CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

In the last, decade considerable research efforts have been spent on seeking systematic approaches to fault diagnosis on process control system. This dissertation analyzes the performance and practical implementation of Fuzzy Neural Network for intelligent process control system. The main goal of intelligent process control systems is the monitoring of the system during its normal working condition so as to detect the occurrences of failures (fault detection), recognize the location (fault isolation), the time evolution and indication (fault identification) hence suggests possible rectification procedures (fault rectification). Every intelligent technique has computational properties (e.g. ability to learn explanation of decisions) that make them suited for particular problems and not for others. Intelligent process control systems are widely used in modern industrial applications due to their reliability, low cost and high performance. Detection and diagnosis of systems faults are very essential for protection of such control systems against failures and permanent damage. In recent years, monitoring and fault detection in control systems have moved from traditional methods to artificial intelligence techniques (Mohamed, 2014) and (Chin-Teng and Lee, 1991).

Neural network are good at recognizing patterns, they are not good at explaining how they reach their decisions (Ding, 1992). Fuzzy logic systems can reason with imprecise information and are good at explaining their decisions but they cannot automatically acquire the rules they use to make those decisions. These limitations have been a central driving force behind the creation of intelligent hybrid systems where two or more techniques are combined in a manner that overcomes the limitations of individual techniques. Neural networks are used to tune membership functions of fuzzy systems that are employed as decision making systems for controlling equipment (De Bollivier and Gallinani,1990). Although fuzzy logic can encode expert knowledge directly using rules with linguistic labels, it usually takes a lot of time to design and tune the membership functions which quantitatively define these linguistic labels. Neural Network learning techniques can automate the process and substantially reduce development time and cost while improving performance (Abraham and Nath, 2001).

1.2 Statement of the Problem

Rigorous effort is required to obtain an appropriate transfer function for use in classical PID control systems. Also the computation of the constants K_p, K_i, K_d (constants of proportion) for a PID control system is difficult even for someone who is familiar with the system that is being modeled.

A new approach that would allow someone who is experienced in a system to model a control system without getting involved in any of the mathematical derivations highlighted above is required. This is the niche which the fuzzy logic based control is expected to fill. Often the result realized through the use of fuzzy logic may be coarse (not exact) for most control systems. It is hoped that neural network can be used to optimize (or fine–tune) the fuzzy process output thereby leading to the emergence of a Fuzzy Neural process control system. This is the direction this research work seeks to follow.

1.3 Aim and Objectives

The aim of this dissertation is to design Intelligent Process Control systems using Fuzzy Neural Network (FNN).

To achieve this aim the following are the objectives.

1. To design a process control system in PID mode using Fuzzy Logic.
2. To optimize the performance of the Fuzzy control system using Neural Network.
3. To simulate a prototype Fuzzy Neural process control system in PID mode.
4. To evaluate the performance of the Fuzzy Logic, Neural Network, Fuzzy Neural Network and classical PID control systems using the results from the simulation.

5. To automate the fault diagnosis in Fuzzy Neural logic based process control system.

1.4 Justification for the Project

Components, machines and processes fail in varying ways depending upon their constituent materials, operating conditions, etc. Failure modes are typically monitored by a sensor which is intended for failure analysis purpose, to capture those failure symptoms that are characteristics of a particular failure mode. Using fuzzy neural network to analyze this is well suited to low cost implementation based on cheap sensors, low resolution analog to digital converters, and 4-bit or 8-bit one chip microcontroller chips (Gupta, 1992).

Moreover such systems can be easily upgraded by adding new rules to improve performance or add new features. Also machine operators will have an idea of the fault and the rectification procedure (Huifang, Ren, Jizhu, and Benxian, 2011).

1.5 Scope of the Dissertation

Intelligent process control systems using fuzzy-neural logic has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experiences can be used in the design of process controllers. This makes it easier to mechanize tasks that are already successfully performed by humans, hence assisting to improve substantially production cycle. The thesis will cover the followings:

a) Design of intelligent controllers based on Fuzzy Logic, Neural network, and hybrid of the two (Fuzzy Neural).
b) The application of the designed controllers in terms of transient and steady state response of the process.
c) Evaluation of the performance of the intelligent controllers in terms of transient and steady state response of the process.
d) Comparing the result of the controller with a conventional PID controller.

1.6 Organization of Dissertation Chapters

In this work, Chapter introduced the dissertation topic, Chapter two handled the literature review, Chapter three discussed the methodology and system analysis, Chapter four covered the system design, Chapter five explained the System implementation and testing while finally Chapter six deals on Summary and Conclusion.

1.7 Proportional Integral Derivative (PID) Controller

PID is a popular control method extensively used in an industrial set up. PID controllers are popular in industrial applications, as they are easy to install and reasonably robust. However, for highly nonlinear systems, the performance of PID controllers can deteriorate quite fast. It is necessary to develop nonlinear PID controllers for controlling nonlinear processes.

The advantages of a PID controller include its simple structure along with robust performance in a wide range of operating conditions. The design

of a PID controller is generally based on the assumption of exact knowledge about the system. The assumption is often not valid since the development of model of any practical system may not include precise information of factors such as friction, backlash, unmodelled dynamics and uncertainty arising from any of the sources (Prabakaran, Kannan, Thirupathi and Hari Prakash, 2014).

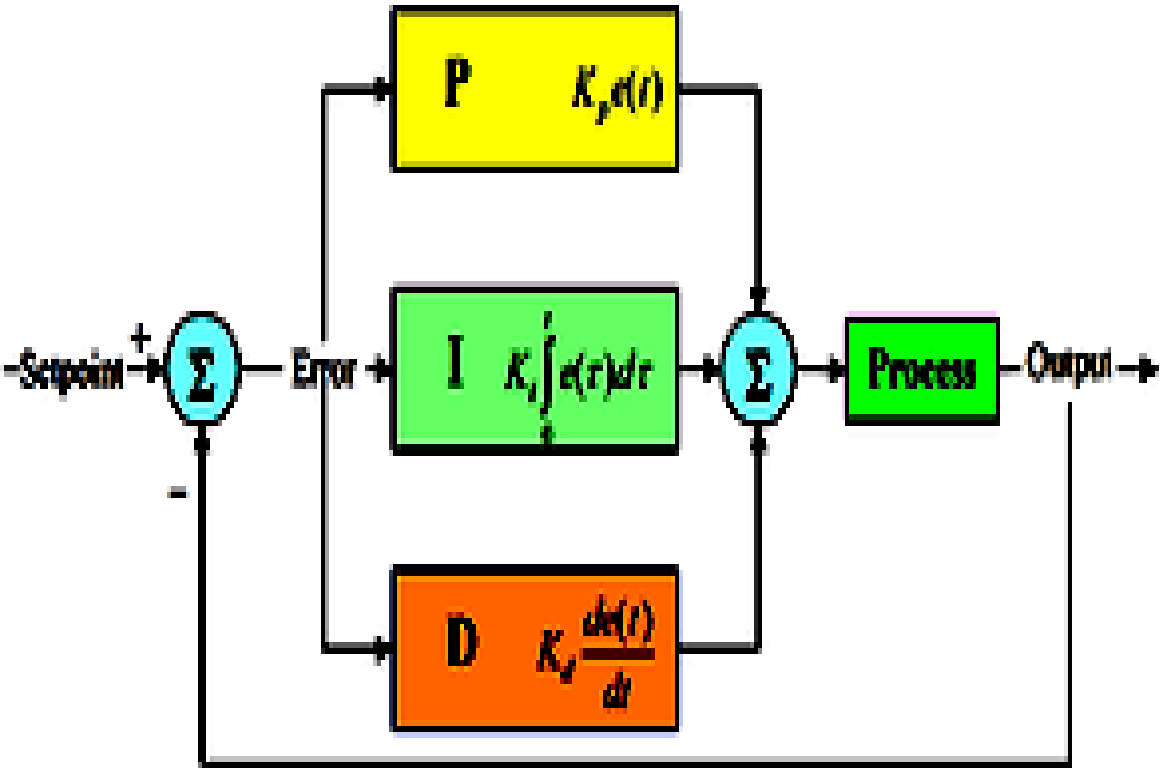


Figure 1.1: Block diagram of a PID controller

A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. A block diagram of a PID controller is shown in figure 1.1. It is the most commonly used feedback controller. A PID controller

calculates an "error" value as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the process control inputs (Onkar, 2010).

It was shown in (Ang, Chong and Li, 2005) that the PID controller calculation (algorithm) involves three separate constant parameters, and is accordingly sometimes called three-term control: the proportional, the integral and derivative values, denoted *P*, *I*, and *D* respectively. Heuristically, these values can be interpreted in terms of time: *P* depends on the *present* error, *I* on the accumulation of *past* errors, and *D* is a prediction of *future* errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, or the power supplied to a heating element.

In the absence of knowledge of the underlying process, a PID controller is the best (Rakesh Gautam, Rajesh Ingle, Milin Nagpure, 2014). By tuning the three parameters in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation.

Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability (Jinghua, 2006).

Some applications may require using only one or two actions to provide the appropriate system control. This is achieved by setting the other parameters to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral term may prevent the system from reaching its target value due to the control action.

1.7.1 Control Loop Basics

A familiar example of a control loop is the action taken when adjusting hot and cold valves to maintain the water at a desired temperature. This typically involves the mixing of two process streams, the hot and cold water. The person touches the water to sense or measure its temperature. Based on this feedback they perform a control action to adjust the hot and cold water valves until the process temperature stabilizes at the desired value. The sensed water temperature is the process variable or process value (PV). The desired temperature is called the set point (SP). The input to the process (the water valve position) is called the manipulated variable (MV). The difference between the temperature measurement and the set point is the error (e) and quantifies whether the water is too hot or too cold and by how much.

After measuring the temperature (PV), and then calculating the error, the controller decides when to change the tap position (MV) and by how much. When the controller first turns the valve on, it may turn the hot valve only slightly if warm water is desired, or it may open the valve all the way if very hot water is desired. This is an example of a simple proportional control. In the event that hot water does not arrive quickly, the controller may try to speed-up the process by opening up the hot water valve more-and-more as time goes by. This is an example of an integral control. Making a change that is too large when the error is small is equivalent to a high gain controller and will lead to overshoot. If the controller were to repeatedly make changes that were too large and repeatedly overshoot the target, the output would oscillate around the set point in a constant, growing, or decaying sinusoid. If the oscillations increase with time then the system is unstable, whereas if they decrease the system is stable. If the oscillations remain at a constant magnitude the system is marginally stable. In the interest of achieving a gradual convergence at the desired temperature (SP), the controller may wish to damp the anticipated future oscillations. So in order to compensate for this effect, the controller may elect to temper its adjustments. This can be thought of as a derivative control method. If a controller starts from a stable state at zero error (PV = SP), then further changes by the controller will be in response to changes in other measured or unmeasured inputs to

the process that impact on the process, and hence on the PV. Variables that impact on the process other than the MV are known as disturbances. Generally controllers are used to reject disturbances and/or implement set-point changes. Changes in feed water temperature constitute a disturbance to the faucet temperature control process.

In theory, a controller can be used to control any process which has a measurable output (PV), a known ideal value for that output (SP) and an input to the process (MV) that will affect the relevant PV. Controllers are used in industry to regulate temperature, pressure, flow rate, chemical composition, speed and practically every other variable for which a measurement exists (King, 2010).

1.7.2 PID Controller Theory

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV) which is a function of time. The proportional, integral, and derivative terms are summed to calculate the output of the PID controller (Kim, Hong and Park, 2002). Defining *u* (*t*) as the controller output, the final form of the PID algorithm is:

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t) \qquad (1.1)$$

where

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

1.7.3 Proportional Term

The proportional term makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain. Figure 1.2 shows the graph.

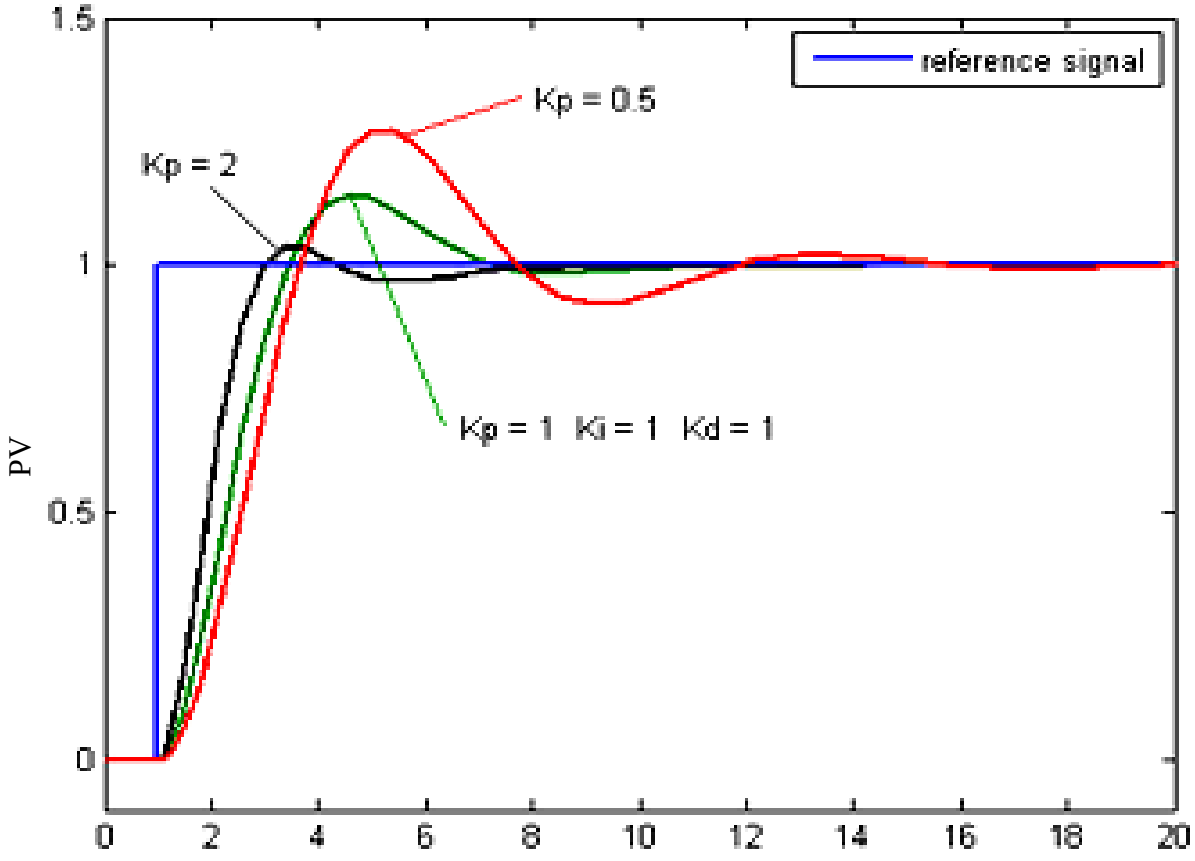


Figure 1.2: Plot of PV vs time, for three values of K_p (K_i and K_d held constant)

The proportional term is given by:

$$P_{\text{out}} = K_p\, e(t) \qquad (1.2)$$

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable (see loop tuning). In contrast, a small gain results in a small output response to a large input error, and a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. Tuning theory and industrial practice indicate that the proportional term should contribute the bulk of the output change (Jinghua, 2006).

1.7.4 Droop

A pure proportional controller will not always settle at its target value, but may retain a steady-state error. Specifically, drift in the absence of control, such as cooling of a furnace towards room temperature, biases a pure proportional controller. If the drift is downwards, as in cooling, then the bias will be below the set point, hence the term "droop".

Droop is proportional to the process gain and inversely proportional to proportional gain (Kim, 2002). Specifically the steady-state error is given by:

$$e = G / K_p \qquad (1.3)$$

Droop is an inherent defect of purely proportional control. Droop may be mitigated by adding a compensating bias term (setting the set point above the true desired value), or corrected by adding an integral term.

1.7.5 Integral Term

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain (K_i) and added to the controller output. The integral term is given by:

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau \tag{1.4}$$

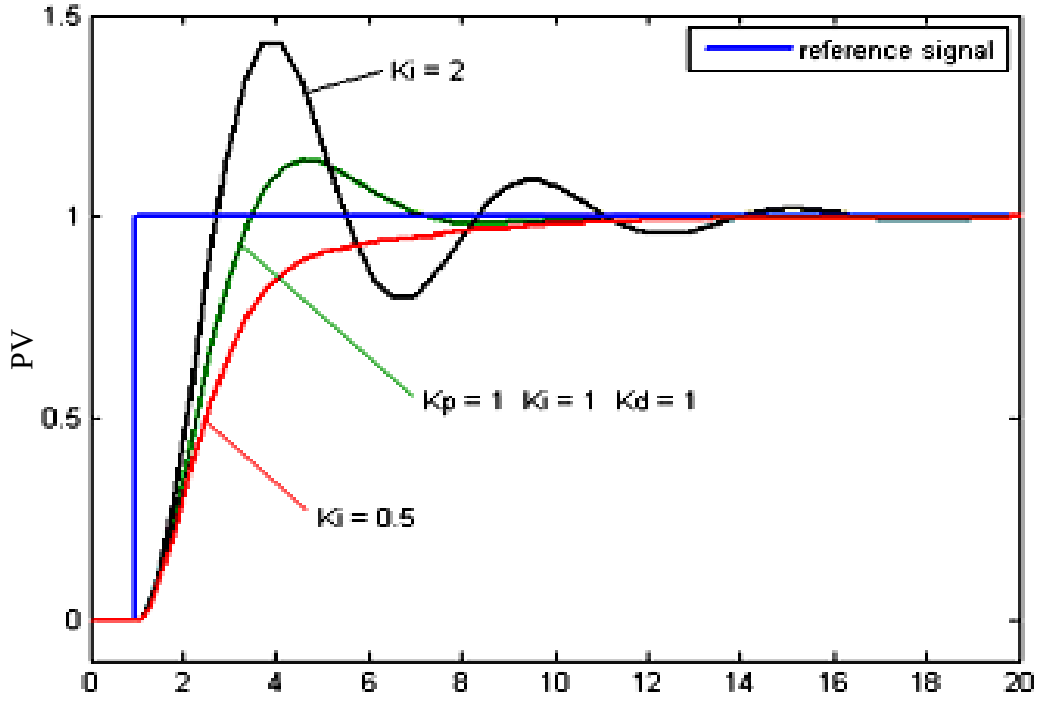


Figure 1.3: Plot of PV vs time, for three values of K_i (K_p and K_d held constant)

The integral term accelerates the movement of the process towards set-point and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the set-point value. Figure 1.3 shows the Plot of PV vs time, for three values of K_i (K_p and K_d held constant) for integral term (Kim etal 2002).

1.7.6 Derivative Term

The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain, K_d.

The derivative term is given by:

$$D_{\text{out}} = K_d \frac{d}{dt} e(t) \tag{1.5}$$

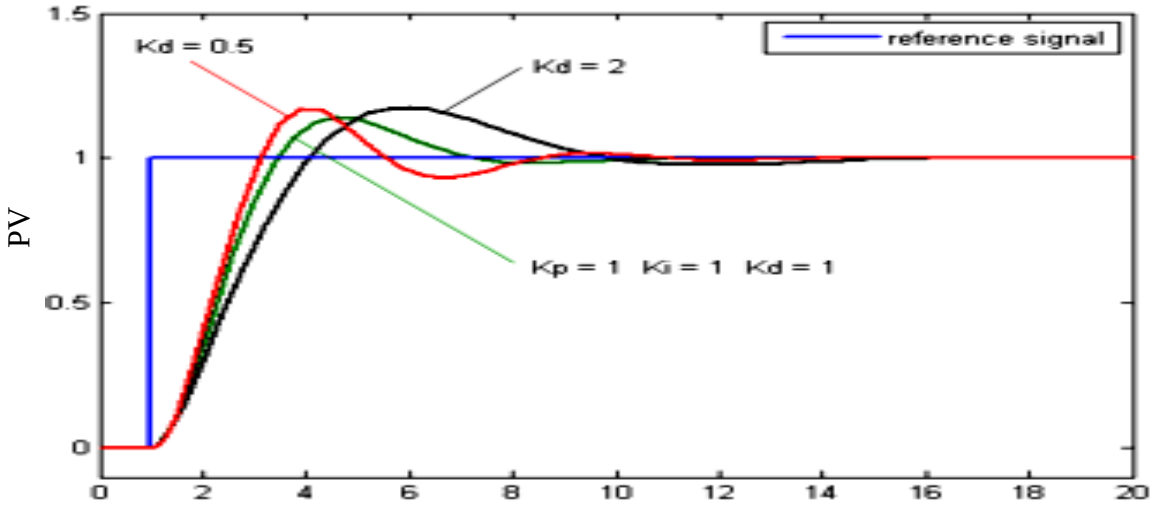


Figure 1.4: Plot of PV vs time, for three values of K_d (K_p and K_i held constant)

The derivative term slows the rate of change of the controller output. Derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, the derivative term slows the transient response of the controller. Also, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large. Hence an approximation to a differentiator with a limited bandwidth is more commonly used. Such a circuit is known as a phase-lead compensator.

(Kim and Cho, 2004)

Figure 1.4 above shows Plot of PV vs time, for three values of K_d (K_p and K_i held constant).

1.7.7 Loop Tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another.

PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy

complex criteria within the limitations of PID control. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents; this section describes some traditional manual methods for loop tuning. (Kim, 2004)

Designing and tuning a PID controller appears to be conceptually intuitive, but can be hard in practice, if multiple (and often conflicting) objectives such as short transient and high stability are to be achieved. Usually, initial designs need to be adjusted repeatedly through computer simulations until the closed-loop system performs or compromises as desired. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling (using different parameters in different operating regions). PID controllers often provide acceptable control using default tunings, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning (Kim, 2004) and (King, 2010).

1.7.8 Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag.

Generally, stability of response is required and the process must not oscillate for any combination of process conditions and set-points, though sometimes marginal stability (bounded oscillation) is acceptable or desired (Tan, Wang and Hang Chang, 1999).

1.7.9 Optimum Behavior

The optimum behavior of a process change or set -point change varies depending on the application. Two basic requirements are regulation (disturbance rejection –staying at a given set point) and command tracking (implementing set point changes), these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the set point if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new set point (Tan et al 1999).

For a PID controller with K_p=5, K =0.7s^{-1}, K_D=0.5s, and $P_I(0)$=20% the error will produce a graph as shown in figure 1.5 .

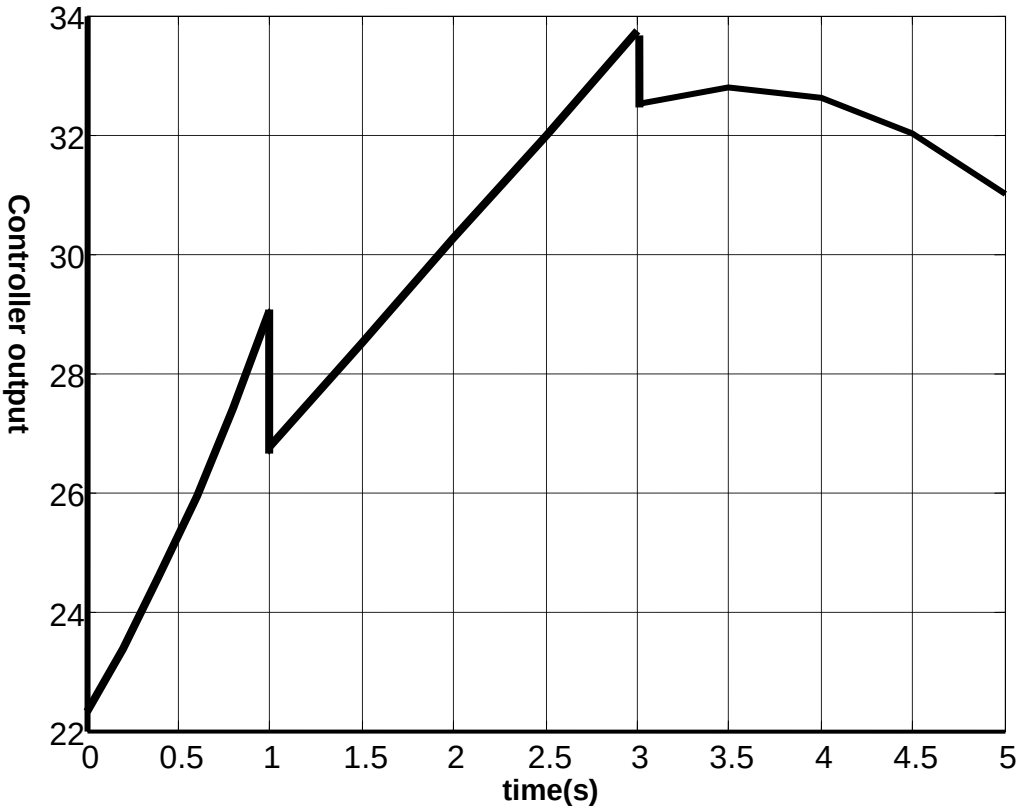


Figure 1.5: Graph of classical PID

CHAPTER TWO

LITERATURE REVIEW

The history of process control can be traced back when people managed bronze and iron producing furnaces by manual practices which, after the First World War, were gradually supplemented by automatic regulatory control of temperatures, levels, pressures, and flow rates. This relieved the process operator of some unsuitable and boring tasks (Antsaklis, 2012).

The first feedback device on record was the water clock invented by the Greek Ktesibios in Alexandria Egypt around the 3rd century Before Christ (B.C).This was certainly a successful device as water clocks of similar design were still being made in Baghdad when the Mongols captured the city in 1258 After Death (A.D). The first mathematical model to describe plant behavior for control purposes is attributed to James Clark Maxwell, of the Maxwell equations' fame, who in 1868 used differential equations to explain instability problems encountered with James Watt's flyball governor. The first known automatic control system, the Flyball governor, was installed on Watts' steam engine over 229 years ago in 1775, the governor was introduced to regulate the speed of steam engine vehicles. (Fu,1970). When J.C. Maxwell used mathematical modeling and methods to explain instability problems encountered with James Watt's flyball governor, it demonstrated the importance and usefulness of

mathematical models and methods in understanding complex phenomena and signaled the beginning of mathematical system and control theory. It also signaled the end of the era of intuitive invention. Control theory made significant strides in the past 120 years, with the use of frequency domain methods and Laplace transforms in the 1930s and 1940s and the development of optimal control methods and state space analysis in the 1950s and 1960s, followed by progress in stochastic, robust, adaptive and nonlinear control methods in the 1960s to today, have made it possible to control more accurately significantly more complex dynamical systems than the original flyball governor (Fu, 1970).

Conventional control systems are designed today using mathematical models of physical systems. A mathematical model, which captures the dynamical behavior of interest, is chosen and then control design techniques are applied, aided by Computer Aided Design (CAD) packages, to design the mathematical model of an appropriate controller. The controller is then realized via hardware or software and is used to control the physical system. The procedure may take several iterations. The mathematical model of the system must be "simple enough" so that it can be analyzed with available mathematical techniques, and "accurate enough" to describe the important aspects of the relevant dynamical behavior. It approximates the behavior of a plant in the neighborhood of an operating point.

