Design of a Neuro Fuzzy Controller

Gurpreet S. Sandhu and Kuldip S. Rattan
Department of Electrical Engineering
Wright State University
Dayton, Ohio 45435
TEL: (937) 775-5052
FAX: (937) 775-5009
EMAIL: krattan@cs.wright.edu

ABSTRACT

Classical control theory is based on the mathematical models that describe the physical plant under consideration. The essence of fuzzy control is to build a model of human expert who is capable of controlling the plant without thinking in terms of mathematical model. The transformation of expert's knowledge in terms of control rules to fuzzy frame work has not been formalized and arbitrary choices concerning, for example, the shape of membership functions have to be made. The quality of fuzzy controller can be drastically affected by the choice of membership functions. Thus, methods for tuning the fuzzy logic controllers are needed. In this paper, neural networks are used in a novel way to solve the problem of tuning a fuzzy logic controller.

The neuro fuzzy controller uses the neural network learning techniques to tune the membership functions while keeping the semantics of the fuzzy logic controller intact. Both the architecture and the learning algorithm are presented for a general neuro fuzzy controller. From this general neuro fuzzy controller, a proportional neuro fuzzy controllers is derived. A step by step algorithm for off-line training is given along with numerical examples.

1. INTRODUCTION

Fuzzy systems and neural networks have attracted the interest of researchers in various scientific and engineering areas [1,2]. The number and variety of applications of fuzzy logic and neural networks have been increasing, ranging from consumer products [3,4] and industrial process control [5] to medical instrumentation [6,7], information systems [8] and decision analysis [9].

The main idea of fuzzy logic control (FLC) is to build a model of a human control expert who is capable of controlling the plant without thinking in terms of a mathematical model. The control expert specifies his control actions in the form of linguistic rules. These control rules are trans-

lated into the framework of fuzzy set theory providing a calculus which can simulate the behavior of the control expert. The specification of good linguistic rules depends on the knowledge of the control expert, but the translation of these rules into fuzzy set theory framework is not formalized and arbitrary choices concerning, for example, the shape of membership functions have to be made. The quality of fuzzy logic controller can be drastically affected by the choice of membership functions. Thus, methods for tuning fuzzy logic controllers are necessary.

Neural networks offer the possibility of solving the problem of tuning. Although a neural network is able to learn from the given data, the trained neural network is generally understood as a black box. Neither it is possible to extract structural information from the trained neural network nor can we integrate special information into the neural network in order to simplify the learning procedure. On the other hand, a fuzzy logic controller is designed to work with the structured knowledge in the form of rules and nearly everything in the fuzzy system remains highly transparent and easily interpretable. However, there exists no formal framework for the choice of various design parameters and optimization of these parameters generally is done by trial and error.

A combination of neural networks and fuzzy logic offers the possibility of solving tuning problems and design difficulties of fuzzy logic [10]. The resulting network will be more transparent and can be easily recognized in the form of fuzzy logic control rules or semantics [11]. This new approach combines the well established advantages of both the methods and avoids the drawbacks of both. In this paper, a neuro-fuzzy controller architecture is proposed, which is an improvement over the existing neuro fuzzy controllers. It overcomes the major drawbacks of the existing neuro-fuzzy approaches; of either keeping neural networks and fuzzy logic as separate entities (co-operative models) working towards a common goal or in most of the existing neuro-fuzzy approaches, the trained controller no longer can be interpreted as fuzzy logic controller. The novelty of this scheme is that the fuzzy controller itself is interpreted as a neural network. So, an error in the resulting control value can be distributed back among the control rules, instead of the integrating neural networks in certain parts of the controller, acting as black boxes to optimize the weights. One of the objective of this paper is to understand adaptation of the membership functions as a reverse mechanism deduced from the forwarding inference machinery of the fuzzy logic controller. The architecture and the learning algorithm of a proportional fuzzy controller (PFLC) is presented. A step by step algorithm for the off-line training of a PFLC is demonstrated by a numerical example.

2. THE NEURO-FUZZY CONTROLLER

We consider a multi-input, single-output dynamic system whose states at any instant can be defined by "n" variables X_1, X_2,...,X_n. The control action that derives the system to a desired state can be described by a well known concept of "if-then" rules, where input variables are first transformed into their respective linguistic variables, also called fuzzification. Then, conjunction of these rules, called inferencing process, determines the linguistic value for the output. This linguistic value of the output also called fuzzified output is then converted to a crisp value by using defuzzification scheme. All rules in this architecture are evaluated in parallel to generate the final output fuzzy set, which is then defuzzified to get the crisp output value.

