



# Simulation of Neuro-Fuzzy control of a Robotic Arm

C. Prabahar Dana Singh<sup>1</sup>, T. RAJAPRATHAB<sup>2</sup>

<sup>1</sup>Post Graduate Student, Department of Electronics and Communication Engineering,  
Satyam college of Engineering.

<sup>2</sup>Assistant Professor & Head, Department of Electronics and Communication Engineering,  
Satyam College of Engineering.

**Abstract**—A new robust Neuro-Fuzzy controller for robotic arm is presented, Neuro-Fuzzy technology is the combination of Neural networks and Fuzzy logic. 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. Here we implement the same in controlling a three joint robotic arm to move around the obstacle to reach the goal. The logic is simulated by creating a simulator and inputs are fed to system from user regarding the obstacle and goal. 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.

**Keywords:** Robot arm, Fuzzy, Cartesian coordinates.

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. Prompted by the weaknesses inherent in the two technologies and their complementary strengths, researchers have looked at ways of combining neural networks and fuzzy logic.

## I. INTRODUCTION

THIS paper is regarding controlling a robotic arm with the aid of neuro fuzzy combinational technique in presence of an obstacle. Using the tech we are going to learn and guide the movement of robotic arm. This technology mimics the human brain, eye and hand coordination. The ability to move an arm around an obstacle and to reach the goal is an intuitive skill for human beings. Translating that skill into instructions for a robotic arm, however, is not an easy task. 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

The technique used in this work replaces the rule-base of a traditional fuzzy logic system with a back propagation neural network. Using neuro-fuzzy techniques, a robotic arm can be trained to plan its movements to avoid a collision with obstacles in its vicinity. Before discussing further let's start with the technology used, the pros and cons of technology and by combining these two technologies how we are attaining the goal efficiently.

## II. ROBOTIC ARM PROBLEM AND SOLUTION

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 paper addresses the problem of planning the trajectory of a three-link robotic arm in the 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 2.1. 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.

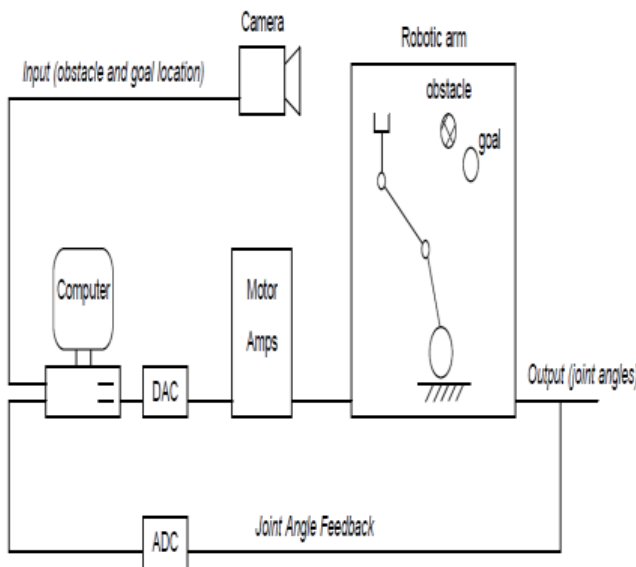


FIGURE 2.1: BLOCK DIAGRAM OF SAMPLE CONFIGURATION FOR THE PROBLEM APPLICATION

The controller will determine a series of joint angles,  $\theta(t)$ , 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  $(x_o, y_o)$ . 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 2.2.

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 is an infinite number of solutions for possible joint angles. 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

$$x_e = l_1 \cos \theta_1 + l_2 \cos (\theta_1 + \theta_2) + l_3 \cos (\theta_1 + \theta_2 + \theta_3) \quad --(2.1)$$

$$y_e = l_1 \sin \theta_1 + l_2 \sin (\theta_1 + \theta_2) + l_3 \sin (\theta_1 + \theta_2 + \theta_3) \quad -- (2.2)$$

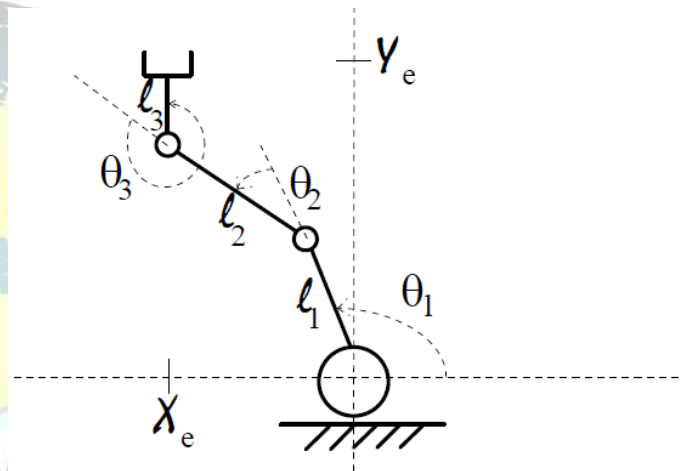


FIGURE 2.2: ROBOTIC ARM VARIABLE DEFINITION

### III. INPUT AND OUTPUT OF THE SYSTEM

There are 7 inputs for the system they are:

