator, which is the robot's arm, consists of segments jointed together with axes capable of motion in various directions allowing the robot to perform work.
The end effector which is a gripper tool, a special device, or fixture attached to the robot's arm, actually performs the work.
Power supply provides and regulates the energy that is converted to motion by the robot actuator, and it may be either electric, pneumatic, or hydraulic.
The controller initiates, terminates, and coordinates the motion of sequences of a robot. Also it accepts the necessary inputs to the robot and provides the outputs to interface with the outside world [2].
Robots exhibit three key elements:

(1) Programmability, implying computational or symbolic manipulative capabilities that a designer can combine as desired (a robot is a computer).

(2) Mechanical capability, enabling it to act on its environment rather than merely function as a data processing or a computational device (a robot is a machine).

(3) Flexibility, it can operate using a range of programs and manipulate and transport materials in a variety of ways [3].

1.1.2 Three laws of robotics

Asimov also proposed three "Laws of Robotics", and later added a 'zeroth law'.
Law Zero: A robot may not injure humanity, or, through inaction, allow humanity to come to harm.

Law one: A robot may not injure a human being, or, through inaction, allow a human being to come to harm, unless this would violate a higher order law.

Law two: A robot must obey orders given it by human beings, expect where such orders would conflict with a higher order law.

Law Three: A robot must protect its own existence as long as such protection does not conflict with a higher order law [4].

1.2CONTROL OF ROBOTIC ARM

The most common manufacturing robot is the robotic arm. A typical robotic arm is made up of seven metal segments, joined by six joints. The computer controls the robot by rotating individual step motors connected to each joint (some larger arms use hydraulics or pneumatics).

The problem addressed in this thesis is that of moving a robotic arm in the presence of an obstacle. For robots to become effectively used in a wide range of applications, they must gain the ability to work in unpredictable and changing environments. Robots used in space exploration and construction will have to sense their environment and carry out their tasks regardless of the presence of objects in the work area.

The ability to move an arm around an obstacle and to a goal is an intuitive skill for human beings. Translating that skill into instructions for a robotic arm, however, is not an easy task [5].The ability of a machine to emulate human behavior has always been the goal of artificial intelligence.

Neural networks and fuzzy logic systems are two of the most important results of research in the area of artificial intelligence. They have been effectively applied to everything from voice and image recognition to toasters and automobile transmissions. Neural networks are best known for their learning capabilities. Fuzzy logic is a method of using human skills and thinking processes in a machine.

While neural networks and fuzzy logic have added a new dimension to many engineering fields of study, their weaknesses have not been overlooked. In many applications, the training of a neural network requires millions of iterative calculations. Sometimes the network cannot adequately learn the desired function. Fuzzy logic systems, on the other hand, acquire their knowledge from an expert who encodes his knowledge in a series of IF/THEN rules. Fuzzy logic systems are easy to understand because they mimic human thinking. The problem arises when systems have many inputs and outputs. Obtaining a rule base for large systems is difficult, if not impossible [6].

Prompted by the weaknesses inherent in the two technologies and their complementary strengths, researchers have looked all ways of combining neural networks and fuzzy logic. Due to the relative youth of this field of study, a consensus on the best way to utilize their individual strengths and compensate for their individual shortcomings has not yet been established. Consequently, research of neuro-fuzzy systems is established. The technique used in this work replaces the rule-base of a traditional fuzzy logic system with a back propagation neural network. Various techniques used or controlling robotic arm are:

1. Fuzzy logic
2. Neural networks
3. Neuro-fuzzy systems

Using neuro-fuzzy techniques, a robotic arm can be trained to plan its movements to avoid a collision with obstacles in its vicinity.

1.3 FUZZY LOGIC

The concept of Fuzzy Logic (FL) was conceived by Lotfi Zadeh, a professor at the University of California at Berkley, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership.

FL is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. FL provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy, or missing input information. FL's approach to control problems mimics how a person would make decisions, only much faster. Fuzzy logic imitates the logic of human thought, which is much less rigid than the calculations computers generally perform [7].

Fuzzy logic has its roots in the work of renegade mathematicians who saw the value of a multivalent logic system. The credit for fuzzy logic's application to the areas of control and in engineering belongs solely to Lotfi Zadeh. In 1965, he formalized fuzzy set theory and in 1973 he brought fuzzy set theory into the context of control systems. According to Lotfi Zadeh, fuzzy logic brings to control systems a "higher machine intelligence quotient" [8].

On a mathematical level, fuzzy logic abandons the strict bivalent logic of TRUE and FALSE, ONE and ZERO, ON and OFF. Fuzzy logic allows for half-truths. Take for an example the scientific classification of a whale. We are taught in grade school, much to our surprise, that a whale is a mammal. A whale is a mammal because, among other reasons, it is warm blooded, gives birth to live baby whales, feeds those young with milk, and also grows hair. This classification system is a perfect example of the traditional bivalent logic that has dominated science for centuries. Despite the fact that it looks like a fish, it swims like a fish, it smells like a fish, and every third grader in the country is skeptical when told that it is not a fish, the whale is 100% mammal, 0%fish. If a fuzzy logician were classifying the whale, he would allow the whale to belong to both the mammal set and the fish set, to certain degrees [9].

On an engineering level, fuzzy logic provides a platform for easily encoding human knowledge into the control of a system. It has been used in an increasing number of applications, especially in Japan. The Sendai railway in Japan is controlled by fuzzy logic controllers. Applications have been developed in tracking problems, tuning, interpolation, classification, handwriting, voice recognition, and image stabilization in video cameras, washing machines, vacuum cleaners, air conditioners, electric fans, hot plates, and Lexus automatic transmissions.

An inverted pendulum experiment was demonstrated in 1987 that "produced balancing responses nearly 100 times shorter than those of conventional PID controller" [10].

PID controllers and variations thereof work well in many control systems. But when the system to be controlled contains uncertainty or is highly complex, poorly understood, or nonlinear, fuzzy control may work better. Fuzzy logic is a method of characterizing knowledge in terms of fuzzy sets and a rule base. A fuzzy system has one or more inputs that are fuzzified, a rule base that is evaluated according to the inputs, and one or more outputs that are defuzzified into "crisp" values. Bringing fuzzy logic to control problems is a way to use a human expert's knowledge about an analog process in a digital computer. Fuzzy logic is not always the best way to solve a control problem, but it offers several advantages. A fuzzy system structure is illustrated in figure 1.1 [11].

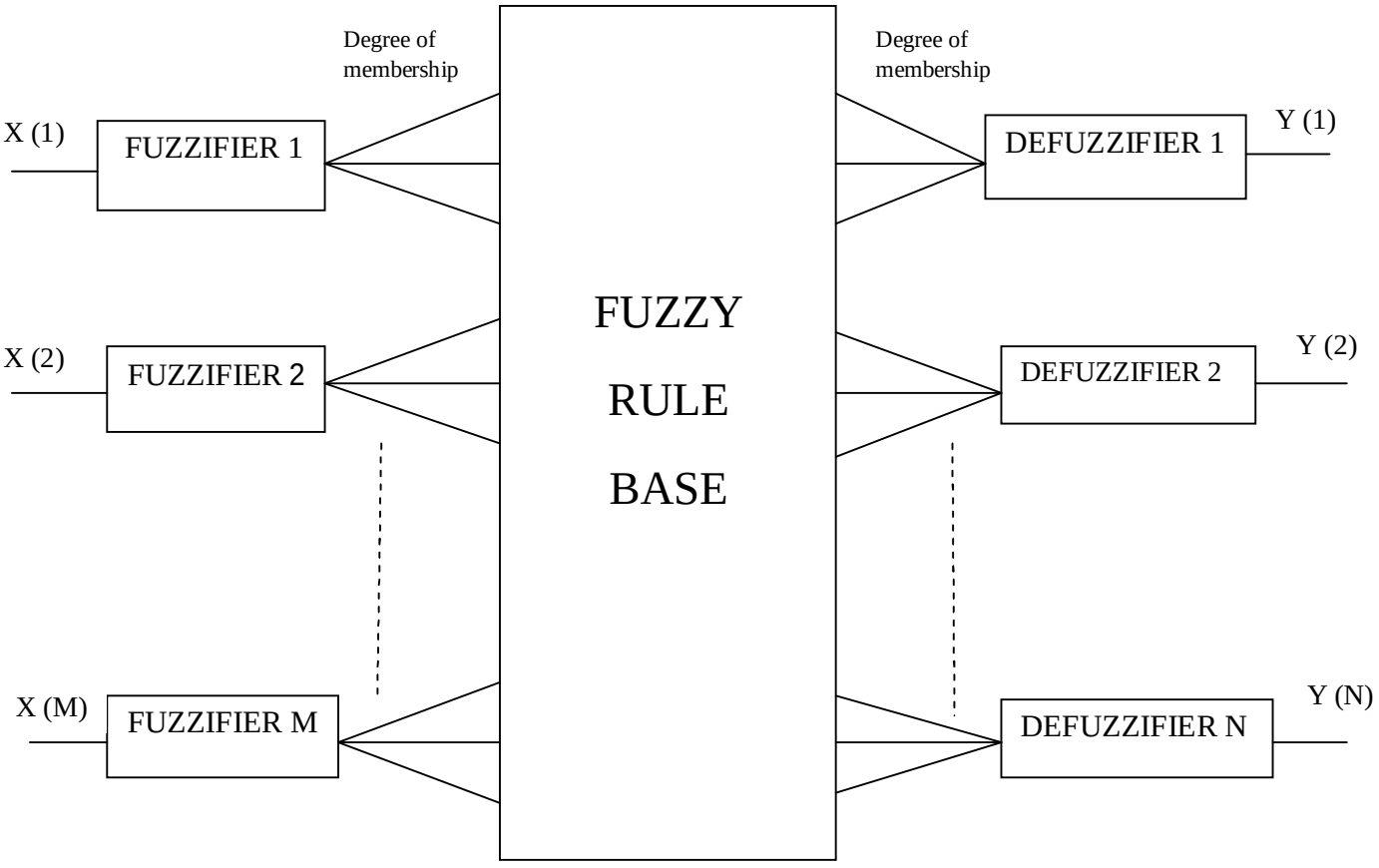


Fig 1.1 A typical fuzzy system

Fuzzy sets are values to which a variable can belong. Fuzzy control involves describing the control procedure in terms of subjective descriptions like "very low", "low", "just right", "high", and "very high". Suppose a human were to describe the operation of an elevator. If the elevator did not stop exactly on the desired floor, he might describe the elevator's error in terms of a fuzzy variable like the one above.

If the elevator stopped three feet too high, the position of the elevator would definitely be "very high". Six inches too high might just be "high." The question is, "At what height did the elevator go from just being 'high' to 'very high'?" Fuzzy logic avoids this problem by its multivalent nature. A fuzzy variable can have a certain degree of membership to "high" and a degree of membership to "very high." Often times these fuzzy sets are graphed as shown in figure 1.2 [12].

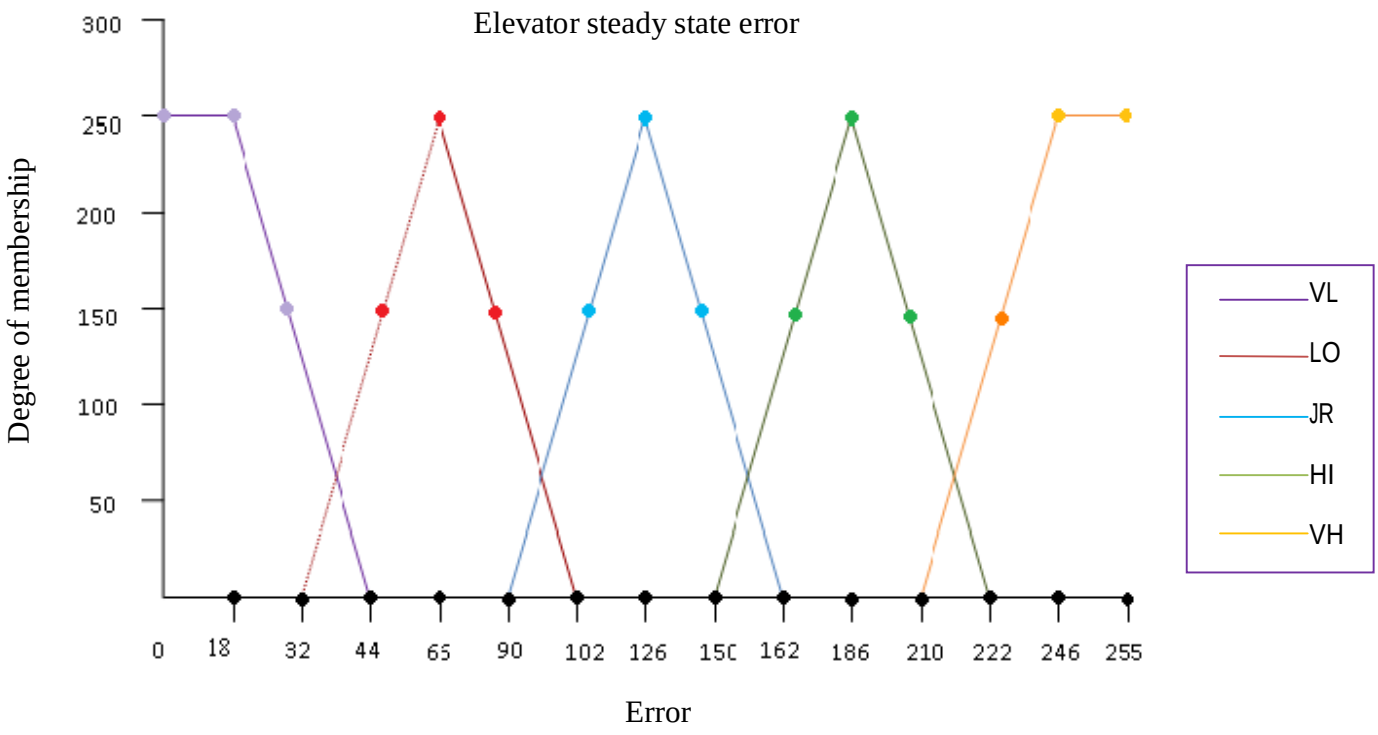


Fig 1.2 A sample fuzzy system.

The control rules are defined with fuzzy sets. The control rules are usually in the form of IF/THEN statements. "If ERROR is VERY HIGH, then VOLTAGE is NEGATIVE LARGE." or "If ERROR is VERY LOW and LOAD is VERY HIGH, then MOTOR POWER is POSITIVE LARGE." Combining several if-then conditions necessitates the use of fuzzy logic operations. Zadeh defines these basic operations in equations 1, 2, and 3 [13].

$$A \cap B = \min(A, B) \quad (1)$$

$$A \cup B = \max(A, B) \quad (2)$$

$$A^{I} = 1\text{-}A \quad (3)$$

Consider for an example that the elevator was located a distance of 1.5 feet above the desired floor. The distance of 1.5 feet might have a 0.4 degree of membership in ERROR's fuzzy value VERY HIGH. The weight of a child in the elevator might cause a 0.5 degree of membership in LOAD's fuzzy value ZERO. According to equations, the rule's antecedent "if ... and ..." would evaluate as min (0.4, 0.5) = 0.4. Therefore, in determining the control, the output variable would have a 0.4 degree of membership in MOTOR POWER's fuzzy value of NEGATIVE SMALL. Usually, an output of a fuzzy system is determined by more than one rule and the total output is calculated according to the centroid of the output membership functions [14].

Sometimes a fuzzy variable has a degree of membership to a fuzzy value of 0.5. "The elevator is not quite low enough to qualify as 'high', but it isn't 'very high' either." For the special case of A = 0.5, evaluates as *(not A) =A*. This contradicts traditional, bivalent logic. Notice, however, that traditional zero-or-one logic is a special case of fuzzy logic. If the state space is considered to be a hyperspace cube, whose axes correspond to individual fuzzy variables, traditional logic holds true for the corners of the cube. Fuzzy logic applies to all points in the state space [15].When is fuzzy logic appropriate? Fuzzy logic is helpful in situations where the control variables are continuous.

Fuzzy rules often take the place of a math model. Therefore, fuzzy logic is useful if a mathematical model of a process does not exist, is too difficult to encode, is too complex to be evaluated in real-time, or requires too much memory.

Other situations that may make fuzzy control advantageous are when there are high ambient noise levels, it is important to use inexpensive sensors, or it is important to use low precision microcontrollers [16]. They are easier to prototype and implement and simpler to describe and verify. They can be maintained and extended with greater accuracy in less time.

What are the weaknesses of fuzzy logic control? One problem is that the control of some systems cannot be easily specified in terms of an IF/THEN rule base. An example of such a system would be a robotic arm operating in the presence of an obstacle. Also, sometimes the 'experts' providing the rule base disagree among themselves. This was demonstrated at the Kawasaki Steel Corporation in Japan. A fuzzy logic control system was installed to help operators make decisions regarding control of a blast furnace [17]. The researchers summarized, "There is a slight difference in knowledge between a multiple numbers of experts. In the case of expert systems which process ill-structured problems, and 100% success could not be possible." Their data suggests that in practice, the operators ignored the expert system's suggestion over 15% of the time.

1.3 NEURAL NETWORKS

A neural network is a computational model of the brain. Neural network models usually assume that computation is distributed over several simple units called neurons, which are interconnected and operate in parallel (hence, neural networks are also called parallel-distributed-processing systems or connectionist systems). The most popular neural network is the multi-layer perceptron, which is a feed forward network. For artificial neural network to give any results it must be trained with series of examples and conditions. During the training neural network “learns” the governing relationships in given data sets e.g. input vectors to produce right solutions e.g. output vectors. For this purpose, back-propagation training algorithm is used. It is an iterative algorithm for minimizing the mean square error between predicted and desired output values [18].

All signals flow in a single direction from the input to the output of the network. Feed forward networks can perform static mapping between an input space and an output space: the output at a given instant is a function only of the input at that instant.

Recurrent networks, where the outputs of some neurons are fed back to the same neurons or to neurons in layers before them, are said to have a dynamic memory, the output of such networks at a given instant reflects the current input as well as previous inputs and outputs. Implicit ‘knowledge’ is built into a neural network by training it.

Some neural networks can be trained by being presented with typical input patterns and the corresponding expected output patterns. The error between the actual and expected outputs is used to modify the strengths, or weights, of the connections between the neurons. This method of training is known as supervised training. In a multi-layer perceptron, the back-propagation algorithm for supervised training is often adopted to propagate the error from the output neurons and compute the weight modifications for the neurons in the hidden layers [19].

Some neural networks are trained in an unsupervised mode, where only the input patterns are provided during training and the networks learn automatically to cluster them in groups with similar features. The attitudes of researchers towards neural networks have experienced an evolution since their inception in the early 1940s. According to Leon Cooper, these attitudes progressed from skepticism to what we have at present: general realistic acceptance of neural networks as the preferred -- most efficient, most economic- solution to certain classes of problems" [20].

Neural networks are composed solely of two elements -- processing elements and interconnections. The processing elements are called neurons and the connections are termed synapses. A processing element generally has many inputs and a single output as shown in figure 1.3 [21]. The neural network performance is governed by the architecture of the processing element's interconnection, the transfer functions for the processing elements, and the learning law. There are two popular models of neural networks -- the feed-forward model and the feedback model [22].

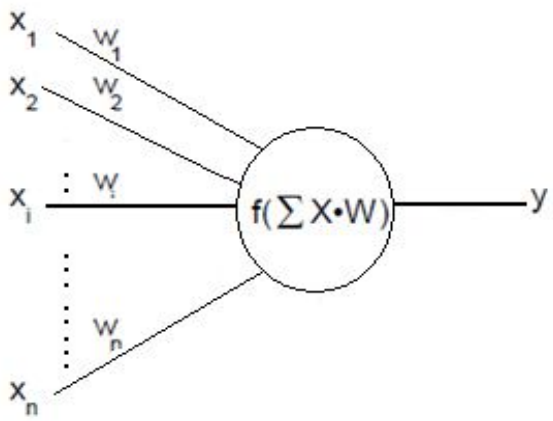


Fig 1.3 An artificial neuron.

The feed-forward model, as illustrated in figure 1.4, is composed of layers where the output from each level is the input for the next level. Input is applied to the input layer. The signals go to the hidden layers and then out to the output layer. Its operation is similar to the operation of a combinational logic circuit. Feed-forward neural networks work well for "natural" problems such as pattern recognition [23].