The conjunction of fuzzified inputs is usually done by either min or product operation (we use product operation) and for generating the output max or sum operation is generally used. For defuzzification, we have used simplified reasoning method, also known as modified center of area method.

For simplicity, triangular fuzzy sets will be used for both input and output. The whole working and analysis of fuzzy controller is dependent on the following constraints on fuzzification, defuzzification and the knowledge base of an FLC, which give a linear approximation of most FLC implementations.

CONSTRAINT 1: The fuzzification process uses the triangular membership function.

CONSTRAINT 2: The width of a fuzzy set extends to the peak value of each adjacent fuzzy set and vice versa. The sum of the membership values over the interval between two adjacent sets will be one. Therefore, the sum of all membership values over the universe of discourse at any instant for a control variable will always be equal to one. This constraint is commonly referred to as fuzzy partitioning.

CONSTRAINT 3: The defuzzification method used is the modified center of area method. This method is similar to obtaining a weighted average of all possible output values.

An example of a very simple neuro fuzzy controller with just four rules is depicted in Figure 1. This architecture can be readily understood as a "neural-like" architecture. At the same time, it can be easily interpreted as a fuzzy logic controller. The modules X_1 and X_2 represent the input

variables that describe the state of the system to be controlled. These modules deliver crisp input values to the respective membership modules (μ-modules) which contain definitions of membership functions and basically fuzzify the input. Now, both the inputs are in the form of linguistic variables and membership associated with the respective linguistic variables. The μ-modules are further connected to R-modules which represent the rule base of the controller, also known as the knowledge base. Each μ-module gives to its connected R-modules, the membership value $\mu(x_i)$ of the input variable X_i associated with that particular linguistic variable or the input fuzzy set. The R-modules use either min-operation or product-operation to generate conjunction of their respective inputs and pass this calculated value forward to one of ν-modules. The ν-modules basically represent the output fuzzy sets or store the definition of output linguistic variables. If there are more than two rules affecting one output variable then either their sum or the max is taken and the fuzzy set is either clipped or multiplied by that resultant value. These ν-modules pass on the changed output fuzzy sets to C-module where the defuzzification process is used to get the final crisp value of the output.

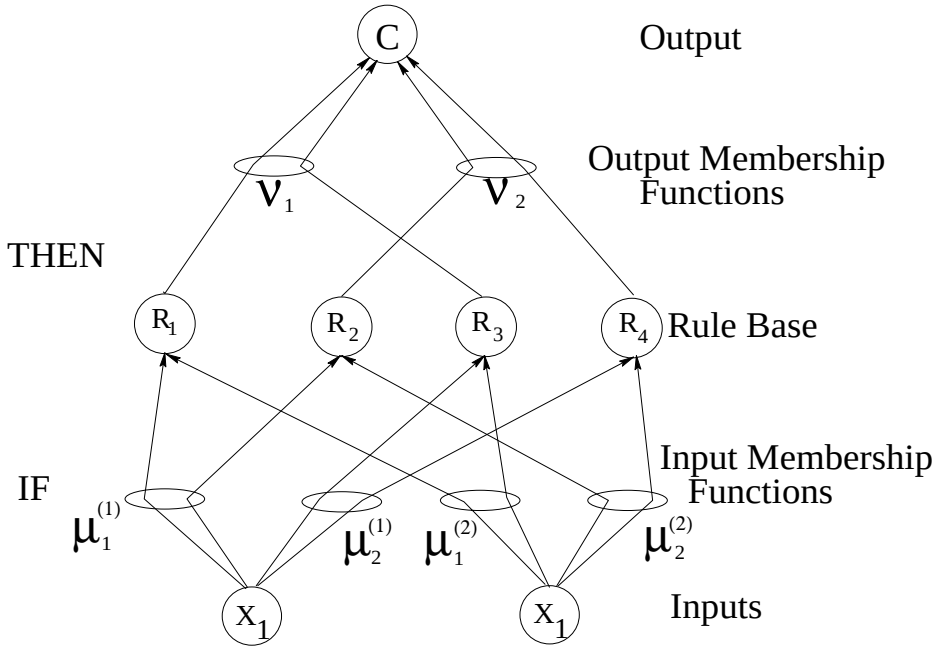

Figure 1. Architecture of four rule fuzzy controller from neural networks point of view.