The control methods and the underlying mathematical theory were developed to meet the ever increasing control needs of our technology. The need to achieve the demanding control specifications for increasing complex dynamical systems has been addressed by using more complex mathematical models such as non linear and stochastic ones, and by developing more sophisticated design algorithm for, say, optimal control. The use of highly complex mathematical models however, can seriously inhibit our ability to develop control algorithms. Fortunately, simpler plant models like linear models, can be used in control design, this is possible because of the feedback used in control which cat tolerate significant model uncertainties. When the fixed feedback controllers are not adequate, then adaptive controllers are used. Controllers can then be designed to meet the specifications around an operating point, where the linear model is valid and then via a scheduler a controller emerges which can accomplish the control objectives over the whole operating range. This is, for example, the method typically used for aircraft flight control and it is a method to design fixed controllers for certain classes of nonlinear systems. Adaptive control in conventional control theory has a specific and rather narrow meaning. In particular it typically refers to adapting to variations in the constant coefficients in the equations describing the linear plant, these new coefficient values are identified and then used, directly or indirectly, to reassign the values of the constant

coefficients in the equation describing the linear controller. Adaptive controllers provide for wider operating ranges than the fixed controllers and so conventional adaptive control systems can be considered to have higher degrees of autonomy than control systems employing fixed feedback controllers (Sklansky, 1966)

There are cases where there is need to significantly increase the operating range of the system. There is need to deal effectively with significant uncertainties in models of increasingly complex dynamical systems in addition to increasing the validity range of our control methods. The need to cope with significant unmodelled and unanticipated changes in the plant, in the environment and in the control objective arises. This will involve the use of intelligent decision making processes to generate control actions so that certain performance level is maintained even though there are drastic changes in the operating conditions. However, it is quite clear that in the control systems there are requirements today that cannot be successfully addressed with the existing conventional control theory and thence is the introduction of Intelligent Process Control System (IPCS) (Saridis and Valavanis, 1988).

Intelligent control describes the discipline where control methods are developed that attempt to emulate important characteristics of human intelligence. These characteristics include adaptation and learning, planning under large uncertainty and coping with large amounts of data.

Intelligent control is interdisciplinary as it combines and extends theories and methods from areas such as control, computer science and operation research, It uses theories from mathematics and seeks inspiration and ideas from biological systems. In control engineering, a state-space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations. "State space" refers to the space whose axes are the state variables (Kataria, 2008).

Intelligent control methodologies are being applied to robotics (Vijay, Popat, Khot Sachin and Burle, 2014) and automation, communication, manufacturing, traffic control to mention but a few application areas (Antsaklis, 2012). Neural networks, fuzzy control, genetic algorithms, petri-nets (Sijimol, Pooja and Soji, 2014) and planning systems, expert systems, and hybrid systems are all intelligent control methodologies. The areas of computer science and in particular artificial intelligence provide knowledge representation ideas, methodologies and tools such as semantic networks, frames, reasoning techniques and computer languages. Concepts and algorithms developed in the areas of adaptive control and machine learning help intelligent controllers to adapt and learn. Advances in sensors, actuators, computation technology and communication networks help provide what is necessary for the implementation of intelligent control hardware (Antsaklis, 2012).

The fact is that there are problems of control today that cannot be formulated and studied in the conventional differential or difference equation mathematical framework using conventional control. Intelligent control attempts to build upon and enhance the conventional control methodologies to solve new challenging control problems (Antsaklis and Passino, 2013). The intelligent control methodology used in this work is hybrid network.

2.1 Previous Works

Dorf and Bishop, (2004) stated that robust control is a branch of control theory that explicitly deals with uncertainty in its approach to controller design. Robust control methods are designed to function properly so long as uncertain parameters or disturbances are within some (typically compact) set. Robust methods aim to achieve robust performance and/or stability in the presence of bounded modeling errors. The early methods of Bode and others were fairly robust; the state-space methods invented in the 1960s and 1970s were sometimes found to lack robustness, prompting research to improve them. This was the start of the theory of Robust Control, which took shape in the 1980s and 1990s and is still active today. In contrast with an adaptive control policy, a robust control policy is static rather than adapting to measurements of variations, the controller is designed to work with the assumption that certain variables will be unknown.

Ellis, (2012) stated that adaptive control is the control method used by a controller which must adapt to a controlled system with parameters which vary, or are initially uncertain. For example, as an aircraft flies, its mass slowly decrease as a result of fuel consumption, a control law is needed that adapts itself to such changing conditions. Adaptive control is different from robust control in that it does not need *a* priori information about the bounds on these uncertain or time-varying parameters; robust control guarantees that if the changes are within given bounds the control law need not be changed, while adaptive control is concerned with control law changing them.

Kataria and Sons, (2008) showed that automatic control as the application of control theory for regulation of processes without direct human intervention. An automatic control system is a preset closed-loop control system that requires no operator action. In the simplest type of an automatic control loop, a controller compares a measured value of a process with a desired set value, and processes the resulting error signal to change some input to the process, in such a way that the process stays at its set point despite disturbances. This closed-loop control is an application of negative feedback to a system. The mathematical basis of control theory was begun in the 18th century, and advanced rapidly in the 20th.Designing a system with features of automatic control generally requires the feeding of electrical or mechanical energy to enhance the

dynamic features of an otherwise sluggish or variant, even errant system. The control is applied by regulating the energy feed. An automatic control system has two process variables associated with it: a controlled variable and a manipulated variable. A controlled variable is the process variable that is maintained at a specified value or within a specified range.

Dubrova, E (2013) stated in fault tolerant control that fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naïvely designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems. A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. The term is most commonly used to describe computer systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software. An example in another field is a motor vehicle designed so it will continue to be drivable if one of the tires is puncture (Menychtas

and Konstanteli, 2012). A structure should be able to retain its integrity in the presence of damage caused by fatigue, corrosion, manufacturing flaws, or impact.

Lee, Yu, and Chien, (2011) in Adaptive Neural Network Controller Design for a Class of Nonlinear Systems Using Simultaneous Perturbation Stochastic Approximation (SPSA) Algorithm proposed a novel SPSA-based on-line adaptive decoupled control scheme by using PID neural network for a class of nonlinear systems. The update laws of parameters with adaptive optimal learning rate were proposed based on the Lyapunov stability theorem and this guarantees the stability and performance of closed-loop system. The proposed approach was applied in the Translational Oscillations Rotational Actuator (TORA) system and the experimental results realized by Digital Signal Processing (DSP) demonstrated its performance and efficiency. This did not eliminate the rigorous mathematical approach for solving non linear PID controllers. (Abu-Mustaf, 1993).

Seema, Mitra and Vijay (2007) designed a Neural Network Tuned fuzzy controller for Multiple Input Multiple output (MIMO) system featuring a neural network based tuned fuzzy controller for controlling the degree of freedom for MIMO systems. The coupling effect of the system was added into the main fuzzy controller for each step to improve the performance. A data set generated was partitioned into a set of clusters based on

subtractive clustering method. A fuzzy IF-THEN rule was then extracted from each cluster to form a fuzzy rule base from which the Fuzzy Neural Network is designed. The Neural Network designed contains only three layers and the hybrid learning algorithm was used to refine the parameters on fuzzy rule base. There was improvement in performance no consideration for a PID controller and the fault diagnosis mechanism of the system was considered (Aleksander (1989)

Lobbrecht and Solomatine (1999) applied intelligent control schemes for reproducing optimal control actions for polder water level. The Artificial Neural Network and Fuzzy Adaptive System reproduced the corresponding control actions with high accuracy. Nevertheless, the adaptive models performances were dependent on choice of training data set so that the two different data sets results in different levels of model performance. Inclusion of variables resulted from the extreme events and also the use of moving average and moving sum values into the training pattern improves the models performance. Artificial Neural Network and Fuzzy Adaptive System exhibited properties of adaptation during training but once trained on the basis of one water system model they cannot represent the hydrologic behavior in other model areas and so have to be retrained. However, the fault diagnosis of the system was not considered (Aluja,Teodorescu and Gil Lafuente, 1992).

Anthonio (2002) in Intelligent Car Parking Using Fuzzy Neural Networks analyzes the performance and practical implementation of Fuzzy Neural Networks for the autonomous motion of mobile robot. The mobile robot with Fuzzy Neural controller presents good positioning and tracking performance for different types of desired trajectories. In this work the fault diagnosis of the system was not considered (Abiyev, Kaynak, Alshanableh and Mamedov, 2011).

Mitchell et al (2000) worked on using a Neural Network to predict the Dynamic Frequency Response of a power system to an Under-frequency Load Shedding Scenario. It showed a method to quickly and accurately predict the dynamic response of a power system during an under frequency load shedding. Emergency actions in a power system due to loss of generation typically calls for under-frequency load shedding measures, to avoid potential collapse due to lack of time in which to correct the imbalance via other means. This is a slow and repetitious use of dynamic simulators so a neural network was used to obtain a fast and accurate procedure during optimal load shedding. In this work the fault diagnosis of the system was not considered Amit (1989).

Wu, Karray and Song (2003) in Water Level Control by Fuzzy Logic and Neural Network designed an Intelligent controllers for controlling water level system by building a prototype of water level control system first with Fuzzy Logic control and then with Neural Network. The

performance of both was noted and compared and it shows that the Neural Network showed a better performance than that of Fuzzy Logic. In this work the hybrid of the two networks was not considered. Fuzzy Neural Networks implement the process of fuzzy reasoning through a neural network structure so that they behave as fuzzy system with learning capabilities.

Pislaru, Trandabat and Olariu (2003) in their work on Neurofuzzy system for industrial processes fault diagnosis featured a methodology to monitor and diagnose machine faults in complex industrial processes. They developed the diagnostic model capable of providing diagnosis for single and multiple faults based on noisy data. In their work, they only showed the faults incurred and ignored monitoring of the system while on operation, the type and time of fault occurrence and also the possible rectification techniques (Stathacopoulou, Magoulas, Grigoriadou, and Samarakou, 2011).

2.1.1 Intelligent Process Control Systems Using Fuzzy-Neural Network

In this research, Fuzzy Neural Network is to be applied to a PID controller. The intelligent control will be used to monitor the system during its normal working condition so as to detect the occurrences of failures, recognize the fault location, the time of fault and hence suggests the possible rectification procedures (fault rectification). The system will be first realized using fuzzy logic and then be improved upon with Fuzzy

Neural Network. The data obtained will be plotted in a graph and the results obtained will be compared with the graph of a classical PID.

2.2 Fuzzy Logic Control

Yager and Zadeh, (1992) stated that Fuzzy Logic is the branch of artificial intelligence that deals with the reasoning algorithms used to emulate human thinking and decision making in machines. The term itself inspires certain skepticism, sounding equivalent to "half-baked logic" or "bogus logic", but the "fuzzy" part does not refer to a lack of rigor in the method, rather to the fact that the logic involved can deal with fuzzy concepts i.e. concepts that cannot be expressed as "true" or "false" but rather as "partially true". The fuzzy logic creates a way to express in-between data values. It deals with uncertainty in engineering by attaching degrees of certainty to answer to a logical question. Fuzzy logic requires knowledge in order to reason (Bezdek and Pal, 1992) and (Binaghi, 1992). This knowledge which is provided by an expert who knows the process or machine is stored in the fuzzy system. For example, an expert may specify that a steam valve should be turned clockwise a “little bit” if the temperature rises in a batching operation. The fuzzy system might interpret this expression as a 10-degree clockwise rotation that closes the output valve opening by five percent. As the name implies the description like a “little bit” is a fuzzy description, meaning that it does not have a

definite value. A Fuzzy Logic system takes this vague description and translates it into a decisive output (Ahlawat, Ashu, and Nidi, 2014).

2.2.1 Fuzzy Logic Principles

The figure 2.1 illustrates the operation of a Fuzzy Logic control system (Fuzzy Logic Controller). It consists of three main components, or actions that must be completed sequentially to determine the appropriate output value. These components are

1. Fuzzification
2. Fuzzy processing
3. Defuzzification

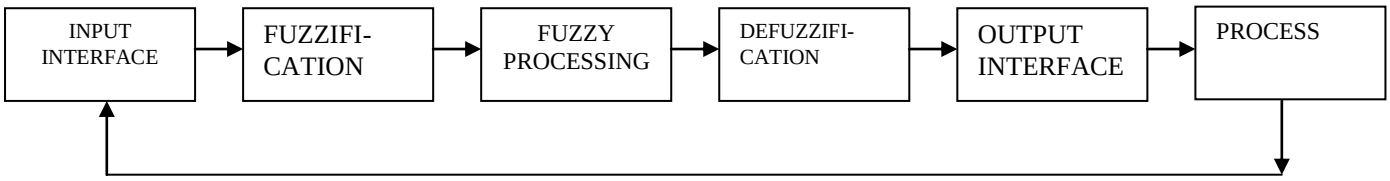


Figure 2.1: Fuzzy Logic controller

When a fuzzy controller receives input data it translates it into a fuzzy form. This process is called fuzzification. The controller then performs fuzzy processing which involves the evaluation of the input information according to IF-THEN rules created by the user during the fuzzy control system's design stage. Once the fuzzy controller finishes the rule processing stage and arrives at an outcome conclusion, it begins the defuzzification process (Chen, 1988). In this final step, the fuzzy controller converts the output conclusions into "real" output data (analog

counts) and sends this data to the process via an output module interface. If the fuzzy logic controller is located in the Programmable Logic Controller (PLC) rack and does not have a direct or built in input/output interface with the process then it will send the defuzzification output to the PLC memory location that maps the processors output interface module (Langavi and Berenji, 1992), (Lee, 1990) and (Kandel, 1991).

2.2.2 Fuzzy Sets

Fuzzy set theory was originally proposed by Prof Lotfi A. Zadeh of University of California at Berkley quantitatively and effectively handles problems involving uncertainty, ambiguity and vagueness. The theory, which is now well-established, was specifically designed to mathematically represent uncertainty and vagueness and provide formalized tools for dealing with the imprecision that is intrinsic to many real world problems. The ability of fuzzy logic to deal with uncertainty and noise has led to its use in control. Designing a fuzzy controller requires describing the operators control knowledge/experience linguistically. The controller captures these traits in the form of fuzzy sets, fuzzy logic operations and fuzzy rules. Thus, Fuzzy logic control can be used to emulate human expert knowledge and experience. The Fuzzy sets and fuzzy rules can be formulated in terms of linguistic variables, which help the operator to understand the functioning of the controller. "Fuzzy sets" can be a complex mathematical term in multi-

valued logic. A fuzzy set is an object with elements, or members, which can belong to it in degrees (Lim, Rahardja and Gwee, 1995). The membership function is obviously a crucial component of a fuzzy set. It is therefore natural to define operations on Fuzzy sets by means of their membership function. In practice, the fuzzy rule sets usually have several antecedents that are combined using fuzzy operators, such as AND, OR, and NOT, though again the definitions tend to vary: AND, in one popular definition, simply uses the minimum weight of all the antecedents, while OR uses the maximum value. There is also a NOT operator that subtracts a membership function from 1 to give the "complementary" function (Dubois and Prade, 1985) and (Fuzzy CLIPS, 1994)

2.2.3 Assigning Zero or One Values to Fuzzy Sub-Sets

A fuzzy control system is a control system based on fuzzy logic - a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or digital logic, which operates on discrete values of either 0 and 1 (true and false). When implementing fuzzy logic control with human originated rules in the loop, we must have a way to assign some numeric value to humans' intuitive assessments of fuzzy sets (Lim and Takefuji, 1990). We must translate from human fuzziness to numbers that can be used by a computer. We do this by assigning fuzzy sub-set conditions a value from zero to 1. In setting up a control system for room temperature,

for example, we could assign a membership of "1.0" in the sub-set of "just right" when the temperature is 75 degrees F. Then, if the temperature drops to 70 degrees F, we might design the system for a membership in the "just right" sub-set of ".8". Fuzzy logic makes use of human common sense. This common sense is either applied from what seems reasonable for a new system, or from experience for a system that has been previously handled by a human operator. The input variables in a fuzzy control system are in general mapped into sets of membership functions similar to this, known as "fuzzy sets". The process of converting a crisp input value to a fuzzy value is called "fuzzification". A control system may also have various types of switch, or "ON-OFF", inputs along with its analog inputs, and such switch inputs of course will always have a truth value equal to either 1 or 0, but the scheme can deal with them as simplified fuzzy functions that happen to be either one value or another (Dubois and Prade, 1988).

Given "mappings" of input variables into membership functions and truth values, the microcontroller then makes decisions for what action to take based on a set of "rules", each of the form: For instance, IF brake temperature is warm AND speed is not very fast THEN brake pressure is slightly decreased. In this example, the two input variables are "brake temperature" and "speed" that have values defined as fuzzy sets. The output variable, "brake pressure", is also defined by a fuzzy set that can

have values like "static", "slightly increased", "slightly decreased", and so on. This rule by itself is very puzzling since it looks like it could be used without bothering with fuzzy logic, but remembers the decision is based on a set of rules which are as follows:

- All the rules that apply are invoked, using the membership functions and truth values obtained from the inputs, to determine the result of the rule.
- This result in turn will be mapped into a membership function and truth value controlling the output variable.
- These results are combined to give a specific ("crisp") answer, the actual brake pressure, a procedure known as "defuzzification".

This combination of fuzzy operations and rule-based "inference" describes a "fuzzy expert system" (Zimmermann, 1996).

2.2.4 Fuzzy Control Design

The design of fuzzy controllers consists of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each, then combines the results of the rules. Finally**,** the output stage converts the combined result back into a specific control output value (Zadeh, 1984). The most common shape of membership functions is triangular, although trapezoidal and bell curves

are also used, but the shape is generally less important than the number of curves and their placement. From three to seven curves are generally appropriate to cover the required range of an input value, or the "universe of discourse" in fuzzy jargon. The processing stage is based on a collection of logic rules in the form of IF-THEN statements, where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy control systems have dozens of rules.

Consider a rule for a thermostat:

IF (temperature is "cold") THEN (heater is "high")

This rule uses the truth value of the "temperature" input, which is some truth value of "cold", to generate a result in the fuzzy set for the "heater" output, which is some value of "high". This result is used with the results of other rules to finally generate the crisp composite output. Obviously, the greater the truth value of "cold", the higher the truth value of "high", though this does not necessarily mean that the output itself will be set to "high", since this is only one rule among many. In some cases, the membership functions can be modified by "hedges" that are equivalent to adjectives. Common hedges include "about", "near", "close to", "approximately", "very", "slightly", "too", "extremely", and "somewhat". These operations may have precise definitions, though the definitions can vary considerably between different implementations. "Very", for one example, squares membership functions; since the membership values are

always less than 1, this narrows the membership function. "Extremely" cubes the values to give greater narrowing, while "somewhat" broadens the function by taking the square root. There are several different ways to define the result of a rule, but one of the most common and simplest is the "max-min" inference method, in which the output membership function is given the truth value generated by the premise. Rules can be solved in parallel in hardware, or sequentially in software (Zimmerman, 1987). The results of all the rules that have fired are "defuzzified" to a crisp value by one of several methods. There are dozens in theory, each with various advantages and drawbacks. The "centroid" method is very popular, in which the "center of mass" of the result provides the crisp value. Another approach is the "height" method, which takes the value of the biggest contributor. The centroid method favors the rule with the output of greatest area, while the height method obviously favors the rule with the greatest output value (Ross, 1995) and (Kazuo and Hua, 2001)

Figure 2.2 demonstrates centroid defuzzification max-min inference used for a system with input variables "x", "y", and "z" and an output variable "n". Note that "mu" is standard fuzzy-logic nomenclature for "truth value":

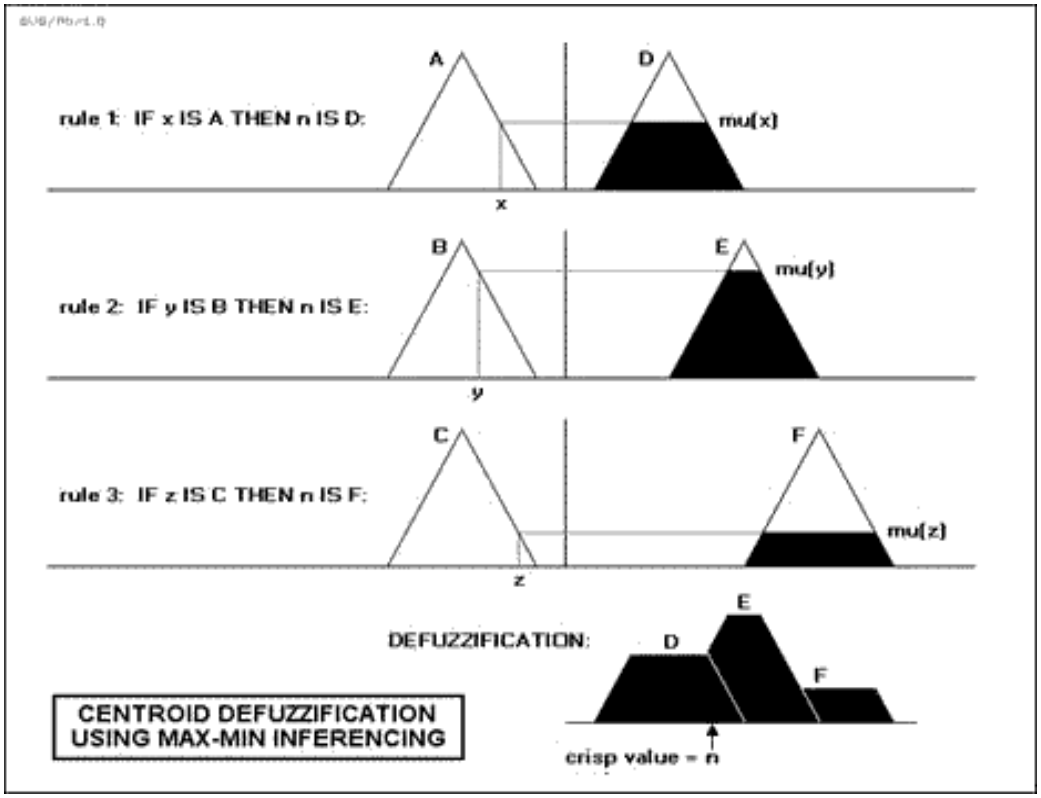


Figure 2.2: Centroid Defuzzification using max-min inference

Notice how each rule provides a result as a truth value of a particular membership function for the output variable. In centroid defuzzification the values are ORs, that is, the maximum value is used and values are not added, and the results are then combined using a centroid calculation.

Fuzzy control system design is based on empirical methods, basically a methodical approach to trial-and-error (Wie and Da fa, 2010). The general process is as follows:

- Document the system's operational specifications and inputs and outputs.
- Document the fuzzy sets for the inputs.
- Document the rule set.

- Determine the defuzzification method.
- Run through test suite to validate system, adjust details as required.
- Complete document and release to production.

2.3 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons (Anderson, 1995).

Neural network (NN) methods have become very popular recently involving mapping of input-output vectors for cases where no theoretical model works satisfactorily. An Artificial Neural Network (ANN) is an information processing paradigm inspired by the manner in which the heavily interconnected, parallel structure of the human brain processes information. They are collections of mathematical processing units that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. Neural Networks (NN) are trainable systems whose learning abilities, tolerance

to uncertainty and noise and generalization capabilities are derived from their distributed network structure and knowledge representation (Lee et al., 2011).

Learning of a Neural Network typically implies adjustments of connection weights and biases so that the square error (between Neural Network output and desired output) is minimized. However NN is often called a black box and it is difficult to interpret the knowledge stored by a it. Knowledge in NN is represented in the values of the weights and biases which forms part of a large and distributed network (Hert, Krogh and Palmer, 1991)

2.3.1 The significance of a Neural Network

Zhou and Quek (1996) showed that Neural Networks with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an expert in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer what if questions (Borisyuk, Holden and Kryukov, 1991).

Other advantages include

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage. (Kohers, 1992)

2.3.2 Neural Networks versus Conventional Computers:

Neural networks take a different approach to problem solving than that of conventional computers. Convention computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do. (Verma, Verma and Bhandari, 2014). Neural Networks process information in a similar way the human brain does. The network is

composed of a large number of highly interconnected processing element (neurons) working in parallel to solve a specific problem. Neural Networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly .The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable (Kosco, 1991).

On the other hand, conventional computers use a cognitive approach to problem solving, the way the problem is to be solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable, if anything goes wrong is due to a software or hardware fault.

Neural Networks and conventional algorithmic computers are not in competition but complement each other. There are tasks more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to Neural Networks. Even more, a large number of tasks, require systems that use a combination of two approaches (normally a conventual's computer is used to supervise the neural network) in order to perform at maximum efficiency.

Neural Networks do not perform miracles. But if used sensibly they can produce some amazing results (Beale and Jackson, 1990).

2.3.3 How the Human Brain Learns

Much is still unknown about how the brain trains itself to process information, so, theories abound. In the human brain, a typical neuron (Figure 2.3 shows the component of a neuron) collects signal from others through a host of fine structures called dendrites. The neuron sends out spikes of electrical activity through a long thin strand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse illustrated in figure 2.4 converts the activity from the axon into electrical effects that inhibit or excite activity into the connected neurons (Bulsara, Jacobs and Zhou, 1991). When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes (Aleksander and Morton, 1993) and (Arbib, 1995).

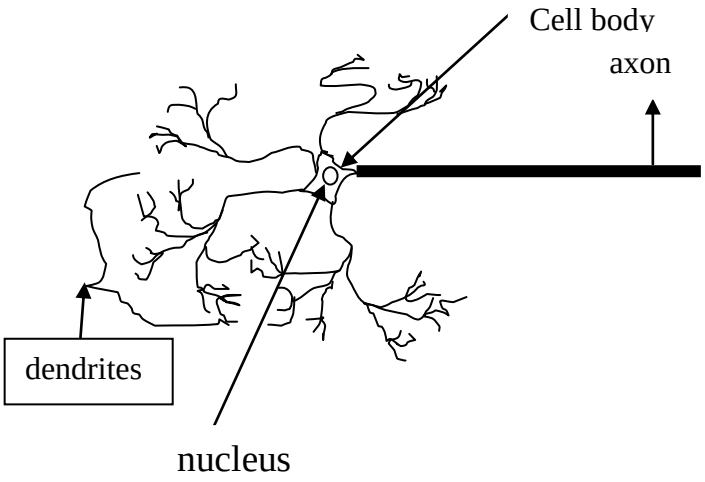


Figure 2.3: Components of a neuron

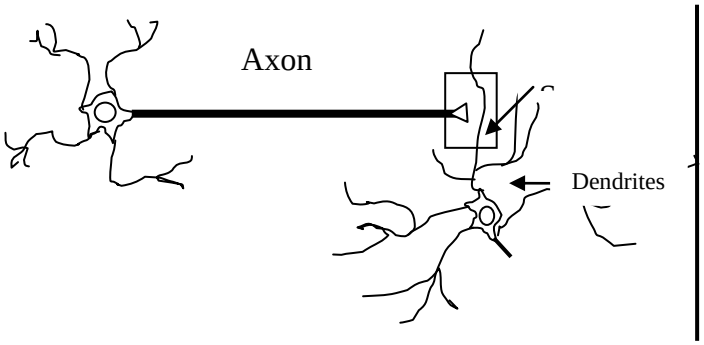


Figure 2.4: The Synapse

2.3.4 From Human Neurons to Artificial Neurons

These Neural Networks were first deduced with the essential features of neurons and their interconnections. Figure 2.5 shows the neuron model. We then typically program a computer to simulate these features. However because our knowledge of neurons is incomplete and our computing power is limited, our models are necessarily gross idealizations of real networks of neurons (Eckmiller and Napp-Zinn, 1993).