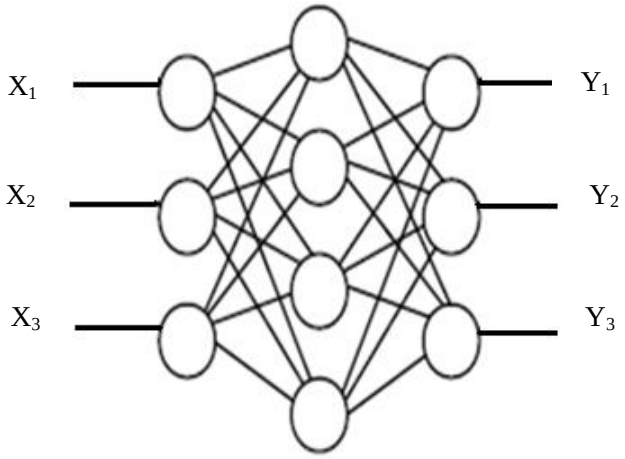


Fig 1.4 A feed-forward neural network.

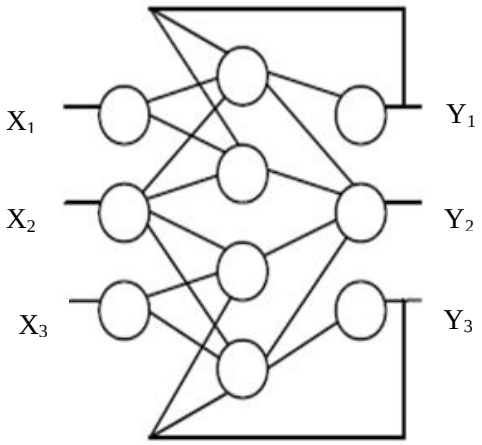


Fig 1.5 A feedback neural network.

Figure 1.5 illustrates the feedback model, which has connections between different levels, forward and backward. This is more like an asynchronous logic circuit in which the nodes evolve to a final state. Optimization problems (like the Traveling Salesman problem) are best implemented on feedback neural networks [24].

Each input to a neuron has an associated weight. If the sum of all the weighted inputs is above a certain threshold, the neuron's output is triggered. An input that tends to lower the weighted sum (probably because of a negative weight) is inhibitory. Excitatory inputs increase the weighted sum. It is useful to consider the inputs, (x_1, x_2,…., x_N) and the weights (w_1, w_2,….., w_N) to be vectors X and W [24]. The output is then the dot product of the two vectors. This can give us a visual image of what goes on in a neural network. Consider the weight vector to be pointing in a direction in space. An X vector will produce a large dot product if that X vector points in the same direction as the W vector. The system in this way leads to the development of a neural network of grandmother cells.

In a neural network of grandmother cells, each cell has its weight vector set to a pattern in which that cell is responsible for identifying. For *n* patterns, *n* grandmother cells are needed. Unknown input is entered into all the grandmother cells. The one that has the highest value on the output has weights closest to the input. This is a very simple neural network. There is no learning involved [25].

Most neural network applications involve training. Training a neural network to provide the desired outputs can be either *supervised* or *graded* training. Supervised training provides the input and the desired output to the network. Graded training provides the training input and a grade telling the network how close its output is to the desired output. The weights of the neuron inputs are adjusted during the learning process according to a *learning law*. An example of supervised learning is the Delta rule or Least Mean Squared training law, developed by Bernard Widrow and Ted Hoff [26].

To implement the LMS training law, the processing elements must be modified to provide the ability to compare its own output to the desired output. The input to the neuron that supplies the desired output is called the mentor input. For LMS training the weights are modified according to equation (4) [27].

$$W_{new} = W_{old} + \beta \frac{X}{|X^2|}(y_{desired} - y_{actual}) \qquad (4)$$

Where W is the weight vector for a processing element

X is the input vector

β *is* a programmer defined constant between 0 and 1

y are the desired and actual outputs (scalar)

The value of β determines how fast the weighting matrices converge to the point of the minimum least square error. The larger β, the faster the learning curve should reach its final destination. If the training data is noisy, however, a large learning constant may prolong the training time. The path followed to the least squared error point is the negative gradient of the error hyper parabola. Therefore, the learning should always take the most efficient route to minimum error [28].

Once trained the network must be tested. If the network produces unacceptable results, there are several options. First, try more training. If that does not work, reevaluate the learning law and any programmed constants. Make sure that the training data is similar to the test data. Examine the architecture of the network. Are more nodes required? More layers? Is the coding scheme used for the input information adequate?

By far the most popular training method is supervised back propagation. A back propagation neural network consists of at least three layers. The input layer accepts the input from the outside world. Therefore, the number of nodes in the input layer is equal to the number of inputs. The output layer produces the result and must also have an appropriate number of nodes. The middle layers are sometimes called hidden layers [29].

Although there is no way to determine what the best number of hidden nodes is, there are some general rules of thumb. If there are too many neurons in the hidden layer, the network will have the tendency to memorize the input patterns rather than generalize the input into features. If on the other hand, the middle layer contains too few neurons, the accuracy of recall will decrease and the number of iterations required for training will increase significantly [30].

Before training begins, all the neuron weights are set to random numbers. Then, the inputs are fed to the neural network, a summation is carried out in each layer, and the results of each layer are passed as the input to the next layer. The summation is shown in equation (5) [31].

$$I = f(\Sigma_i(w_i * x_i)) \qquad (5)$$

Where i is the input index

w is the weight of that input

x is the input signal

f (x) is the activation function

The activation function f(x), determines the activity, or excitation level, generated in the neuron as a result of the magnitude of the input. "For a back propagation network, this function should be sigmoid; that is, it must be continuous, S-shaped, monotonically increasing, and asymptotically approaching fixed values as the input approaches plus or minus infinity" [32]. If the neural network is used as a classifier, we must assume +1 or 0 values above and below a certain threshold on the output layer. Equation (6) and figure 1.6 is an example of such a function.

$$f(x) = \frac{1}{1+e^{-x}} \quad (6)$$

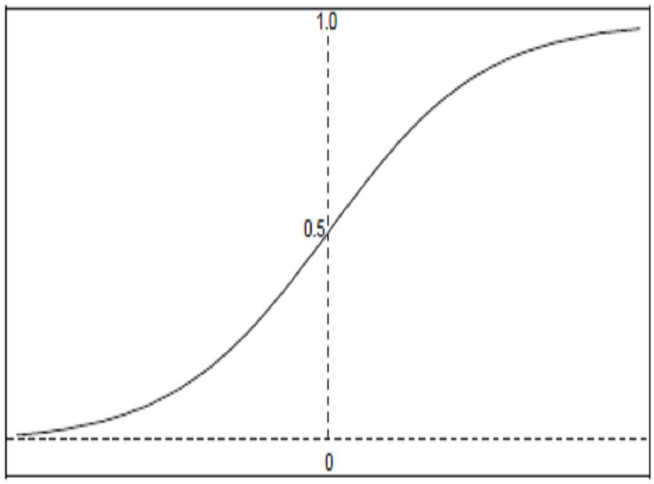


Fig 1.6 Graph of sample activation function.

The delta rule can be used to adjust the weights of a back propagation neural network. The delta rule is based on minimizing the error. It is easy to apply to the output layer because we know the input and the desired output. However, there is no way for us to know what the desired output is in middle layers. Middle layers require a determination of their error based on the layer receiving their outputs. The term back propagation refers to the way that the error calculation is back propagated from the output to the input. The error is calculated with equations (7) and (8) respectively [33].

$$e_i = f'(I) * \sum_j (w_{ij} * E_j) \quad (7)$$

$$f'(x) = f(x)\,(1-f(x)) \quad (8)$$

Where i refer to a particular node in the middle layer

j refers to the nodes in the following layer

e_i is the error of a middle layer node

E_j is the error in the nodes of the following layer

W_{ij} is the weight between node i and j

f (x) is the activation function

The derivative term contributes to stability ad helps prevent excessive blame being attached to the middle layer nodes. The training of a BPN therefore requires two passes through the network -- once forward, once backward. Because of this, BPNs are usually trained off-line to determine the weights. Once trained, the network can operate at much faster speeds.

The primary benefit of using a neural network is that the system "learns." Another benefit is that a problem does not have to be well understood before applying a neural network solution. The speed of computation (if many parallel processors act as the neurons) is a potential benefit as hardware implementation of neural networks improves. Finally, there is a better fault tolerance in parallel computer architecture than in the traditional von Neumann architecture [34].

Neural networks have their problems too. Neural networks are not good at precise mathematical computation. Therefore, neurocomputing will not replace algorithmic computing. The two have different, but complementary strengths. Artificial neural networks often require a huge number of iterations for training. They may get stuck in a local minimum of the error versus possible weights curve. Most researchers agree that the starting point vector is as important as the other aspects of the network. One way to help guard against the local minimum problem is to include a momentum term in the Delta rule. Equation (9) adds a momentum term to the delta rule [35].

$$W_{new} = W_{old} + \beta E \frac{X}{|X^2|} + \alpha (W_{new} - W_{old})_{prev} \qquad (9)$$

Where W is the weight vector for a processing element

X is the input vector

β is a programmer defined constant between 0 and 1

y are the desired and actual outputs (scalar)

α determines how much weight to put on momentum

One of the major problems with the use of fuzzy system is the difficulty of choice and design of membership functions to suit a given problem.

A systematic procedure for choosing the type of membership function and the ranges of variables in the universe of discourse is still not available. Tuning of fuzzy controller by trial and error is often necessary to get a satisfactory performance. However, the neural networks have the capability of identification of a system by which the characteristic features of a system can be extracted from the input output data. This learning capability of the neural network can be combined with the control capabilities of a fuzzy logic system resulting in a neuro-fuzzy inference system [36].

1.4NEURO-FUZZY SYSTEMS

Neuro-fuzzy approach combines two powerful computing disciplines: Adaptive neural networks and fuzzy set theory. Neural networks are well known for its ability to learn and adapt to unknown or changing environment to achieve better performance. Fuzzy set theory, on the other hand, can by its effectiveness in handling linguistic information, incorporate human knowledge, deal with imprecision and uncertainty, and clarify the relation between input and output variables. A neuro-fuzzy can be used to study both neural as well as fuzzy logic systems. A neural network can approximate a function, but it is impossible to interpret the result in terms of natural language. The fusion of neural networks and fuzzy logic in neuro fuzzy models provide learning as well as readability. Control engineers find this useful, because the models can be interpreted and supplemented by process operators [37].

Recently, the combination of neural networks and fuzzy logic has received attention. The idea is to lose the disadvantages of the two and gain the advantages of both. Neural networks bring into this union the ability to learn. Fuzzy logic brings into this union a model of the system based on membership functions and a rule base.

This field of study is still in its infancy. Universally accepted techniques and a general consensus on the direction of research have not yet been established. Most of the work done in this area is still associated with individual researchers and has not been adopted as standard strategy [38].

Determining the fuzzy membership functions from sample data using a neural network is the most obvious method of using the two together. The definition of the membership function has a huge impact on the system response. Often, the programmer must use trial and error to find acceptable values.

Assuming a certain shape and finding the beginning and endpoints for the fuzzy values in a fuzzy set is a neural network optimization problem [39]. Figure 1-7 is a diagram of such a system.

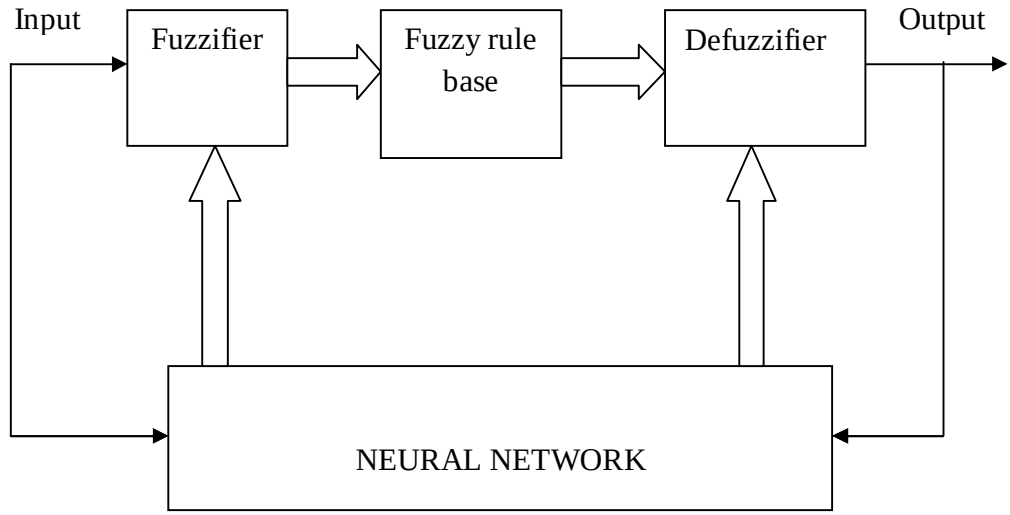


Fig 1.7 A fuzzy system whose membership functions is adjusted by a neural network.

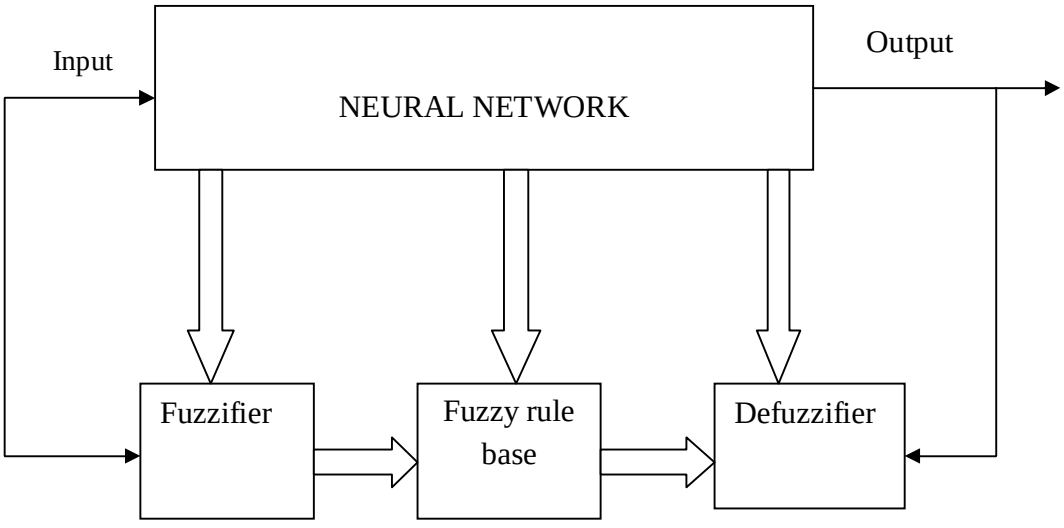


Fig 1.8 A fuzzy system defined by a neural network.

Figure 1.8 shows a more complex integration -- the use of neural networks to determine both the fuzzy membership functions and the rule base.

National Semiconductor scientists have developed a system that converts input and output data into nonlinear membership functions and a rule base. The nonlinearity of the membership functions is unique to membership functions derived by neural networks. They help minimize the number of rules [40].

Another approach is to incorporate fuzzy logic into the neurons of the neural networks. This approached developed because of the original neuron model proposed by McCulloch and Pitts [40]. The McCulloch-Pitts cell produced an all-or-none output. It was quickly realized that neurons with output in the range of [0, 1] produced much better results. The concept of a fuzzy neuron, however, has advanced beyond simply expanding the range of outputs on a crisp neuron. Some researchers have incorporated membership functions and rule bases into the individual neurons, as shown in figure 1.9.

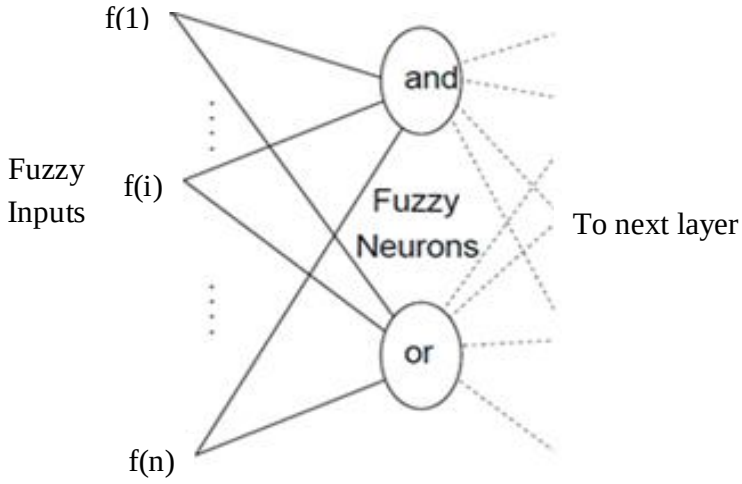


Fig 1.9 A neural network of fuzzy neuron

Finally, the idea of fuzzification of control variables into degrees of membership in fuzzy sets has been integrated into neural networks. If the inputs and outputs of a neural network are fuzzified and defuzzified, significant improvements in the training time, in the ability to generalize, and in the ability to find minimizing weights can be realized.

Also, the membership function definition gives the designer more control over the neural network inputs and outputs. It is this technique that is implemented in this thesis for the control of a robotic arm in the presence of an obstacle [41].

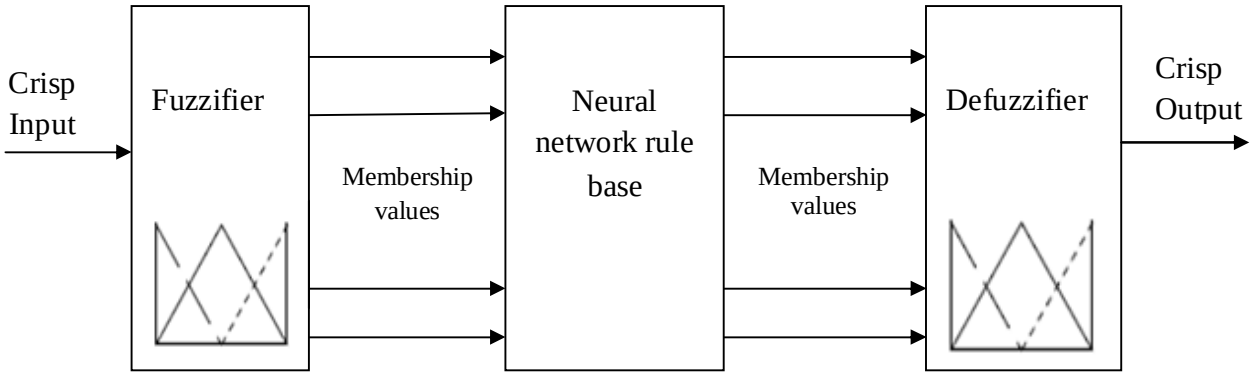


Fig 1.10 A fuzzy system with neural network rule base

1.5.1 Adaptive neuro-fuzzy inference system

Adaptive Neuro-Fuzzy Inference System is a feedforward adaptive neural network which implies a fuzzy inference system through its structure and neurons. Jang was one of the first to introduce ANFIS. He reported that the ANFIS architecture can be employed to model nonlinear functions, identify nonlinear components on-line in a control system, and predict a chaotic time series. It is a hybrid neuro-fuzzy technique that brings learning capabilities of neural networks to fuzzy inference systems. The learning algorithm tunes the membership functions of a Sugeno-type Fuzzy Inference System using the training input-output data [42]. ANFIS is the fuzzy-logic based paradigm that grasps the learning abilities of ANN to enhance the intelligent system's performance using a priori knowledge.

Using a given input/output data set, ANFIS constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone, or in combination with a least squares type of method.

This allows your fuzzy systems to learn from the data they are modelling [43].These techniques provide a method for the fuzzy modelling procedure to learn information about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input/output data. This learning method works similarly to that of neural networks.

The basic idea behind this neuro-adaptive learning technique is to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input output data. This learning works similar to that of neural networks. The knowledge base is developed from the training data set by training the system to a number of epochs to minimize the error rate [44].

The fuzzy inference process involves the following steps.

1.Fuzzify inputs. Determine the degree to which inputs belong to each of the appropriate fuzzy sets via membership functions.
2.Apply fuzzy operator. If the antecedent of a given rule has more than one part, apply relevant fuzzy operators to obtain one membership value that represents the match to the antecedent for that rule.
3.Apply implication method. Shape the consequent based on the antecedent.
4.Defuzzify. The input for the defuzzification process is a fuzzy set and the output is a single number. In Sugeno inference, the centroid is simply the weighted average of a few data point.

Sugeno model

For a first order Sugeno type of rule base with two inputs x, y and one output, the structure of ANFIS is shown in Figure 1.11.