The architecture given in Figure 1 of a fuzzy logic controller resembles a feedforward neural network. The X-, R-, and C-modules can be viewed as the neurons in a layered neural network and

the μ- and ν-units as the adaptable weights of the network. The X-module layer can easily be identified as the input layer of a multi-input neural network whereas the C-module layer can be seen as the output layer. The R-module layer serves as the hidden or intermediate layer that constitutes the internal representation of the network. The fact that one μ-module can be connected to more than one R-module is equivalent to the connections in a neural network that share a common weight. This is of key importance for keeping the structural integrity of the fuzzy controller intact.

3. THE LEARNING PROCEDURE

The actual output signal, C_a, is generated by the controller and now if the desired output of the controller, C_d, is also known at every instant, then error, e_c, can be generated. The goal now is to generate the adjustments of the input and output membership functions, by back- propagating the error through the "neural-like" architecture of the fuzzy controller.

The essence of back propagation algorithm in this case is to reward those rules which contribute towards taking the actual control action towards the desired control action and to discourage the rules which tend to take the control action away from the desired path.

The error, e_c, is assumed to be due to the bad choice of membership functions. Membership functions can be adjusted by laterally moving the domain or by bending the segments of the function. The error, e_c, is due to a combination of errors resulting from wrong lateral placement of the domains and from specification of function shapes. These partial errors are then distributed back to the architecture. The error due to lateral placement of domain goes on to affect the output membership function domain, whereas the error due to function shapes modifies the input membership function in order to reduce that error.

This subdivision of error is related to the position of actual output and the desired output values on the output domain. Using simplified reasoning defuzzification, there can be four different possible cases depending upon the relative position of actual and desired output values as shown in the Figure 2. It can be readily seen from simplified reasoning defuzzification scheme that the actual output is always going to lie between the μ_i and μ_{i+1} adjacent active output fuzzy sets. Now, if the error is other than zero, i.e., the actual and desired output values are different, the following four cases can arise,

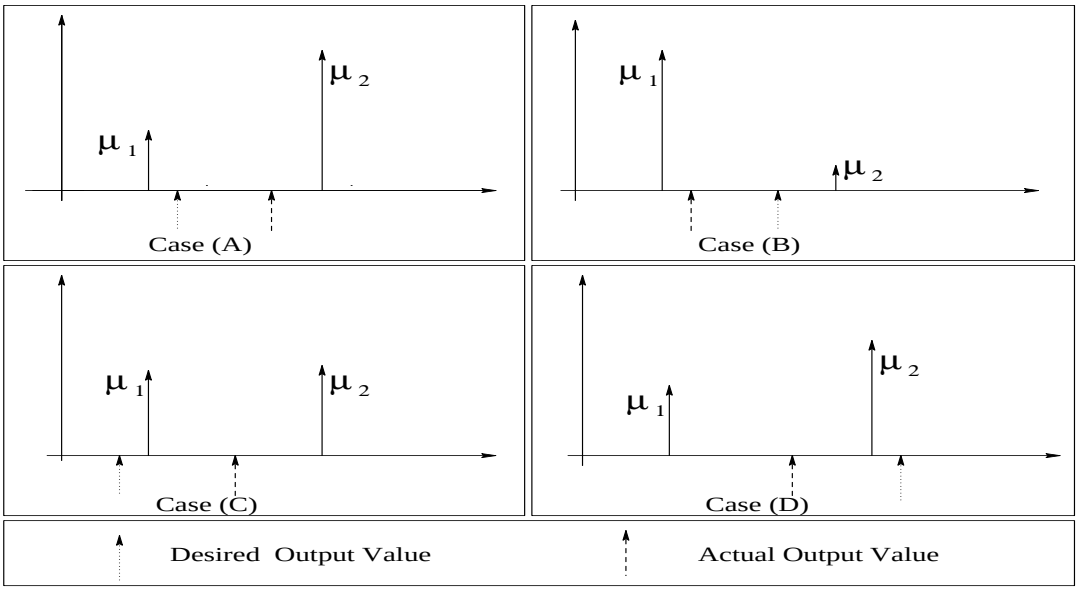

Figure 2. Four possible cases of error.