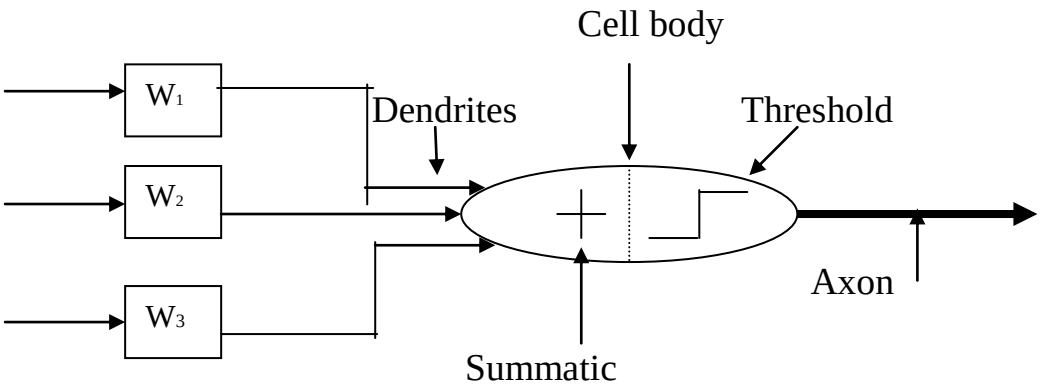


Figure 2.5: Neuron model.

2.3.5 A Simple Neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation, the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input pattern. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not. Figure 2.6 shows the diagram of a simple neuron (Feigenbaum 1989) and (Lippman, 1987).

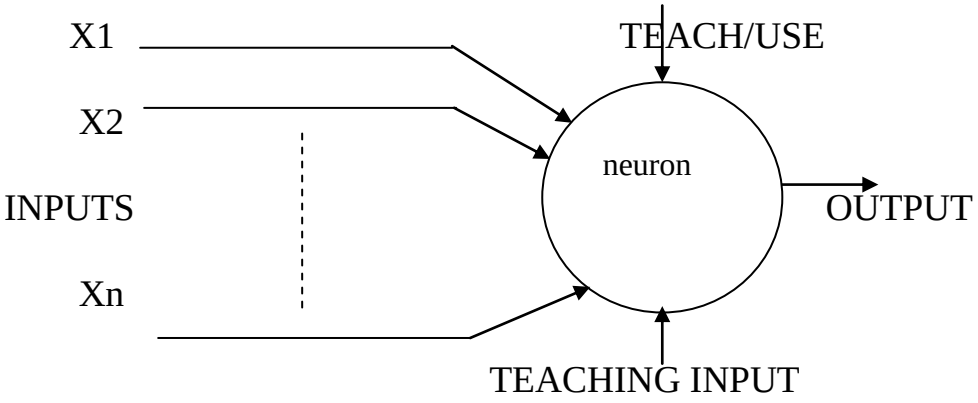


Figure 2.6: A Simple Neuron

2.3.6 The Firing Rules

The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained (Eaton and Oliver, 1992). A simple firing rule can be implemented by using Hamming distance technique. The rule goes as follows:

Take a collection of training patterns for a node, some of which cause it to fire (the 1- taught set of patterns) and others which prevent it from doing so (the 0 - taught set).Then the patterns not in the collection cause the node to fire if, on comparison, they have more input elements in common with the nearest pattern in the 1 taught set than with the nearest pattern in the 0 taught set. If there is a tie, then the pattern remains in the undefined stat (Ajanagadde and Shastri, 1991).

For example , a 3 input neuron is taught to output 1 when the input (X1.X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001.Then , before applying the firing rule the truth table (Table 2.1) is

Table 2.1: Truth table for 3 input neuron

X1		0	0	0	0	1	1	1	1
X2		0	0	1	1	0	0	1	1
X3		0	1	0	1	0	1	0	1
OUT		0	0	0	0/1	0/1	1	1	1

As an example of the way the firing rule is applied, take the pattern 010 for instance, it differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from 111 in 2 elements. Therefore, the nearest pattern is 000 which belongs in the 0 taught set. Thus the firing rule requires that the neuron should not fire when the input is 001.On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1) (Fletcher and Goss, 1993) and (Freeman and Skapura, 1992).

By applying the firing in every column the following truth table is obtained as shown in Table 2.2.

(Table 2.2: Truth table of 3 input neuron)

X1		0	0	0	0	1	1	1	1
X2		0	0	1	1	0	0	1	1
X3		0	1	0	1	0	1	0	1
OUT		0	0	0	0/1	0/1	1	1	1

The difference between the two truth tables is called generalization of the neuron. Therefore the firing rule gives the neuron a sense of similarity and enables it to respond sensibly to patterns not seen during training (Funahashi, 1989).

2.4 NeuroFuzzy System

A neurofuzzy system is a fuzzy system that uses a learning algorithm derived from or inspired by neural network theory to determine its parameters (fuzzy sets and fuzzy rules) by processing data samples (Lin and Lee, 1996).

2.4.1 Fuzzy Neural network

A Fuzzy Neural Network or neurofuzzy system is a learning machine that finds the parameters of a Fuzzy system (i.e., fuzzy sets, fuzzy rules) by

exploiting approximation techniques from Neural Networks (Feldkamp, Puskorius and Yuan, 1992).

A Fuzzy Neural system is based on a Fuzzy system which is trained by a learning algorithm derived from Neural Network theory. The (heuristical) learning procedure operates on local information, and causes only local modifications in the underlying Fuzzy System (Kar, Das, and Ghosh, (2014).

A Fuzzy Neural system can be viewed as a 3-layer feed forward Neural Network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as (fuzzy) connection weights. It is not necessary to represent a Fuzzy System like when you want to apply a learning algorithm to it. However, it can be convenient, because it represents the data flow of input processing and learning within the model (Gallinari, Thinia and Fogelman-Soulie, 1988)

A Fuzzy Neural system can be always (i.e. before, during and after learning) interpreted as a system of fuzzy rules. It is also possible to create the system out of training data from scratch, as it is possible to initialize it by prior knowledge in form of fuzzy rules (Park, 2002).

The learning procedure of a Fuzzy Neural system takes the semantical properties of the underlying Fuzzy System into account. This results in constraints on the possible modifications applicable to the system

parameters. A Fuzzy Neural system approximates an n-dimensional (unknown) function that is partially defined by the training data. The fuzzy rules encoded within the system represent vague samples, and can be viewed as prototypes of the training data. A Fuzzy Neural system should not be seen as a kind of (fuzzy) expert system, and it has nothing to do with Fuzzy Logic in the narrow sense (Abiyev, Kaynak, Alshanableh and Mamedov, 2011).

2.4.2 Combining Fuzzy Systems with Neural Networks

Both neural networks and fuzzy systems have some things in common. They can be used for solving a problem (e.g. pattern recognition, regression or density estimation) if there does not exist any mathematical model of the given problem. They solely do have certain disadvantages and advantages which almost completely disappear by combining both concepts. Neural Networks can only come into play if the problem is expressed by a sufficient amount of observed examples. These observations are used to train the black box. (Hech- Nielsen, 1988) and Holland, 1975). On the one hand no prior knowledge about the problem needs to be given. On the other hand, however, it is not straightforward to extract comprehensible rules from the neural network's structure. On the contrary, a Fuzzy System demands linguistic rules instead of learning examples as prior knowledge. Furthermore the input and output variables

have to be described linguistically. If the knowledge is incomplete, wrong or contradictory, then the fuzzy system must be tuned. Since there is not any formal approach for it, the tuning is performed in a heuristic way.

Table 2.3: Comparison of neural control and fuzzy control

Neural Networks	**Fuzzy Systems**
no mathematical model necessary	no mathematical model necessary
learning from scratch	Apriori knowledge essential
several learning algorithms	not capable to learn
black-box behavior	Simple interpretation and implementation

This is usually very time consuming and error-prone. (Kasabov, 1995)

It is desirable for Fuzzy Systems to have an automatic adoption procedure which is comparable to Neural Networks. As it can be seen in Table 2.3, combining both approaches should include advantages and exclude disadvantages.

Characteristics

- Compared to a common Neural Network, connection weights and propagation and activation functions of Fuzzy Neural Network differ a lot. Although there are many different approaches to model a Fuzzy Neural Network, most of them agree on certain characteristics such as the following: A Fuzzy Neural system based on an underlying fuzzy system is trained by means of a data-driven

learning method derived from neural network theory. This heuristic only takes into account local information to cause local changes in the fundamental fuzzy system.

- It can be represented as a set of fuzzy rules at any time of the learning process, i.e., before, during and after. Thus the system might be initialized with or without prior knowledge in terms of fuzzy rules.
- The learning procedure is constrained to ensure the semantic properties of the underlying Fuzzy System.
- A Fuzzy Neural system approximates a n-dimensional unknown function which is partly represented by training examples.
- Fuzzy rules can thus be interpreted as vague prototypes of the training data.
- A Fuzzy Neural system is represented as special three-layer feed forward Neural Network as it is shown in Figure 2.7,there the

  The first layer corresponds to the input variables, the second layer symbolizes the fuzzy rules and the third layer represents the output variables (Keller and Chen, 1992) and (Maren, 1990)

  The fuzzy sets are converted as fuzzy connection weights.

  Some approaches also use five layers where the fuzzy sets are encoded in the units of the second and fourth layer, respectively. However, these models can be transformed into three-layer

architecture. One can basically distinguish between three different kinds of Fuzzy Neural Networks (FNN), i.e., cooperative, concurrent and hybrid FNNs. (Abraham, 1997) and (Bart, 1991)

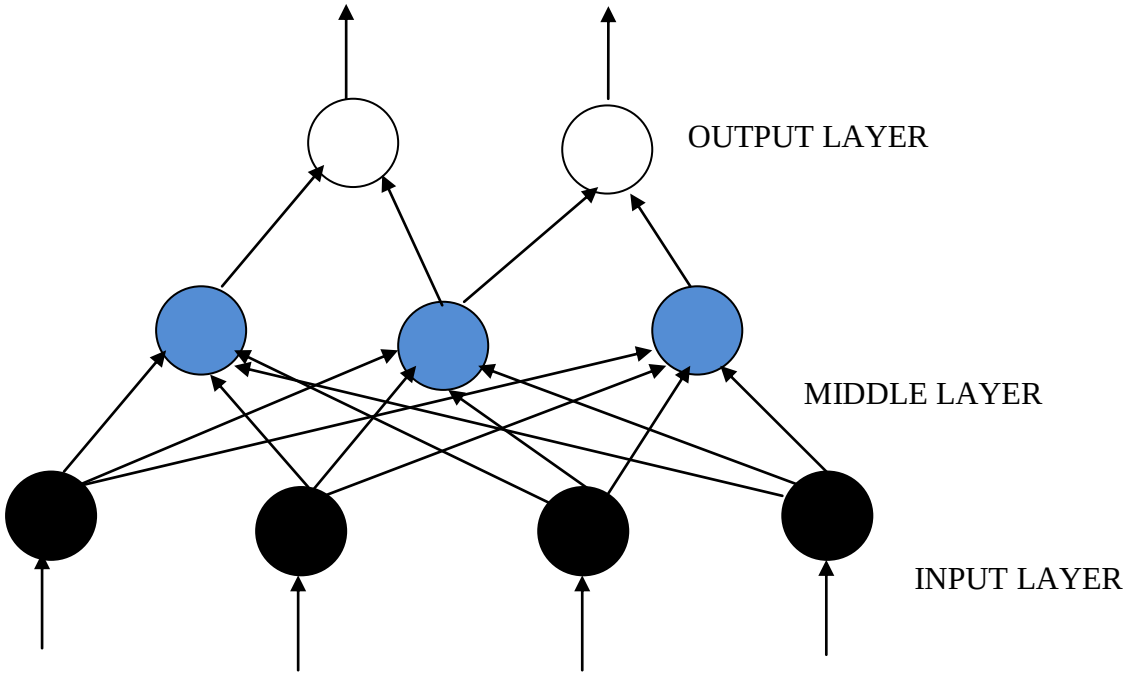


Figure 2.7: The architecture of a Fuzzy Neural system

2.4.3 Cooperative Fuzzy-Neural Network

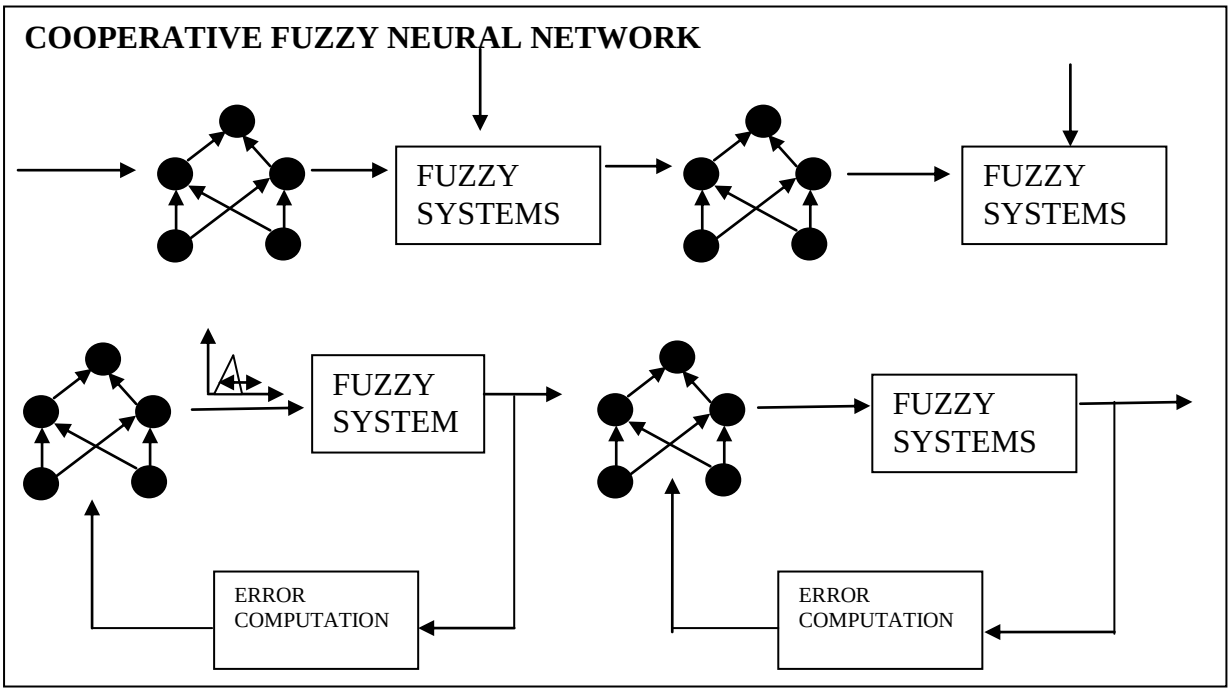


Figure 2.8: Cooperative Fuzzy Neural Networks

In the case of Cooperative Neural Fuzzy systems, both Artificial Neural Network and Fuzzy System work independently from each other. The ANN tries to learn the parameters from the Fuzzy System. This can be either performed offline or online while the Fuzzy System is applied. Figure 2.8 depicts four different kinds of Cooperative Fuzzy Neural Networks. The upper left Fuzzy Neural Network learns fuzzy set from given training data. This is usually performed by fitting membership functions with a neural network. The fuzzy sets are then determined offline. They are then utilized to form the Fuzzy System by fuzzy rules that are given (not learned) as well (Limkens and Nie, 1992).

The upper right Fuzzy Neural System determines fuzzy rules from training data by a Neural Network. Here as well, the neural networks learn offline before the Fuzzy System is initialized. The rule learning is usually done by clustering on self-organizing feature maps. It is also possible to apply fuzzy clustering methods to obtain rules.

In the lower left Fuzzy Neural model, the system learns all membership function parameters online, i.e. while the Fuzzy System is applied. Thus initially fuzzy rules and membership functions must be defined beforehand. Moreover, the error has to be measured in order to improve and guide the learning step (Peymanfar, Khoei and Hadidi, 2010).

The lower right one determines rule weights for all fuzzy rules by a neural network. This can be done online and offline. A rule weight is

interpreted as the influence of a rule. They are multiplied with the rule output. The authors argue that the semantics of rule weights are not clearly defined. They could be replaced by modified membership functions. However, this could destroy the interpretation of fuzzy sets. Moreover, identical linguistic values might be represented differently in dissimilar rules (Yi and Oh 1992), (Fogelman, Lamy, and Viennet, 1993) and (Fukushima, Miyake and Ito, 1993).

2.4.4 Hybrid Fuzzy Neural Network

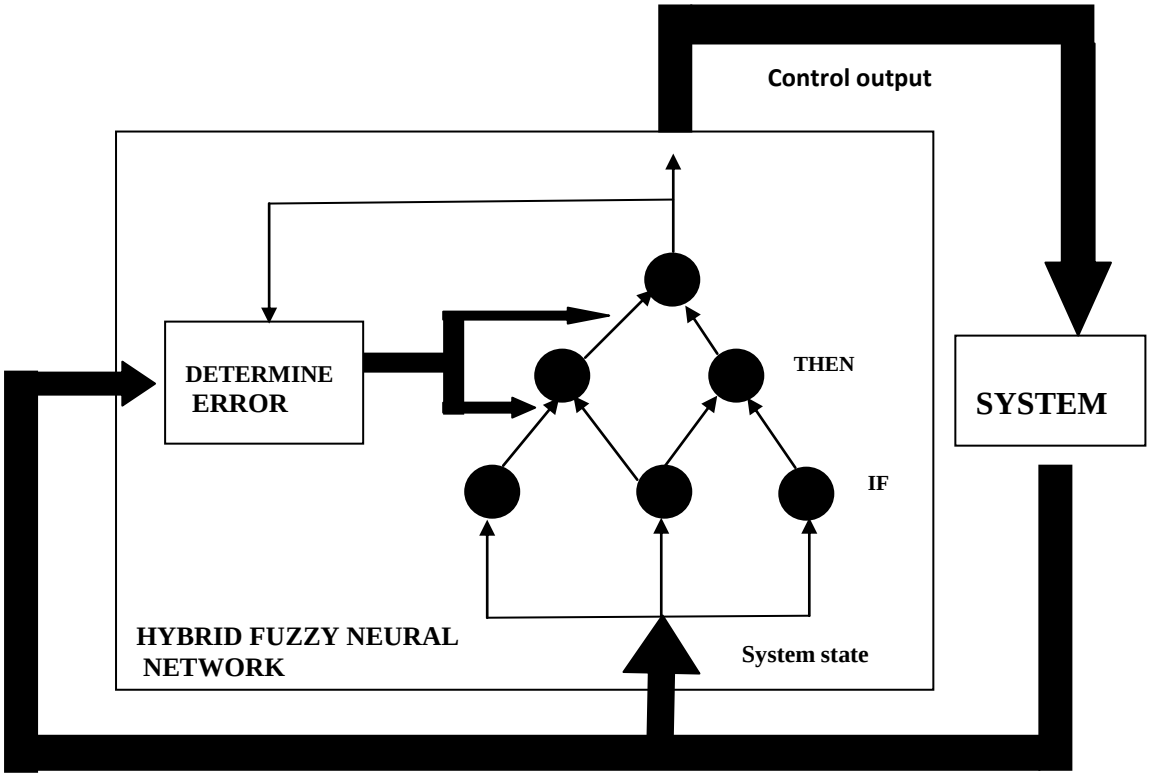


Figure 2.9: A Hybrid Fuzzy Neural Network

Hybrid Fuzzy Neural systems are homogeneous and usually resemble neural networks. Here, the Fuzzy System is interpreted as special kind of Neural Network. The advantage of such hybrid NFS is its architecture

since both Fuzzy System and Neural Network do not have to communicate any more with each other. They are one fully fused entity (Goonatilake and Khebbal, 1994). These systems can learn online and offline. Figure 2.9 shows such a hybrid FNN.

The rule base of a Fuzzy System is interpreted as a Neural Network. Fuzzy sets can be regarded as weights whereas the input and output variables and the rules are modeled as neurons. Neurons can be included or deleted in the learning step. Finally, the neurons of the network represent the fuzzy knowledge base. Obviously, the major drawbacks of both underlying systems are thus overcome.

In order to build a fuzzy controller, membership functions which express the linguistic terms of the inference rules have to be defined. In fuzzy set theory, no formal approach to define these functions. Any shape (e.g., triangular, Gaussian) can be considered as membership function with an arbitrary set of parameters. Thus the optimization of these functions in terms of generalizing the data is very important for Fuzzy Systems. Neural Networks can be used to solve this problem.

By fixing a distinct shape of the membership functions, say triangular, the neural network must optimize their parameters by gradient descent. Thus, aside information about the shape of the membership functions, training data must be available as well (Barlett, 1993)

Quek and Zhou (1999) suggested another approach which is to group the training data $\{(\mathbf{x}_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, l\}$ into M clusters. Every cluster represents a rule R_m where $m = 1, 2, \ldots, M$. Hence these rules are not defined linguistically but rather by crisp data points $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Thus a Neural Network with n input units, hidden layers and M output units might be applied to train on the pre-defined clusters. For testing, an arbitrary pattern $\boldsymbol{X}$ is presented to the trained neural network. Every output unit m will return a degree to which extend $\boldsymbol{X}$ may fit to the antecedent of rule $\boldsymbol{R_m}$.) (Zurada, 1992)

To guarantee the characteristics of a Fuzzy System, the learning algorithm must enforce the following mandatory constraints:

- Fuzzy sets must stay normal and convex.
- Fuzzy sets must not exchange their relative positions (they must not *pass* each other).
- Fuzzy sets must always overlap.

Additionally there exist some optional constraints like the following:

- Fuzzy sets must stay symmetric.
- The membership degrees must sum up to 1.

An important Hybrid Fuzzy Neural Network has been introduced .The ARIC (Approximate Reasoning-based Intelligent Control) is presented as

a neural network where a prior defined rule base is tuned by updating the network's prediction. Thus the advantages of Fuzzy Systems and Neural Networks are easily combined as presented in Table 2.3

The ARIC is represented by two feed-forward Neural Networks, the Action-state Evaluation Network (AEN) and the Action Selection Network (ASN). The ASN is a multilayer neural network representation of a fuzzy system. It then again consists of two separate. The first one represents the fuzzy inference and the second one computes a confidence measure based on the current and next system state. Both parts are eventually combined to the ASN's output.

As it is shown in Figure 2.7, the first layer represents the rule antecedents, whereas the second layer corresponds to the implemented fuzzy rules and the third layer symbolized the system action. The network flow is at follows. In the first layer the system variables are fuzzified. In the next step these membership values are multiplied by the attached weights of the connections between the first and second layer. In the latter layer, every rule's input corresponds to the minimum of its input connections (Hayashi, 1991).

A rules conclusion is installed as membership function. This function maps the inverse rule input value. Its output values are then multiplied by the weights of the connections between second and third layer. The final

output value is eventually computed by the weighted average of all rules' conclusions (Iniyia and Lalitha, 2014) .

The AEN (which is as three-layer feed-forward Neural Network as well) aims to forecast the system behavior. The hidden layer obtains as input both the system state and an error signal from the underlying system. The output of the networks shall represent the prediction of the next reinforcement which depends on the weights and the system state. The weights are changed by a reinforcement procedure which takes into consideration the outputs of both networks ASN and AEN, respectively. ARIC was successfully applied to the cart-pole balancing problem (Hornik 1991).

Whereas the ARIC model can be easily interpret as a set of fuzzy-if-then rules, the ASN network to adjust the weights is rather difficult to understand. It is a working Neural Network architecture that utilizes aspects of Fuzzy Systems. However, a semantic interpretation of some learning steps is not possible (Ang, Quek and Pasquier, 2003) .

2.5 Genetic Algorithms

Genetic Algorithms are search algorithms based on mechanics of natural selection and natural genetics. They combine survival of the fittest among the string structures with randomized, yet organized, information exchange to form a search algorithm with capabilities of natural evolution. A genetic algorithm starts with a random creation of a

population of strings and thereafter generates successive populations of strings that improve over generations. The processes involved in the generation of new populations mainly consist of operations such as reproduction, cross over and mutation. Genetic Algorithms have proven their robustness and usefulness over other search techniques because of their unique procedures that differ from other normal search and optimization techniques. Genetic Algorithm based search and optimization techniques have recently found increasing use in machine learning, robot motion planning scheduling, pattern recognition, image sensing and many other engineering applications. (Davis, 1991), and (Goonatilake and Campbell, 1994)

Since they work on coding of the parameter set, and not on the derivative function, they are capable of solving a vast range of optimization problems including optimization of the rule set of a fuzzy logic controller.

2.6 Process Control

Process control is extensively used in industry and enables mass production of continuous processes such as oil refining, paper manufacturing, chemicals, power plants and many other industries. Process control enables automation, with which a small staff of operating personnel can operate a complex process from a central control room.

All control systems have at least three parts to them. An input that takes information into the control system, a process that uses the input

information to create the output information and an output that passes information out of the control system. There is an Open Loop control system and a Closed Loop Control system (Onkar, 2010).

An Open Loop control system is a system where the information from the output is not sent back to the input. For example an electric heater, the input is the power, the element is the process and the heater is the output. When one switches on, however hot the room gets, the heater keeps producing heat until someone switches it off hence no feedback to the input. If the heater had a thermostat in, it would switch off by itself when the room reached a set temperature (the 'input' to the system). In this case information from the output of the system (heat) has been fed back to the input, as shown in the block diagram of a closed loop system in figure 2.10. Here, the control system is

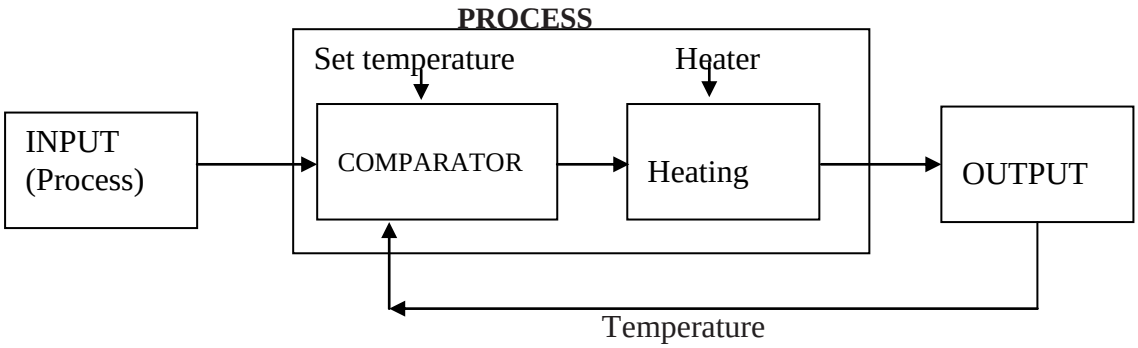


Figure 2.10: Block diagram of closed loop system

now a Closed Loop system. Information from the output goes back to the input in a Feedback loop, a thermostat can be considered as the heater comparator in this control system. It compares the set temperature with

the output temperature and the difference between these two temperatures is the error. When the control system detects an error it tries to make it smaller by changing the output and then the thermostat turns ON or OFF the heater (King, 2010). Table 2.4 shows the comparison between open loop system and closed loop system.