The typical rule set can be expressed as,

Rule 1: If x_1 is A_1 AND x_2 is B_1, THEN $f_1 = p_1x+q_1y+r_1$

Rule 2: If x_1 is A_2 AND x_2 is B_2, THEN $f_2 = p_2x+q_2y+r_2$

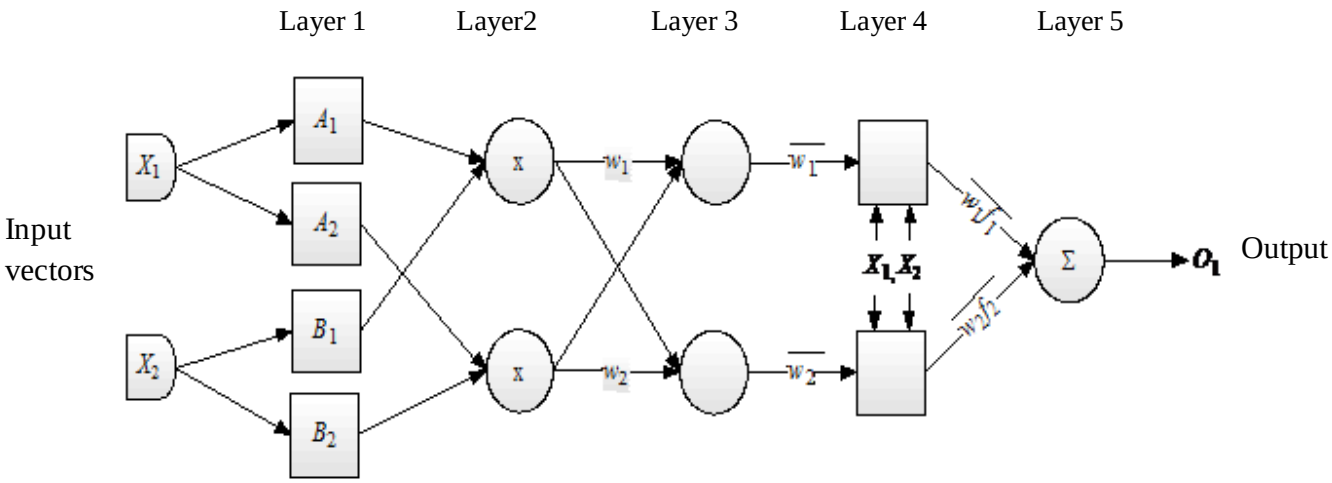


Figure 1.11: ANFIS architecture with two fuzzy if-then rules

Layer 1

$O_{l,i}$ is the output of the ith node of the layer l. Every node i in this layer is an adaptive node with a node function. The output of each node is:

$$O_{1,i} = \mu_{A_i}(x) \qquad for\ i = 1,2$$

$$O_{1,i} = \mu_{B_{i-2}}(y) \qquad for\ i = 3,4$$

x (or y) is the input node i and Ai (or Bi−2) is a linguistic label associated with this node.

The membership function given by:

$$\mu_A(x) = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b_i}}$$

where a_i, b_i, c_i are parameters and are the premise parameters.

Layer 2

Every node in this layer is a fixed node labelled Prod. The output is the product of all the incoming signals.

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), \quad i = 1,2$$

Each node represents the fire strength of the rule .Any other T-norm operator that perform the AND operator can be used.

Layer 3

Every node in this layer is a fixed node labelled Norm. The ith node calculates the ratio of the ith rule's firing strength to the sum of all rule's firing strengths.

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}$$

Outputs are called **normalized firing strengths**.

Layer 4

Every node i in this layer is an adaptive node with a node function:

$$O_{4,i} = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i)$$

w_i is the normalized firing strength from layer 3. The parameters in this layer (p_i, q_i, r_i) are to be determined and are referred to as the consequent parameters.

Layer 5

The single node in this layer is a fixed node labelled sum, which computes the overall output as the summation of all incoming signals:

$$O_{5,i} = \sum_i \overline{w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

The adjustment of modifiable parameters is a two step process. First, information is propagated forward in the network until Layer-4 where the parameters are identified by a least-squares estimator. Then the parameters in Layer-2 are modified using gradient descent. The only user specified information is the number of membership functions in the universe of discourse for each input and output as training information.

ANFIS uses back propagation learning to learn the parameters related to membership functions and least mean square estimation to determine the consequent parameters. Every step in the learning procedure includes two parts. The input patterns are propagated, and the optimal consequent parameters are estimated by an iterative least mean square procedure. The premise parameters are assumed fixed for the current cycle through the training set [45].

The pattern is propagated again, and in this epoch, back propagation is used to modify the premise parameters while the consequent parameters remain fixed.

To use ANFIS for classification problem, the designer needs to perform the following steps:

1. Design a Sugeno FIS appropriate for the classification problem.
2. Hands optimize the FIS, given actual input classification data.
3. Set up training and testing matrices. The training and testing matrices will be composed of inputs and the desired classification corresponding to those inputs.
4. Run the ANFIS algorithm on the training data.
5. Test the results using the testing data.

ANFIS has a network-type structure similar to that of a neural network which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map. ANFIS uses either back propagation or a combination of least squares estimation and back propagation for membership function parameter estimation.

CHAPTER 2
REVIEW OF LITERATURE

The use of neural networks, fuzzy logic and neuro-fuzzy technology in control has progressed in that order. Neural networks and fuzzy logic, used independently, have both demonstrated their value in control systems. More recently, neuro-fuzzy control has brought even more improvement to the quality of intelligent control. The following is a survey of the research done in the areas of intelligent control, followed by a survey of trajectory planning techniques.

2.1 FUZZY LOGIC

Li and Lau look at the application of fuzzy algorithms to servo system control [3]. They see the challenge of servo control lying in the accuracy and speed requirements. In comparing fuzzy control, PI control, and MRAC, they conclude that the fuzzy control is better than PI control and as good as MRAC. The authors note that their fuzzy controller could only offer an optimum solution to a narrow range of inputs.

Neffenger concludes that the value of fuzzy motor control is in the fact that a math model of the system is not needed [5]. A position control experiment is performed using a variable speed AC motor. Neffenger varied the mass of the object being controlled without the fuzzy system being affected. A change in system parameters is a drawback for model based control. The experiment demonstrates that a fuzzy controller can yield sufficient responses as parameters in the system change. Also, Neffenger points to the short design time required for fuzzy systems.

Kosko introduces the concept of FAM -- *fuzzy associative memory*. It is based on his view of fuzzy state space as a hyperbolic unit cube. FAMs map input "balls" to output "balls". The balls are clusters of data in the state space associated with certain conditions. Adaptive fuzzy associative memories, AFAM, change over time as new data is sampled and processed.

Kosko uses his FAM concept to build on the work of Nguyen and Widrow by designing a fuzzy truck backer-upper [20]. His fuzzy controller consistently chose a smooth path to its final destination. Kosko points to the fact that Nguyen and widrow's neural controller did not always

converge. He also points out that the fuzzy controller did not require 20,000 iterations for training. Training was achieved by encoding "common sense" FAM rules.

Kong and Kosko extend Kosko's own fuzzy controller for a truck and trailer by implementing an adaptive fuzzy controller [6]. In adaptive FAM, the rules are inferred from sample data using *differential competitive learning* (DCL) clustering. The rule inference for the AFAM was much faster than the training required for neural net weight adjustments. The researchers suggest that their fuzzy controller is robust, as demonstrated by the fact that up to 50% of the control rules could be removed without a significant deterioration in performance.

Pacini and Kosko compare the use of AFAM and Kalman-filter for real-time target tracking [7]. The comparison shows that AFAM could track as well as a Kalman filter, but not better than. Again, DCL is used to cluster sample data into rules. The rules that were generated from the sample data produced nearly exactly the same results as the human rule base.

Berenji, Chen, Lee, Murugesan, and Jang perform experiments with fuzzy control and a cart-pole balancing problem [9]. In their comparison of the fuzzy logic controller and a state-feedback controller, the fuzzy logic controller scored high marks in the areas of ease of implementation, robustness and percentage overshoot. The state feedback controller had better settling times and ease of modification. They felt that the largest limitation of the fuzzy method is in the calibration of the membership functions. They suggest research in the area of "automatic learning of approximate control rules."

Shao addresses one of the problems associated with fuzzy systems -- sometimes the rule base is difficult to define. He presents a method for developing fuzzy systems that adjust their rule base to conform to predetermined system requirements. The strength of his approach lies in its application to nonlinear systems and systems with a large time lag. Shao applies his technique to the control of both a boiler and a DC motor [11].

Qioa, Wang, Heng, and Shan design a Rule Self-regulating Fuzzy Controller (RSFC) [13]. They are interested in adjusting a fuzzy rule base on-line in real-time. In order to give the system the ability to quickly modify the fuzzy controller's rule base, the authors use a special type of rule base which is more easily modified. The example given is that of a fuzzy logic controller adjusting the proportional and integral gain constants in a simple feedback control system.

Bagchi and Hatwal use fuzzy logic to plan the path of an object moving in the presence of obstacles [14]. The researchers justify the use of fuzzy logic for two reasons -- ease of design and the ability of a fuzzy system to work with inaccurate sensors. First, appropriate fuzzy variables and values are chosen. The researchers develop a collision avoidance strategy and encoded that strategy into a simple rule base. A simulation is performed with five objects moving in a plane.

Jou and Wang present their work in the area of adaptive fuzzy logic systems [16]. Error back propagation is applied to a fuzzy logic system to adjust the rule base according to a pre-specified training set. The specific application addressed is that of backing up a truck to a dock. The researcher's system performed as well as the neural network controller of Nguyen and Widrow, but required far fewer training sessions. They note that in order to develop a rule base capable of generalization, training samples that are "homogeneously distributed throughout the entire state space" are necessary.

2.2 NEURAL NETWORKS

Guez, Eilbert, and Kam propose architecture for neural network control that can serve as an adaptive control system [18]. A comparison is made to traditional *model reference adaptive control* (MRAC). In their example of a robotic manipulator, the neural network approach shows significant improvement in performance over the MRAC. The neuro-controller is more general, making it possible to be trained for many problems. It shows the most improvement over MRAC as the system increased in order. It is also more stable and less sensitive to plant dynamics.

Psaltis, Sideris, and Yamamura address the problem of training a neuro controller over a large state space. They suggest training the neural net in two modes – generalized training and specialized training. The generalized training session extracts the major features. The specialized training works to define detailed boundaries between samples. This decreases the iterations necessary to train the neuro-controller and provides better response in important operating regions.

Nguyen and Widrow use two neural networks to control a truck and trailer as it backs up to a dock from any initial position [19]. The first neural network, called an *emulator,* learns to identify the system's dynamic characteristics.

The second provides the actual control. After 20,000 simulated backups, the controller is able to move the truck to within 3% of the desired position. The network also develops a control strategy, as demonstrated by the fact that initial moves sometimes increase the current error in order to prepare the truck and trailer for ultimate success.

Newton and Xu outline the work being done at Carnegie Mellon University in the area of neural network control of a robotic arm [21]. A seven degree-of-freedom space manipulator is controlled with a neural network resulting in 85% less trajectory error than recorded under PID control. The neural network is trained on-line and made use of a *moving average feedback*. The feedback provides the network with the ability to plan current control based on both the current input and the recent response of the network to past inputs.

Arai, Rong, and Fukuda use neural networks to control a three link robotic arm that is manipulating a flexible plate [23]. The problem is complicated by the fact that the control must take into account not only the trajectory planning, but also the vibration of the flexible material being held. Four neural networks are used to learn the error caused by the vibrations of the flexible plate. The networks have 4, 10, and 1 nodes in the input, hidden and output layers, respectively. After training, the model of the system's error, contained in the neural networks, is used to improve control.

Liu and Asada describe a neural network control of a deburring robot [25]. Initially, a neural network is trained off-line to data obtained by recording a human's control of the robot. This training data teaches general control strategy and task planning. The final control system also makes neural network adjustments on-line based on data acquired during operation. This training fine tunes precise controlled motion. Because of this second learning capability, the neural network controller was able to exceed the performance of the human controller.

T. C. Hsia and Z. Mao propose a scheme for obstacle avoidance of redundant manipulators by neural network [26]. Their scheme, called *q-system*, translates a desired Cartesian coordinate position, X_d, into four joint angles, q. The training samples are taken from known solutions given by the forward kinematics solution. In this manner, a neural network was trained for the inverse kinematics solution of a four-link arm. The system is improved by performing a coordinate transformation of q, which localizes the search for an inverse solution.

Cooperstock and Milios incorporate neural network control into a vision guided robotic arm [27]. Neural networks take the place of complex numerical solution techniques for both the solution of the inverse kinematics of the arm and for the solution of the perspective projection of the stereo vision system. Cooperstock and Milios's controller is comprised of five neural networks. The networks specialize in the operations of approaching, centering, paralleling, reaching, and adjusting. The researchers describe their control system as being competitive with traditional systems.

Yegerlehner and Meckl utilize the learning capabilities of neural networks to adapt the control of a robot experiencing large changes in payload weight [28]. One neural network is trained for the inverse kinematics, while another is trained to estimate the mass of the payload. The authors reveal that the artificial neural network did not model the inverse kinematics as well as a comparison model of least squares fitting parameters. They point to the fact that the network inputs were only the joint angles and their two time derivatives. They suggest that a "richer set of inputs" would be necessary for improving performance.

2.3 NEURO-FUZZY TECHNOLOGY

Nauck, Klawonn, and Kruse research the fusing of neural networks and fuzzy systems in an attempt to overcome the disadvantages expressed by other researchers using only one of the technologies [29]. Their inverted pendulum application is similar to Berenji's, et al. The researchers' "neural network oriented fuzzy control" system adjusts its fuzzy set definitions. The application of this system to an inverted pendulum was able to balance the pendulum and reduce the rules required from 512 to fewer than 40. Not only did learning take place, as in a neural network, but the resulting system was defined in the linguistic variables.

Jang designs a self-learning fuzzy controller based on temporal back propagation [31]. The current state of the system is compared to the desired state and the error is back propagated through the system to adjust individual fuzzy parameters. Tests on an inverted pendulum show that significant adjustments were made on membership function definitions. The trained system exhibited robustness and fault tolerance.

Archer and Wang propose a method for using neural networks to define membership functions [32]. Their algorithm incorporates what they term as a *Fuzzy Membership Model*. Neural networks are used to learn how to classify patterns that fall near regions of uncertainty in pattern space. In their example, one neural network is used to learn the sharp boundary between two classes, while two other neural networks are trained to determine a pattern's fuzzy membership in a particular class.

Berenji and Khedkar also use neural networks to define fuzzy membership functions [33]. They emphasize the importance of selecting the correct granularity for describing the values of each linguistic variable. Berenji and Khedkar implement a *generalized approximate-reasoning-based intelligent control* (GARIC). One neural network is used to evaluate the performance of the fuzzy system. One neural network is used to adjust the membership function based on the evaluation network's output. Simulations for the classical cart-pole balancing problem are performed.

Blanco and Delgado present their work in the area of neuro-fuzzy techniques [34]. They suggest that a neural network's strength lies in its ability to approximate a function from sample data. The parallel in fuzzy system applications would be the need to infer an output from a predefined rule base. Blanco and Delgado suggest training neural networks to the knowledge contained in a fuzzy rule base. The neural networks can then be used in the place of a rule base.

Keller, Yager and Tahani justify this approach by pointing to the fact that in fuzzy logic systems "as the number of antecedents' clauses increases, the storage and the computation in the inference process grow exponentially". Their research indicates that for complex systems, a fuzzy rule base is more efficiently stored in a neural network. Also, due to the parallel nature of a neural network, the inference procedure would require less time in a parallel implementation.

Pedrycz takes a similar approach to combining neural networks and fuzzy logic [36]. His approach, however, does not involve training a generic neural network to emulate a rule base. He introduces two new classes of fuzzy neurons. The aggregative neurons realize AND, OR, and mixed AND/OR operations. The referential neurons realize binary relations of matching, difference, inclusion and dominance. The fuzzy rule base is "encoded" into a neural network by appropriate architecture design and weight selection using these two types of fuzzy neurons.

Pal and Mitra adapt neural networks to include fuzzy inputs and outputs [37]. In their combination of the two technologies, each input is first fuzzified according to the input membership functions. The desired outputs are also fuzzified in order to perform the back propagation during training. The resulting network produces fuzzy membership values for a given input. Pal and Mitra compared their neuro-fuzzy system of classifying vowel sounds to conventional speech recognition systems.

Rahman outlines his use of neuro-fuzzy technology in a practical, home appliance problem [38]. The problem is that a toaster's darkness setting does not take into account the initial temperature of the toaster. Rahman equips a toaster with a temperature sensor and microcontroller. The temperature is read by an eight bit microcontroller that controls the toaster coils. Data was collected off-line and used by a neural network to derive 52 rules and 3 membership functions. After training, the toaster produced toast with almost identical degrees of darkness regardless of the initial temperature of the toaster.

Altrock and Krause have researched neuro-fuzzy technology in embedded automotive control [39]. A 600 rule fuzzy control system was implemented in a 20 inch model and in a full size sedan to provide an anti-skid steering system. Due to the large number of rules and the real-time demands of such a system, the authors used the Gamma aggregational operator and FAM inference. Through a serial link to a PC, a neural network was able to optimize parameters during operation.

Lea, Jani, and Berenji explore the use of neuro-fuzzy control for space shuttle rendezvous and docking [40]. An architecture composed of two networks is used. The two networks are referred to as the *evaluation network* and the *action network*. The evaluation network is a multilayer back propagation neural network dedicated to the "if" side of a fuzzy rule base. Similarly, the action network is dedicated to the "then" side of the rule base.

S. C. Lee and E. T. Lee introduce the fuzzy neuron. The fuzzy neuron incorporates fuzzy logic into the neurons of the neural networks. Their approach is based on the fact that the original neuron model proposed by McCulloch and Pitts produced an all-or-none output [41].

It was quickly realized that neurons with output in the range of [0, 1] produced much better results. The authors say that their fuzzy neurons are best used in areas of "soft sciences, such as in prediction making, pattern recognition, and decision making processes.

Pedrycz successfully trained a fuzzy-neuron neural network to control a system with two state variables and two control variables [42]. His approach attaches linguistic terms in the place of numeric weights between the individual processing elements. This makes it easier to interpret what goes on in the network. Pedrycz also points out that fuzzy-neuron controllers make it easier to understand and adjust the trade-off between information granularity and learning capabilities.

Keller and Hunt address the problem of a neural network classifier's inability to terminate training when training samples from two classes are nonseparable [44]. They suggest that training samples that are nonseparable are probably atypical of their respective class. Fuzzy techniques can be incorporated into the perceptron model to put less weight on these atypical samples during training. The researchers respond to the rhetorical question, "Why not just identify the atypical vectors and ignore them?" By making a sample vector's influence during training dependent on how closely it represents its class, the boundaries between the classes can be better characterized than if a vectors are ignored.

2.4 TRAJECTORY PLANNING

W. J. Chung, W. K. Chung, and Y. Youm introduce a method for dealing with the problems of inverse kinematics in robotic manipulators with redundant degrees of freedom [26]. They propose the use of *virtual links* and *displacement distribution*. The virtual link scheme divides a redundant manipulator into a series of two or three link manipulators. The displacement distribution assists in planning movements by assigning responsibility for the end-effector's position to each joint. The researcher's simulate their algorithm for planar arms with as many as 10 degrees of freedom.

Paredis and Khosla also address the problem of solving inverse kinematics equations in redundant manipulators [27]. They suggest, however, a numerical approach for obtaining a solution. A system of inequalities and an objective equation describing the robotic arm's task is determined.

Then the objective equation is optimized by the simulated annealing technique to iteratively solve for the arm's joint angles. The researcher's numerical solution for a two DOF serial link manipulator agrees with the analytical solution.

Lumelsky applies his dynamic path planning (DPP) approach to various types of two joint robotic arms operating in the presence of obstacles [28]. His non-heuristic approach is based on transforming the arm and obstacle arrangement into *image space*. The DPP image space is made up of regions of virtual boundaries caused by the obstacles in the environment. A path is planned in image space and then transformed into joint space. The DPP algorithm possesses a significant requirement for computation.

Wang and Hamam suggest trajectory planning and collision avoidance by representing the arm and objects as a set of convex polyhedra [29]. A *doubly connected edge table* (DCET) is used to determine the interaction between objects in three dimensional space. A path is iteratively calculated based on the geometric distances from each link of a robotic arm to the obstacle. Their FORTRAN simulation on a VAX II/GPX took "a few minutes" to find an acceptable path. The researchers suggest the use of heuristics to accelerate path planning.

Galicki examines the use of global optimization of potential functions to plan collision-free trajectories [30]. His specific application is that of a three link robotic arm operating in 2d-space in the presence of a single obstacle. Because Galicki's algorithm involves the optimization of both geometric trajectory path and the determination of the time parameterization of the trajectory, an iterative solution of nonlinear equations is necessary. Galicki summarizes, "Numerical solution of the problem ... is a complex and time-consuming task."