1. The desired value is located in between the center values for μ_i and μ_{i+1} functions but to the left of the actual value.
2. The desired value is located in between the center values for μ_i and μ_{i+1} functions but to the right of the actual value.
3. The desired value is not located in between the center values for μ_i and μ_{i+1} functions but lies to the left of the center of μ_i.
4. The desired value is not located in between the center values for μ_i and μ_{i+1} functions but lies to the right of the center of μ_{i+1}.

In the first two cases the output value is being produced by the correct fuzzy sets, but we need to adjust the shape of input membership functions. So, we try to change the form of active input membership sets such that error is reduced. The details of this are given in section 4. For the cases 3 and 4, since the desired output does not lie in the range of centers of adjacent active output fuzzy sets, no amount of modification of input fuzzy sets can result in error going to zero. So, in this case the problem lies with the lateral displacement of the output sets or in some cases it is due to the wrong choice of the rule base. In this paper, as will become clear later, such error is considered to be due to the wrong position of the output set and the output set is moved to cover the desired value. It is seen that if rule base is chosen with care occurrence of such errors is rare.

4. ERROR PROPAGATION

As discussed in section 3, the generated error is propagated back to the R-module and action is taken on the input membership functions or the output membership functions according to the four different possibilities of relative positions of actual and desired output value. In this section, we give a detailed step by step algorithm to propagate the error back through the controller in order to reduce the error.

Step 1: After the C-module produces the actual output, C_a, it along with the desired value, C_d, are propagated to the R-modules unit.

Step 2: Check if the desired value, C_d, lies in the range of centers of active output fuzzy μ_i and μ_{i+1}. If it is, then go to step 3 else if C_d does not lie in the range of centers fuzzy sets μ_i and μ_{i+1} then move to step 6.

Step 3: Check if desired output value C_d is greater than the actual output C_a. If it is, then go to step 4, else go to step 5.

Step 4: For a case, where $C_d > C_a$, we need to increase the influence of μ_{i+1} fuzzy set while reducing the weight of μ_i set. This will shift the weighted average towards right resulting in reduced error, i.e. we need to reward the rule which affects the μ_{i+1} and discourage the rule(s) affecting μ_i fuzzy set. This can be achieved in two ways as explained in Figure 3 by either moving μ_{i+1} input fuzzy set closer to the other one or by moving the μ_i set away from its initial position. Go to step 7.

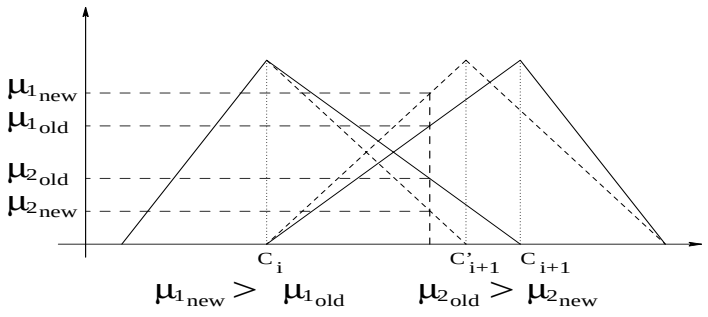

Figure 3. Adjusting the shapes of fuzzy sets to reduce error.

Step 5: For a case, where $C_d < C_a$, we need to increase the influence of μ_i fuzzy set while reducing the weight of μ_{i+1} set, which will move the weighted average towards left and thus reduce error. That means that we need to reward the rule(s) which affects the μ_i

and discourage the rule(s) affecting μ_{i+1} fuzzy set. This can be achieved in two ways as explained in Figure 4 by either moving μ_i input fuzzy set closer to the other one or by moving the μ_{i+1} set away from its initial position. Move to step 7.

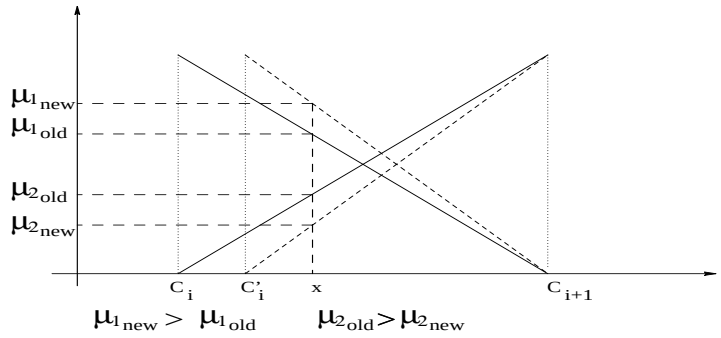

Figure 4. Adjusting the shapes of fuzzy sets in order to reduce the error

Step 6: If C_d lies outside the range of the centers of μ_i and μ_{i+1}, then we will have to move the output fuzzy sets in the appropriate direction to get the desired value to lie in the range and go on to next step.