Table 2.4 Comparison between open loop system and closed loop system

	OPEN LOOP SYSTEM	**CLOSED LOOP SYSTEM**
1	They are not reliable	These are reliable
2	It is easier to build	It is difficult to build
3	If calibration is good, they perform accurately	They are accurate because of feedback
4	They are generally more stable	They are less stable
5	Optimization is not possible	Optimization is possible

2.6.1 Advantages of a Closed Loop System

1. They are more reliable
2. They are faster
3. It can handle a number of variables simultaneously

2.6.2 Disadvantages of a Closed Loop System

1. Closed loop systems are expensive
2. Its maintenance is difficult

3. It has complicated installation
4. It is difficult to build.

2.6.3 Types of Control Systems

In practice, process control systems can be characterized as one or more of the following forms:

- Discrete – Found in many manufacturing, motion and packaging applications. Robotic assembly, such as that found in automotive production, can be characterized as discrete process control. Most discrete manufacturing involves the production of discrete pieces of product, such as metal stamping (Wayne, 2003).

- Batch – Some applications require that specific quantities of raw materials be combined in specific ways for particular durations to produce an intermediate or end result. One example is the production of adhesives and glues, which normally require the mixing of raw materials in a heated vessel for a period of time to form a quantity of end product. Other important examples are the discrete production of individual products at a time and batch production of group products at a time.

- Continuous process –A physical system which variables are smooth and uninterrupted in time can be classified as a continuous

process. For example, the control of the water temperature in a heating jacket. Other examples of continuous processes are seen in production of fuels, chemicals and plastics. Continuous processes in manufacturing are used to produce very large quantities of product annually.

2.6.4 Control System Parameters

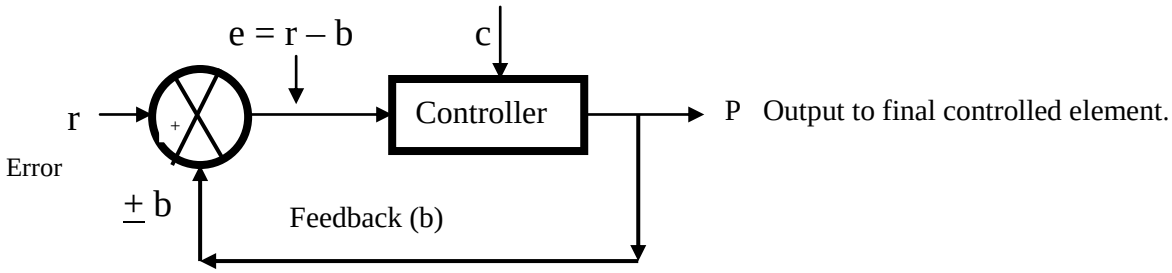


Figure 2.11: Control system parameters

In Figure 2.11 the reference input (r) shows the externally produced input while the error detector receives the measured signal (b) from the feedback from the output and compares it with the reference input (r). The difference of the two signals gives the error signal which is given by

$$e = r - b \qquad 2.1$$

Where, e = error

b = measured indication of variable or the feedback

r = reference input

The controller (c) represents the actual variable that is being controlled and regulates the output (p) according to the signal obtained from the detector. The feedback feeds back the output to the error detector for

comparison with the reference input (Tan, Wang, Quing and Chang, 1999).

2.6.5 Process Characteristics

a. Process Load: - The term process load refers to all the set of parameters resulting in the controlled variable having the set point value excluding the controlled variable. This set of parameters is called the nominal set. The required controlling variable value under these conditions is the nominal value of that parameter. If the set point is changed, the control parameter is altered to cause the variable to adopt this new operating point. The load is still nominal, however because the other parameters are assumed to be unchanged. Suppose one of the parameters changes from nominal, causing a corresponding shift in the controlled variable, then a process load change has occurred. The controlling variable is then adjusted to compensate for this load change and its effect on the dynamic variable to bring it back to the set point. In practice, concern is only shown on the variation in the controlling parameter bringing the controlled variable back to the set point.

b. **Transient**: Another type of change involves a temporary variation of one of the load parameters. After the excursion, the parameter returns to its nominal value. This variation is called a transient. A transient cause variation of the controlled variable and the control system must

make equally transient changes of the controlling variable to keep error to a minimum. A transient is not a load change because it is not permanent (Tanomaru and Omatu, 1992).

c. **Self Regulation:** The tendency to adopt a specific value of the controlled variable for nominal load with no control operation is referred as self regulation. For instance, for a process with a steam valve at 50% that has its' control loop open so that no change in valve position is possible. The liquid heats up until the energy carried away by the liquid equals that input energy from the steam. If the load changes, a new temperature is adopted because the system temperature is not controlled. The process is self regulating, however, because the temperature will not "run away" but stabilizes at some value under given conditions. (Ang, Chong and Li, 2005).

2.7 Limitations of PID control:

While PID controllers are applicable to many control problems, and often perform satisfactorily without any improvements or even tuning, they can perform poorly in some applications, and do not in general provide optimal control.

The fundamental difficulty with PID control is that it is a feedback system, with constant parameters, and no direct knowledge of the process and thus overall performance is reactive.

PID controllers, when used alone, can give poor performance when the PID loop gains has be reduced so that the control system does not overshoot, oscillate or hunt about the control set point value (Lima, Jacobina and Souza, 1997).

Linearity: Another problem faced with PID controllers is that they are linear, and in particular symmetric. Thus, performance of PID controllers in non-linear systems is variable. For example, in temperature control, a common use case is active heating (via a heating element) but passive cooling (heating off, but no cooling), so overshoot can only be corrected slowly – it cannot be forced downward. In this case the PID should be tuned to be over damped, to prevent or reduce overshoot, though this reduces performance, it increases settling time.

Noise in derivative: A problem with the derivative term is that small amounts of measurement or process noise can cause large amounts of change in the output. It is often helpful to filter the measurements with a low-pass filter in order to remove higher-frequency noise components. However, low-pass filtering and derivative control can cancel each other out, so reducing noise by instrumentation means is a much better choice. Alternatively, a nonlinear median filter may be used, which improves the filtering efficiency and practical performance. In some case, the differential band can be turned off in many systems with little loss of

control. This is equivalent to using the PID controller as a Proportional Integral controller. (Tanamaru and Omatu, 1992)

Recent advances in instrumentation, telecommunications and computing are making available to manufacturing company's new sensors and sensing strategies, plant-wide networking and information technologies that are assisting to improve substantially the production cycle. Recently soft computing method, integrating quantitative and qualitative modeling information, has been developed to improve in FD reasoning capabilities. In order to develop better - fast accurate and robust – process control, model – based modern control methods and efficient adaptive and learning techniques are required. The adoption of effective fault diagnosis techniques is becoming critical to ensure higher levels of safety and reliability in automated plants and autonomous systems. (Hoskins and Himmelbaum, 1990)

In many practical situations, uncertainty in the process can affect the performance of the system significantly no matter how the uncertainty that is described as vagueness or ambiguity.

This realization provides the motivation for a possible fuzzy logic approach to Fault Diagnosis Logic (FDL). This has the ability to directly describe the potential failure modes in the parameters while handling a class of nonlinear systems. (Pislaru etal, 2003)

CHAPTER THREE

METHODOLOGY AND SYSTEM ANALYSIS

3.1 METHODOLOGY

The methodology of this research involves the fuzzy logic technique, the neural network and the application of Fuzzy Neural Network which fine tunes the Fuzzy Logic output.

3.1.1 The Fuzzy Logic Principle

Once a clear understanding of the process control is obtained, the rigorous mathematical steps involved in a linear modeling are overcome using fuzzy logic technique. The technique includes the inputs which are the systems variables, the fuzzy sets were generated from the system variables and used to form the membership functions. The fuzzy helps in generating the matrix table, the fuzzy rule is obtained easily using the matrix table. In this work Visual Basic. Net (VB.Net) was used to solve the PID equation to obtain the data that is used to generate the matrix table (See Appendix E). In the rule base stage, the inference engine contains all the system processes. It assists in depiction of an action. The deffuzification stage which is the last stage outputs the approximated actions from the system. Another advantage of this technique is that it can handle multiple inputs and multiple outputs.

3.1.2 The Neural Network

The artificial neuron model simulates multiple inputs and one output, the switching function of input-output relation and the adaptive synaptic weights. In this work, the feed forward neural network and unsupervised learning algorithm was adopted. The feed-forward network is a filter which outputs the processed input signal while the unsupervised learning algorithm use the mechanism that changes synaptic weight values according to input values to the network. A four layer Neural Network was implemented.

3.1.3 Fuzzy Neural Network

The hybrid of the Fuzzy Logic and Neural Network was implemented to fine tune and optimize the fuzzy output. Figure 3.1 shows the block diagram of Fuzzy Neural system used. In response to linguistic statements, the fuzzy interface block provides an input vector to a multi-layer neural network. The Neural Network is trained to yield desired command outputs or decisions. The output of the fuzzy above was fed into the Neural Network and outputs the decision as shown in figure 3.1. The learning algorithm was used to train the neurons.

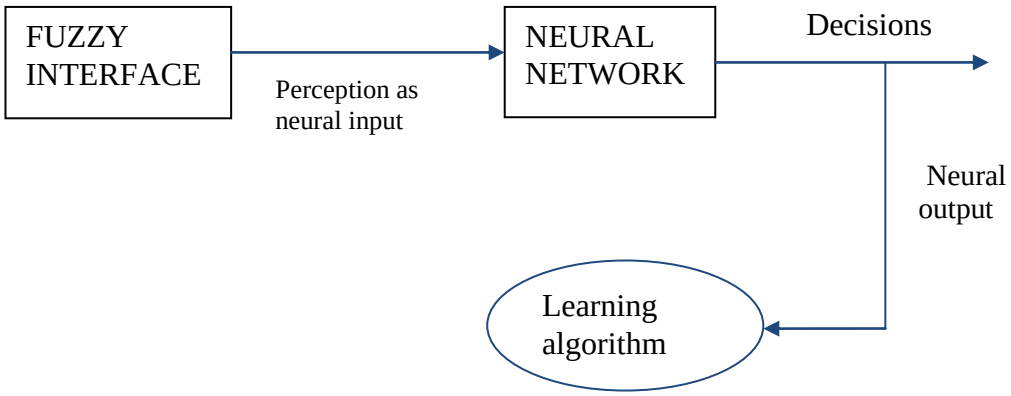


Figure 3.1: Block diagram of Fuzzy Neural Network

3.2 System Analysis

In analyzing the process control, the controller theory and PID Continuous Process Control Unit was discussed.

3.2.1 The Controller Theory

The block diagram of a controller is shown in figure 3.2. The output (controlled variable) is first measured by the instrument. The measurement signal goes to the comparator in form of signal (where it is compared with the set point signal). The difference is called the deviation or error .The resulting current output of the comparator is the error signal which is given as input to the controller. According to the error signal, the controller gives the correction signal (output of controller) to the final control element. The final control element implement the correction by physically adjusting the value of controlled variable of the process to bring the new process output which is fed back to be compared with the set point.

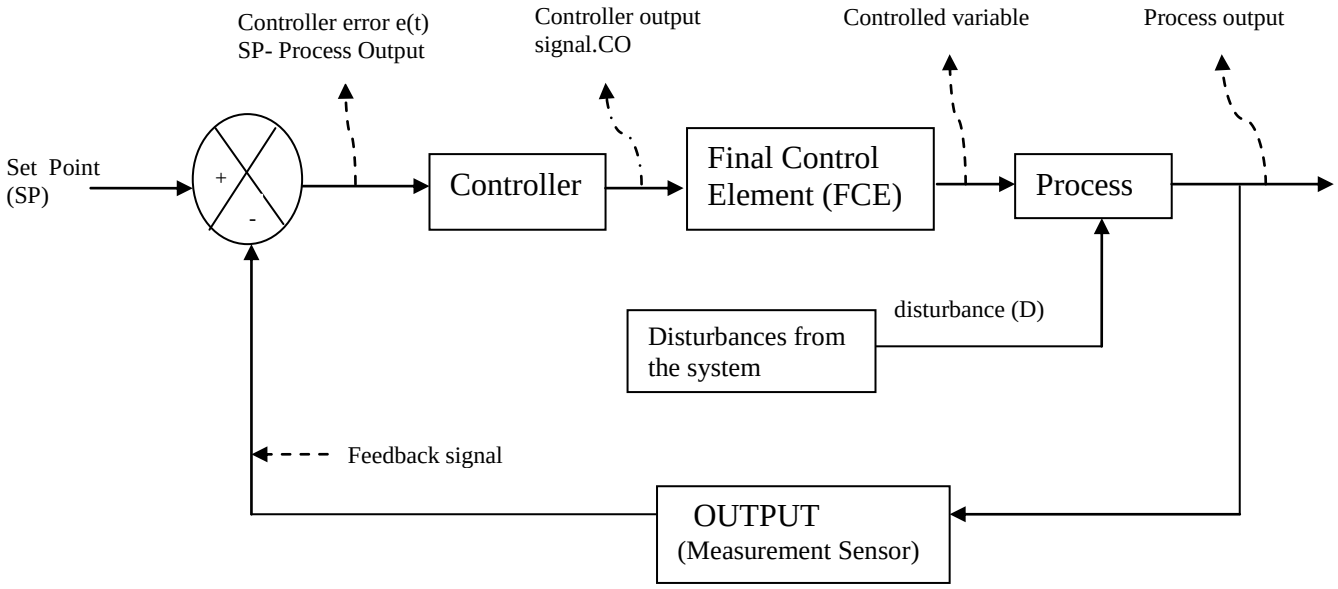


Figure 3.2: Block diagram of a controller

3.2.2 PID Continuous Process Control Unit

Here, the three PID parameters required are the proportional gain value, K_p; integral gain value K_i; and the derivative gain value K_d that guarantee system stability in spite of the time delay. Figure 3.3 shows the water inside a process that is made hot by the heater. The objective of the process is to keep the temperature of the water at a steady temperature (in this case at 50^oc) s as the temperature ranges between (30-70) ^{o}c .The system should also ensure that the liquid level should be at a steady set level as water flows into and out of the tank. This is being controlled by their respective valves. The feedback of the temperature measurement is transmitted by the temperature sensor to the controller. The comparator compares the actual temperature with the set point and the output current of the comparator is the error signal. (i.e. E = Ts-Tm).The error signal goes to the controller. The controller makes the necessary correction and

the output of the controller (4-20 milliampere) is given to the E/P converter. A constant air supply (137.8KN/m^2) is given to the E/P converter. According to the output of the controller, the outflow of air from E/P converter is regulated i.e. pneumatic signal (20.6KN/m^2 - 103.4KN/m^2). The pneumatic signal goes to the control valve and according to the pneumatic signal, the valve position changes to manipulate the water inlet flow to minimize error. The stirrer automatically turns on when the system is at it working condition to establish uniform liquid temperature. The heat is generated by passing hot air through a tube embedded in water whose inlet valve determines the volume of hot air that is passed through the tube.

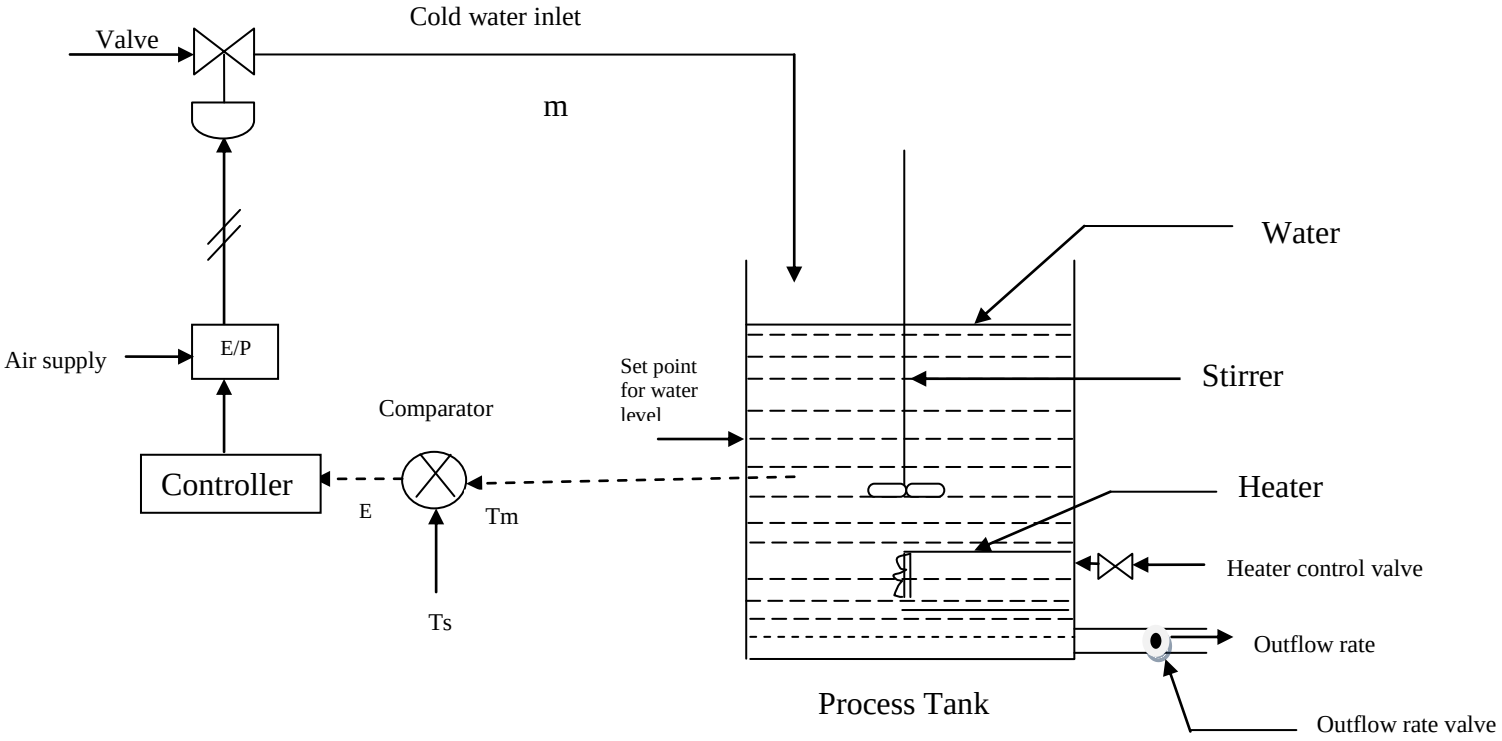


Fig 3.3: PID Continuous Process Control Unit

The equation for the above process is given as below:

$$P = k_p e_p + k_p k_1 \int_0^T e_p dt + k_p k_D \frac{d e_p}{dt} + P_1 (o) \qquad (3.1)$$

where $K_i = 0.7/s$

$K_p = 5$

$K_D = 0.5$ (s)

p_1 (o) = 20%

$e_p = t\%$

T = the interval of time until the system stabilizes.

These values were obtained as shown below:

$$e_p = [(x - x_o)/Kp] \times 100 \qquad (3.2)$$

Assume $e_p = 2\%$ and X=120cm3/s and Xo =110cm3/s

this gives $K_p = 5$

To obtain K_D

$$p(t) = K_D (de_p/dt) \qquad (3.3)$$

p(t) is taking to be 5s and since a positive rate of change of error produces a positive derivative mode output , then K_D is given to be 5s.

To obtain K_i

K_i = - 0.15% of controller output per second per percentage error, this is given to be -4.67/s/%, therefore $K_i = 0.7005/s$

Having obtained these values, it was applied to the three mode equation for controller output (see eqn 3.1)

We have that $p_1=5t+1.75t^2+22.5$; $p_2=3.5(t-1)+1.75t^2+26.75$;

$P_3=-0.875t^2+6.25t+21.625$ (3.4)

3.3 Problem Formulation:

In addition to other limitations stated in section 2.7 PID controllers operate in single input single output (SISO) mode. Also the tuning of the PID variables involves trial and error method and rigorous mathematical procedures.

CHAPTER FOUR

SYSTEM DESIGN

The design stage of the intelligent process control is discussed below.

4.1 The Design of Intelligent Process Control System Using Fuzzy Network

The fuzzy control system design follows the following steps:

i) Obtain the system' operational specifications and inputs and outputs.

ii) Obtain the fuzzy sets for the inputs

iii) Determine the rule set

iv) Determine the defuzzification

v) Run through test suite to validate system and adjust systems as required.

vi) Complete document and release to production.

The detailed discussion of what happens at each step now follows:

4.1.1 Obtaining the system' operational specifications and inputs and outputs:

This involves the Universe of Discourse (Ù) which is the range of all possible values that comprise the input to the fuzzy system and also the output.

It is given by:

$$Ù=UL+UP+UT \quad (4.1)$$

Where UL=fuzzy parameters for liquid level

UP=fuzzy parameters for pressure

UT=fuzzy parameters for temperature

4.1.2 Obtaining the fuzzy sets (U) for the inputs:

Fuzzy set (U) is any set that empowers its members to have different grades of membership in an interval and the values obtained are based on the system adaptation during training. The fuzzy set for liquid level (U_L), pressure (U_P) and temperature (U_T) is depicted below:

U_L= {H5, H4, H3, H2, H1} = {12, 11, 10, 9, 8,} (4.2)

U_P= {P5, P4, P3, P2, P1} = {15, 12, 9, 6, 3} (4.3)

U_T= {T5, T4, T3, T2, T1} = {70, 60, 50, 40, 30.} (4.4)

To determine the fuzzy set for both the input and output variables, we have that:

For change in liquid level (dl) with reference to the set point

H_5 - Very high 12

H_4 - High 11

H_3 - Zero (at set point) 10

H_2 - low 9

H_1 - very Low 8

For valve settings (Vo):

V_w	-	Very Wide	5
W	-	Wide	4
N_m	-	Normal	3
Na	-	Narrow	2
V_{Na}	-	Very Narrow	1

For the outflow rate (Rout) in (cm^3/s):

V_H	-	Very High	150
H	-	High	120
N	-	Normal	90
L	-	Low	60
V_L	-	Very Low	30

The fuzzy inference system for liquid level in MATLAB environment is shown in Figure 4.1a (See Appendix C)

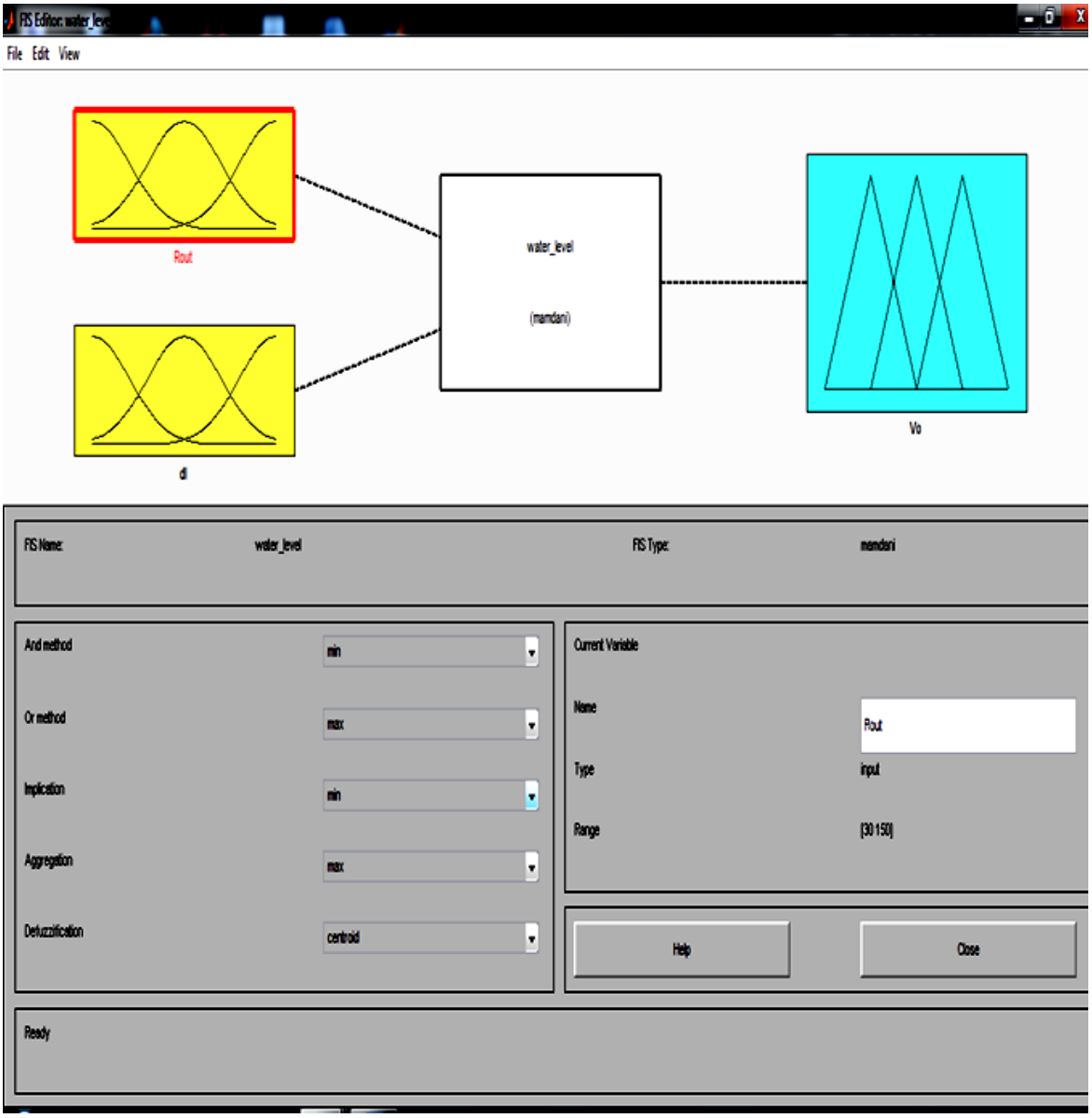


Figure 4.1a: Fuzzy inference system for water level in MATLAB

R_{out} = Rate of outflow

dL = Change in water level

V_o = Valve setting

The fuzzy parameters for temperature in (^oC)

$T_5 = \theta_5 =$ very hot (70)

$T_4 = \theta_4 =$ hot (60)

$T_3 = \theta_3$ = warm (50)

$T_2 = \theta_2$ = cold (40)

$T_1 = \theta_1$ = very cold (30)

The heater settings:

HT_5 = Very High

HT_4= High

HT_3= medium

HT_2= Low

HT_1= Very Low

The heater control settings:

Hc_5 = Very High

Hc_4= High

Hc_3= Normal

Hc_2= Low

Hc_1= Very Low

The fuzzy inference system for temperature in MATLAB environment is shown in Figure 4.1b