Lin and Fu take a divide-and-conquer approach to the collision avoidance and motion planning problem [31]. Their work deals with trajectory planning of an *n* degree of freedom manipulator where *n* is an arbitrary parameter of the system rather than a predefined, small constant. First, the space of operation is divided into subspaces and cells. Then, the problem of finding a collision free path is further decomposed into two separate problems. Because, the path planning is all done in configuration space, the researchers concede that there is not yet an efficient way to practically implement their algorithm.

CHAPTER 3
ROBOTIC ARM PROBLEM

3.1 INTRODUCTION

For the given the initial and final positions of a robot end effector, the problem of finding an optimal path to be followed is considered in this paper. The problem consists of locating a specific path that requires the least amount of energy amongst several possible paths. It is evident that the end effector, in moving between any two specified end points, can follow a variety of paths. All such paths require different amounts of energy depending upon the distance covered the velocity and the acceleration achieved, and the payload carried.

3.1 PROBLEM FORMULATION

For robots to become effectively used in a wide range of application, they must gain the ability to work in unpredictable and changing environments. Robots used in space exploration and construction will have to sense their environment and carry out their tasks regardless of the presence of objects in the work area. This thesis addresses the problem of planning the trajectory of a three-link robotic arm in the presence of an obstacle [32].

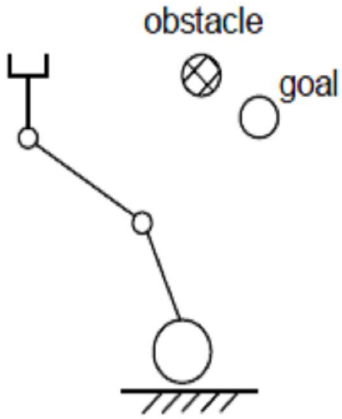


Fig 3.1 A diagram of robotic arm in presence of an obstacle

For the purpose of designing the control system, the position of the obstacle and goal can be assumed to be available from sensory feedback. A vision system, for example, could be used to generate Cartesian coordinates of the objects. Also, the joint angle values are assumed to be available from position feedback sensors in the arm. The system configuration that inspired this problem is shown in figure 3.2 [47].

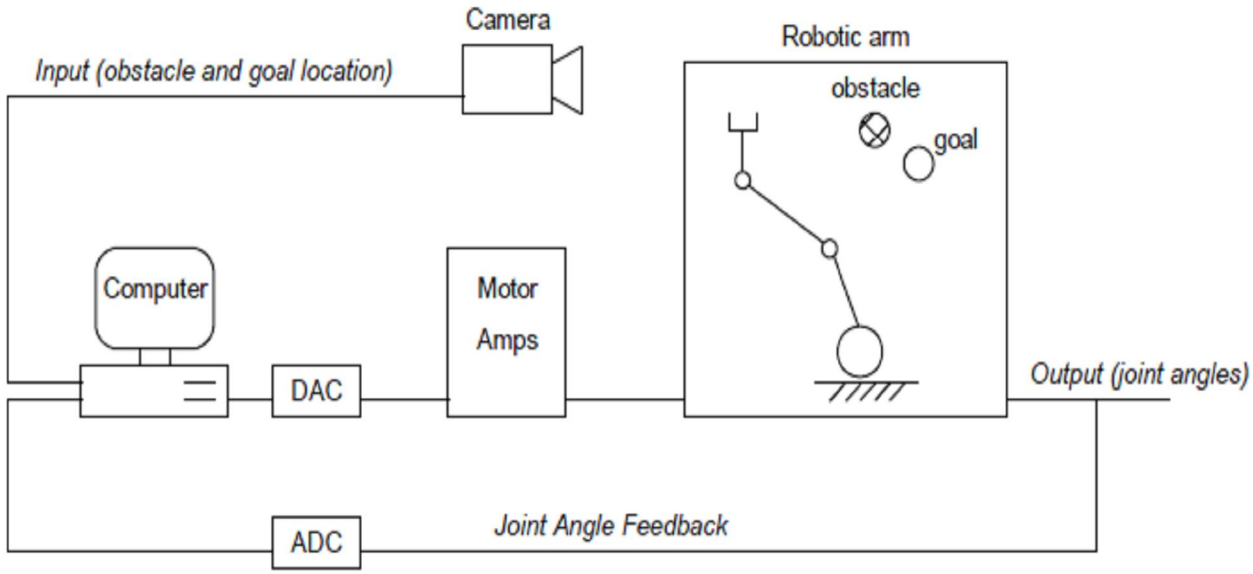


Fig 3.2 A sample configuration for the problem application

The arm should operate in two dimensions in an environment containing a randomly placed obstacle. The starting position and desired position of the arm are arbitrary, as well. The arm will be modeled as a three-link planar manipulator. The controller will determine a series of joint angles, Q(t), that move the end effector from a given starting position (x_S, y_S) to a desired final position (x_g, y_g) without colliding with the obstacle at (x_o, y_o). For a manipulator with n degree of freedom, at any instant of time joint variables is denoted by $\theta_i = \theta$ (t), i =1, 2, 3,..., n and position variables $x_j = x(t)$, j = 1, 2, 3,...,m [48].

The relations between the end-effector position x (t) and joint angle θ(t) can be represented by forward kinematic equation,

$$x(t) = f(\theta(t)) \quad (10)$$

where f is a nonlinear, continuous and differentiable function.

On the other hand, with the given desired end effector position, the problem of finding the values of the joint variables is inverse kinematics, which can be solved by,

$$\theta(t) = f'(x(t))$$

The robotic arm's end-effector position in Cartesian space can be directly related to its link lengths and joint angles by the following equations and figure 3.3 [49].

$$x_e = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2) + l_3\cos(\theta_1+\theta_2+\theta_3) \quad (11)$$

$$y_e = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2) + l_3\sin(\theta_1+\theta_2+\theta_3) \quad (12)$$

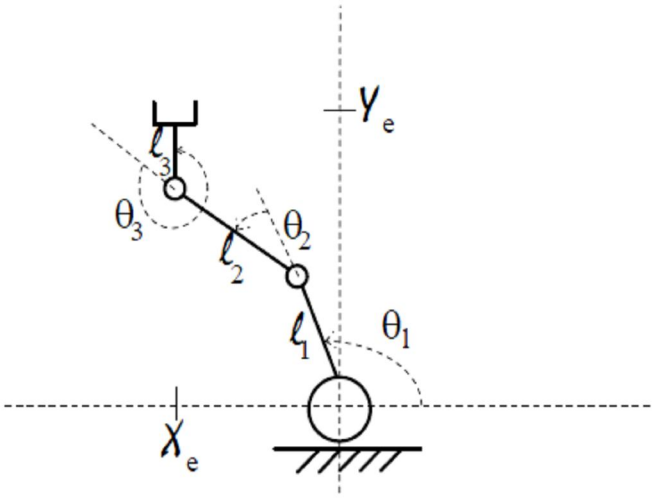


Fig 3.3 Robotic arm definition variables

Notice that because the model of the arm contains three joints and is operating in a two-dimensional plane, redundancy exists in possible configurations of the joint angles for a given Cartesian coordinates location for the end-effector.

In one respect, this redundancy complicates the problem because there is no unique mapping available to convert a coordinate in Cartesian space to a specific point in joint space. In fact,

given a desired position of the end-effector, there are an infinite number of solutions for possible joint angles [49].

On the other hand, this gives the designer the freedom to select the arm movements based on power efficiency, or a straight line trajectory, or even on the observed behavior of an expert's control of an arm. The characteristics of importance for this problem are the link lengths and the physical constraints of the joints [50].

Table 3.1 lists these values.

Physical characteristic	Minimum angle	Normalized length	Maximum angle
Link length l_1	-	1	-
Link length l_2	-	0.824	-
Link length l_3	-	0.737	-
Joint constraints θ_1	0^o	-	180^o
Joint constraints θ_2	-90^o	-	90^o
Joint constraints θ_3	-25^o	-	25^o

Table 3.1 The physical characteristics of a remote robotic arm

The goal must be assumed to lie within the possible reach of the end-effector and the obstacle must be assumed to lie outside the path of link, l_1. Using the direct kinematics equations, the constraints in table 3.1, and one final constraint that $y_e \geq 0$, all possible locations for the end-effector can be determined and are illustrated in figure 3.4.

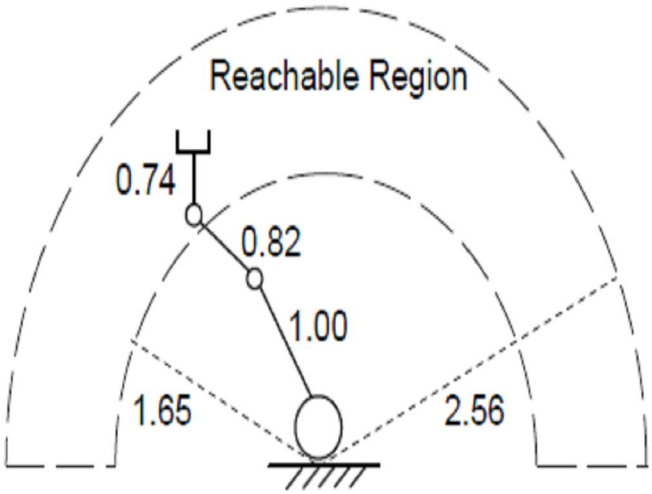


Fig 3.4 Normalized dimensions for remote arm and reachable region

The problem therefore can be summarized as follows:

Given a robotic arm with:

- The current joint angles, Q(0),
- An obstacle located at a known position, (x_o, y_o),
- A desired position, or goal, (x_g, y_g);

Determine a trajectory, θ (t), such that:

- The end-effector reaches the goal,
- The arm does not touch the obstacle,
- The calculations can be performed in real-time with current hardware.

CHAPTER-4
NEURO FUZZY SOLUTION

4.1 INTRODUCTION

The problem addressed in this thesis is actually a series of sub-problems. Planning the trajectory of a robotic arm has been itself a topic of many research projects. Planning the trajectory of a redundant robotic arm adds a new dimension to the problem -- more degrees of freedom, more nonlinear equations. Place an obstacle in the work area and the problem becomes a challenge of optimizing differential equations, iterative numerical solutions, and "guess and shoot" methods [51].

The largest disadvantage of previous solutions is the computer resources necessary to solve a system of equations or optimize a performance index. Researchers acknowledge that "minutes" are necessary to plan one trajectory on some of the world's fastest computers. Simply determining the equations to be solved is a formidable task. What is needed is an approach that is easy to apply and will work in real-time.

This problem is well addressed by neuro-fuzzy techniques because a solution is not easily found by analytical or numerical techniques. While an analytical technique is difficult, moving an arm in the presence of an obstacle can be instinctively performed by a child. Neuro-fuzzy systems excel in using sample data to determine an input-output relationship. Neuro-fuzzy technology is the fusing of both neural networks and fuzzy logic. Neural networks bring to this solution the ability to learn while fuzzy logic is based on mimicking an expert's thinking. In addition, as hardware technology progresses, more and more value will be placed on solutions that can utilize parallel processing, like neural networks [52].

The field of neuro-fuzzy technology has gone in many directions. The neuro-fuzzy technique implemented in this thesis is that of replacing the fuzzy logic rule base of a traditional fuzzy logic system with a multilayer back propagation neural network. Figure 4.1 illustrates the type of neuro-fuzzy system used in this problem.

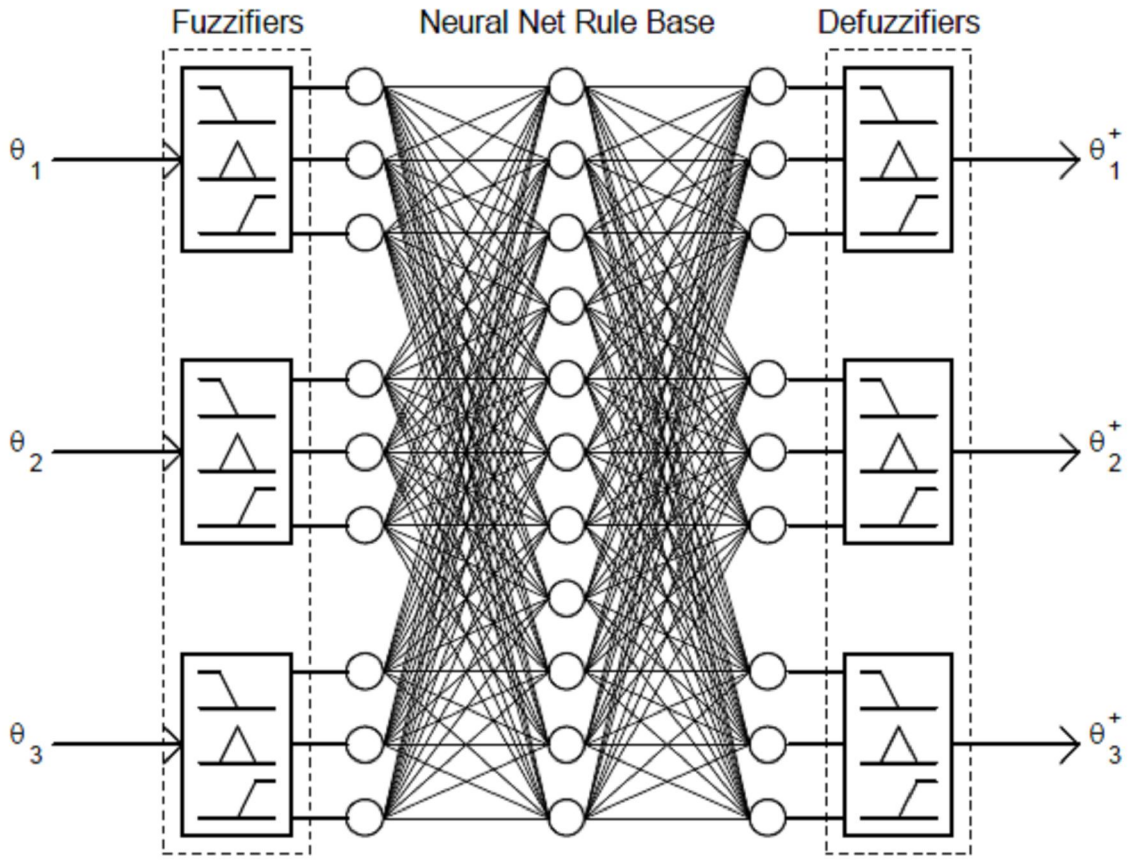


Fig 4.1 A sample Neuro-Fuzzy system

This type of system is beneficial for several reasons. While it is true that a child is able to move an arm around an obstacle to reach a desired goal, that ability is intuitive. Putting the instructions for performing such a task into a neat, fuzzy logic, IF/THEN rule base is not easy. Thus, there is a necessity for the neural network to learn the rules. The fuzzifier and defuzzifiers necessary for any fuzzy system provide an interface between an expert's control of a simulated arm and the neural network.

The problem was addressed in the following steps, which will serve as an outline for the remainder of chapter. The state representation of the system, which would serve as the controller's input, was considered. A simulator of the Remote robotic arm was written. The simulator included a graphical display of a three-link planar manipulator, an obstacle and a goal.

The simulator was controllable from the keyboard and recorded all movements of the arm under "expert" control. A generic neuro-fuzzy system was programmed. The recorded data was used as the training set for the neuro-fuzzy system. The simulator and neuro-fuzzy controller were combined to simulate the arm under neuro-fuzzy control [53].

4.2 STATE REPRESENTATION

One of the primary considerations in the design of the neuro-fuzzy controller is the representation of the state of the system. The ability of the final neuro-fuzzy controller to generalize a solution from training data will depend largely on the data representation scheme in the system. The question about whether to input joint angles, Cartesian coordinates, relative distances, or a combination of these into the neuro-fuzzy network will also determine whether or not the system can converge during training [54].

The following considerations guided the selection of state representation and are based on the fact that situations important to the trajectory chosen should be an important part of the state representation scheme.

- The state representation must excite the system if the end-effector is not at the correct position. In other words, a steady state error of the end-effector position must produce an error reducing response. Therefore, the distance from the end-effector to the desired location goal should be an input to the controller.
- The state representation should be similar for similar relative arrangements. For example, the two arrangements of arm, obstacle, and goal would have similar acceptable trajectories and should therefore have similar state representation. A scheme based solely on Cartesian coordinates may not allow a neuro-fuzzy system to generalize similar strategies for the two arrangements below [55].

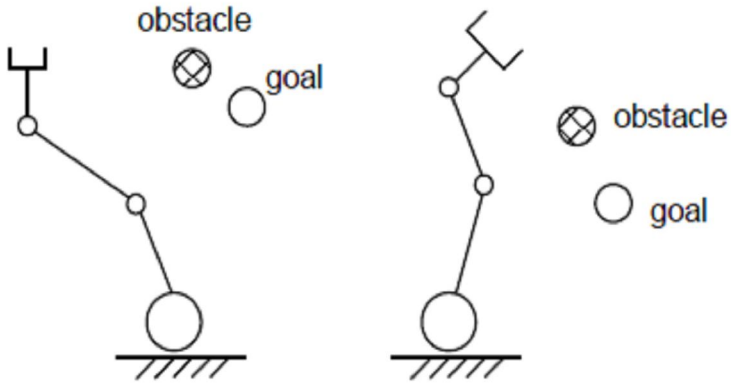


Fig 4.2 The relative position of the objects is same

- A scheme based on joint space might cause a neuro-fuzzy system to interpolate similar strategy for completely different situations. The representation should be different for the cases in figure 4.3.

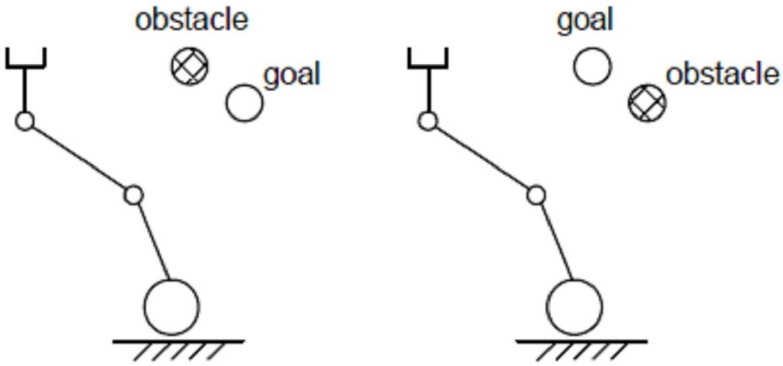


Fig 4.3 The obstacle and goal switch places

- The representation should take into account the physical constraints so that interpolation into the table or past a joint's capabilities does not occur.
- The output of the system will be the three small changes in angle, q*i* that should be made to each of the three joints to move the end-effector toward the goal without hitting the obstacle.

These guidelines are not sufficiently met by using only joint space or only Cartesian coordinates. On the basis of these guidelines, the following inputs and outputs were chosen to represent the system and are illustrated in figure 4.4 [56].

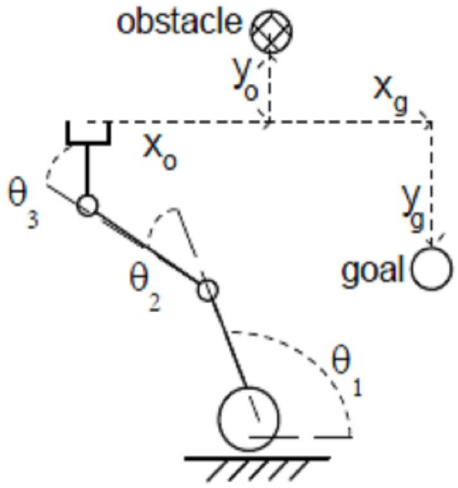


Fig 4.4 The state representation for the obstacle problem

INPUTS TO THE NEURO-FUZZY CONTROLLERS:

θ_1, joint angle of link1 to the base

θ_2, joint angle of link2 from link1's axis

θ_3, joint angle of link3 from link2's axis

X_o, horizontal distance from the end-effector to the obstacle

Y_o, vertical distance from the end-effector to the obstacle

X_g, horizontal distance from the end-effector to the goal

Y_g, vertical distance from the end-effector to the goal

OUTPUTS OF THE NEURO-FUZZY CONTROLLERS:

$\Delta\theta_1$, a small change in the angle between link1 and the base

$\Delta\theta_2$, a small change in the angle between link2 and the link1 axis

$\Delta\theta_3$, a small change in the angle between link3 and the link2 axis

4.3 ROBOTIC ARM SIMULATION

In order to obtain the training data and test the robotic arm controller, it was necessary to write a simulation program of the robotic arm. The simulator was written in C on a PC, as was all the code for this thesis project. It contains a complete listing of the programming done during this project. Included in this chapter are the data structure definitions. The data structure definitions are the foundation of well-organized programming code and they provide insight into the nature of the rest of the program [57].