Step 7: Get the next set of input value(s) and the desired output value and move back to step 1.

These steps are also shown in a flow chart as shown in Figure 5.

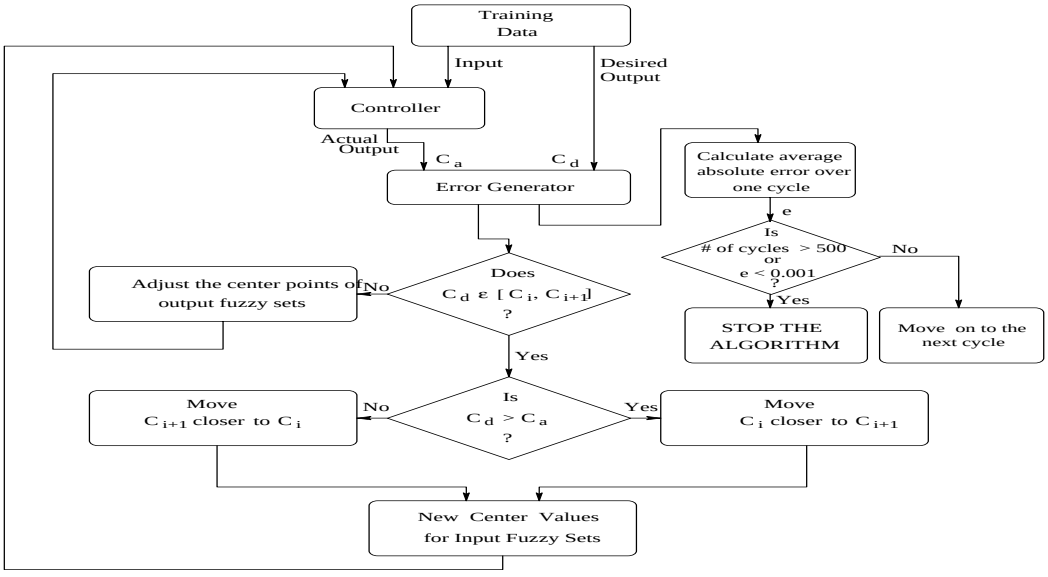

Figure 5. Flow chart of learning procedure

5. OFF-LINE TRAINING OF NEURO FUZZY CONTROLLER

To train a controller off-line for a system, the limiting condition is that either the system can be taken off-line or we should be able to model or simulate the system accurately. Further, since we are applying the supervised learning algorithm discussed earlier, the following restrictions apply,

1. Training data is available for the system to be controlled.
2. The system to be controlled can be modeled or simulated.
3. Knowledge base in terms of rules is available to the designer, however, this is not a limiting assumption since rule base can be generated by using various clustering algorithms available.

6. PROPORTIONAL NEURO FUZZY CONTROLLER

Proportional neuro fuzzy controller (PNFC) presented here is an extension of work given in [12], where proportional fuzzy logic controller (PFLC) is analyzed and a general design method is presented.

6.1 Architecture

A PFLC, which is represented by the above “if - then" rules, can be easily implemented in the form of a neural network as shown in Figure 6. There is one input node, known as an error node. The input node feeds to the intermediate or hidden layer of rules, through weighted connections. These weights are essentially the definitions of linguistic variables or the fuzzy sets. At these shared weights, the input error is converted into linguistic variables and fed to the rule modules or rule layer. Each rule module processes the information available to its input in parallel. Rule module takes the input from input layer (error) conditioned by the fuzzy weights (fuzzification) and finds the conjunction of the inputs. The output of rule module is then fed to the output layer through weighted connections. The weights are definitions of output fuzzy sets and at the output defuzzification is performed which is chosen to be the “modified center of gravity" approach.