- $\theta_1$ , joint angle of link1 to the base.
- $\theta_2$ , joint angle of link2 from link1's axis
- $\theta_3$ , 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

There are totally 3 outputs of the system they are:

- $\Delta\theta_1$ , 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

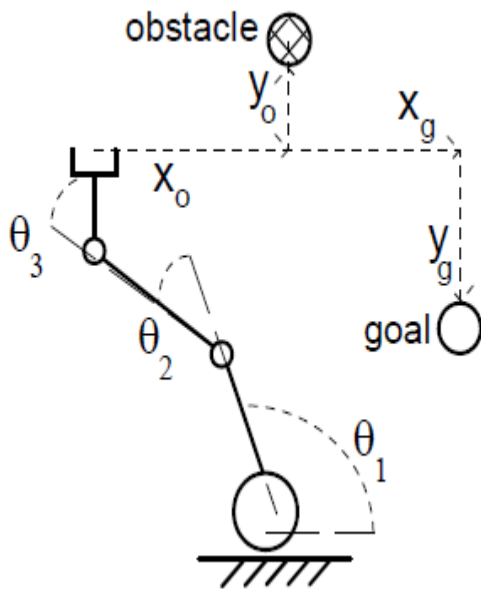


FIGURE 3.1: STATE REPRESENTATION FOR THE OBSTACLE PLATFORM

#### IV. ALGORITHM USED IN THIS SYSTEM

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. Although there is no way to determine, what is the best number of hidden nodes? 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. 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 4.1.

$$I = f(\sum(W_i * X_i)) \quad \text{-- (4.1)}$$

Where  $i$  is the input index,  $w$  is the weight of that input,  $x$  is the input signal and  $f(x)$  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 sigmoidal; that is, it must be continuous, S-shaped, monotonically increasing, and asymptotically approaching fixed values as the input approaches plus or minus infinity. 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 4.2 is an example of such a function.

$$f(x) = 1/(1+e^{-x}) \quad \text{-- (4.2)}$$

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 1.3 and 1.4.

$$e_i = f'(I) * \sum_j (W_{ij} * E_j) \quad \text{-- (4.3)}$$

$$f'(x) = f(x) * (1 - f(x)) \quad \text{-- (4.4)}$$

Where  $i$  refers 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  $j$  and  $f(x)$  is the activation function.

The derivative term contributes to stability and 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. One way to help guard against the local minimum problem is to include a momentum term in the Delta rule. Equation 4.5 adds a momentum term to the delta rule of equation.

$$W_{new} = W_{old} + \beta E * (X/|X_2|) + \alpha (W_{new} - W_{old})_{prev} \quad --(4.5)$$

where  $W$  is the weight vector for a processing element,  $X$  is the input vector,  $\beta$  is a programmer defined constant between 0 and 1,  $y$  are the desired and actual outputs (scalar),  $\alpha$  determines how much weight to put on momentum.

## V. COLLISION AVOIDANCE AND DETECTION

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. The goal of our system is to determine a trajectory  $\theta(t)$ , such that:

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

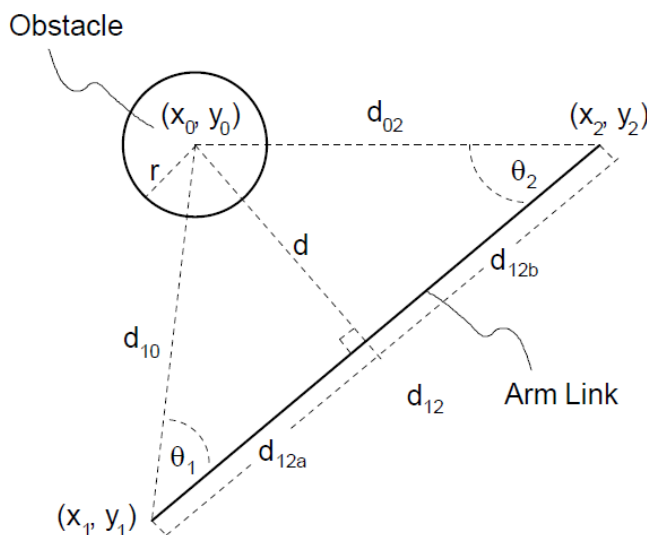


FIGURE 5.1: DETERMINATION OF A COLLISION BETWEEN THE OBSTACLE AND A LINK.