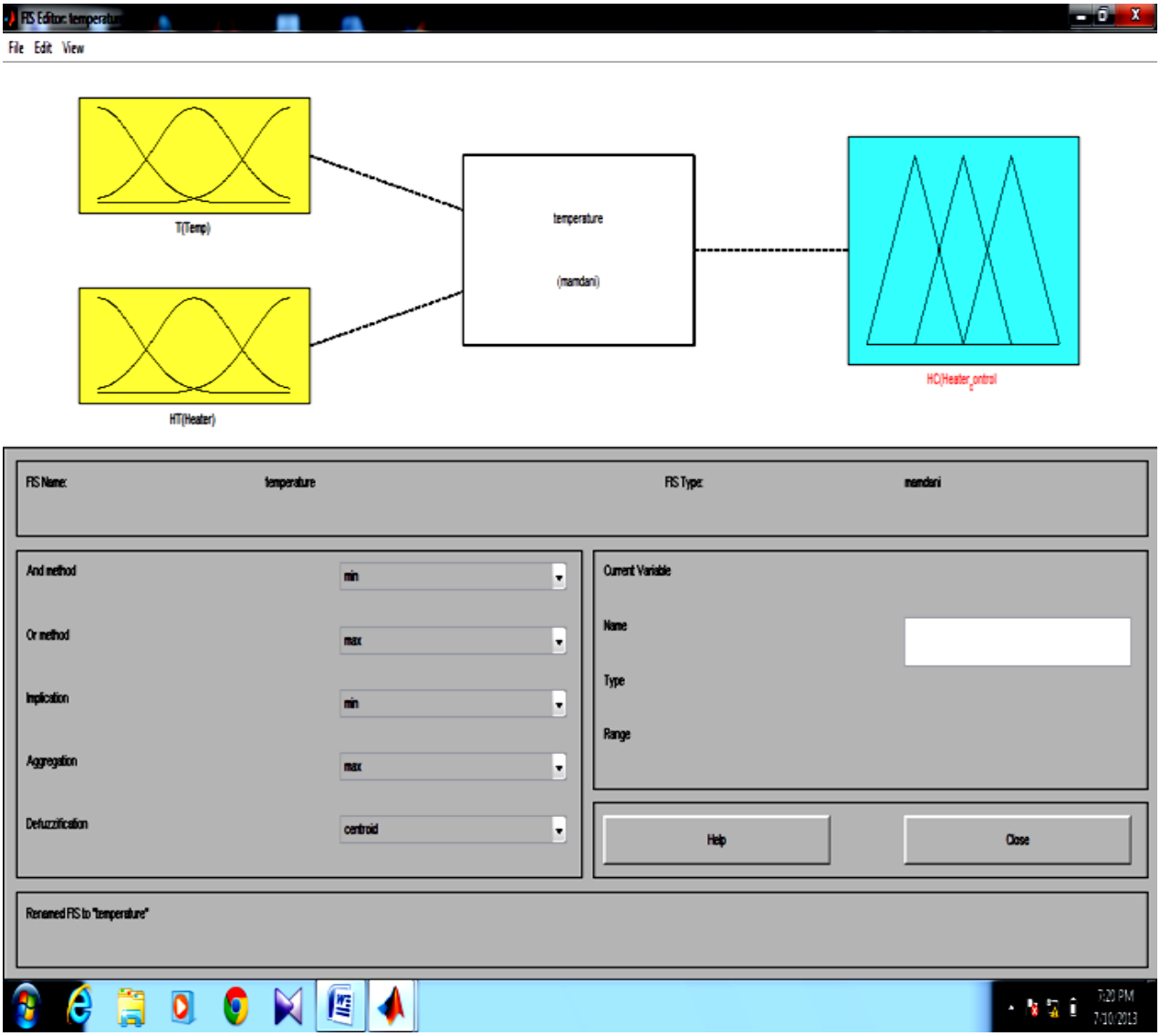


Figure 4.1b: Fuzzy inference system for temperature in MATLAB

The fuzzy parameters for pressure in KN/m^2:

P_5 = Very High 15

P_4 = High 12

P_3 = Normal 9

P_2 = Low 6

P_1 = Very Low 3

4.1.3 Membership functions (U_A)

This is given by

$$U_{APC} = U_{AL} + U_{AT} + U_{AP} \quad (4.5)$$

Where

$$U_{AL} = DL + R_{OUT} + V_O \quad (4.6)$$

$$U_{AT} = T + HT + HC \quad (4.7)$$

$$U_{AP} = P + HL + V_{OP} \quad (4.8)$$

And

$$D_L = \{H_5, H_4, H_3, H_2, H_1\} = \{12, 11, 10, 9, 8\} \quad (4.9)$$

$$R_{OUT} = \{V_H, H, N, L, V_L\} = \{7, 6, 5, 4, 3\} \quad (4.10)$$

$$V_O = \{V_W, W, N, NA, V_{NA}\} = \{5, 4, 3, 2, 1\} \quad (4.11)$$

Using MATLAB, figure 4.2a, fig 4.2b and 4.2c, shows the membership functions of the liquid level, outflow rate, and the valve setting respectively. Membership functions are used in the fuzzification and defuzzification steps of a Fuzzy Logic System to map the non-fuzzy input values to fuzzy linguistic terms and vice versa. A membership function is used to quantify a linguistic term. Note that an important characteristic of fuzzy logic is that a numerical value does not have to be fuzzified using only one membership function. In other words, a value can belong to multiple sets at the same time. For example, figure 4.2a, a liquid level value can be considered as “Very High (H_5)” and “High (H_4)” at the same

time, with different degree of memberships. Note that that H_5 to H_1 are the degree of membership which the system can comprise.

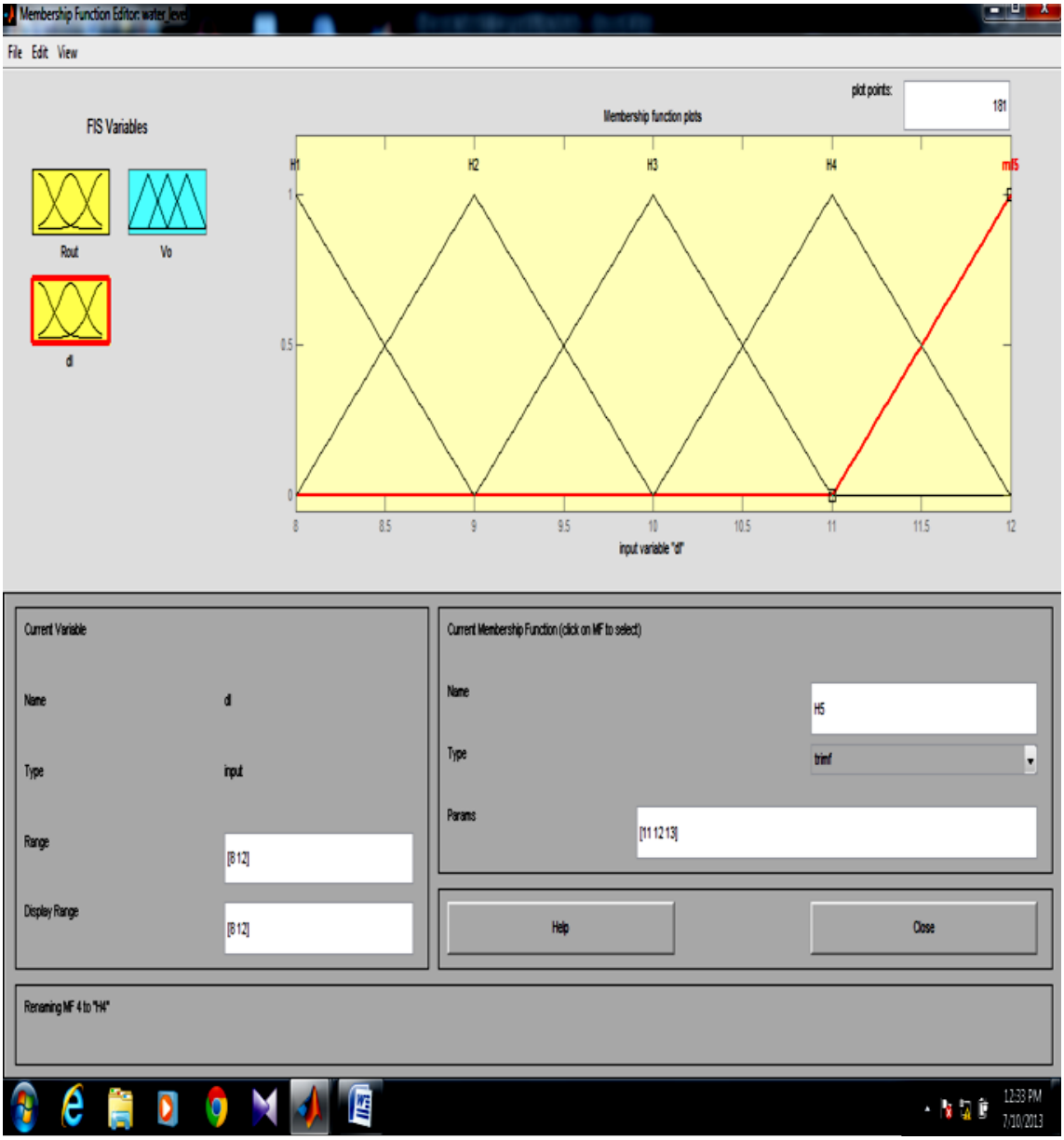


Fig 4.2a: Membership function for liquid level in MATLAB

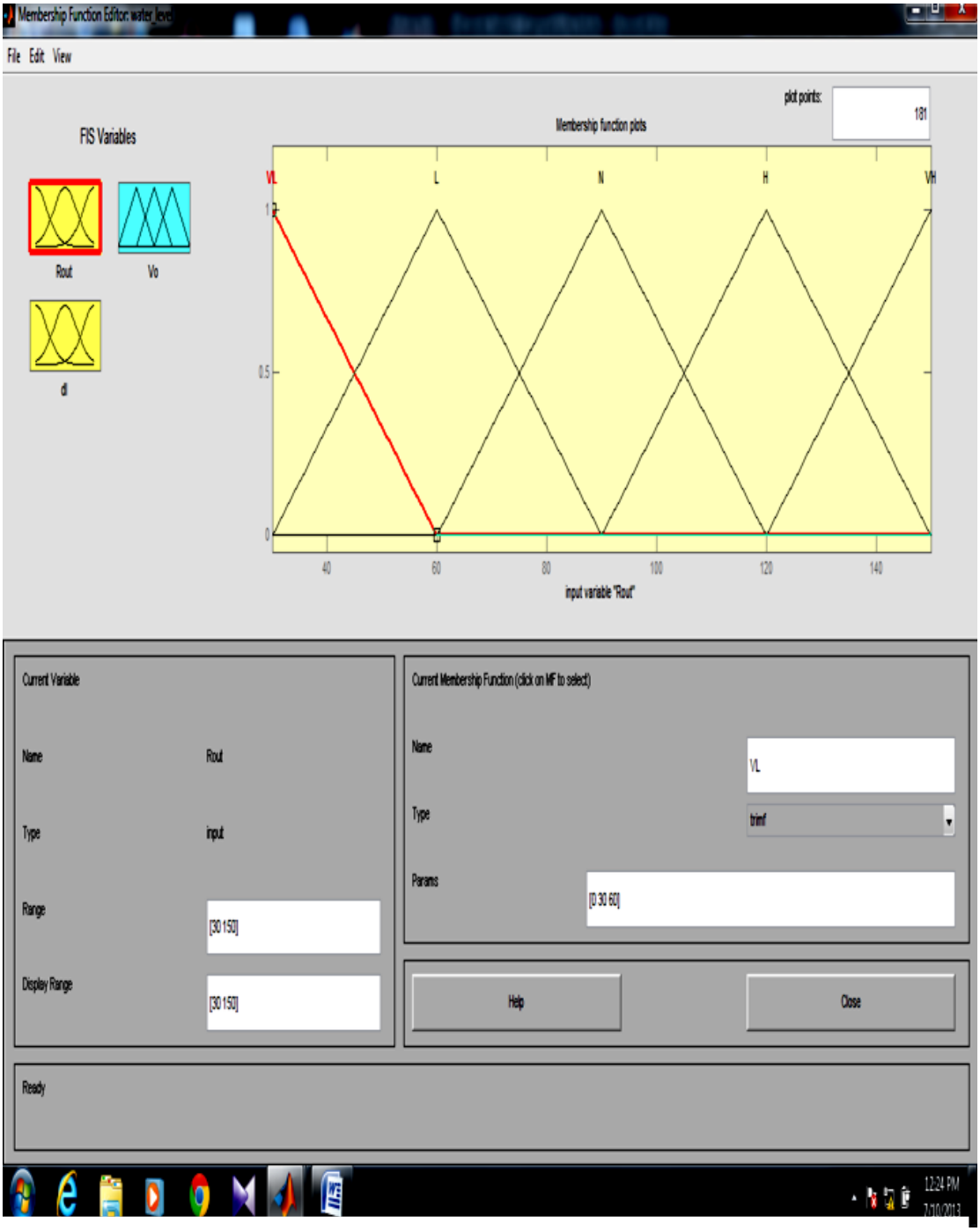


Figure 4.2b: Membership function for outflow rate in MATLAB

In Figure 4.2b, the memberships functions for outflow rate value are "Very High (V_S)", "High (H)", "Normal(N)", Low(L) and Very Low (V_L)" and can be considered as "Very Low" and "Low" at the same time, based on the value it acquired at that point.

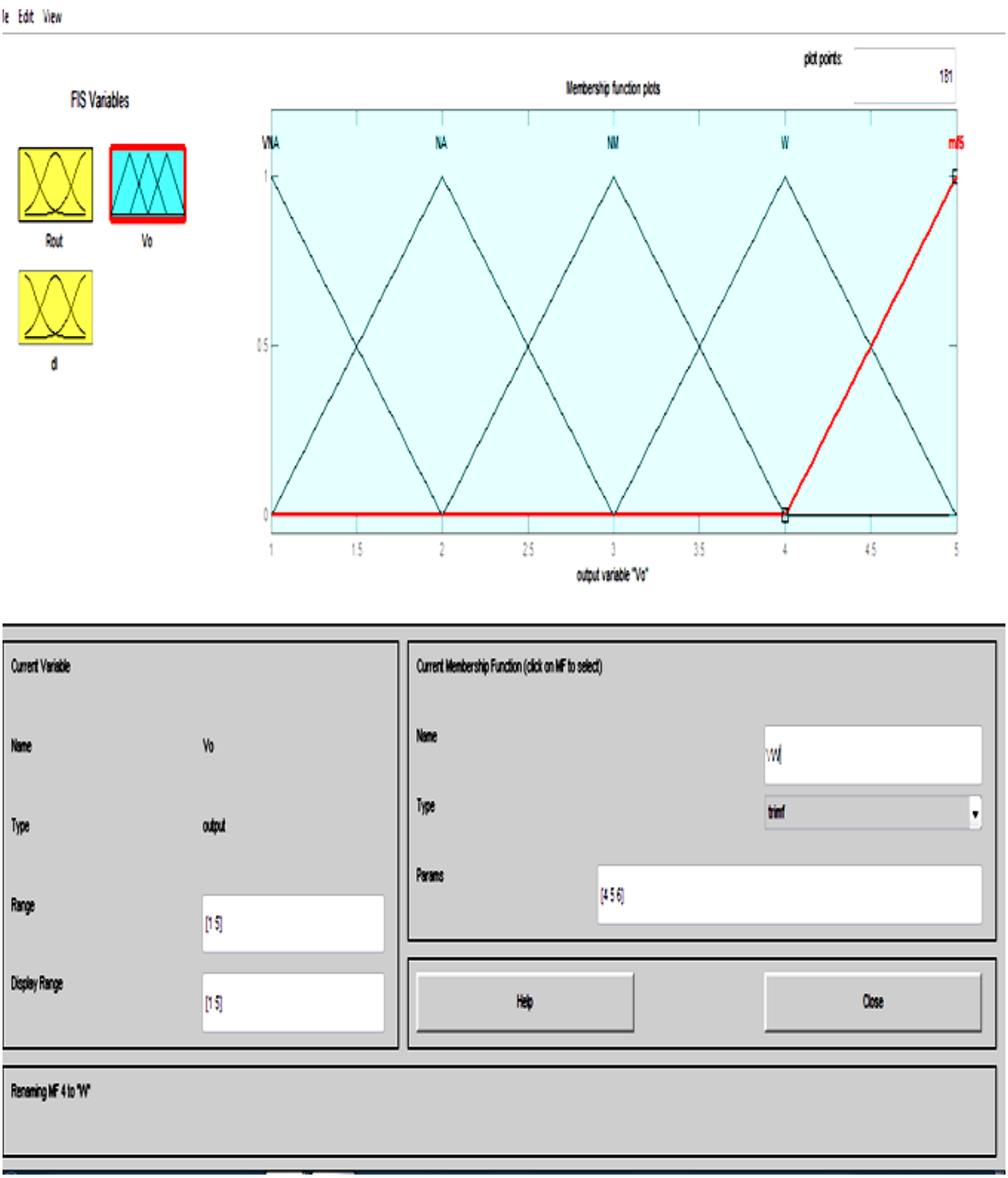


Figure 4.2c: Membership function for valve setting in MATLAB

Figure 4.2c shows the membership functions for valve setting are "Very Wide (V_W)", "Wide (W)", "Normal(N_m)", Narrow(Na) and Very Narrow (V_{Na})" and can be considered as "Very Wide" and "Wide" at the same time, depending on its value at that point.

The fuzzy sets for the temperature (T), Heater (H) and Heater Control (HC) from which the membership functions is formulated is shown below

$T = \{T_5, T_4, T_3, T_2, T_1\} = \{70, 60, 50, 40, 30.\}$ (4.12)

$HT = \{HT_5, HT_4, HT_3, HT_2, HT_1) = \{100, 80, 60, 40, 20\}$ (4.13)

$HC = HC_5, HC_4, HC_3, HC_2, HC_1) = \{5, 4, 3, 2, 1\}$ (4.14)

Figure 4.2d to 4.2f shows the membership functions for temperature, heater and heater control in MATLAB.

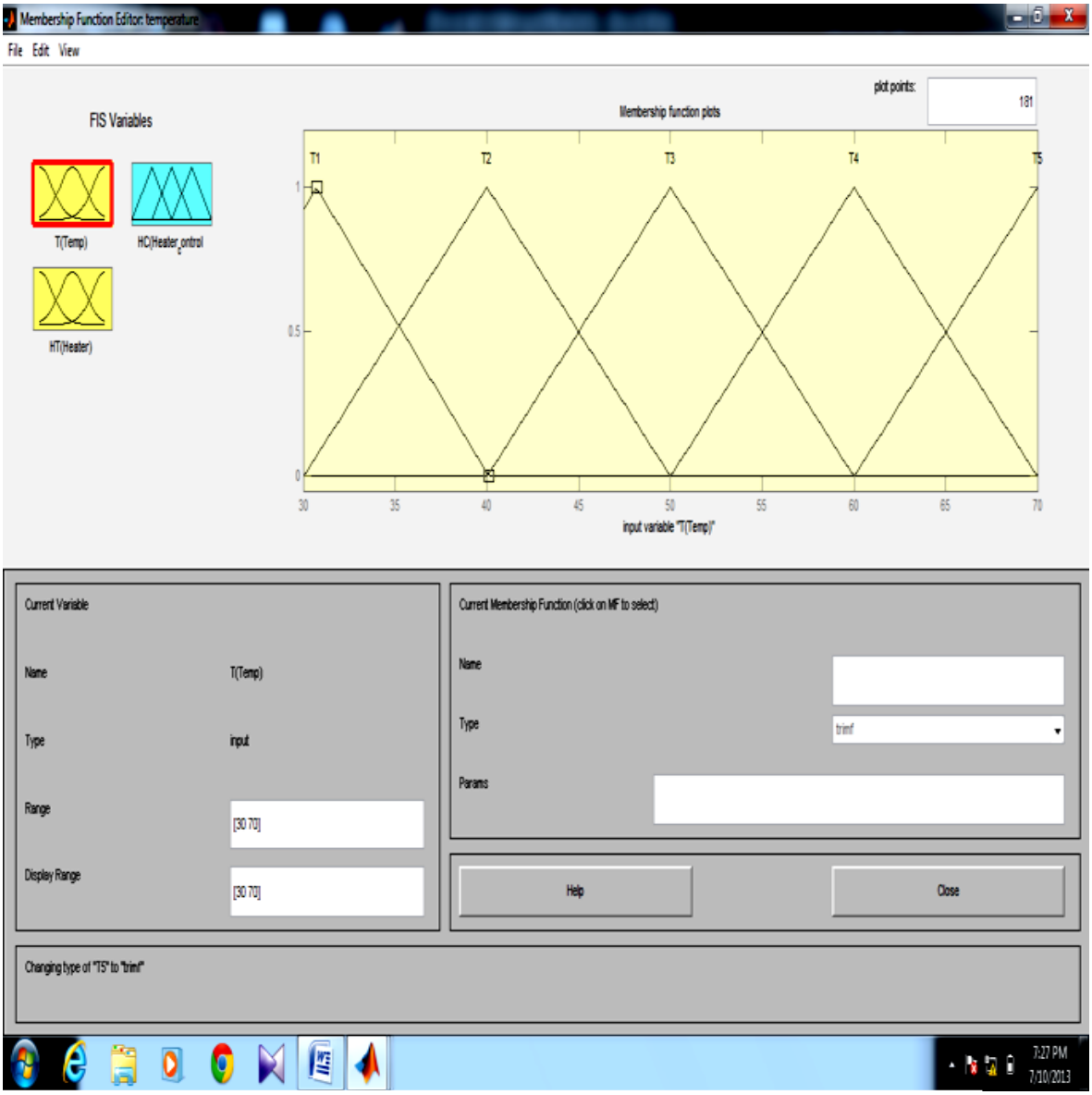


Figure 4.2d: Membership function for temperature in MATLAB

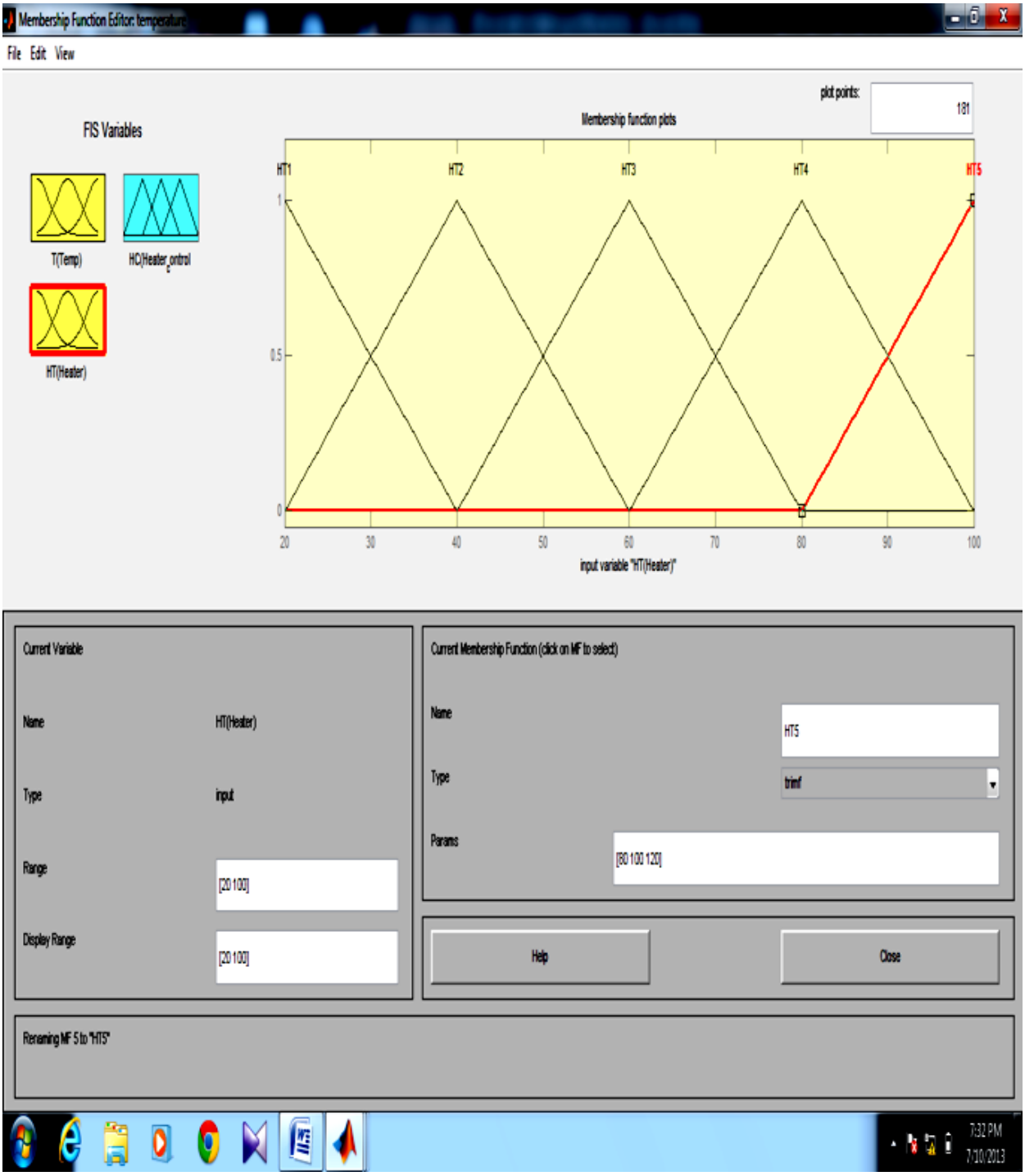


Figure 4.2e: Membership function for the heater in MATLAB

In Figure 4.2e shows the membership function for the heater in MATLAB, the output of the heater control can fall in any point in the fuzzy set axis giving values like 20.5, 80.2, 60.8 e.t.c, The fuzzy

controller cannot take critical decision on what the final output should be and this is where the neural network fine tunes the variables based on its characteristics at that point in time. It approximates it and deduces an action.