The primary data structure in the simulator was the *Link* structure. The *Link* structure is defined as shown in figure 4.5. A robotic arm is programmed as a linked list of *Link*s. The obstacle and goal are represented on the screen as circles. The simulated arm is controlled from the keyboard. The program records its data in two data files. One data file contained the state vectors which acted as the inputs to the neuro-fuzzy system during training. The other data file stored the "expert's" movements and served as the desired output of the training system. All the data was normalized to be in the range of (0, 1) [58].

```
struct Link
{                                /* A link extending from a joint */
  double len;                    /* Length of link in pixels */
  double theta;                  /* Angle relative to previous link */
  double max_theta;              /* Max theta caused by joint constraint */
  double min_theta;              /* Min theta caused by joint constraint */
  double phi;                    /* Angle relative to horizontal */
  struct Link *next;             /* Pointer to next link in linked list */
  };
```

Fig 4.5 The robotic arm link data structure

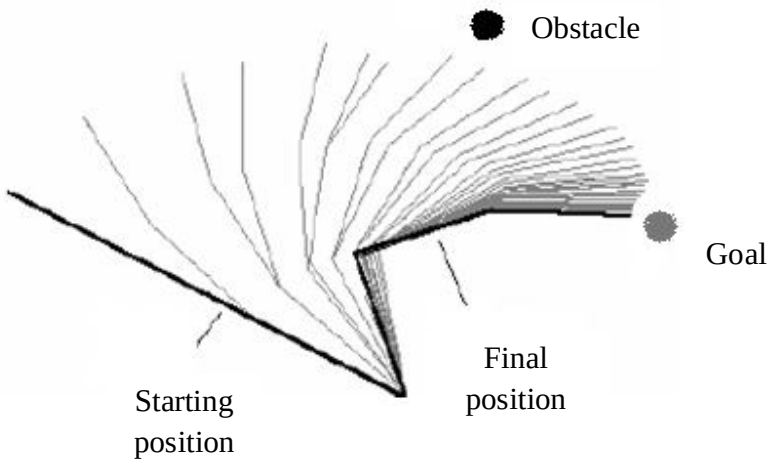


Fig 4.6 A sample training trajectory obtained from the simulator

Figure 4.6 shows a screen snap-shot of the robotic arm simulator and a sample trajectory while under "expert" control. The training samples were obtained by cycling through a series of goal and obstacle positions. The training samples included two starting positions of the arm -- ($\theta_1 = 0$; $\theta_2 = 0$; $\theta_3 = 0$) and ($\theta_1 = 180$; $\theta_2 = 0$; $\theta_3 = 0$). Goal and obstacle positions were taken from the entire reachable region. As the training samples were recorded, each movement created a new input to the neuro-fuzzy system. Therefore, selecting starting positions of the arm at the two extremes of possible movements generated training samples that extended over the entire state space. Obtaining training samples over an entire range of possible state space is important for neuro-fuzzy design.

4.4 NEURO-FUZZY PROGRAMMING

Once the training samples were normalized, and in ASCII data files, the code to run the neuro-fuzzy system was written. The important data structures in the neuro-fuzzy code included *Node*, *Layer*, and *Net*. The definition of these data structures are shown in figure 4.7.The neuro-fuzzy engine is completely void of any constants or programming related to a robotic arm. Consequently, it can be used for any input/output data. The number of inputs, the number of nodes and even the fuzzy membership functions are defined for individual systems in an ASCII text file. This text file is termed the FNN data file. The weights for a particular training set are stored in a text file, which is also specified in the FNN data file [59].

```
struct Node
{                                   /* a neural network node */
  int numweights;                   /* number of inputs to the node */
  double *win;                      /* current weights array */
  double *winlast;                  /* last weights array */
  double sum;                       /* sum of inputs*weights */
  double output;                    /* activation output */
  double error;                     /* BPN error at this node */
  };
struct Layer
{                                   /* an in, out, or hid layer */
  int numnodes;                     /* number of nodes in this layer */
  struct Node *nodes;               /* ptr to next node in this layer */
  };
struct Net
 {                                  /* a neural network */
  int numlayers;                    /* the number of layers in this NN */
  struct Layer *layers;             /* ptr to next layer in this NN */
  };
```

Fig 4.7 The neuro-Fuzzy data structure

A sample FNN data file is shown in figure 4.8.

```
remThe number of inputs and outputs on the neuro-fuzzy network
inputs=7
outputs=3
rem The number of hidden layers and nodes
hidden=1
hidnodes=65
```

```
rem The activation slope constant and training parameters
alpha=1
beta=.5
gamma=.1
epochs=300
maxerror=.01
rem The data files
input_file=input.dat
samples_file=input.dat
desired_file=output.dat
weights_file=weights.dat
error_file=error.dat
rem Number of fuzzy values for each input and output
input_values=7 7 7 3 3 3 3
output_values=3 3 3
rem The fuzzy membership function data
input_max=
0.0 0.166 0.333 0.5 0.666 0.833 1.0
0.0 0.166 0.333 0.5 0.666 0.833 1.0
0.0 0.166 0.333 0.5 0.666 0.833 1.0
0.0 0.5 1.0
0.0 0.5 1.0
0.0 0.5 1.0
0.0 0.5 1.0
output_max=
0.0 0.5 1.0
0.0 0.5 1.0
0.0 0.5 1.0
```

Fig 4.8 A sample FNN data file

4.5 NEURO-FUZZY CONTROLLER

Once training samples are obtained and a neuro-fuzzy system is trained to this data, the controller can be tested on the robotic arm simulator. The tests are performed by placing the objects at random locations and assigning a random starting position for the arm. The controller simulation must quantify the performance of the proposed neuro-fuzzy controller. To do so, the original design specifications of the problem statement should be addressed [60].

The design specification in the following goals of the system:

Determine a trajectory, θ (t), such that:

- The end-effector reaches the goal,
- The arm does not touch the obstacle,
- The calculations can be performed in real-time with current hardware.

In order to judge the controllers based on these guidelines, the simulated neuro-fuzzy controller determines the percentage error in reaching the goal, determines whether or not a collision occurred, and performs all the necessary calculations so that a judgment can be made about the possibility of real-time application. The first and third criteria are straight-forward. A collision with the obstacle on any part of the arm can be detected from the joint angles and obstacle position [61].

Given a robotic arm, composed of a series of links, operating in the presence of an obstacle whose Cartesian coordinate position is known, the shortest distance between the obstacle and the arm is the minimum of the distance between the obstacle and each link of the arm. The distance, *d*, from a link to an obstacle is determined where all the variables are defined [49].

$$d_{12} = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2} \quad (12)$$

$$d_{10} = \sqrt{(x_1 - x_0)^2 - (y_1 - y_0)^2} \quad (13)$$

$$d_{02} = \sqrt{(x_0 - x_2)^2 - (y_0 - y_2)^2} \quad (14)$$

$$\theta_1 = arcos\left(\frac{d_{12}^2 + d_{10}^2 + d_{02}^2}{2 d_{12} d_{10}}\right) \quad (15)$$

$$\theta_2 = arcos\left(\frac{d_{02}^2 + d_{10}^2 + d_{12}^2}{2 d_{02} d_{10}}\right) \quad (16)$$

$$d = d_{10} \qquad for\ \left(\theta_1 > \frac{\pi}{2}\right),$$

$$d = d_{02} \qquad for\ \left(\theta_2 > \frac{\pi}{2}\right),$$

$$d = d_{10} \sin(\theta_1), \qquad otherwise$$

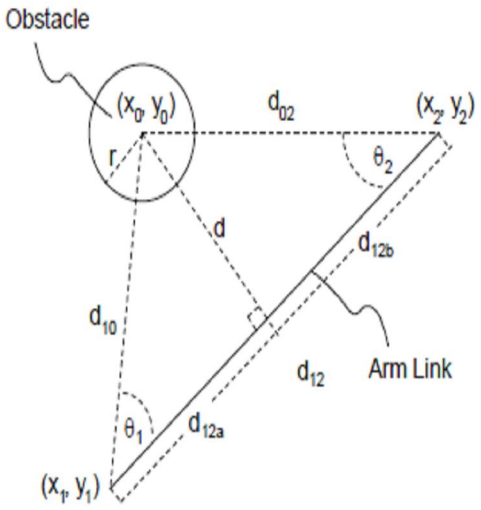


Fig 4.9 Determination of a collision between the obstacle and the link

If $d < r$, a collision has occurred.

The function that checks for a collision is shown in figure 4.10.

```
Determine if the arm is intersecting


int Collision(struct Link *links, struct Point *obstacle, double margin)
{
  double d, d12, d10, d02, d12a, d12b, t0, t1, t2;
  struct Link *l;
  struct complex p0, p1, p2;
  struct Point p;
  Pol2Rec(obstacle, &p0);
  p1.x = 0;
  p1.y = 0;
  for(l = links; l!=NULL; l = l->next)
{
    p.rho = l->len;
    p.phi = l->phi;
    Pol2Rec(&p, &p2);
    p2.x += p1.x;
    p2.y += p1.y;
    d12 = Distance(&p1, &p2);
    d10 = Distance(&p1, &p0);
    d02 = Distance(&p0, &p2);
    t1 = acos( (d12 * d12 + d10 * d10 - d02 * d02) / (2 * d12 * d10));
    t2 = acos( (d12 * d12 + d02 * d02 - d10 * d10) / (2 * d12 * d02));
    d = d10 * sin(t1);
    if(t1 > M_PI_2) d = d10;
    if(t2 > M_PI_2) d = d02;
    if(d < margin) return ON;
    p1.x = p2.x; p1.y = p2.y; }
    return OFF;
}
```

Fig 4.10 Source code for collision detection function

CHAPTER 5
EXPERIMENTAL RESULTS

5.1 INTRODUCTION

To test the use of neuro-fuzzy systems on control of a robotic arm, first a simulator was used to generate training samples from a human's example of controlling of an arm. Then, the training samples were used to train seven different neuro-fuzzy controllers. The seven controllers are named FNN1, FNN2, FNN3, FNN4, FNN5, FNN6, and FNN7 for reference. The controllers varied in the definition of their fuzzy membership functions. This chapter contains the results of the training sessions and the testing of each controller on the simulated arm [50].

The results are primarily shown in three different types of graphs. The fuzzy membership function graphs define the way the crisp values are converted into fuzzy values. There is a fuzzy membership function graph for each of the seven controllers. The RMS training error graph indicates how well each neuro-fuzzy controller was able to learn the training data. The performance histograms quantify the controller's performance in moving the simulated arm from the starting position to the goal without touching the obstacle [51].

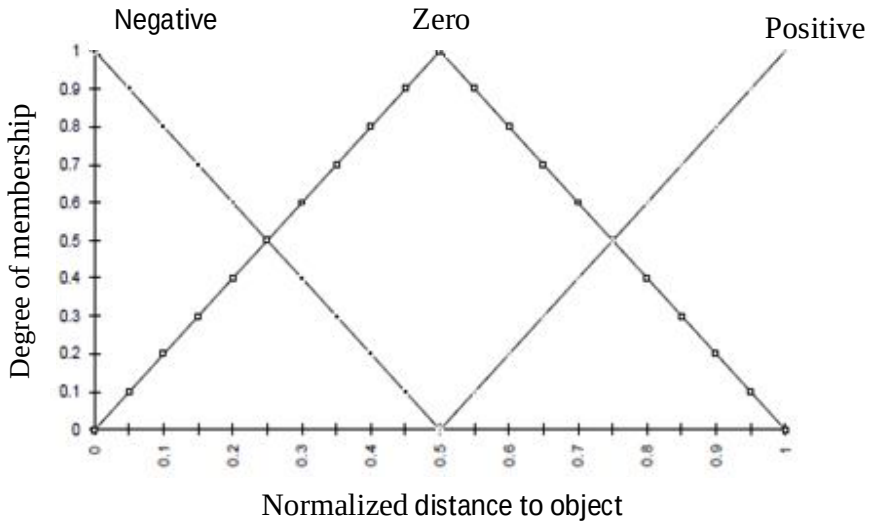


Fig 5.1 The Fuzzy membership function definition FNN1

Figure 5.1 is a graph of the FNN1 fuzzy membership function. It assigns a crisp value's degree of membership to three fuzzy values -- NEG, ZERO, and POS. The fuzzy membership functions used in the controllers FNN2 and FNN3, on the other hand, have seven fuzzy values -- NL, NM, NS, ZR, PS, PM, and PL [52].

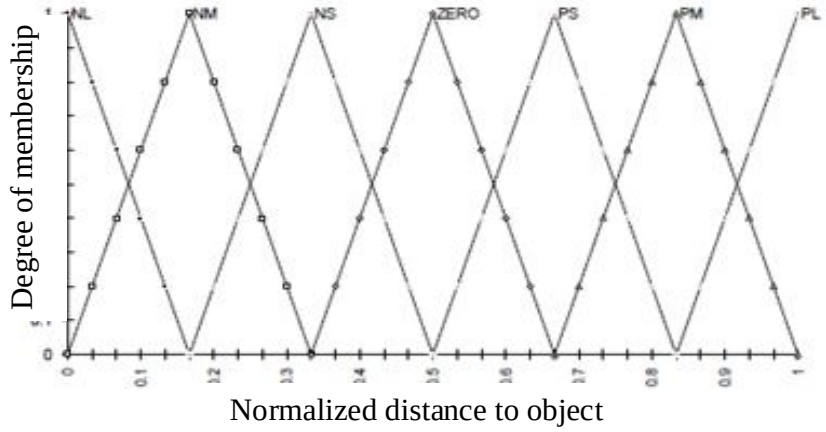


Fig 5.2 The Fuzzy membership function definition FNN2

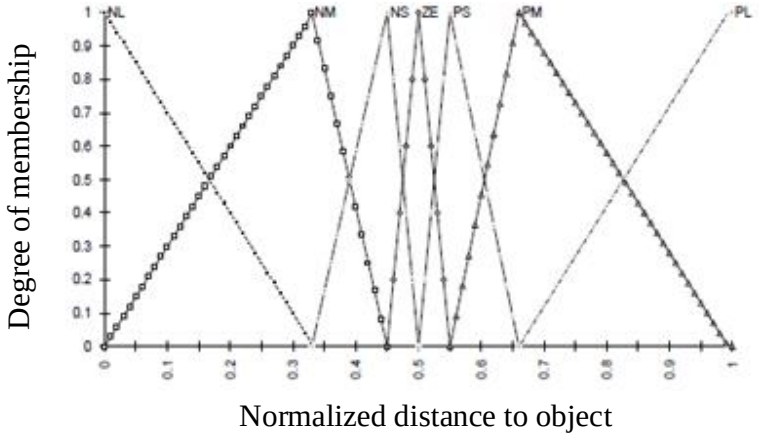


Fig 5.3 The fuzzy membership function definition FNN3

Figures 5.2 and 5.3 illustrate the difference between FNN2 and FNN3. Fnn3 has a concentration of membership values near ZERO. This concentration of values near the center of the function definition is motivated by the fact that the motion strategy changes as the arm approaches the obstacle or goal [62].

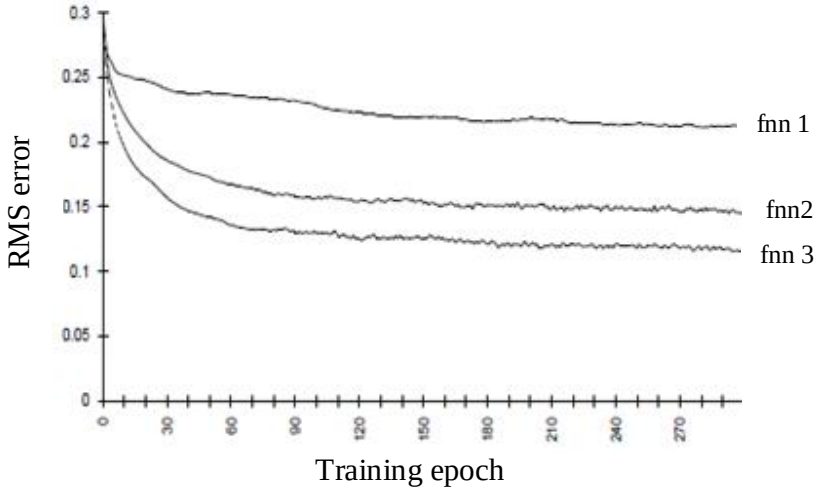


Fig 5.4 The RMS training error of FNN1, FNN2, and FNN3

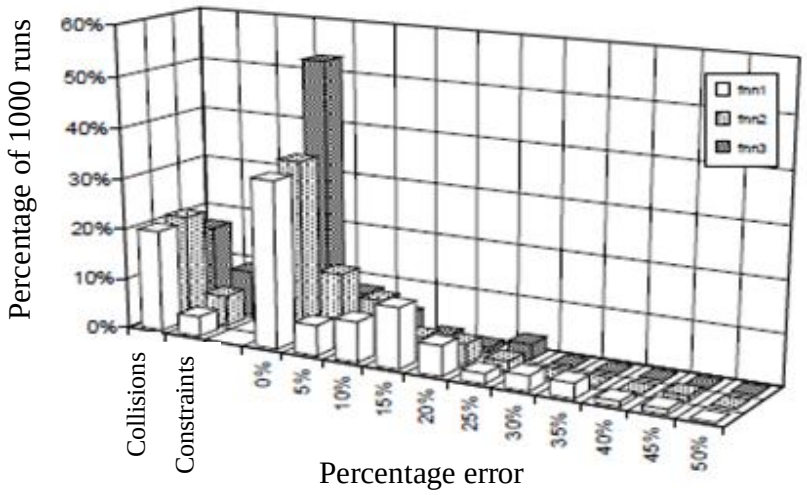


Fig 5.5 Performance histogram of the neuro-fuzzy controller for FNN1, FNN2 and FNN3

A comparison of the different controllers can be made from the performance histograms. The "collisions" category of the histogram records the percentage of runs that resulted in a collision with the obstacle. Figure 5.6 (a) and (b) are screen-shots of the simulator under neuro-fuzzy control. It shows samples of the controller moving the arm into the obstacle. When the neuro-fuzzy controller attempted to move the arm past the joint's capabilities, the "constraint" category was incremented. Figure 5.7 shows samples of this case. The percentage error categories record how close the end effector was to the goal at the arm's final position. Figure 5.8 (a) and (b) are actual trajectories planned by the neuro-fuzzy controller that incremented the 0%-5% error category [53].