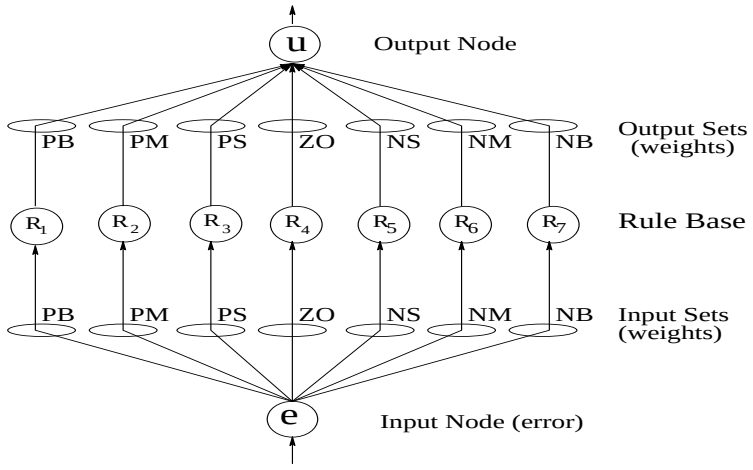

Figure 6. Architecture of a PFLC from neural networks perspective.

6.2 Training

The general algorithm described earlier is used to train the PNFC off-line. The algorithm is implemented in MATLAB using m-files. To use the algorithm, first the training data was generated. The generation of training data is dependent on the type of application. For example, to design a feedforward controller for trajectory control, we basically need to design the inverse of the system. In this case, the data is generated from either the original plant (if it can be taken off-line) or from the simulation. However, if the application is to replace the existing controller, we need the input-output data of the controller.

Further, the heuristic knowledge about the system in terms of "if-then" rules should be available. Then, the training data is used to generate the output and the generated output is compared with the desired output. If there is a mismatch between the desired output and the actual generated output, an error is generated and this error is propagated through the "neural-like" network. The back-propagation algorithm of error is based on the philosophy of rewarding the rules, which bring the actual control action towards the desired control action, and punishing the rules that take the actual output away from the desired path. The rewarding or punishing is achieved by making the corresponding fuzzy sets more or less sensitive, respectively, which is done by changing the shapes of the sets.

The error generated is assumed to be made up of two parts; error due to the shape of the fuzzy sets and error due to the lateral placement of the sets. So, after the error generation, it is divided into these two parts which then are propagated back to the rule modules. Rule modules

decide which rules to punish and which one to reward by varying either the shape of input sets or by moving the domain of output sets in order to reduce the error.

7. SIMULATION RESULTS

Neuro fuzzy controller can basically learn any static input-output characteristics if the training data is available. This means that the learning algorithm can produce a neuro fuzzy controller which can copy the control surface of an existing controller if the input-output data from the controller is known. There are many reasons why we may want to replace an existing controller and one of the reason is that a proportional controller in the fuzzy logic domain has been shown to have the characteristics of a proportional plus derivative controller in the classical domain [12, 13]. Also, we can replace a PD or PID controller with a simple proportional fuzzy logic controller.

7.1 Example 1

The block diagram of a closed-loop system is shown in Figure 7. Let us assume that the plant is a second-order type-zero system with a transfer function given as

$$G(s) = \frac{75}{s^2 + 20s + 75}$$

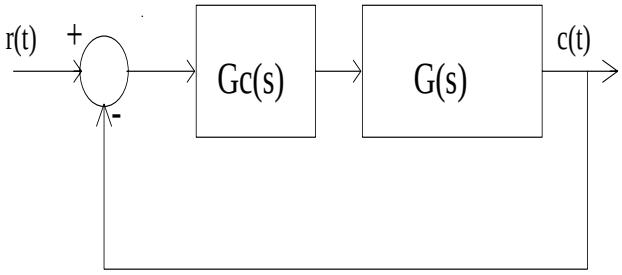

Figure 7. Block diagram of a closed-loop control system.

The existing system has a PD controller, with a transfer function $G_c(s) = 4.0 + 0.2s$, in the feedback path. The goal is to replace the existing PD controller with a PNFC without affecting the performance of the system. The training data is generated by running the simulation for the above system and recording the input as well as the output of the controller. This data is then used in the learning algorithm to generate a PNFC with same input-output characteristics as the PD controller. The results of training algorithm in terms of generated values of center points of input and output fuzzy sets are given in Table I.

Table I. Rule base for controller in example 1.