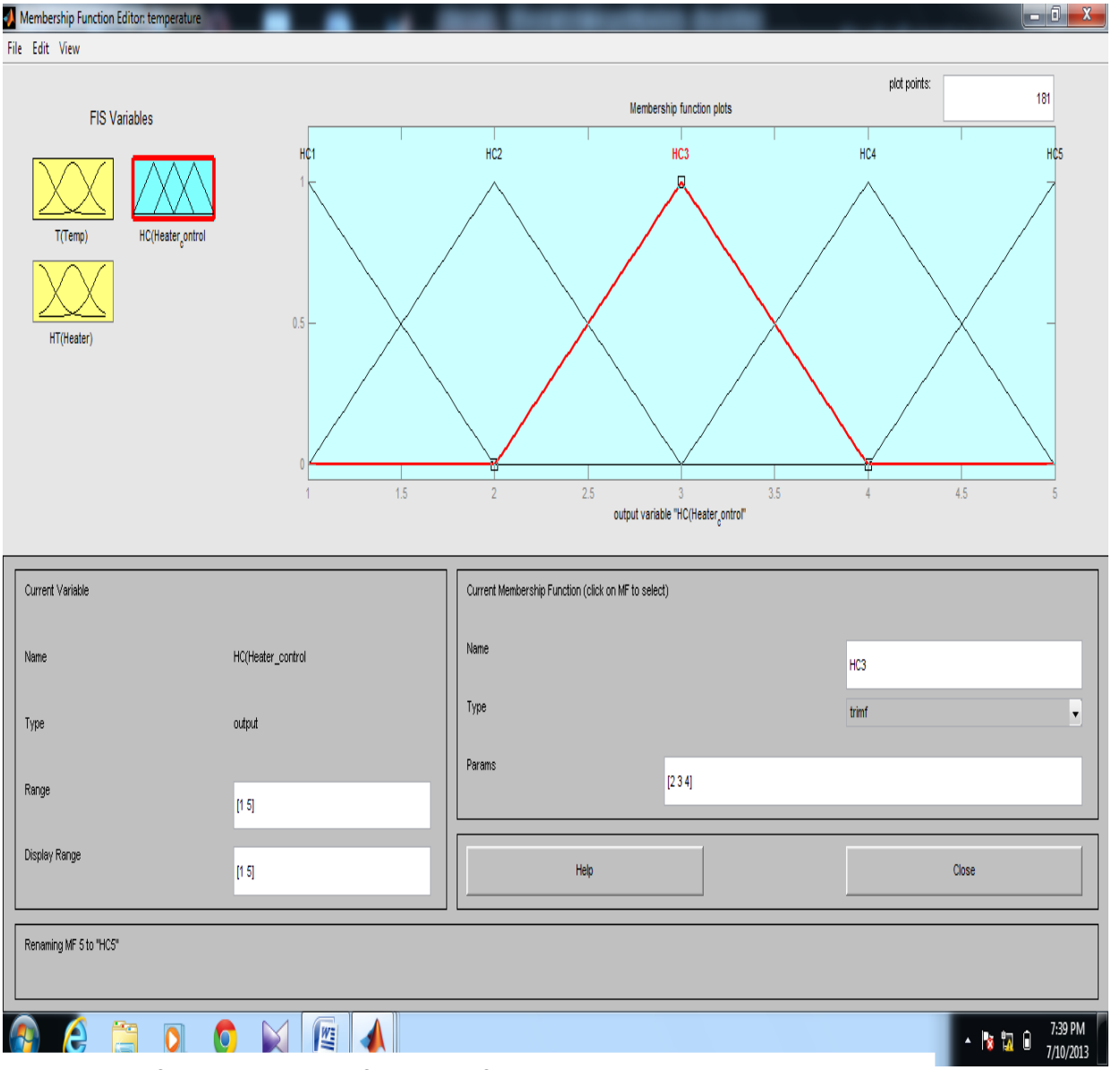


Figure 4.2f: Membership function for heater control in MATLAB

In Figure 4.2f, the membership functions for heater control are “Very High (HC_{5_5})”, “High (HC_4)”, “Normal (HC_3)”, “Low (HC_2)” and “Very Low” (Hc_1)” and can be considered as “Normal” and “Low” at the same time, with different degree of memberships.

The membership functions for the pressure (P), water level (H_L) and pressure valve (V_{OP}) was shown below and Fig 4.2g, 4.2h and 4.2i shows this in MATLAB respectively.

$$P = \{P_5, P_4, P_3, P_2, P_1\} = \{15, 12, 9, 6, 3\} \quad (4.15)$$

$$H_L = \{H_5, H_4, H_3, H_2, H_i = \{12, 11, 10, 9, 8\} \quad (4.16)$$

$$V_{OP} = \{V_W, W, N, NA, V_{NA}\} = \{5, 4, 3, 2, 1\} \quad (4.17)$$

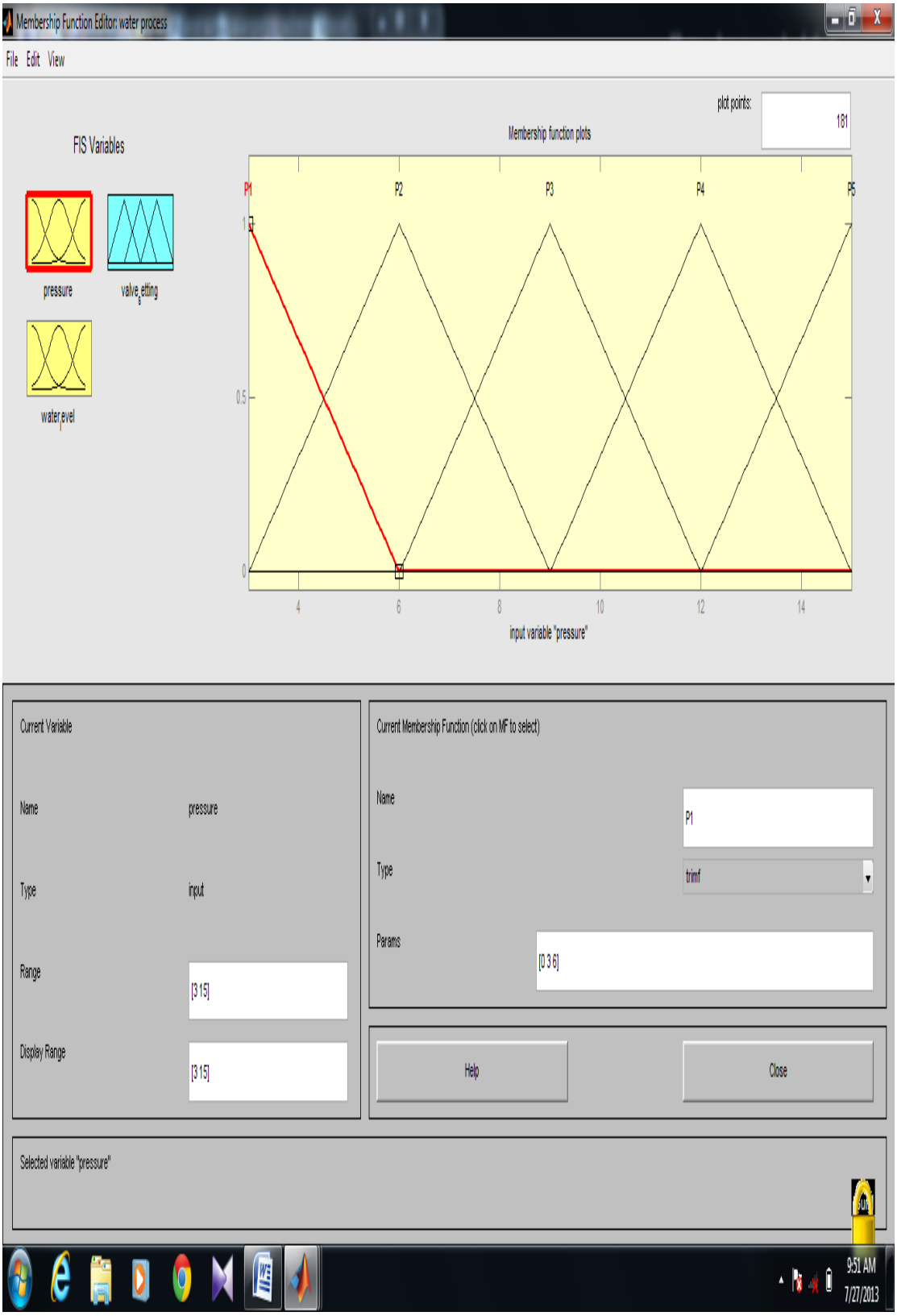


Fig 4.2g: Membership function for pressure in MATLAB

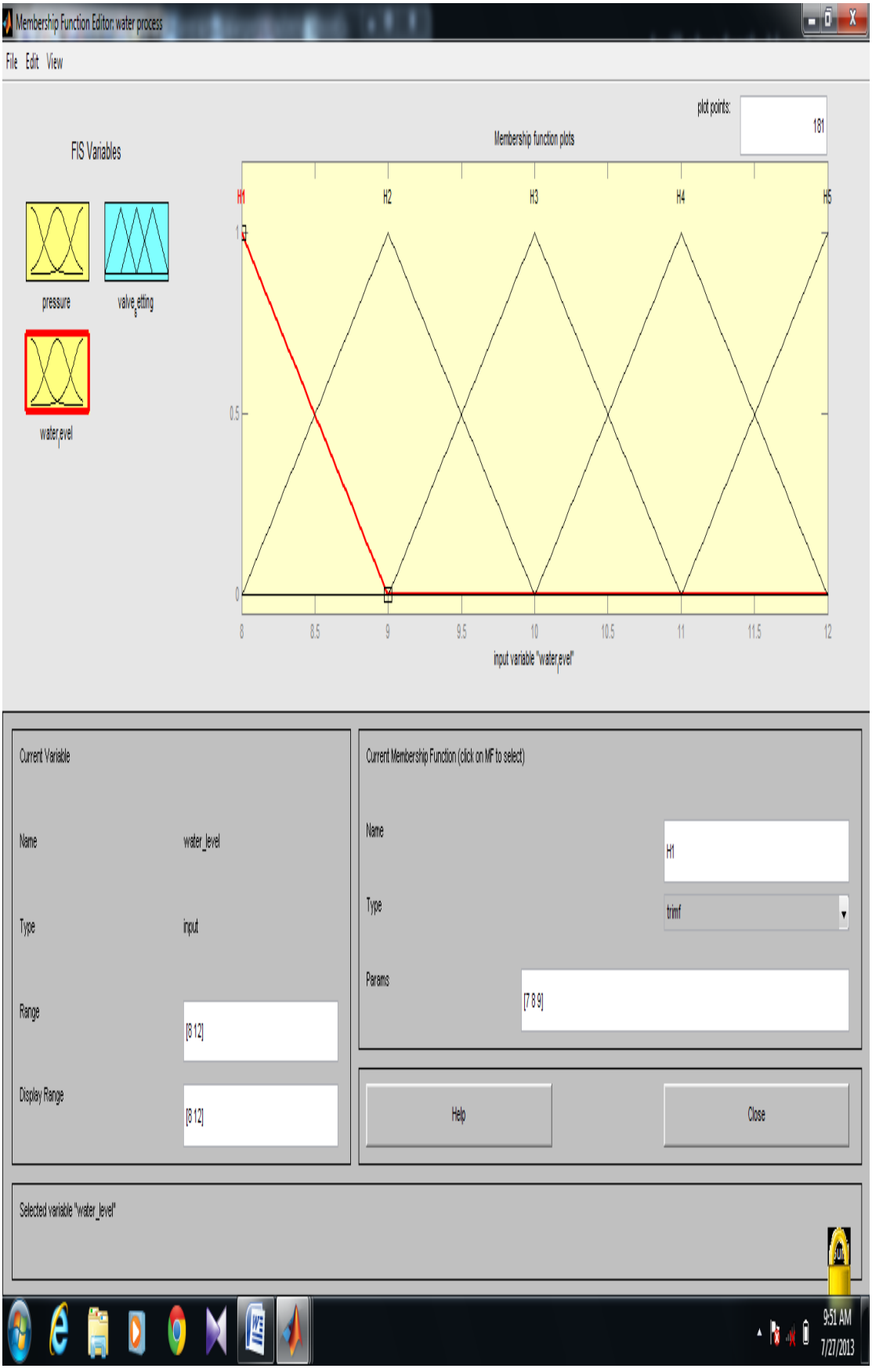


Figure 4.2h: Membership function for water level in MATLAB

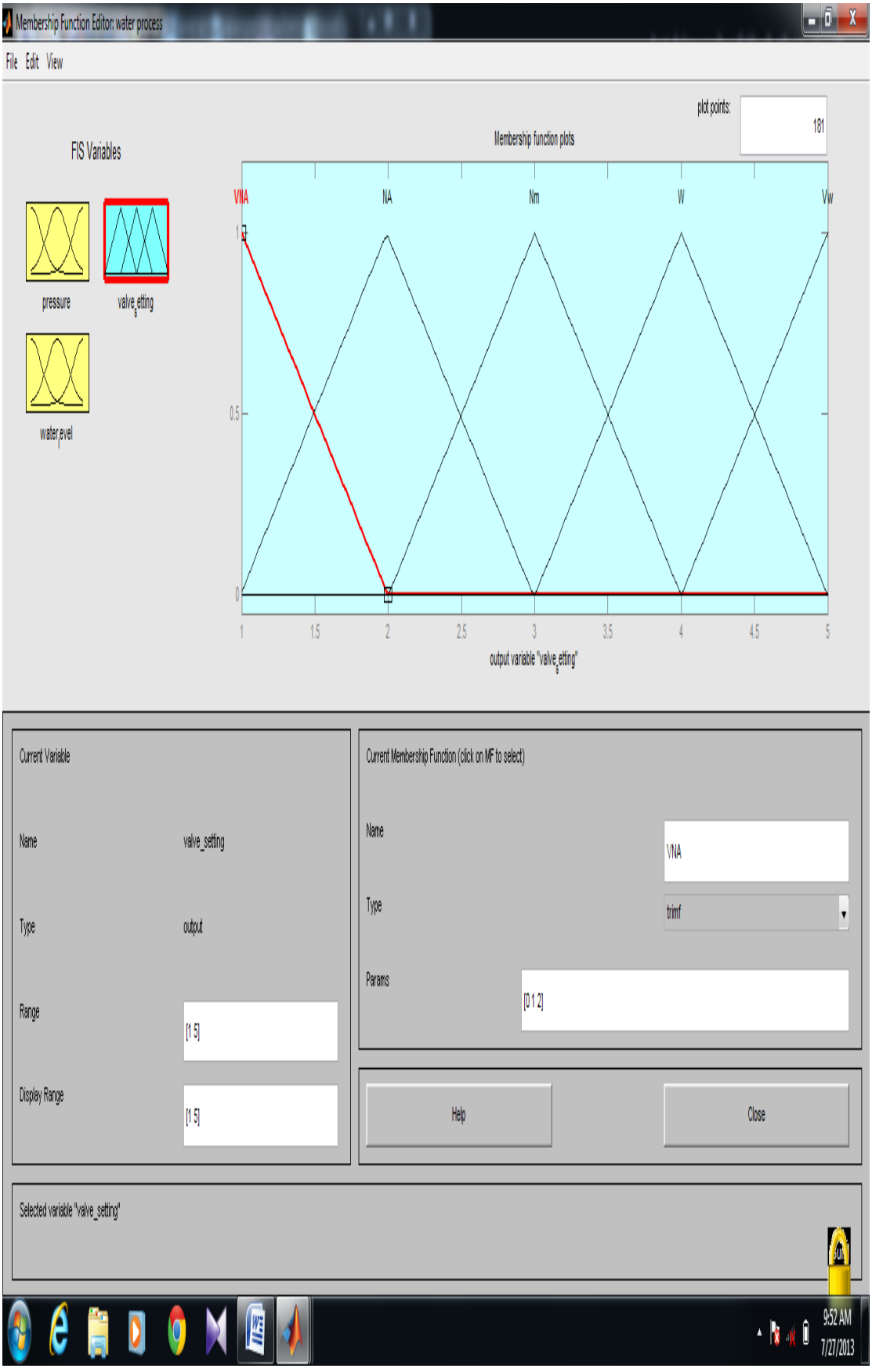


Figure 4.2i: Membership function for valve setting in MATLAB

4.1.4 Depiction of the Fuzzy Sets (f)

$fU_{APC} = (fU_{AL} \cup fU_{AT} \cup fU_{AP})$ (4.18)

where fuzzy sets for liquid level, temperature and pressure is shown in fuzzy set notation below.

$fU_{AL} \rightarrow (D_L \cap R_{OUT}) = V_O$ (4.19)

$fU_{AT} \rightarrow (T \cap HT) = HC$ (4.20)

$fU_{AP} \rightarrow (P \cap HL) = V_{OP}$ (4.21)

4.1.5 Determining the fuzzy rule set: The depiction of the fuzzy sets is used to create a matrix table which makes it easier for the developing of the fuzzy rule. Fig 4.3a, 4.3b, 4.3c and table 4.3a, 4.3b, 4.3c shows the block diagram for the fuzzy control system and matrix table respectively for liquid level, temperature and pressure. The first two blocks contain the fuzzy sets, the second single block represents the fuzzy rule base while the last block depicts the defuzification that is the output of the control respectively for each of the variables under control.

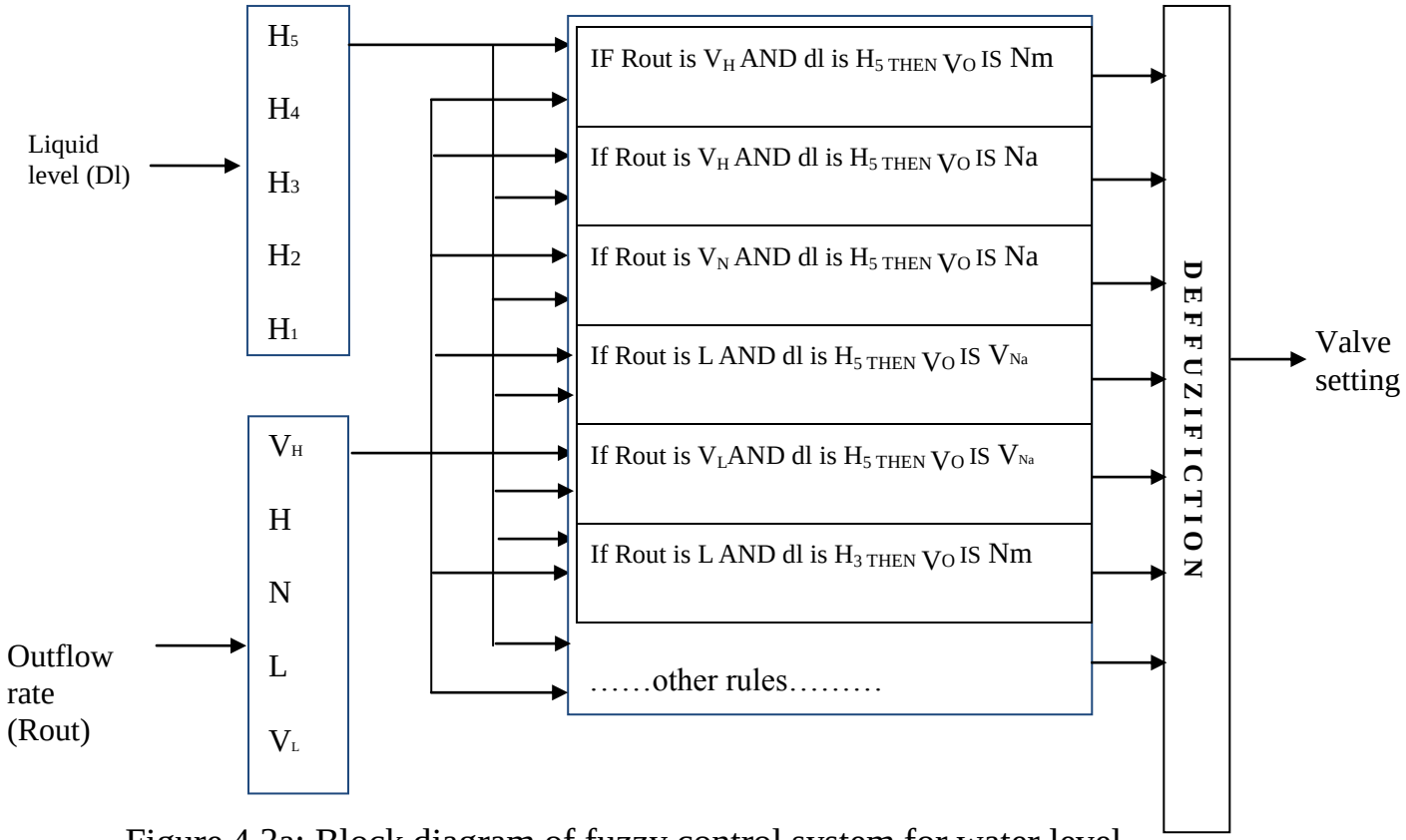


Figure 4.3a: Block diagram of fuzzy control system for water level

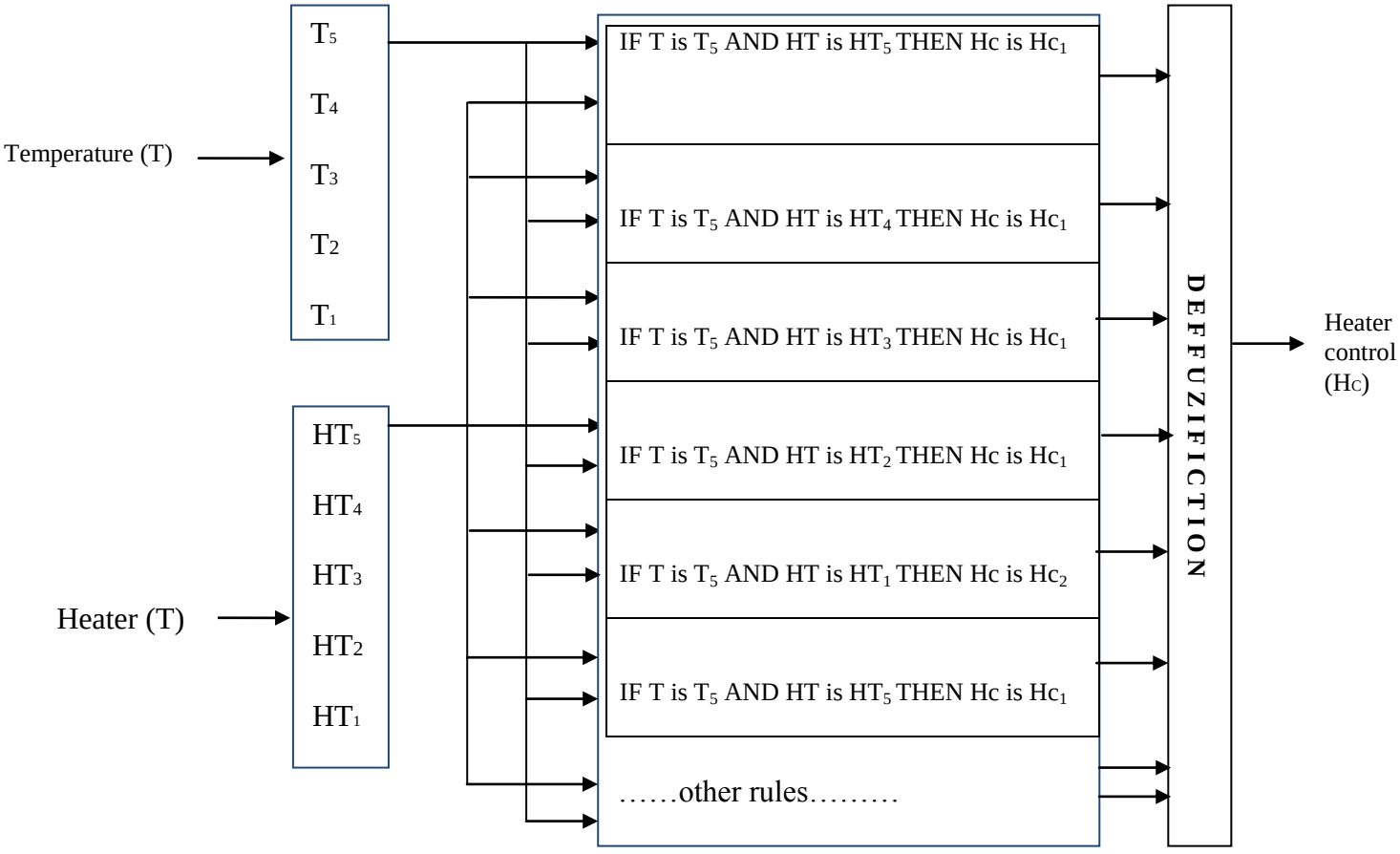


Figure 4.3b: Block diagram of fuzzy control system for temperature

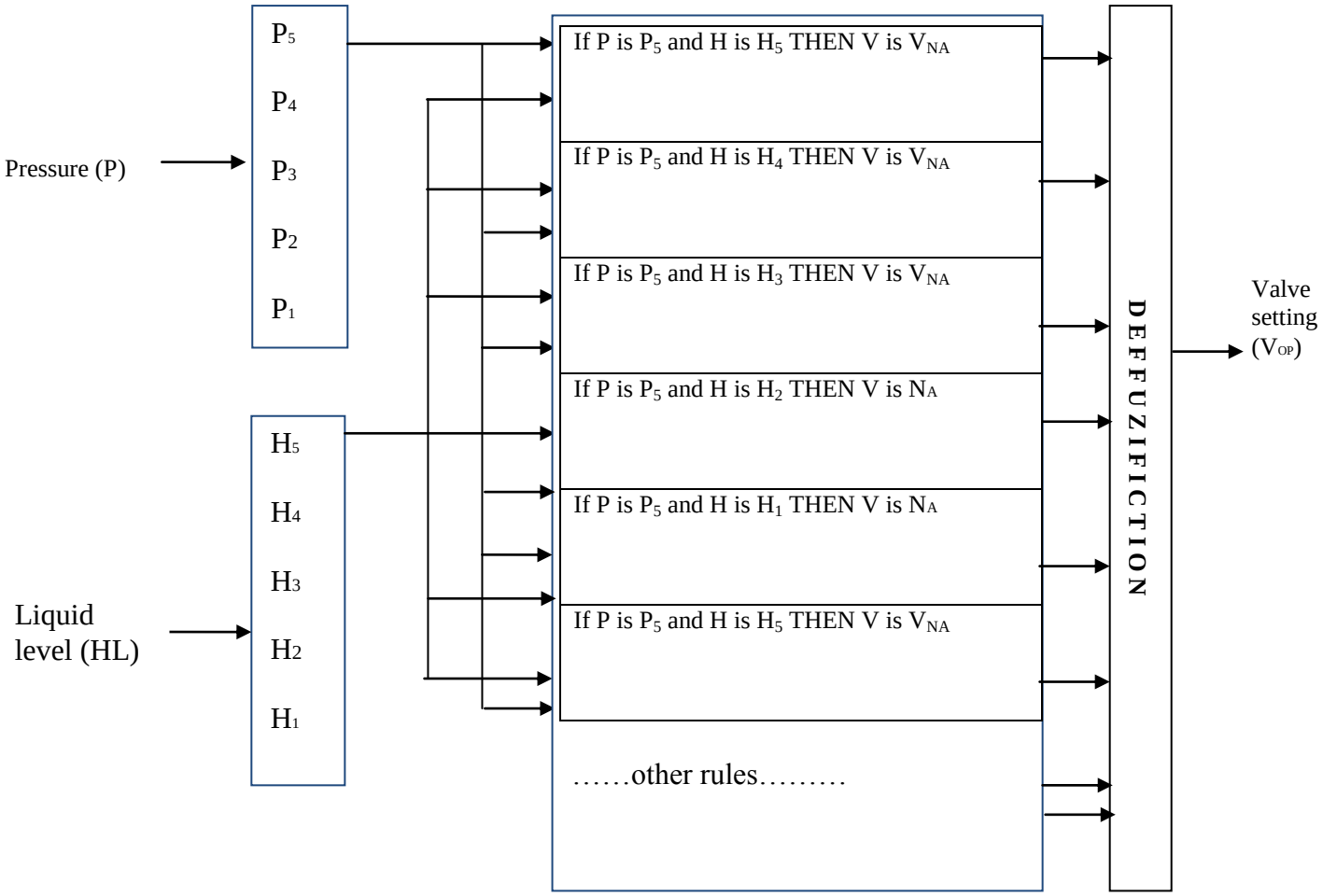


Figure 4.3c: Block diagram of fuzzy control system for pressure

The matrix tables (table 4.1, 4.2, 4.3) are 5 X 5 matrix table with 5 rows and 5 columns in each case. The rows and columns represent one of the inputs each for the parameter being considered whereas the intersection (blocks inside) represents its output. For instance, in the matrix table for water level in table 4.1, the row represents the liquid level, the column represents the outflow rate while the intersection of both (internal blocks) is the output in this case the valve setting. In the matrix table for temperature in table 4.2, the row represents the temperature rating, the column represents the heater while the intersection of both (internal

blocks) is the output in this case the heater control. The matrix table for pressure in table 4.3, the row represents the pressure rating, the column represents the liquid level while the intersection of both (internal blocks) is the output in this case the valve setting for pressure.