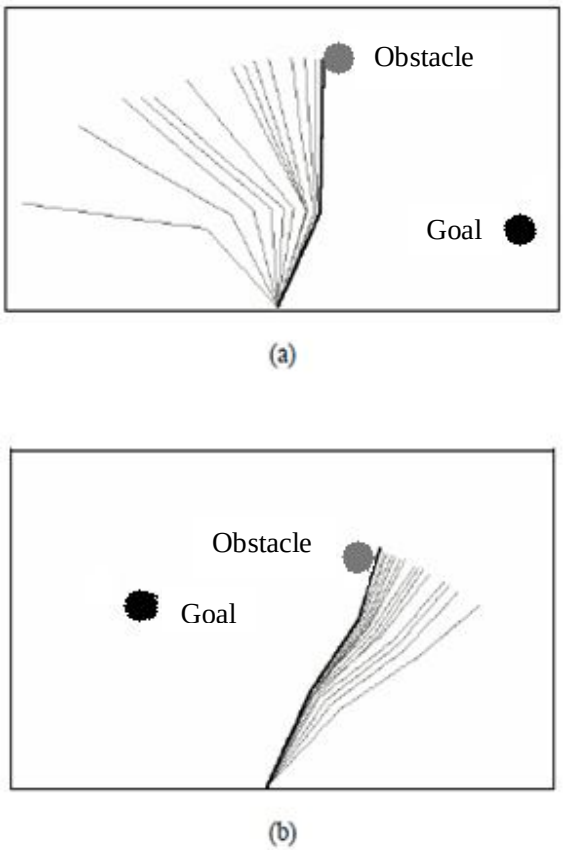


Fig 5.6 Samples of collisions while under neuro-fuzzy control

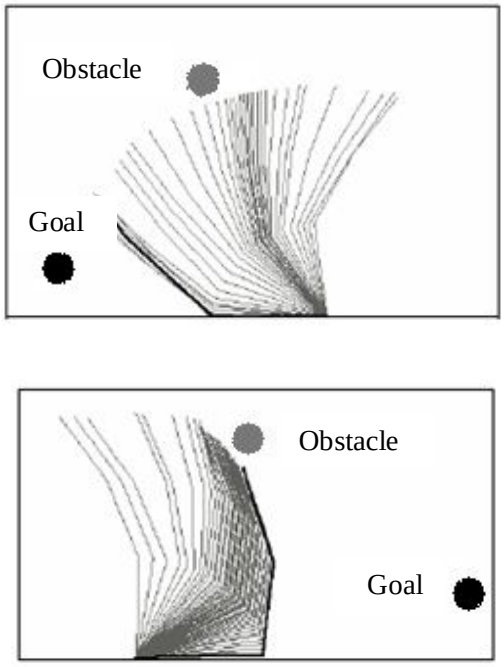
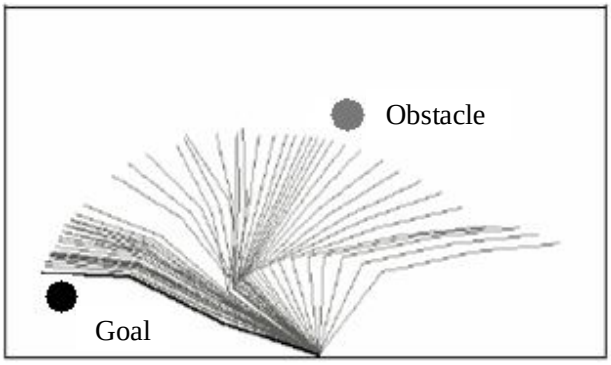


Fig 5.7 Samples of neuro-fuzzy control in which constraint limits were exceeded



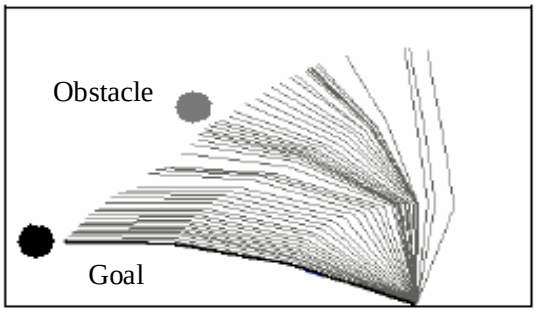


Fig 5.8 Samples of successful control by the neuro-fuzzy controller

The fuzzy membership function definitions FNN4 and FNN5 increase the granularity to 11 fuzzy values as shown in figures 5.9 and 5.10. Similar to FNN3, FNN5 emphasizes the region near the center of operation. The training data and performance histogram are shown in figures 5.11 and 5.12, respectively [61].

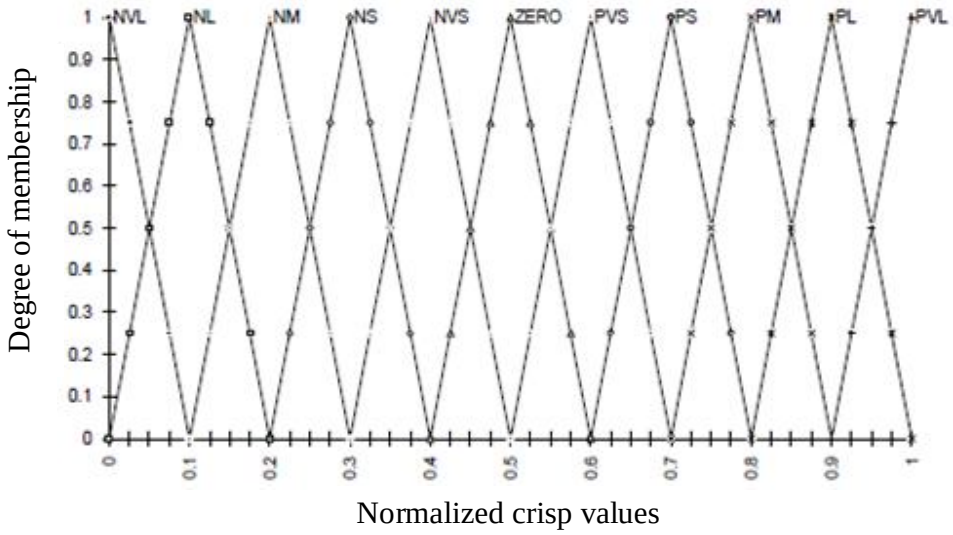


Fig 5.9 The fuzzy membership function definition FNN4

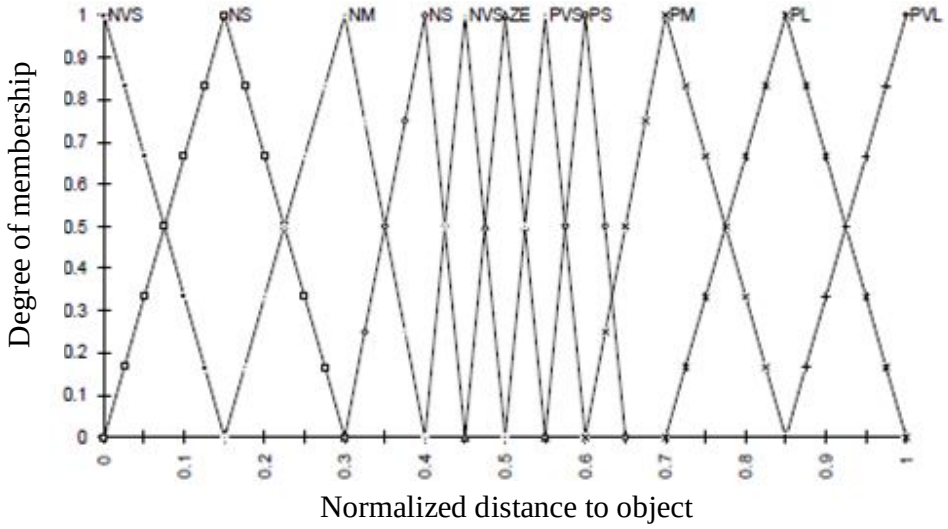


Fig 5.10 The fuzzy membership function definition FNN5

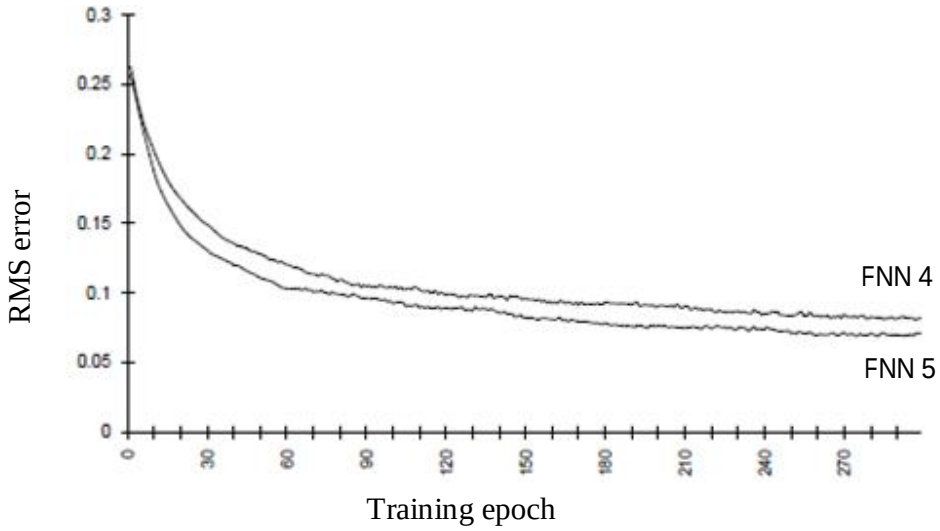


Fig 5.11 The RMS training error of FNN4 and FNN5

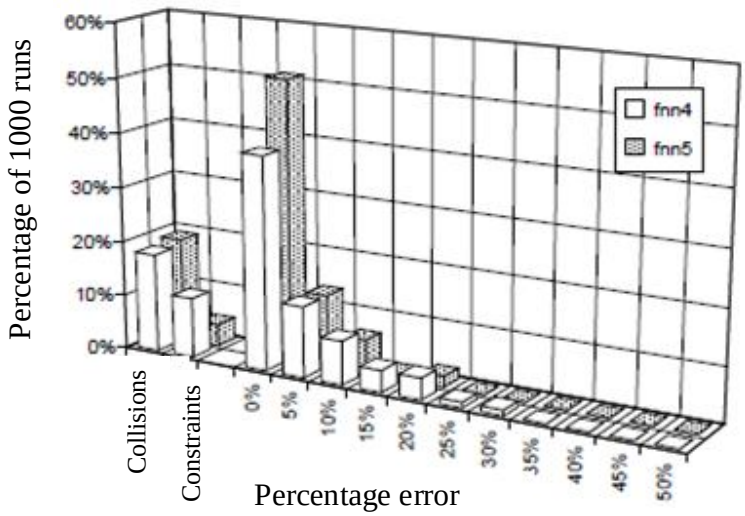


Fig 5.12 Performance histogram of the neuro-fuzzy controller for FNN4 and FNN5

The last two controllers increased the number of fuzzy values to fifteen. FNN6 and FNN7 are shown in figures 5.13 and 5.14, respectively. Figures 5.15 and 5.16 show the controllers' error curves and performance histograms [55].

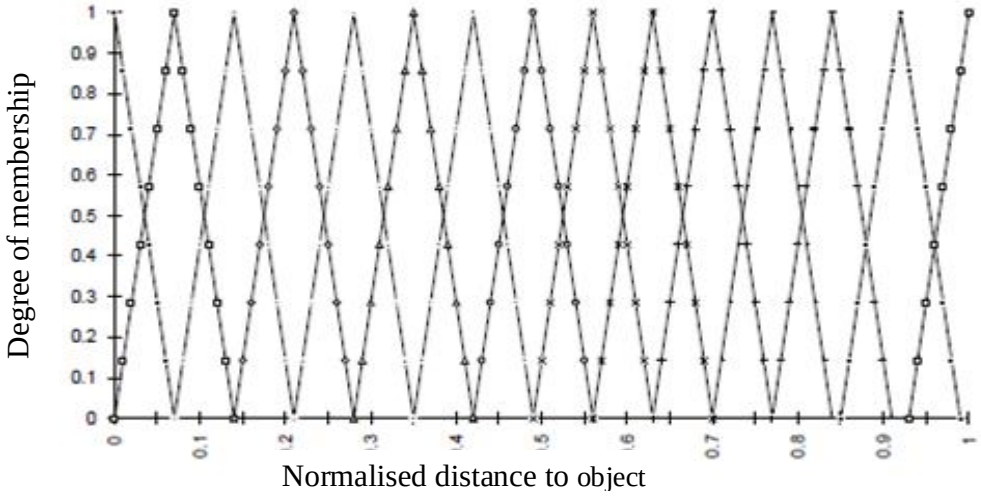


Fig 5.13 The fuzzy membership function definition FNN6

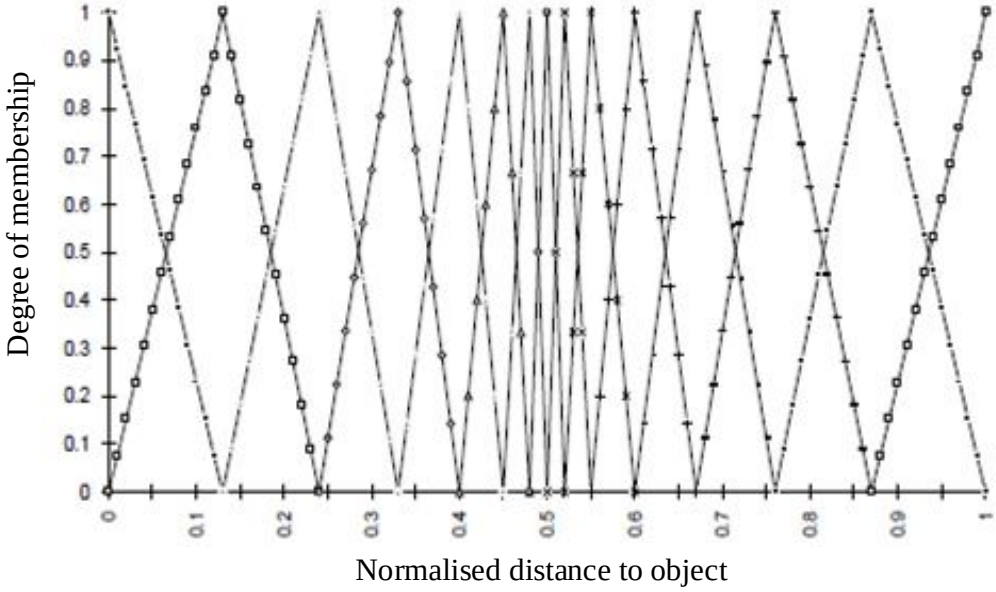


Fig 5.14 The fuzzy membership function definition FNN7

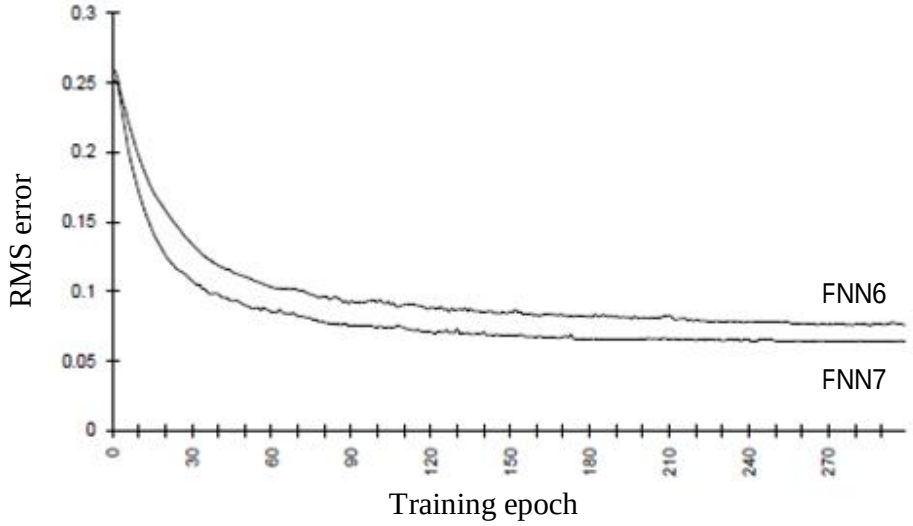


Fig 5.15 The RMS training error of FNN6 and FNN7

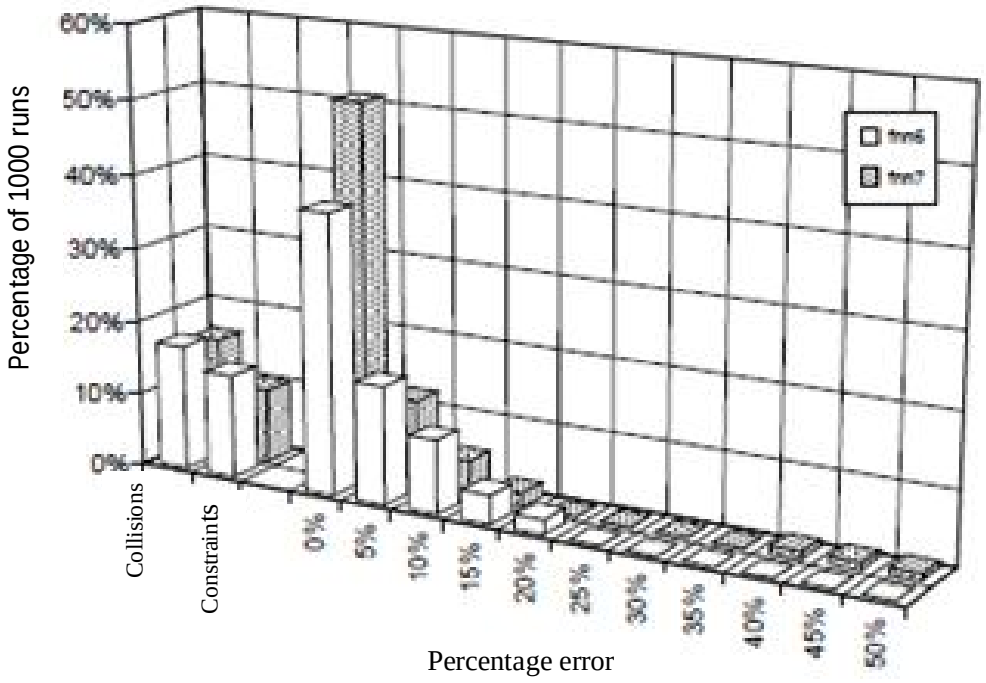


Fig 5.16 Performance histogram of the neuro-fuzzy controller for FNN6 and FNN7

Two more charts are helpful in interpreting the experimental results. Figure 5.17 compares the final RMS training error of the seven controllers. Figure 5.18 compares the collisions and success rate for the seven controllers.

Finally, many attempts were made to train a back propagation neural network to control the simulated arm. The neural networks without the fuzzification layer were not able to learn from the sample data [56].

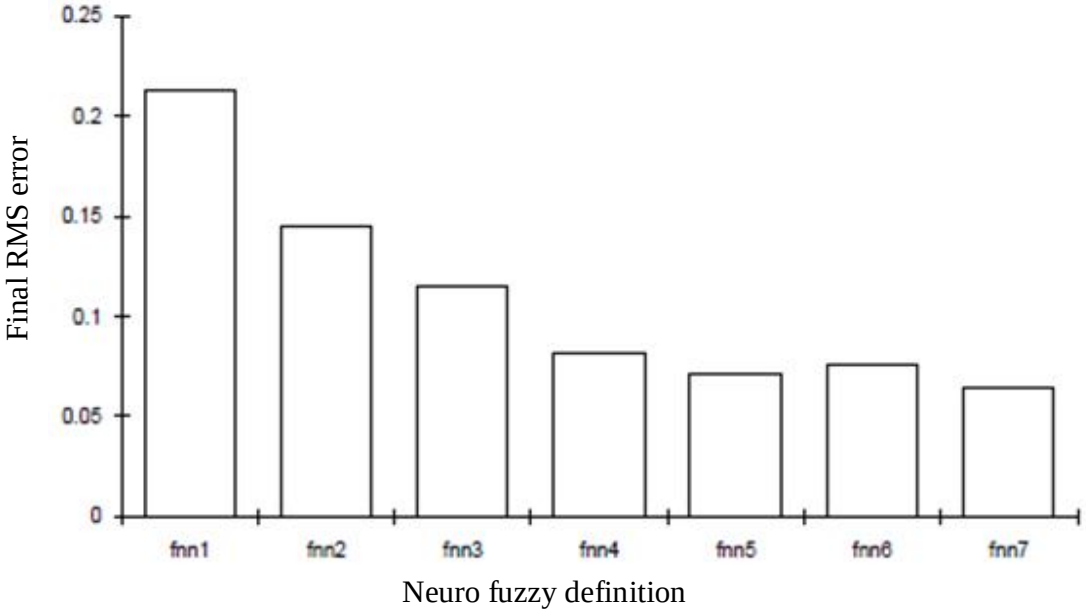


Fig 5.17 A comparison of the final RMS training error

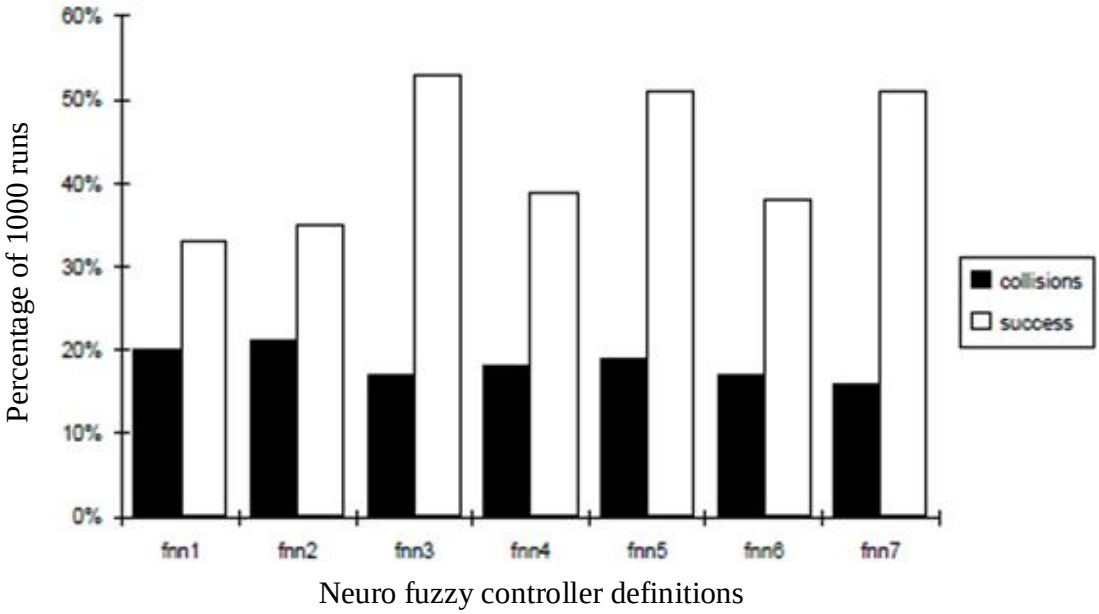


Fig 5.18 A comparison of performance of all the controllers

5.2 SIMULATION RESULTS

After training the ANFIS controller the next objective was to (a) check whether the training is sufficient and produces correct outputs (b) To get the trajectory for new set of obstacle and goal. For this some set of experimental results are generated as follows:

For this complete set of experiment the reference position of robotic arm is x-axis.