Error	-1	-.67	-.33	0	.05	.21	.26
Output	-1	-.67	-.33	0	.20	.56	1.0

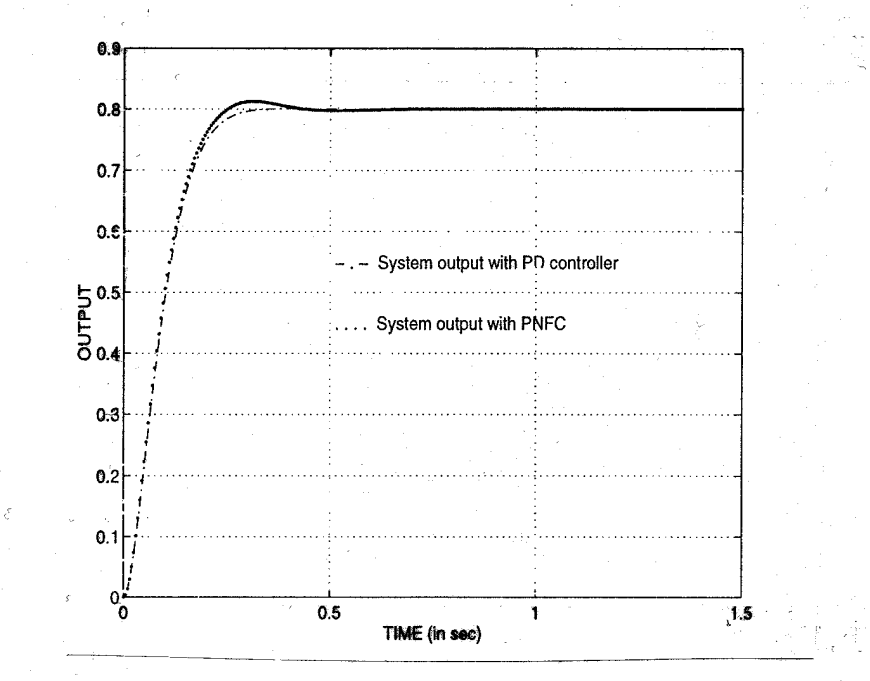

Figure 8. Simulation results for example 1.

The comparison of the results (Figure 8) obtained from the PNFC with those obtained from a classical PD controller show that the algorithm was able to duplicate the input-output characteristics of the existing controller successfully.

7.2 Example 2

Let us take the case of second-order type-one system with a transfer function given as, $G(s) = 7/s^2+7s$. The existing system has a classical PD controller, given as, $G_c(s) = 4.0 + 0.2s$ in the feedback path. The goal is to replace the existing PD controller with a PNFC without affecting the performance of the system. The training data is generated by running the simulation of the above system and recording the input as well as the output of the controller. This data is then used in the learning algorithm to generate a PNFC with the same input-output characteristics as the classical PD controller. The results of the training algorithm in terms of generated values of center points of input and output fuzzy sets are given in Table II.

Table II. Rule base for controller in example 2.

Error	-1	-.67	-.33	0	.09	.17	.26
Output	-1	-.67	-.33	0	.25	.57	1.0

The comparison of the results obtained from the PNFC with those obtained from the existing controller, as shown in Figure 9, suggest that algorithm was able to copy the input output characteristics of the existing controller successfully.

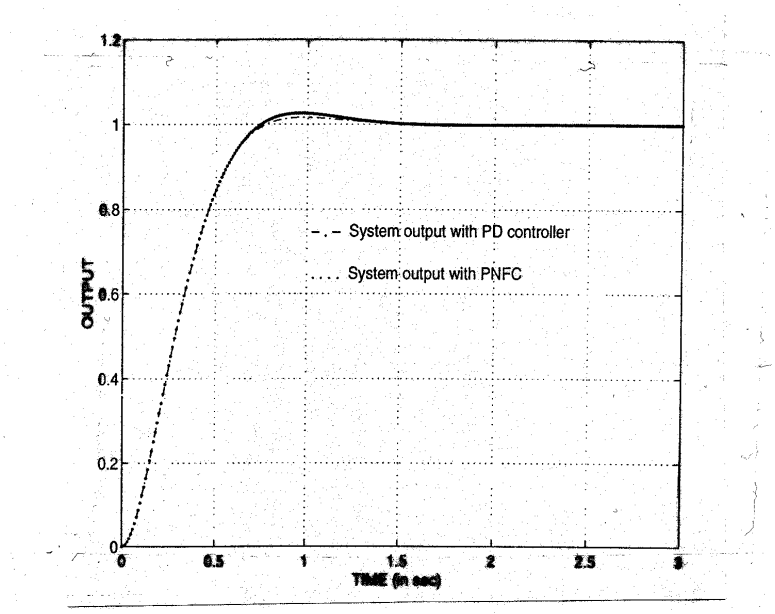

Figure 9. Simulation results for example 2.