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. 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 by equations 4.3 through 4.10 where all the variables are defined.

$$d_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad --(5.1)$$

$$d_{10} = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad --(5.2)$$

$$d_{02} = \sqrt{(x_0 - x_2)^2 + (y_0 - y_2)^2} \quad --(5.3)$$

$$\theta_1 = \arccos((d_{12}^2 + d_{10}^2 - d_{02}^2)/(2d_{12}d_{10})) \quad --(5.4)$$

$$\theta_2 = \arccos((d_{02}^2 + d_{10}^2 - d_{12}^2)/(2d_{02}d_{10})) \quad --(5.5)$$

$$d = d_{10} \text{ for } (\theta_1 > \pi/2) \quad --(5.6)$$

$$d = d_{02} \text{ for } (\theta_2 > \pi/2) \quad --(5.7)$$

$$d = d_{10} \sin(\theta_1) \text{ otherwise} \quad --(5.8)$$

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

## VI. EXPERIMENTAL RESULT

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 a controller. The fuzzy membership function graphs define the way the crisp values are converted into fuzzy values. Presently we have designed a single controller and its output as follows. After training the 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.

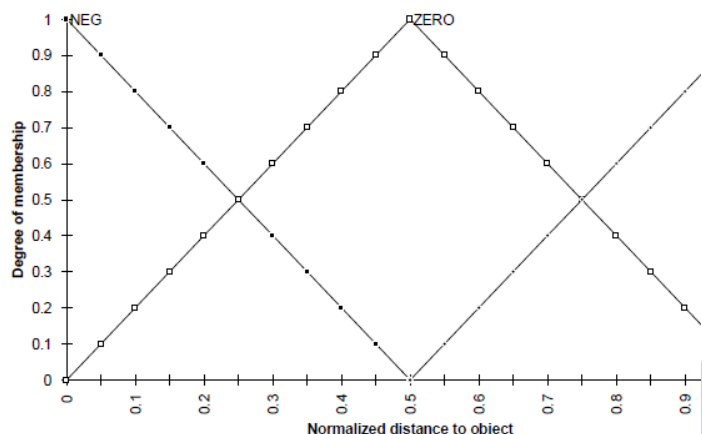


FIGURE 6.1: THE FUZZY MEMBERSHIP FUNCTION DEFINITION FNN1

01	02	03	xd	yd
9	2.5	5	2.50	0.53
18	5	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	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	15	-0.82	2.31

TABLE 6.1: DATA WHEN OBSTACLE IS AT (0.5, 2.3), GOAL AT (-0.82, 2.31).

## VII. 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 neuro-fuzzy controllers were trained to emulate a human's example control of a robotic arm. First, the membership function definitions are an important part of the neuro-fuzzy system. 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 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%. In this only one controller is trained and presented.

## VIII. REFERENCES

- [1] Deb S.R. *Robotics technology and flexible automation*, Tata McGraw-Hill Publishing Company Limited, New-Delhi, 2008.
- [2] Clarke, R. *Asimov's Laws of Robotics: Implications for Information Technology- part II*, *Computer*, 27(1), September (1994): pp.57-66.
- [3] Martin, F.G. *Robotic Explorations: A Hands-On Introduction to Engineering*, Prentice Hall, New Jersey, 2001.
- [4] Sciavicco L. and Siciliano B. *Modelling and Control of Robot Manipulators*, Springer Second edition, Chapter 3, (2000), pp. 96.
- [5] Shimizu M., Kakuya H., Yoon W., Kitagaki K., and Kosuge K. *Analytical Inverse Kinematic Computation for 7-DOF Redundant Manipulators with Joint Limits and its application to Redundancy Resolution*, *IEEE Transaction on Robotics*, 24 No. 5: October. (2008).
- [6] Moradi H. and Lee S. *Joint limit analysis and elbow movement minimization for redundant manipulators using closed form method*, In *Advances in Intelligent Computing*, Berlin/Heidelberg: Springer, Part 2, 3645(2005): pp. 423-432.
- [7] Koivo H. *ANFIS :Adaptive Neuro-Fuzzy Inference System*. European Symposium on Intelligent Technology, Aachen, Germany, 2000.