Result 1:

θ_1	θ_2	θ_3	x_d	y_d
9	2.5	5	2.50	0.53
18	5.0	10	2.32	1.03
27	7.5	15	2.04	1.50
36	10	15	1.74	1.82
45	12.5	15	1.37	2.10
54	15.0	15	0.96	2.31
63	17.5	15	0.52	2.44
72	20	15	0.64	2.48
81	22.5	15	-0.39	2.44
90	25.0	15	-0.82	2.31

Table 5.1: Data when obstacle is at (0.5, 2.3), goal at (-0.82, 2.31)

Result 2:

θ_1	θ_2	θ_3	x_d	y_d
7	5	5	2.50	0.50
14	10	10	2.33	0.99
21	15	15	2.06	1.42
28	20	20	1.71	1.76
35	25	20	1.36	2.01
42	30	20	0.97	2.19
49	35	20	0.56	2.29
56	40	20	0.15	2.31
63	45	20	-0.25	2.26
70	50	20	-0.64	2.13

Table 5.2: Data when obstacle is at (1.3, 2.2), goal at (-0.64, 2.31)

Result 3:

θ_1	θ_2	θ_3	x_d	y_d
12	4.5	5	2.45	0.71
24	9	10	2.14	1.36
36	13.5	15	1.66	1.88
48	18	20	1.06	2.23
60	22.5	20	0.48	2.40
72	27	20	-0.18	2.41
84	31.5	20	-0.78	2.25
96	36	20	-1.31	1.95
108	40.5	20	-1.73	1.53
120	45	20	-2.03	1.01

Table 5.3: Data when obstacle is at (0, 2.44), goal at (-2.03, 1.01)

Result 4:

θ_1	θ_2	θ_3	x_d	y_d
15	7	5	2.39	0.90
30	14	10	1.89	1.67
45	21	10	1.22	2.18
60	28	10	0.42	2.42
75	35	10	-0.39	2.38
90	42	10	-1.13	2.06
105	49	10	-1.71	1.53
120	56	10-	-2.05	0.84
135	63	10	-2.14	0.10
150	70	10	-1.97	-0.59

Table 5.4: Data when obstacle is at (1, 2.44), goal at (-1.97, -0.59)

Result 5:

θ_1	θ_2	θ_3	x_d	y_d
8	5	5	2.49	0.55
16	10	5	2.33	1.02
24	15	10	2.04	1.48
32	20	10	1.70	1.83
40	25	15	1.24	2.12
48	30	15	0.80	2.28
56	35	15	0.34	2.36
64	40	15	-0.12	2.34
72	45	15	-0.55	2.20
80	50	15	-0.96	2.04

Table 5.5: Data when obstacle is at (0, 2.4), goal at (-.96, 2.04)

Result 6:

θ_1	θ_2	θ_3	x_d	y_d
4	8	-5	2.5	0.33
8	16	-10	2.46	0.65
12	24	-15	2.33	0.96
16	32	-20	2.16	1.23
20	40	-25	1.96	1.48
24	32	-20	1.97	1.5
28	24	-15	1.98	1.6
32	16	-10	1.98	1.6
36	8	-5	1.98	1.62
40	0	0	1.96	1.65

Table 5.6: Data when obstacle is at (2.25, 1.2), goal at (1.96, 1.65)

Result 7:

θ_1	θ_2	θ_3	x_d	y_d
9	9	0	2.47	0.64
18	18	5	2.17	1.28
27	27	10	1.70	1.78
36	36	15	1.10	2.11
45	45	20	0.45	2.22
54	54	20	-0.12	2.17
63	63	20	-0.64	1.97
72	72	20	-1.07	1.64
81	81	20	-1.36	1.22
90	90	20	-1.52	0.75

Table 5.7: Data when obstacle is at (0, 0.23), goal at (-1.52, 0.75)

Result 8:

θ_1	θ_2	θ_3	x_d	y_d
10	15	-5	2.42	0.77
20	30	-10	2.03	1.48
30	45	-15	1.48	1.93
40	60	-20	0.72	2.18
50	75	-25	0.02	2.15
60	60	-20-	-0.04	2.3
70	45	-15	-0.14	2.41
80	30	-10	-0.24	2.48
90	15	-5	-0.34	2.51
100	0	0	-0.44	2.52

Table 5.8: Data when obstacle is at (0.2, 2.3), goal at (-0.44, 2.52)

Result 9:

θ_1	θ_2	θ_3	x_d	y_d
5	3	5	2.53	0.37
10	6	5	2.46	0.66
15	9	5	2.36	0.95
20	12	5	2.22	1.22
25	15	5	2.05	1.47
30	18	5	1.86	1.70
35	21	5	1.64	1.90
40	24	5	1.39	2.07
45	27	5	1.13	2.21
50	30	5	0.85	2.31

Table 5.9: Data when obstacle is at (1.5, 2.1), goal at (0.85, 2.31)

Result 10:

θ_1	θ_2	θ_3	x_d	y_d
13	8	5	2.41	0.84
26	16	5	2.01	1.53
39	24	10	1.36	2.07
52	32	10	0.65	2. 34
65	40	10	-0.10	2.37
78	48	10	-0.86	2.16
91	56	10	-1.39	1.74
104	64	10	-1.78	1.17
117	72	10	-1.96	0.52
130	80	10	-1.92	-0.12

Table 5.10: Data when obstacle is at (-1.5, 1.3), goal at (-1.92,-0.12)

Result 11:

θ_1	θ_2	θ_3	x_d	y_d
18	7	5	2.36	1.02
36	14	5	1.76	1.82
54	21	5	0.93	2.33
72	28	5	-0.02	2.47
90	35	5	-0.75	2.24
108	42	10	-1.72	1.62
126	49	10	-2.14	0.82
144	56	10	-2.22	-0.06
162	63	10	-1.90	-0.88
180	70	10	-1.41	-1.50

Table 5.11: Data when obstacle is at (0.1, 1.5), goal at (-1.41,-1.50)

Result 12:

θ_1	θ_2	θ_3	x_d	y_d
17	5	5	2.38	0.94
34	10	5	1.90	1.69
51	15	10	1.14	2.24
68	20	10	0.30	2. 48
85	25	15	-0.62	2.37
102	30	15	-1.18	2.18
119	35	20	-1.96	1.31
136	40	20	-2.25	0.55
153	45	20	-2.26	-0. 26
170	50	20	-1.98	-0.99

Table 5.12: Data when obstacle is at (-1.5, 1.5), goal at (-1.98,-.0.99)

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

The field of neuro-fuzzy technology will become an important part of intelligent control. The ability to learn how to control a process from sample data is its biggest asset. In this thesis, seven neuro-fuzzy controllers were trained to emulate a human's example control of a robotic arm. Two conclusions can be drawn about the application of neuro-fuzzy control.

First, the membership function definitions are an important part of the neuro-fuzzy system. The final training error was reduced 50% by increasing the number of fuzzy values in a fuzzy set. Membership functions, defined with an important aspect of the problem in mind, consistently improved performance by more than 25%.

Second, the fuzzification of a neural network's inputs and outputs allows neural networks to learn more complex functions than ever before. The function being learned in this thesis project is very complex. An AI researcher would have a difficult time training a traditional neural network to the sample data used in this work. Not only were the neuro-fuzzy networks able to converge to a solution, they did so in a relatively few number of training epochs and with as many as 100 hidden nodes.

The performance of the neuro-fuzzy controllers in this specific application, however, is less than perfect. Even the best controller had a collision rate of 17%. What would improve the performance? What more needs to be understood about this new technology?

Programming some heuristic rules into the control of the arm could improve performance. A large percentage of the "failures" resulted from the controller attempting to move past the physical joint constraints. The control program could move the arm away from such limitations. Another situation that decreases performance occurs when the arm starts oscillating between two points.

While ideally, these situations would naturally not occur in the neural network, a few heuristics in the control program could decrease these problems. A trained neuro-fuzzy system is only as good as the training data used to train it. The seven FNN controllers of this project were trained to 2150 training samples.

6.2 FUTURE WORK

Future work in this area should look into determining the adequacy of training samples. Are there enough training samples? Are all areas of the state space represented? A study of the relationship of the training samples and the fuzzy membership functions would be particularly helpful.

The use of neuro-fuzzy systems for control has been examined. It is the opinion of this researcher that fuzzification of a neural network's inputs and outputs will become standard procedure in neural network applications.

This work can be extended to a robot having more degrees of freedom as well as there can be more constraints i.e. while working more than one obstacle should be avoided. This thesis work simply manages to avoid the collision; This work can be extended:

- As obstacle avoidance with minimum kinetic energy consumed (using Evolutionary Algorithms).
- More precision can be obtained using more no. of membership functions for various inputs and outputs.
- A more complex model of robotic arm can be simulated using more and more precise work.
- The work can be integrated along with MRAC (model reference adaptive control) for more precise and fast results with less training data required.

REFERENCES

[1] J.J. Craig, Introduction to Robotics: Mechanisms and Controls, Addison-Wesley, Reading, MA, 1989.

[2] Clarke, Roger. “Asimov’s Laws of Robotics: Implications for Information Technology”. September 1994.

[3] Wise, E., Applied Robotics, Prompt Publications, Indianapolis, 1999.

[4]Martin, F.G., Robotic Explorations: A Hands-On Introduction to Engineering, Prentice Hall, New Jersey, 2001.

[5] G.C.S. Lee, “Robot Arm Kinematics, Dynamics and Control,” Computer, 15(12), 62-79, 1982.

[6] Andeen, Gerry B., ed. “Robot Design Handbook”. McGraw-Hill Book Company: USA, 1988.

[7] Fu, K.S., R.C. Gonzales, and C.S.G. Lee. “Robotics: Control, Sensing, Vision, and Intelligence”, 1986.

[8] L.A. Zadeh, Fuzzy Sets, Information and Control, 1965

[9] 1. Jacek M. Zurada, Introduction to Artificial Neural Systems,1994.

[10] E. Cox, "Fuzzy Fundamentals" IEEE Spectrum, (October 1993), pp. 58-61,1993.

[11] J.F. Baldwin,” Fuzzy logic and fuzzy reasoning,” in Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (eds.), London: Academic Press, 1981.

[12] L.A. Zadeh, ”Fuzzy algorithms,” Info. & Ctl., Vol. 12, pp. 94-102,1968.

[13] Zimmermann H.J., Fuzzy Sets, Decision Making and Expert Systems, Boston, Kluwer, 1987.

[14] S. Haack,”Do we need fuzzy logic?” Int. Jrnl. of Man-Mach. Stud., Vol. 11, pp.437-445, 1979.

[15] R. Kruse, J. Gebhardt, F. Klawon ”Foundations of Fuzzy Systems”, Wiley, Chichester 1994.

[16] F. Esragh and E.H. Mamdani, ”A general approach to linguistic approximation,” in Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (eds.), London: Academic Press, 1981.

[17] J.F. Baldwin, “Fuzzy logic and fuzzy reasoning,” in Fuzzy Reasoning and Its Applications, E.H. Mamdani and B.R. Gaines (eds.), London: Academic Press, 1981.

[18] M. Caudill, "Neural Networks Primer, Part I", AI Expert, pp. 46-52, (December, 1987).

[19] M. Caudill, "Neural Networks Primer, Part II", AI Expert, pp. 55-61, (February, 1988).

[20] M. Caudill, "Neural Networks Primer, Part III", AI Expert, pp. 53- 59(June, 1988).

[21] A. Bulsari. Neural networks for chemical engineers. Elsevier, Amsterdam, 1995.

[22]. Martin T. Hagan, Howard B. Demuth, Mark Beale, Neural Network Design.1990.

[23]M.A. Hussain, M. A. “Review of the applications of neural networks in chemical process control. Simulation and on-line implementations”, Artificial Int. in Eng, Vol. 13 pp. 55-68,1999.

[24] R. Jang and C. Sun. Neuro-fuzzy modeling and control. Proceedings of the IEEE 1995, Vol. 83 n. 3 pp. 378-405, 1995.

[25] L. N. Cooper, "Hybrid Neural Network Architectures: Equilibrium Systems That Pay Attention", in Neural Networks: Theory and Applications (ed. Richard J. Mammon and Yehoshua Y. Zeevi) Boston: Academic Press, Inc., 1991, pp. 81-96. [8] A. Guez, J. Eilbert, and M. Kam, "Neural Network Architecture for Control", IEEE Control Systems Magazine, pp. 22 – 25, April, 1988.

[26] D. V. Hillman, "Integrating Neural Nets and Expert Systems", AI Expert pp. 54-59, June, 1990.

[27] R. Hecht-Nielson, "Neurocomputer Applications" in Proceedings of the 1987 IEEE Asilomar Signals & Systems Conference. IEEE Press, 1988.

[28] B. Kosko, Neural Networks and Fuzzy Systems. Englewood Cliffs: Prentice Hall Inc., 1992.

[29] L. McLauchlin, Supervised and Unsupervised Learning Applied to Robotic Manipulator Control, (Master's Thesis) Kingsville, TX: Texas A&I University, August, 1993.

[30]M. Sadiku and M. Mazzara. "Computing with Neural Networks" IEEE Potentials, pp. 14-16, October, 1993.

[31] D. Psaltis, A. Sideris, and A. Yamamura, "A Multilayered Neural Network Controller", IEEE Control Systems Magazine, pp. 17-21, April, 1988.

[32] S. Rahman, "Neural-Fuzzy Consumer Appliance Applications", Proceedings of Fuzzy Logic, pp. M234-1 - M234-7,1993.

[33] [12] E. Khan, "An Elegant Combination of Fuzzy Logic & Neural Nets", Proceedings of Fuzzy Logic, pp. A223-1 - A223-7,1994.

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

[35] S. C. Lee, and E. T. Lee, "Fuzzy sets and neural networks" Journal of Cybernetics Vol 4. pp. 83 -103, 1974.

[36] J.-S. R. Jang, "ANFIS: Adaptive-Network-based Fuzzy Inference Systems," IEEE Transactions on Systems, Man, and Cybernetics, 665-685, May 1993.

[37] Jang, J., Sun, C., and Mizutani, E., Neuro Fuzzy and Soft Computing, Printice-Hall, Upper Saddle River, NJ, 1997.

[38] Rasit Coker, Cemil Oz, Turk Cakar, Hussein Elis, "A study of neural network based inverse kinematics solution for a three-joint robot," Robotics and Autonomous Systems, 49, 227-234, 2004.

[39]Jang J.S.: ANFIS: Adaptive-Network-based Fuzzy Inference System. IEEE Trans. On system, Man and Cybernetics. Vol.23, No 3, pp. 665-685, May/June 1993.

[40] J. Engin, J. Kuvulmaz and E. Omurlu. "Fuzzy control of an ANFIS model representing a nonlinear liquid level system". Neural Computing and Appl., Vol. 13, n. 3 pp. 202-210, 2004,.

[41]M. Denai, F. Palis and A. Zeghbib. "ANFIS based modeling and control of nonlinear systems: A tutorial", Proceedings of the IEEE Conference on SMC, pp. 3433-3438,2004.

[42] Kong and B. Kosko, "Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems" in Neural Networks and Fuzzy Systems. Englewood Cliffs: Prentice Hall Inc., 1992.

[43] J. Jang, "Self-Learning Fuzzy Controllers Based on Temporal Back Propogation", IEEE Transactions on Systems, Man, and Cybernetics, 1992.

[44]H. Berenji, Y. Chen, C. Lee, S. Murugesan, and J. Jang, "An Experiment-based Comparative Study of Fuzzy Logic Control" Proceedings of the American Association for Artificial Intelligence Conference, pp. 2751 - 2753, 1988.

[45]H. Leegwater. Industrial experience with double quality control. In W. L. Luyben (Ed.), Practical distillation control. New York, USA: Van Nostrand Reinhold, 1992.

[46] S. Park and C. Han. " A non-linear soft sensor based on multivariate smoothing procedure for quality estimation in distillation columns", Comp. and Chem. Eng. Vol. 24 ,pp. 871-877,2000.

[47] P. Fleming and R. Purshouse – Evolutionary algorithms in control systems engineering: a survey. Control Engineering Practice, Vol. 10 pp. 1223–1241,2002.

[48] M. Diehl, I. Uslun and R. Findeisen.”Real-time optimization for large scale processes: Nonlinear predictive control of a high purity distillation column”, On Line Optimization of Large Scale System: State of the Art, Springer-Verlag, 2001.

[49] J. Fernández de Cenete, S. Gonzalez-Perez and P. del Saz Orozco. “A development of tools for monitoring and control of multivariable neurocontrolled systems with application to distillation columns”, Proc. EANN 2007 Int. Conf., Thessaloniki, Greece, pp. 296-305,2007.

[50] R. Lea, Y. Jani, and H. Berenji, "Fuzzy Logic Controller with Reinforcement Learning for Proximity Operations and Docking", Fifth IEEE International Symposium on Intelligent Control, 1990.

[51] Y. Li and C. Lau, "Development of Fuzzy Algorithms for Servo Systems", in IEEE Control Systems Magazine, pp. 65 - 71, April, 1990.

[52]P. Pacini and B. Kosko, "Comparison of Fuzzy and Kalman-Filter Target- Tracking Control Systems" in Neural Networks and Fuzzy Systems. Englewood Cliffs: Prentice Hall Inc., 1992.

[53] L. A. Zadeh. "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes" IEEE Transactions on Systems, Man, and Cybernetics, pp. 28 – 44, January 1973.

[54] O. Iida, M. Sato, M. Kiguchi, T. Iwamura, S. Fukumura. "Blast Furnace Control by Artificial Intelligence", Artificial Intelligence in Real-time Control (ed. M. G. Rodd and G. J. Suski) Swansea, UK: Pergamum Press, p. 73, 1988.

[55] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Imminent in Nervous Activity," Bulletin of Mathematical Biophysics, Vol 5, 1973.

[56] V. J. Lumelsky, "Effect of Kinematics on Motion Planning for Planar Robot Arms Moving Amidst Unknown Obstacles," Journal of Robotics and Automation Vol. RA-3, pp. 207-222, June, 1987.

[57] M. Galicki, "Optimal Planning of a Collision-free Trajectory of Redundant Manipulators," The International Journal of Robotics Research, Vol 11, No. 6, December, 1996.

[58] D. Wang and Y. Hamam, "Optimal Trajectory Planning of Manipulators With Collision Detection and Avoidance", The International Journal of Robotics Research, Vol 11, No. 5, pp. 460 – 468, October 1992.

[59] C. H. Lin and L. C. Fu, "Motion Planning of Robot Manipulators with Arbitrary Number of DOF", Proceedings of the 30th IEEE Conference on Decision and Control, pp. 359 – 370,1990.

[60] F. Arai, L. Rong, and T. Fukuda, "Trajectory Control of Flexible Plate Using Neural Network", Proceedings of the IEEE International Conference on Robotics and Automation, Vol, pp. 155 – 160, 1 May 1993.

[61] S. Liu and H. Asada, "Teaching and Learning of Deburring Robots Using Neural Networks", Proceedings of the IEEE International Conference on Robotics and Automation, Vol 3, pp. 339 – 345,May 1993.

[62] T. C. Hsia and Z. Mao, "Obstacle Avoidance Inverse Kinematics Solution of Redundant Manipulators by Neural Networks", Proceedings of the IEEE International Conference on Robotics and Automation Vol. 3, p.1014,May 1993.

APPENDIX A

The programming and debugging of the neuro-fuzzy engine was an extensive process. This appendix includes the data from that testing process and verifies the validity of the neuro-fuzzy engine.

LINEAR FUNCTION

The first function used to test the neuro-fuzzy system was $y = x$. After 500 training epochs, the neuro-fuzzy system produced the output shown in figure A.1.