8. CONCLUSION

Classical control theory is based on mathematical models that describe the system under consideration. The underlying principle of fuzzy control is to build a model of a human expert who is capable of controlling the plant without thinking in terms of a mathematical model. The control expert specifies the control actions in the form of linguistic rules, generated from heuristic knowledge of the system. The specification of good linguistic rules depends on the heuristic knowledge of control expert. However, the translation of these linguistic rules into fuzzy sets theory framework is not formalized and arbitrary choices concerning, for example, the shape of the membership functions have to be made. The quality of fuzzy logic controller can be drastically affected by the choice of membership functions. Thus, methods for tuning fuzzy logic controllers are necessary.

A combination of neural networks and fuzzy logic offers the possibility of solving the tuning problems and design difficulties of fuzzy logic. In this paper, an innovative approach of hybrid

neuro fuzzy controller is presented which lends neural networks learning capabilities to the fuzzy logic controllers. A generic learning algorithm is presented for learning and training. The generic learning algorithm for hybrid neuro fuzzy controller is used for off-line training of proportional neuro fuzzy controller and is tested with various simulations.

REFERENCES

[1] K. Asakawa and H. Takagi, "Neural Networks in Japan," Communication of the ACM, Vol. 37, No. 3, 1994, pp. 106-112.

[2] C. von Altrock, B. Krause, and H. J. Zimmerman, "Advanced fuzzy logic control technologies in automotive applications," Proc. IEEE Int. Conf. Of Fuzzy Systems, San Diego, 1992, pp. 835-842.

[3] S. Shao, "Fuzzy self-organizing controller and its application for dynamic processes," Fuzzy Sets and Systems, Vol. 26, 1988, pp. 151-164.

[4] H. Takagi, "Application of neural networks and fuzzy logic to consumer products," Proc. Int. Conf. On Industrial Fuzzy Electronics, Control, Instrumentation, and Automation, Vol. 3, San Diego, Nov. 1992, pp. 1629-1639.

[5] T. Culliere, A. Titli, and J. Corrieu, "Neuro-fuzzy modeling of nonlinear systems for control purposes," Proc. IEEE Int. Conf. On Fuzzy Systems, Yokohama, 1995, pp. 2009-2016.

[6] N. Bridgett, J. Brandt, and C. Harris, "A neurofuzzy route to breast cancer diagnosis and treatment," Proc. IEEE Int. Conf. On Fuzzy Systems, Yokohama, 1995, pp. 641-648.

[7] T. Chen, "Fuzzy neural network applications in medicine," Proc. IEEE Int. Conf. On Fuzzy Systems, Yokohama, 1995, pp. 627-634.

[8] R. Kruse, J. Gebhardt, and R. Palm, editors, Fuzzy Systems in Computer Science, Vieweg, Braunschweig, 1994.

[9] J. Hollatz, "Neuro-fuzzy in legal reasoning," Proc. IEEE Int. Conf. On Fuzzy Systems, Yokohama, 1995, pp. 655-662.

[10] P. J. Werbos, "Neurocontrol and fuzzy logic: connections and design," Int. J. Approximate Reasoning, Vol. 6, Feb. 1992, pp. 185-220.

[11] D. Nauck, F. Klawonn, and R. Kruse, "Combining neural networks and fuzzy controllers," In E. P. Klement and W. Slany, editors, Fuzzy Logic in Artificial Intelligence, Springer-Verlag, Berlin, 1993, pp. 35-46.

[12] K. S. Rattan, T. Brehm, and G. S. Sandhu, "Analysis and design of proportional fuzzy controller," Proc. Sixth International Fuzzy Systems and Intelligent Control Conference, Hawaii, 1996.

[13] K. S. Rattan and G. S. Sandhu, "Analysis and design of proportional plus derivative fuzzy logic controller, Proc. National Aeronautics and Electronics Conference, Dayton, 1996.