TABLE 4.1: Matrix table for outflow rate

Dl / Rout	**V_H**	**H**	**N**	**L**	**V_L**
H_5	N_m	Na	Na	V_{Na}	V_{Na}
H_4	N_m	N_m	Na	V_{Na}	V_{Na}
H_3	V_w	W	N_m	N_a	V_{Na}
H_2	V_w	W	W	N_a	N_m
H_1	V_w	V_W	W	W	W

Table 4.1 was used to generate the 25 fuzzy rules as shown in table 4.5a

TABLE 4.2: Matrix Table for Temperature

T/HT	**HT_5**	**HT_4**	**HT_3**	**HT_2**	**HT_1**
T_5	Hc_1	Hc_1	Hc_1	Hc_1	Hc_2
T_4	Hc_1	Hc_1	Hc_1	Hc_2	Hc_2
T_3	Hc_1	Hc_2	Hc_3	Hc_4	Hc_5
T_2	Hc_1	Hc_2	Hc_3	Hc_4	Hc_5
T_1	Hc_2	Hc_3	Hc_4	Hc_4	Hc_5

TABLE 4.3: Matrix Table for Pressure

P/HL	**H_5**	**H_4**	**H_3**	**H_2**	**H_1**
P_5	V_{NA}	V_{NA}	V_{NA}	N_A	N_A
P_4	V_{NA}	V_{NA}	N_A	N_A	N_A
P_3	V_{NA}	V_{NA}	N_M	W	W
P_2	V_{NA}	N_M	N_M	W	W
P_1	N_M	N_M	W	V_W	V_W

Table 4.2 was used to generate the 25 fuzzy rules as shown in table 4.5b while table 4.3 was used to generate the 25 fuzzy rules as shown in table 4.5c

4.1.6 The Algorithm and Flowchart

The algorithm shown below is the stepwise procedures taken to develop the source code of the (R_{OUT}, D), 5x5 matrix table and the flowchart (fig 4.4a (flowchart for the matrix) shows the diagrammatic representation of the flow or execution of the algorithm. VB.Net software was used to develop a generalized 5x5 matrix table (e_{ij}) for all the variables under consideration. Table 4.4 shows the matrix table used while fig 4.4b shows the matrix table of e_{ij} in VB.Net environment (See Appendix). In the table above e_{ij} where i =1 to 5 and j =1 to5.

1. Start
2. DIM (R_{OUT} , D)
3. For R_O = 1 TO 5
4. For D_I = 1TO 5
5. INPUT (R_O , D_I)
6. NEXT D_I
7. NEXT R_O
8. END

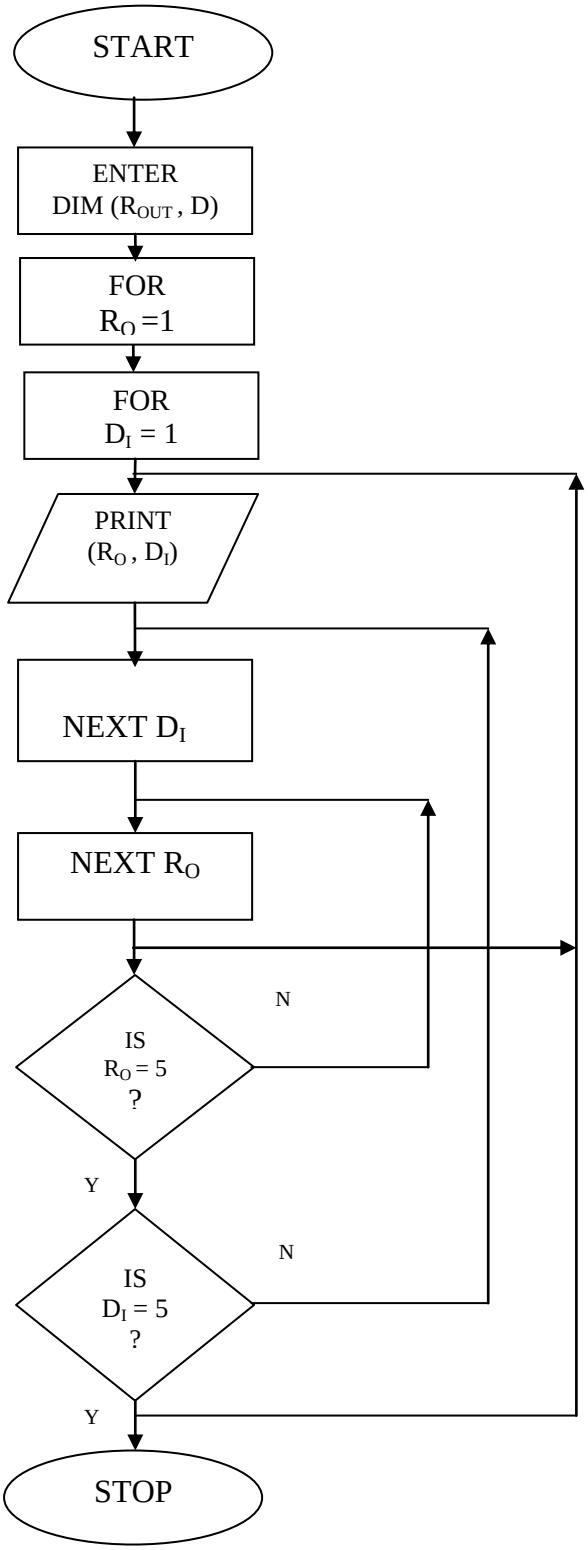


Figure 4.4a: Flowchart of the Matrix

Table 4.4: Matrix table for e_{ij}

Rout /D	**R_{1out}**	**R_{2out}**	**R_{3out}**	**R_{4out}**	**R_5**
D_1	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}
D_2	e_{21}	e_{22}	e_{23}	e_{24}	e_{25}
D_3	e_{31}	e_{32}	e_{33}	e_{34}	e_{35}
D_4	e_{41}	e_{42}	e_{43}	e_{44}	e_{45}
D_5	e_{51}	e_{52}	e_{53}	e_{54}	e_{55}

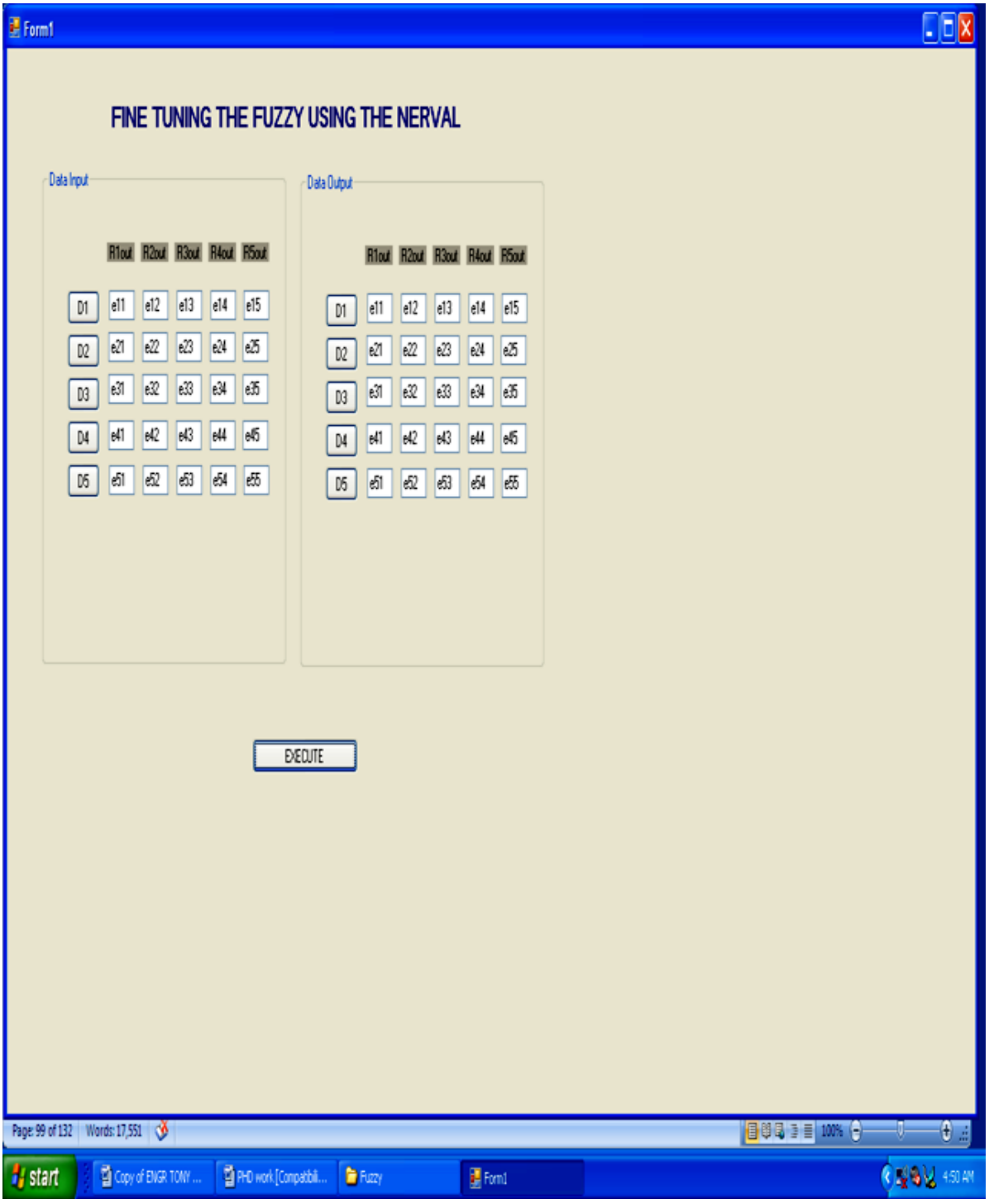


Figure 4.4b Matrix table in VB.Net environment

4.1.7 The Fuzzy Rule Table

Tables 4.5a, 4.5b and 4.5c show the fuzzy rule table for liquid level, temperature, and pressure respectively. They contained possible fuzzy operations in “AND operator” on the sets comparatively.

In Table 4.5a the Rule Base for liquid level/Outflow rate was derived from the Matrix Table of table 4.3a. For example, rule one was formed from IF “column 1” AND “row 1”, THEN the “Intersection” become the output. This is how the twenty five rules were generated.

Table 4.5b shows the Rule Base for temperature and was derived from the Matrix Table of table 4.3b. For example, rule one was formed from IF “IF T(Temperature) is T_5 and HT(Heater) is HT_5THEN Hc is Hc_1 (which is the output). Rule two was formed from IF “IF T (Temperature) is T_5 and HT (Heater) is HT_4THEN Hc is Hc_1). Rule three was formed from IF “IF T (Temperature) is T_5 and HT (Heater) is HT_3THEN Hc is Hc_2 and etcetera. This is also how the twenty- five rules for each of the variables were generated.

However, Table 4.5c shows the Rule Base for pressure and was derived from the Matrix Table of table 4.3c just as discussed above.

TABLE 4.5a: Rule Base for Outflow rate

RULE	CONDITION	ANTECEDENT
1.	If Rout is V_H AND dl is H_5	THEN V_o is N_m
2.	If Rout is V_H AND dl is H_5	THEN V_o is N_a
3.	If Rout is V_N AND dl is H_5	THEN V_o is N_a
4	If Rout is L AND dl is H_5	THEN V_o is V_{Na}
5	If Rout is V_LAND dl is H_5	THEN V_o is V_{Na}
6	If Rout is V_H AND dl is H_4	THEN V_o is N_m
7	If Rout is H AND dl is H_4	THEN V_o is N_m
8	If Rout is N AND dl is H_4	THEN V_o is N_a
9.	If Rout is L AND dl is H_4	THEN V_o is V_{ma}
10	If Rout is VL AND dl is H_4	THEN V_o is V_{Na}
11	If Rout is V_H AND dl is H_3	THEN V_o is V_m
12	If Rout is H AND dl is H_3	THEN V_o is W
13	If Rout is N AND dl is H_3	THEN V_o is N_m
14	If Rout is L AND dl is H_3	THEN V_o is N_a
15	If Rout is V_L AND dl is H_3	THEN V_o is V_{Na}
16	If Rout is V_H AND dl is H_2	THEN V_o is V_w
17	If Rout is H AND dl is H_2	THEN V_o is W
18	If Rout is N AND dl is H_2	THEN V_o is W
19	If Rout is L AND dl is H_2	THEN V_o is N_a
20	If Rout is V_L AND dl is H_2	THEN V_o is N_m
21	If Rout is V_H AND dl is H_1	THEN V_o is V_m
22	If Rout is H AND dl is H_1	THEN V_o is V_w
23	If Rout is N AND dl is H_1	THEN V_o is W
24.	If Rout is L AND dl is H_1	THEN V_o is W
25	If Rout is V_L AND dl is H_1	THEN V_o is W

TABLE 4.5b: Rule Base for Temperature

RULE	CONDITION	ANTECEDENT
1.	If T is T_5 and HT is HT_5	THEN Hc is Hc_1
2.	If T is T_5 and HT is HT_4	THEN Hc is Hc_1
3.	If T is T_5 and HT is HT_3	THEN Hc is Hc_1
4	If T is T_5 and HT is HT_2	THEN Hc is Hc_1
5	If T is T_5 and HT is HT_1	THEN Hc is Hc_2
6	If T is T_4 and HT is HT_5	THEN Hc is Hc_1
7	If T is T_4 and HT is HT_4	THEN Hc is Hc_1
8	If T is T_4 and HT is HT_3	THEN Hc is Hc_1
9.	If T is T_4 and HT is HT_2	THEN Hc is Hc_2
10	If T is T_4 and HT is HT_1	THEN Hc is Hc_2
11	If T is T_3 and HT is HT_5	THEN Hc is Hc_1
12	If T is T_3 and HT is HT_4	THEN Hc is Hc_2
13	If T is T_3 and HT is HT_3	THEN Hc is Hc_3
14	If T is T_3 and HT is HT_2	THEN Hc is Hc_4
15	If T is T_3 and HT is HT_1	THEN Hc is Hc_5
16	If T is T_2 and HT is HT_5	THEN Hc is Hc_1
17	If T is T_2 and HT is HT_4	THEN Hc is Hc_2
18	If T is T_2 and HT is HT_3	THEN Hc is Hc_3
19	If T is T_2 and HT is HT_2	THEN Hc is Hc_4
20	If T is T_2 and HT is HT_1	THEN Hc is Hc_5
21	If T is T_1 and HT is HT_5	THEN Hc is Hc_2
22	If T is T_1 and HT is HT_4	THEN Hc is Hc_3
23	If T is T_1 and HT is HT_3	THEN Hc is Hc_4
24.	If T is T_1 and HT is HT_2	THEN Hc is Hc_4
25	If T is T_1 and HT is HT_1	THEN Hc is Hc_5

TABLE 4.5c: Rule Base for Pressure

RULE	CONDITION	ANTECEDENT
1.	If P is P_5 and H is H_5	THEN V is V_{NA}
2.	If P is P_5 and H is H_4	THEN V is V_{NA}
3.	If P is P_5 and H is H_3	THEN V is V_{NA}
4	If P is P_5 and H is H_2	THEN V is N_A
5	If P is P_5 and H is H_1	THEN V is N_A
6	If P is P_4 and H is H_5	THEN V is V_{NA}
7	If P is P_4 and H is H_4	THEN V is V_{NA}
8	If P is P_4 and H is H_3	THEN V is N_A
9.	If P is P_4 and H is H_2	THEN V is N_A
10	If P is P_4 and H is H_1	THEN V is N_A
11	If P is P_3 and H is H_5	THEN V is V_{NA}
12	If P is P_3 and H is H_4	THEN V is V_{NA}
13	If P is P_3 and H is H_3	THEN V is N_M
14	If P is P_3 and H is H_2	THEN V is W
15	If P is P_3 and H is H_1	THEN V is W
16	If P is P_2 and H is H_5	THEN V is V_{NA}
17	If P is P_2 and H is H_4	THEN V is N_M
18	If P is P_2 and H is H_3	THEN V is N_M
19	If P is P_2 and H is H_2	THEN V is W
20	If P is P_2 and H is H_1	THEN V is W
21	If P is P_1 and H is H_5	THEN V is N_M
22	If P is P_1 and H is H_4	THEN V is N_M
23	If P is P_1 and H is H_3	THEN V is W
24.	If P is P_1 and H is H_2	THEN V is V_W
25	If P is P_1 and H is H_1	THEN V is V_W

The fuzzy rule for liquid level, pressure and temperature was implemented respectively in the MATLAB environment. The evaluations of the fuzzy rules and the combination of the results of the individual rules are performed using fuzzy set operations.

Figures 4.5a, 4.5c and 4.5e show the fuzzy rule editor while figures 4.5b, 4.5d and 4.5f show the surface viewer of the fuzzy rule in MATLAB.

Based on the descriptions of the input and output variables defined earlier on, the Rule Editor allows you to construct the rule statements automatically, by clicking on and selecting one item in each input variable box, one item in each output box, and one connection item based on the fuzzy rule for the process under control in this case “AND”. Rules may be changed, deleted, or added, by clicking on the appropriate button.

The Surface Viewer represents the mapping of the variables that is outflow rate, temperature and pressure in each case. Since this is a three-input three-output case, we can see the entire mapping in one plot as they generate three-dimensional plots that MATLAB can adeptly manage