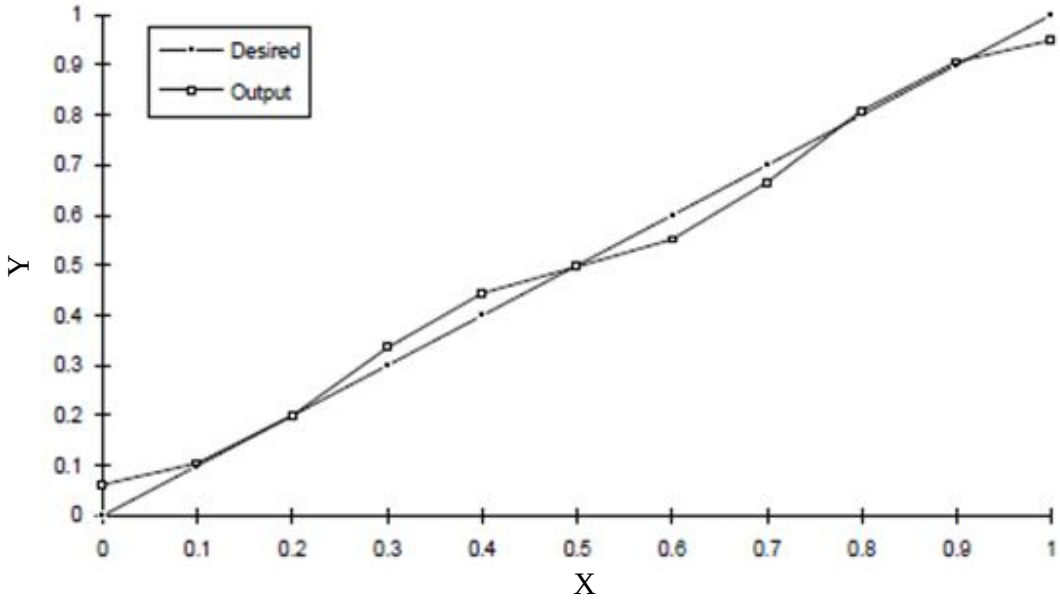


Fig A.1 The performance of the neuro-fuzzy system on a linear function.

SINUSOIDAL FUNCTION

The ability of the neuro-fuzzy system to learn more complex data was tested with the function *y = sin(x)*. Figure A.2 shows the desired output and the network's output after 250 training epochs.

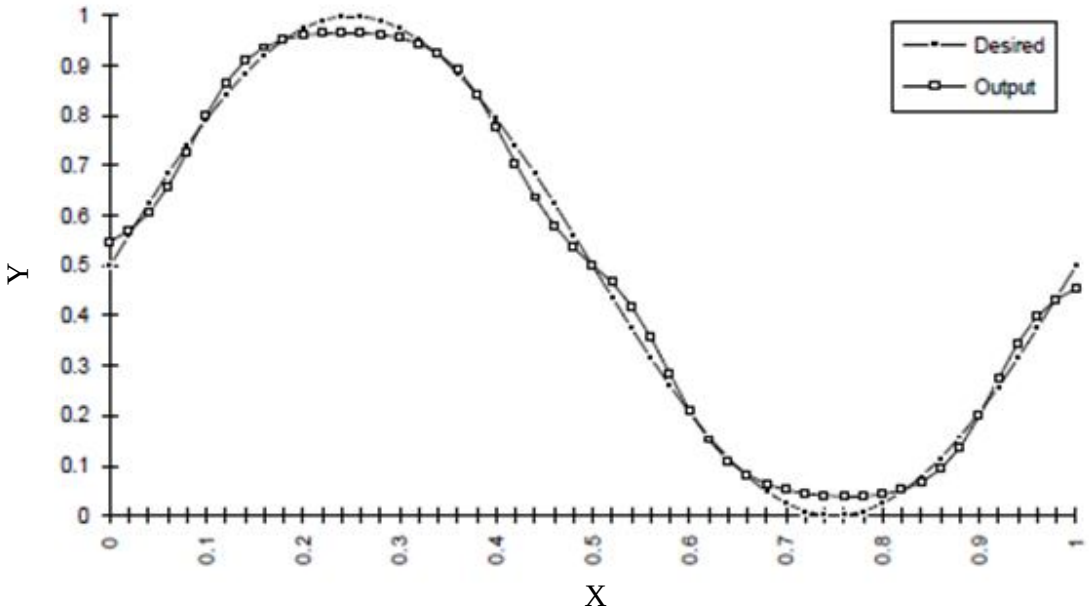


Fig A.2 The performance of the neuro-fuzzy system on a sinusoidal function

TWO DIMENSIONAL FUNCTIONS

A programming bug existed in the neuro-fuzzy engine that was not a problem for single-input, single-output functions. Testing with $z = x\ 2|sin(y)|$ revealed the error. After correcting the programming bug and training the network for 400 epochs, the network learned the function, as shown in figure A.3.

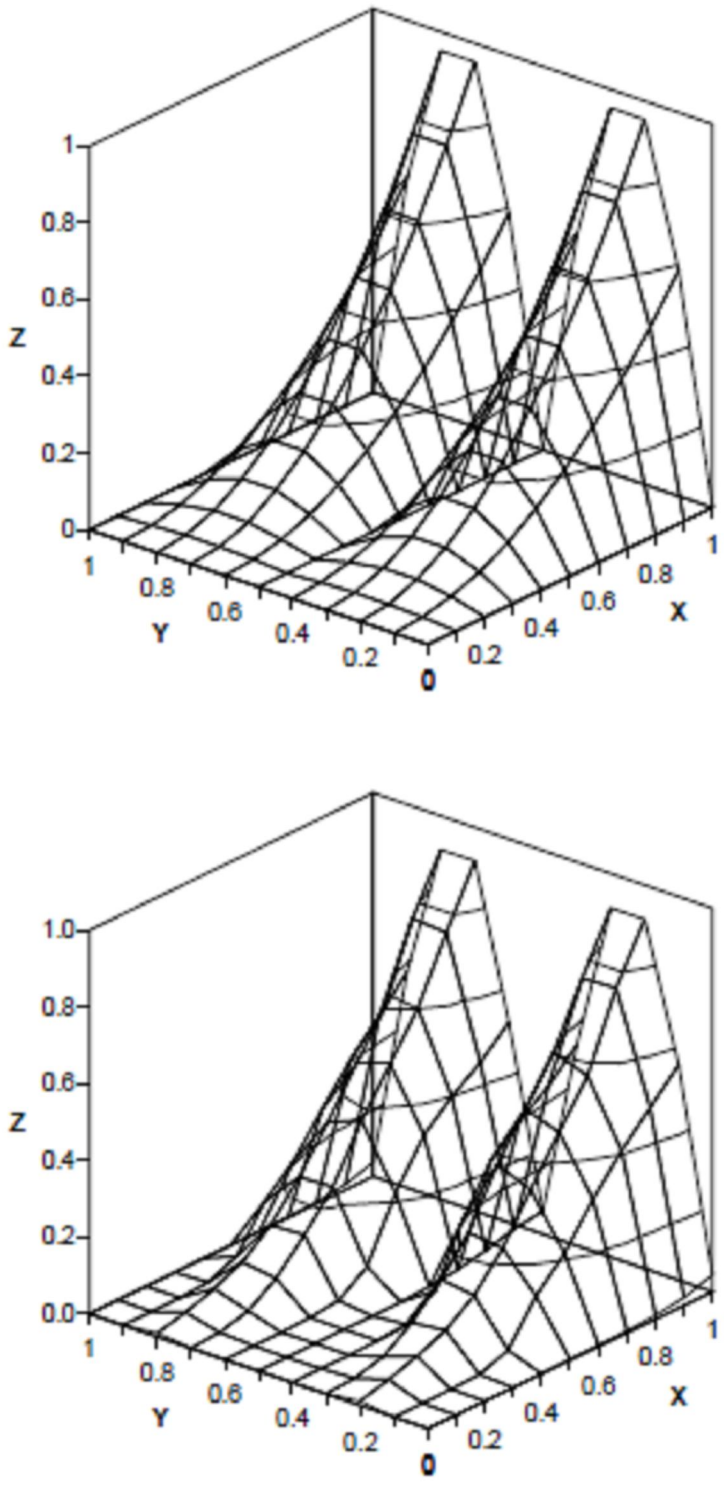


Fig A.3 The desired output (top) and the actual output for the 2D test data

THE LEARNING RATE CONSTANT

Another important part of any system involving a back propagation neural network is the learning rate constant. The learning rate constant, b, determines how much the weights in the neural network are adjusted as the error is back propagated. The learning equation are used in most BPNs. The effect of the learning rate constant on three training sessions for the sine function is shown in figures A.4. All three training sessions began the networks with the same initial weights. The training error decreases faster for the network with the higher learning rate.

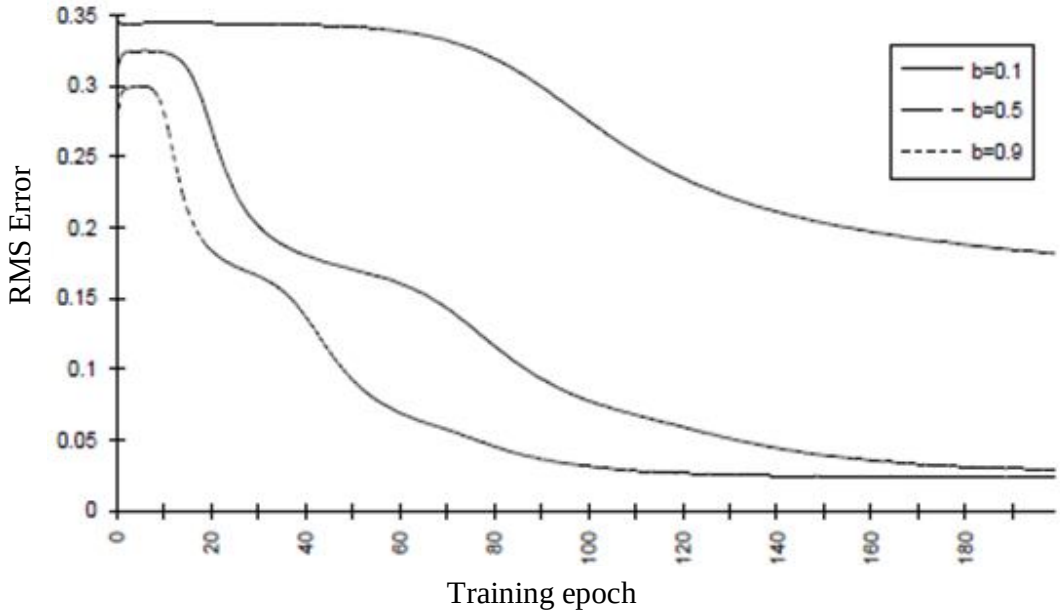


Fig A.4 The effect of the learning rate constant

THE MOMENTUM CONSTANT

It became apparent during the early stages of this thesis project that the neuro-fuzzy system would benefit from the addition of a momentum term in the learning rule. Experiments show that the momentum term causes the system to converge more quickly.

Figure A.5 is a comparison of the network output after two training sessions that were identical except for the inclusion of a momentum term.

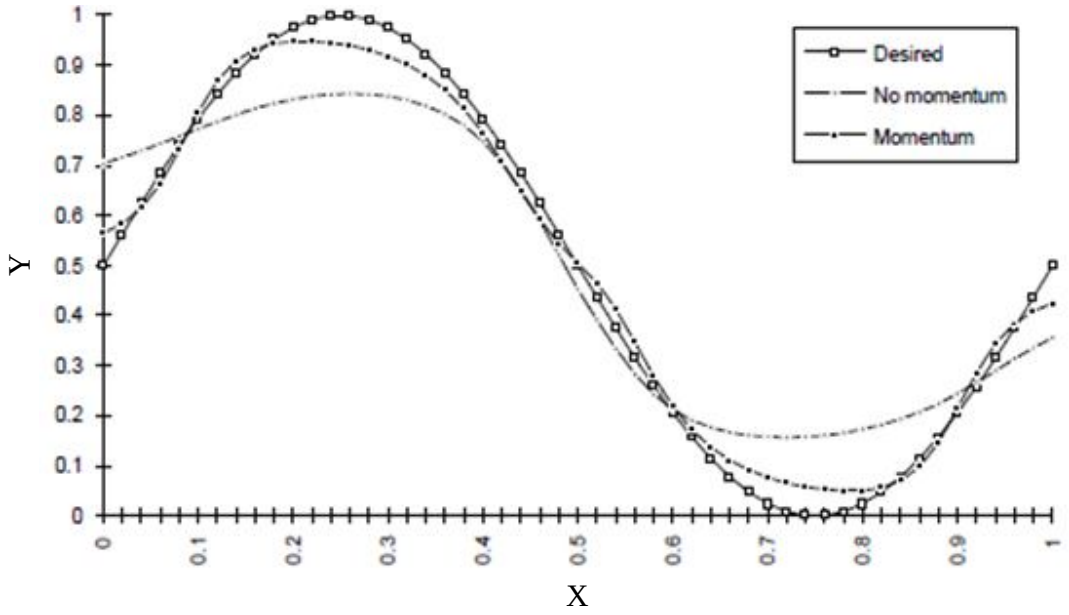


Fig A.5 The effect of momentum term

THE FUZZY MEMBERSHIP FUNCTIONS

In order to test the fuzzy membership function features, the neuro-fuzzy engine was tested for several functions with varying membership function definitions. Figure A.6 reveals the effect of the fuzzy membership function definition on a network training to a sinusoidal function. The "error1" line corresponds to the output error of the network with input and output fuzzy membership function definition #1, shown in figure A.7 (a). The plot of "error2" corresponds to definition #2, shown in figure A.7 (b). The error was less for the fuzzifier/defuzzified with the higher granularity.

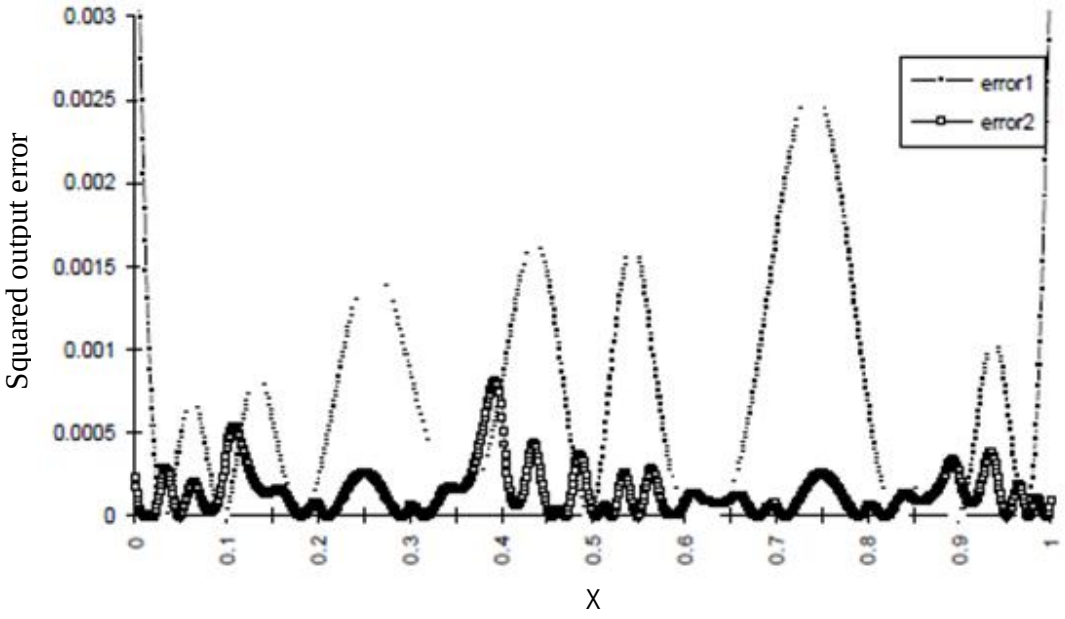


Fig A.6 The effect of the fuzzy membership function definition

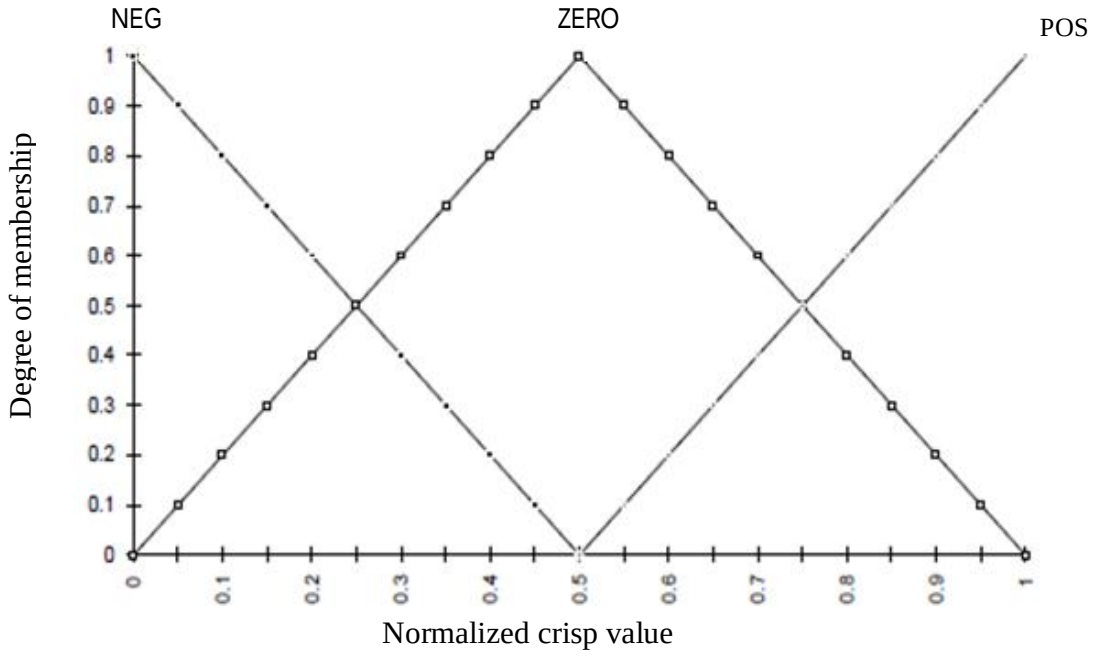


Fig A.7 (a) The fuzzy membership function, test definition #1

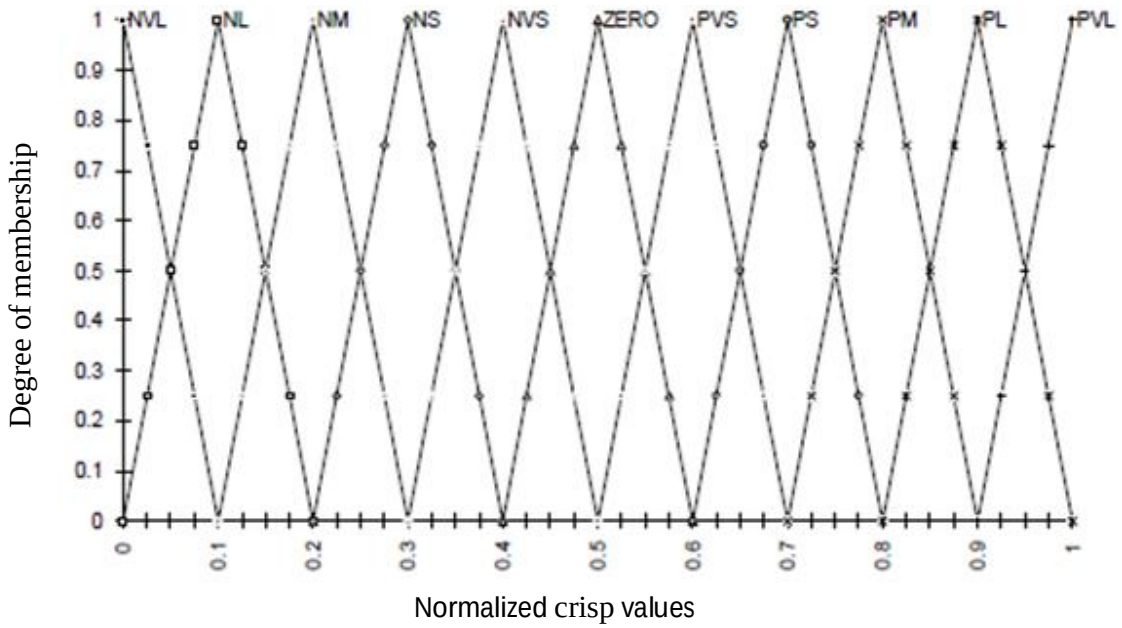


Fig A.7 (b) The fuzzy membership function, test definition #2