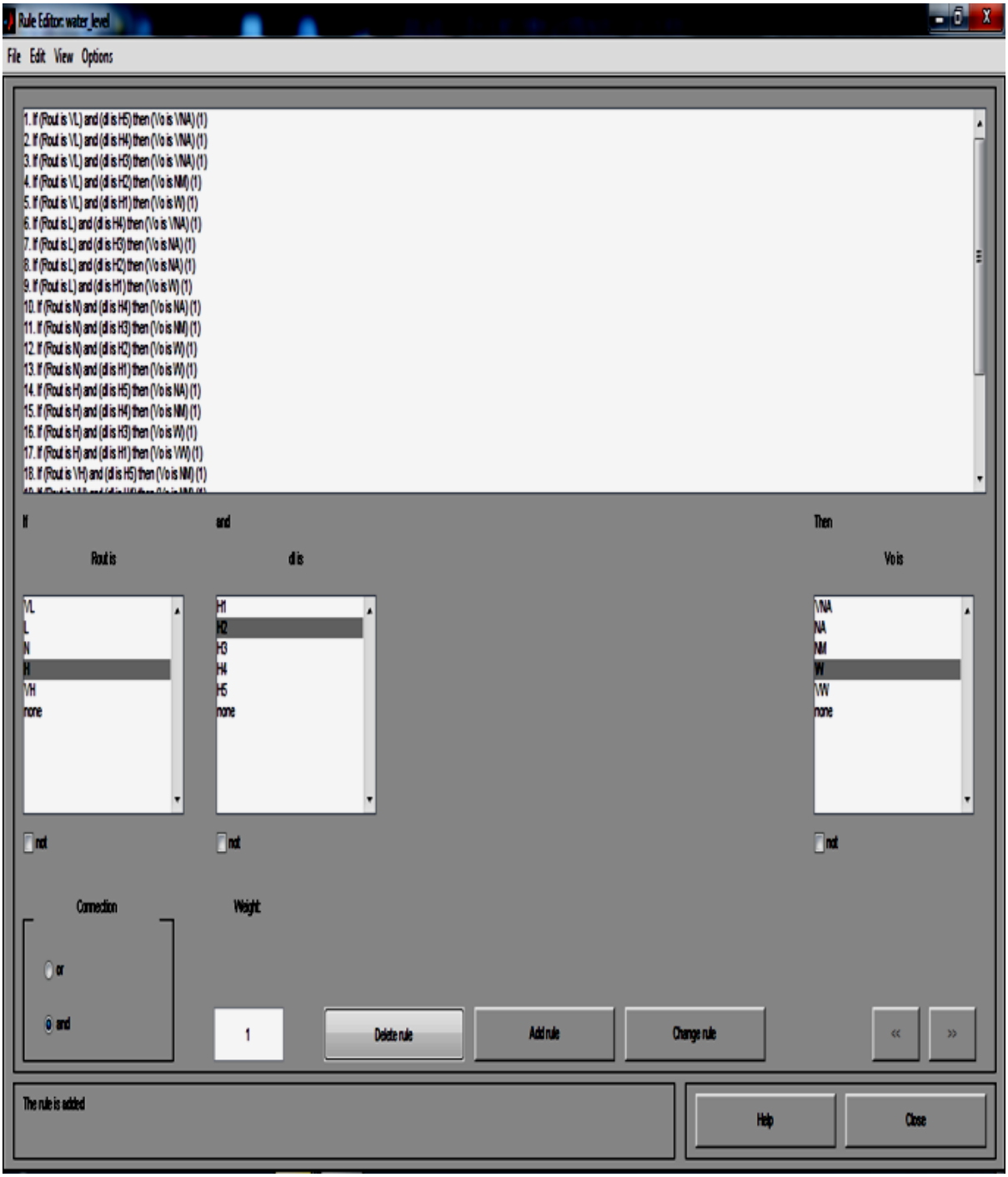


Figure 4.5a Fuzzy rule editor for water level in MATLAB

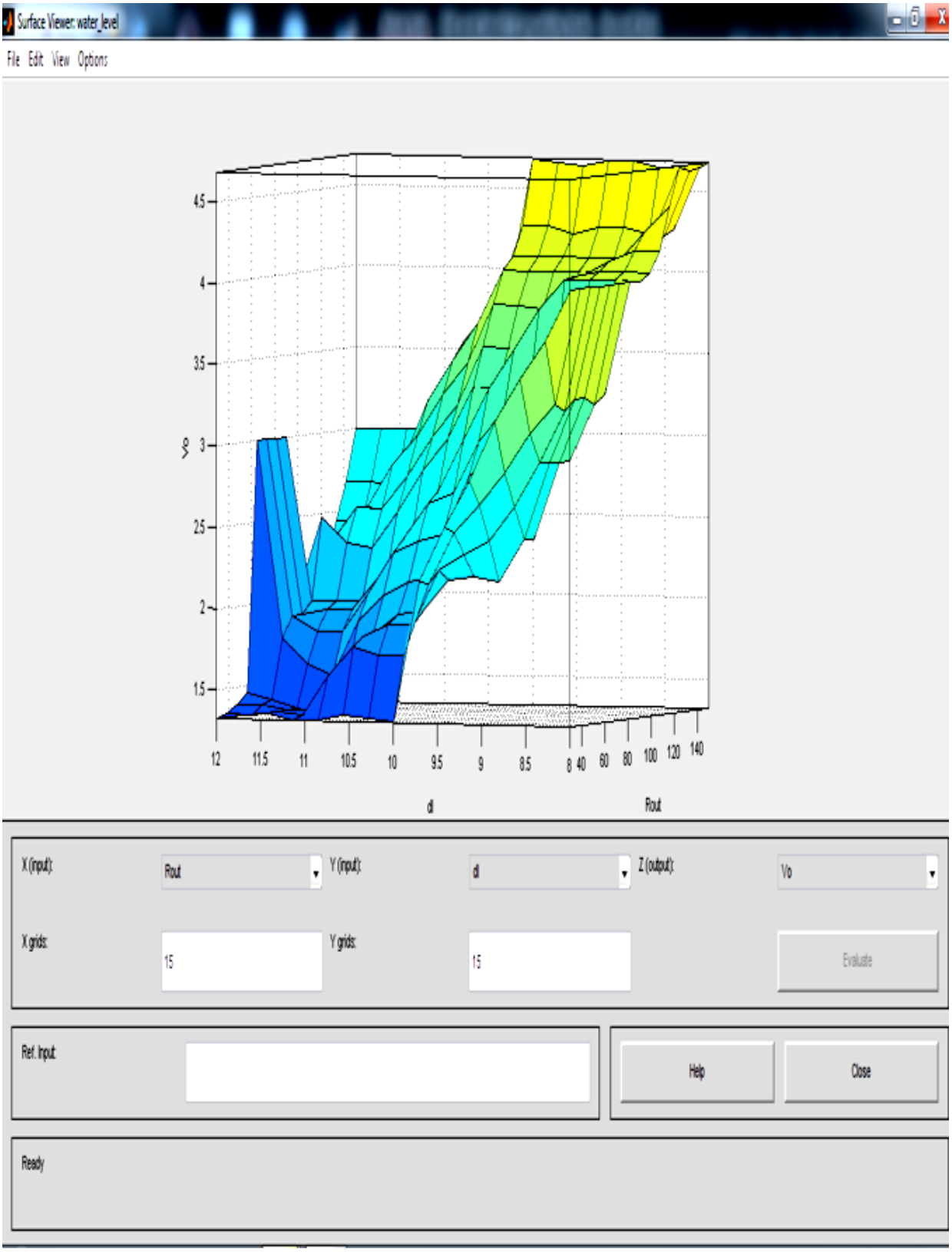


Figure 4.5b Fuzzy rule surface viewer for water level in MATLAB

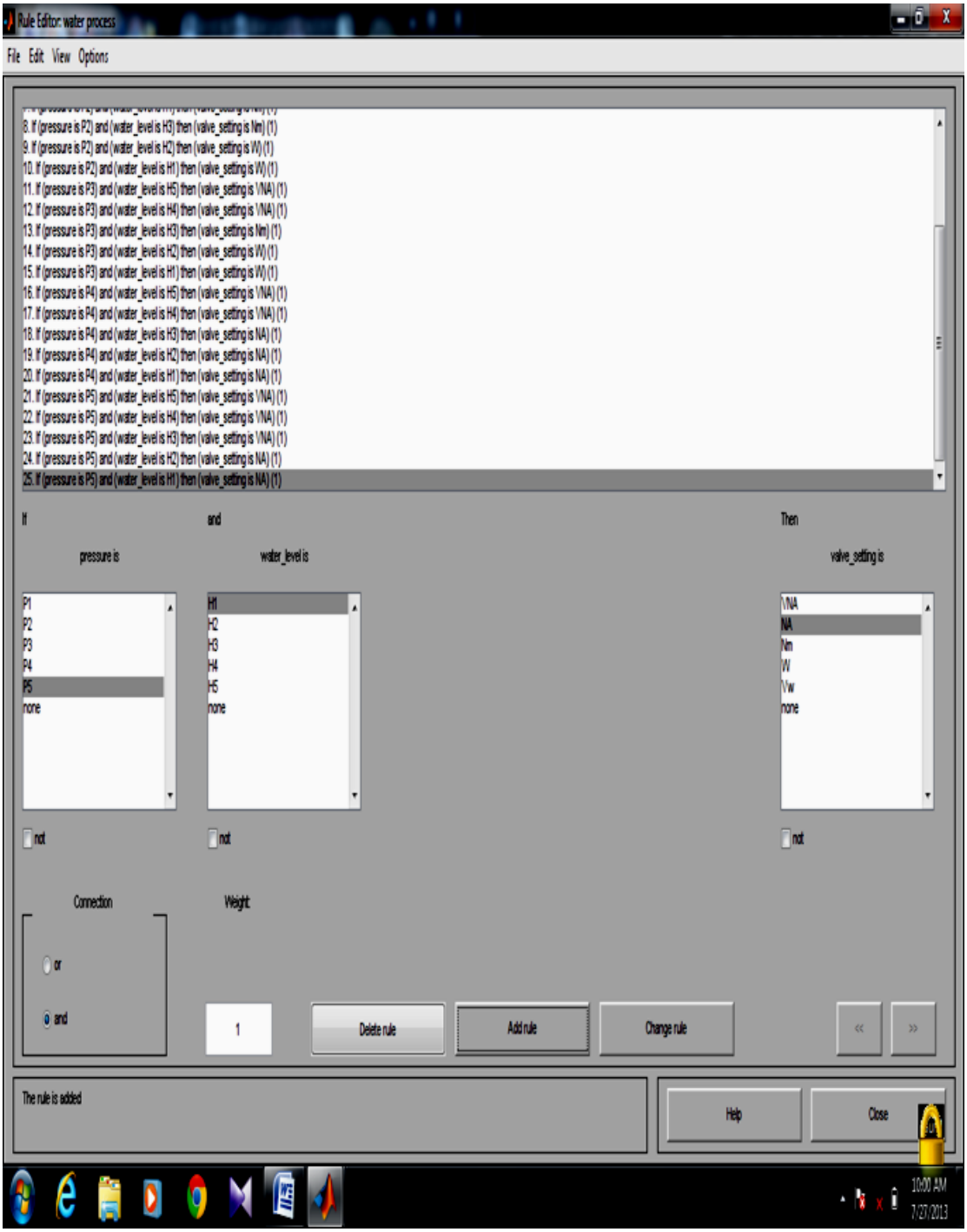


Figure 4.5c: Fuzzy rule editor for pressure in MATLAB

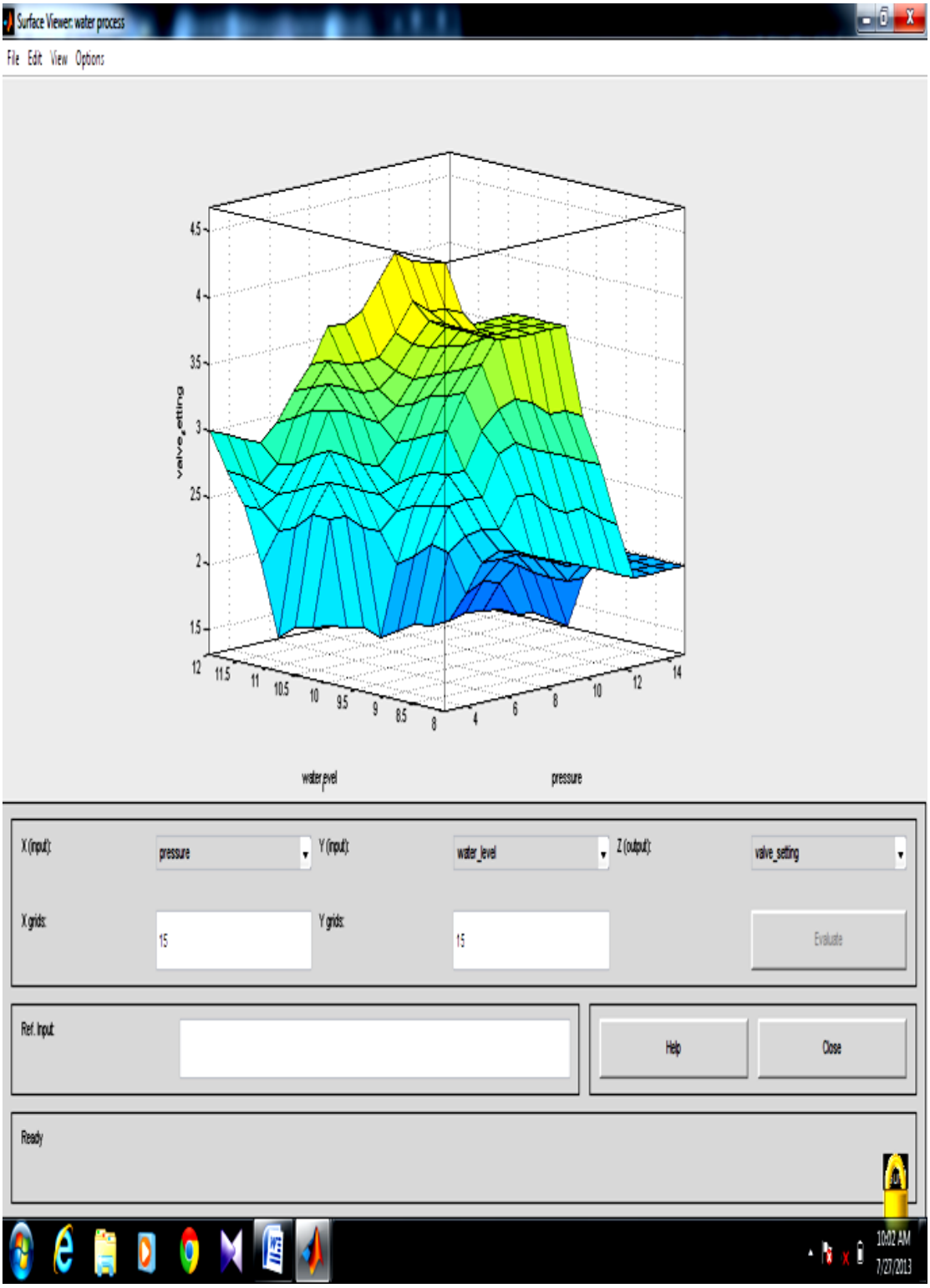


Fig 4.5d: Surface viewer of Fuzzy rule for pressure in MATLAB

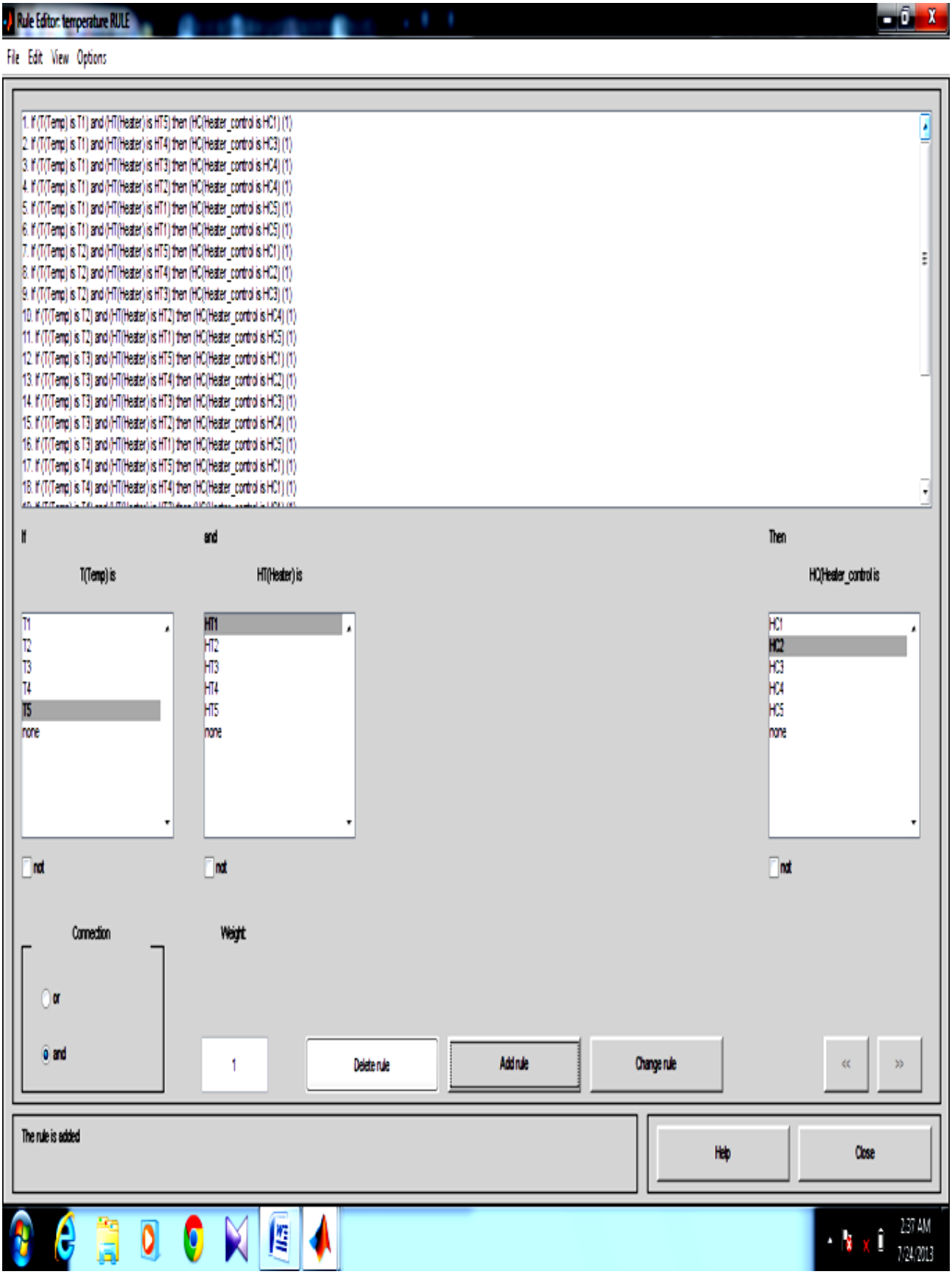


Figure 4.5e: Fuzzy rule editor for temperature in MATLAB

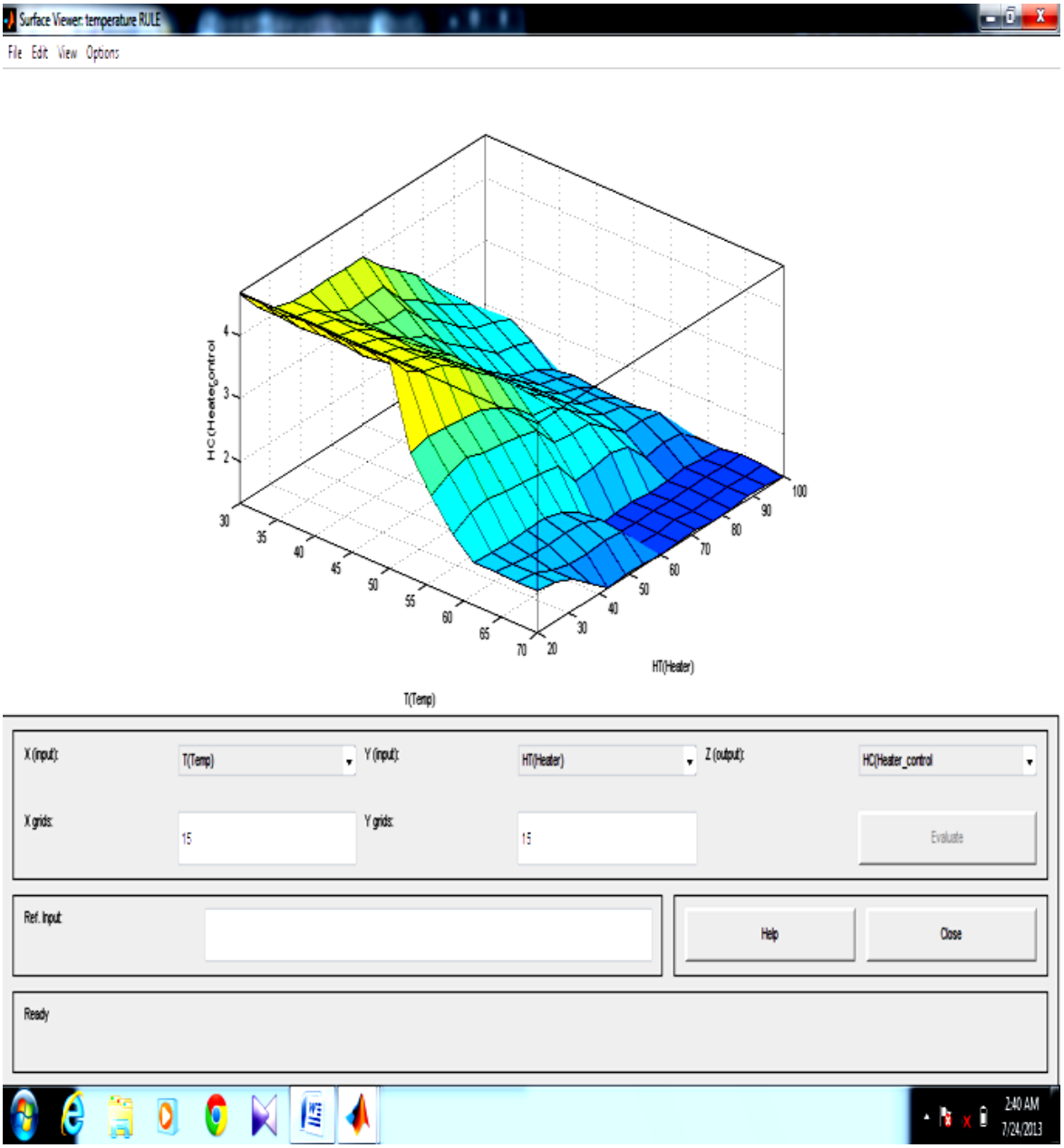
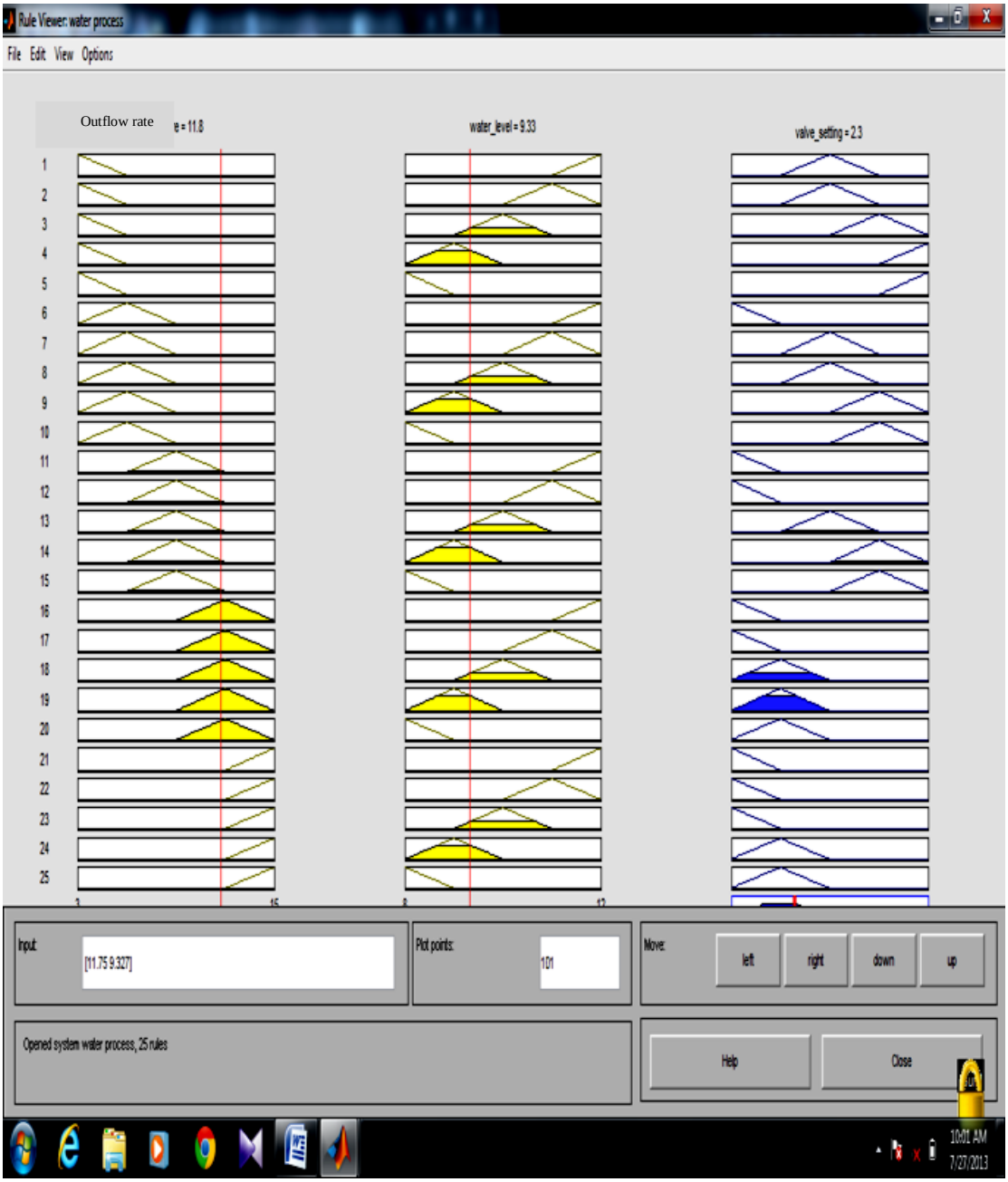


Fig ure 4.5f: Surface viewer of fuzzy rule for temperature in MATLAB

4.1.8 Determining the Defuzzification.

Since fuzzy logic controller can have only one output, it must complete a process called deffuzification to determine the actual final output value. Defuzzification is performed according to the membership function of the output variable. In figures 4.6a, 4.6b and 4.6c shows the Rule Viewer for outflow rate, temperature and pressure respectively. They display a

roadmap of the whole fuzzy inference process. The three small plots across the top of the figure represent the antecedent and consequent of the first rule. Each rule is a row of plots, and each column is a variable. The first two columns of plots show the membership functions referenced by the antecedent, or the if-part of each rule. The third column of plots shows the membership functions referenced by the consequent, or the then-part of each rule and that is the column for defuzzification. If we follow rule 1 across the top of the diagram through the twenty-fifth rule, we can see the consequents has been truncated to exactly the same degree as the (composite*)* antecedent--this is the implication process in action. The aggregations occur down the third column, and the resultant aggregate plot and the defuzzified output value is shown through the aggregate fuzzy set. The Rule Viewer also shows how the shape of certain membership functions influences the overall result..



` Figure 4.6a: Rule viewer for outflow rate

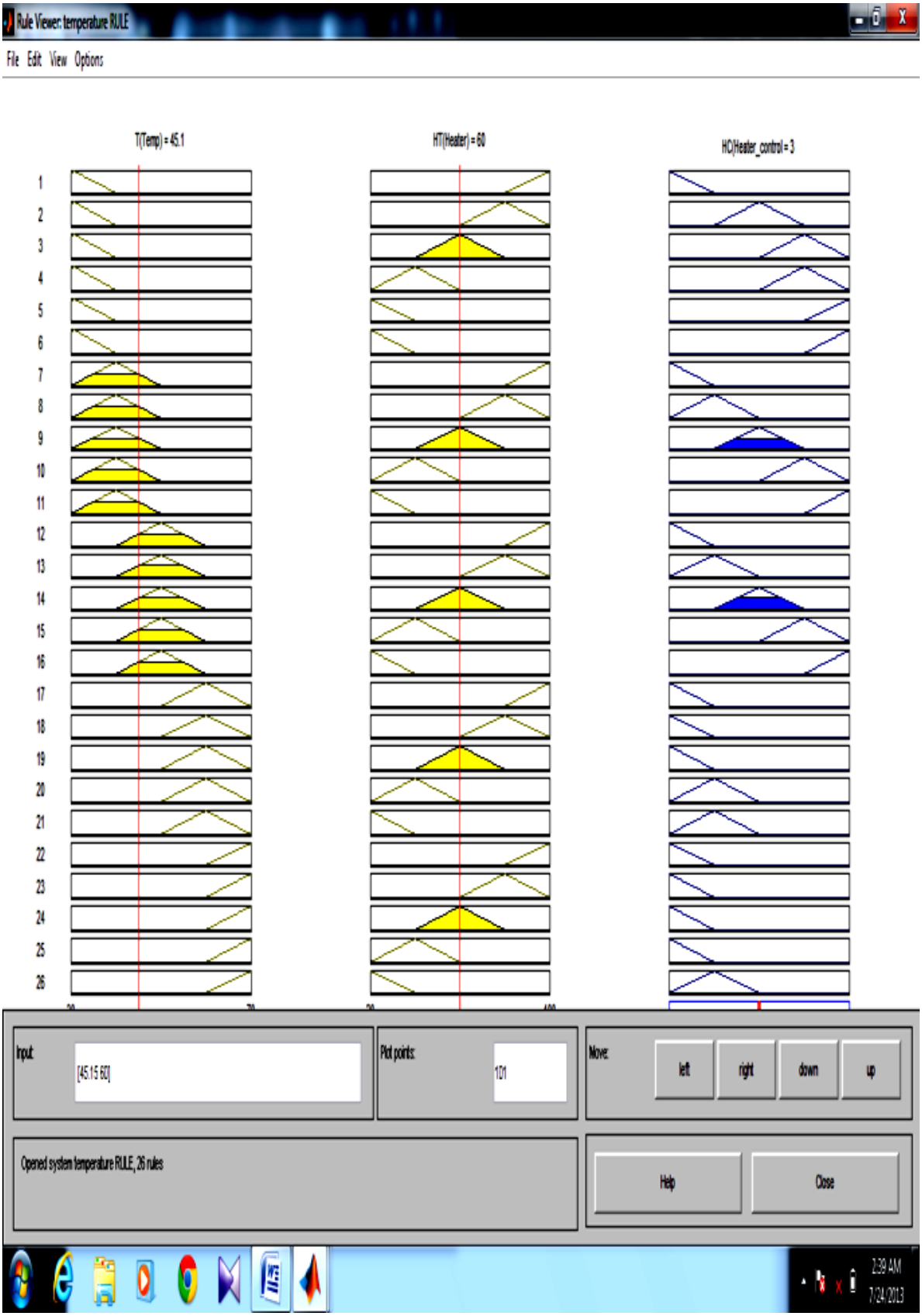


Figure 4.6b: Rule viewer for temperature

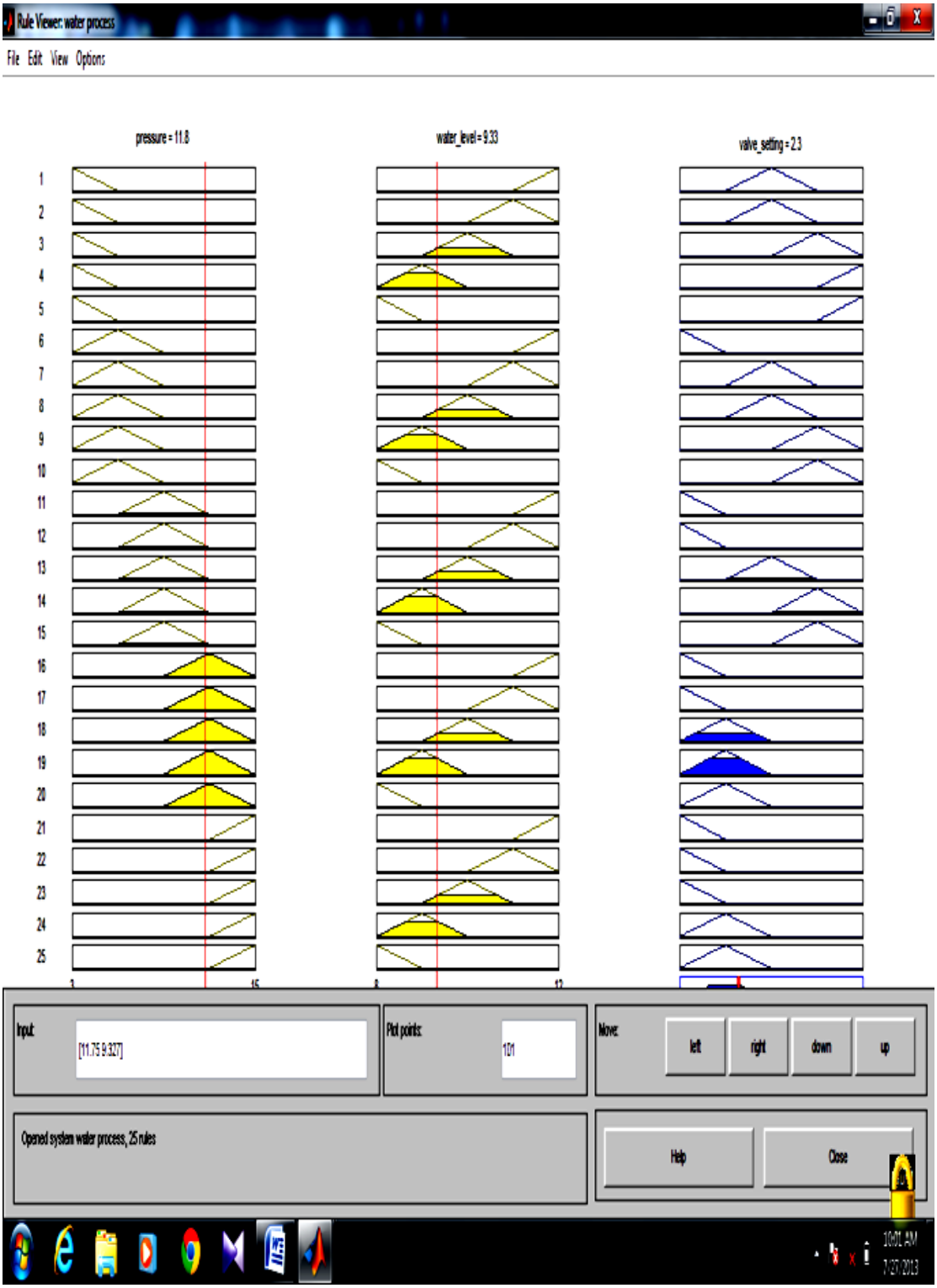


Figure 4.6c Rule viewer for pressure

4.1.9 Running Test Suite to Validate System, Adjusting Systems, Completing Document and Releasing for production.

After the defuzification stage, the fuzzy logic network was tested and necessary adjustment made before it is implemented in the process control as shown in Figure 4.7. It shows the block diagram of the PID fuzzy controller of the tank water level. The process under control has a valve that controls the inflow of liquid into the tank and another one that controls the outflow from the tank. The control system tries to keep the liquid level in the tank constant within the set-point for level. This is achieved by adjusting the inflow rate and outflow rate dynamically as appropriate by the microcontroller. The heater is being controlled by the heater control while the stirrer ensures that the liquid is at uniform temperature. Four sensors where used:

1) To sense the liquid level in the tank,

2) To sense the outflow rate from the tank

3) To sense the temperature and

4) To sense the pressure

The four sensor outputs in millivolt after being amplified are selected one at a time via 4 X 1 analog multiplexer. The selected signal is then converted to digital pattern via an analog to digital converter. The digitalized sensor output is then sent to the microcontroller which controls the process.

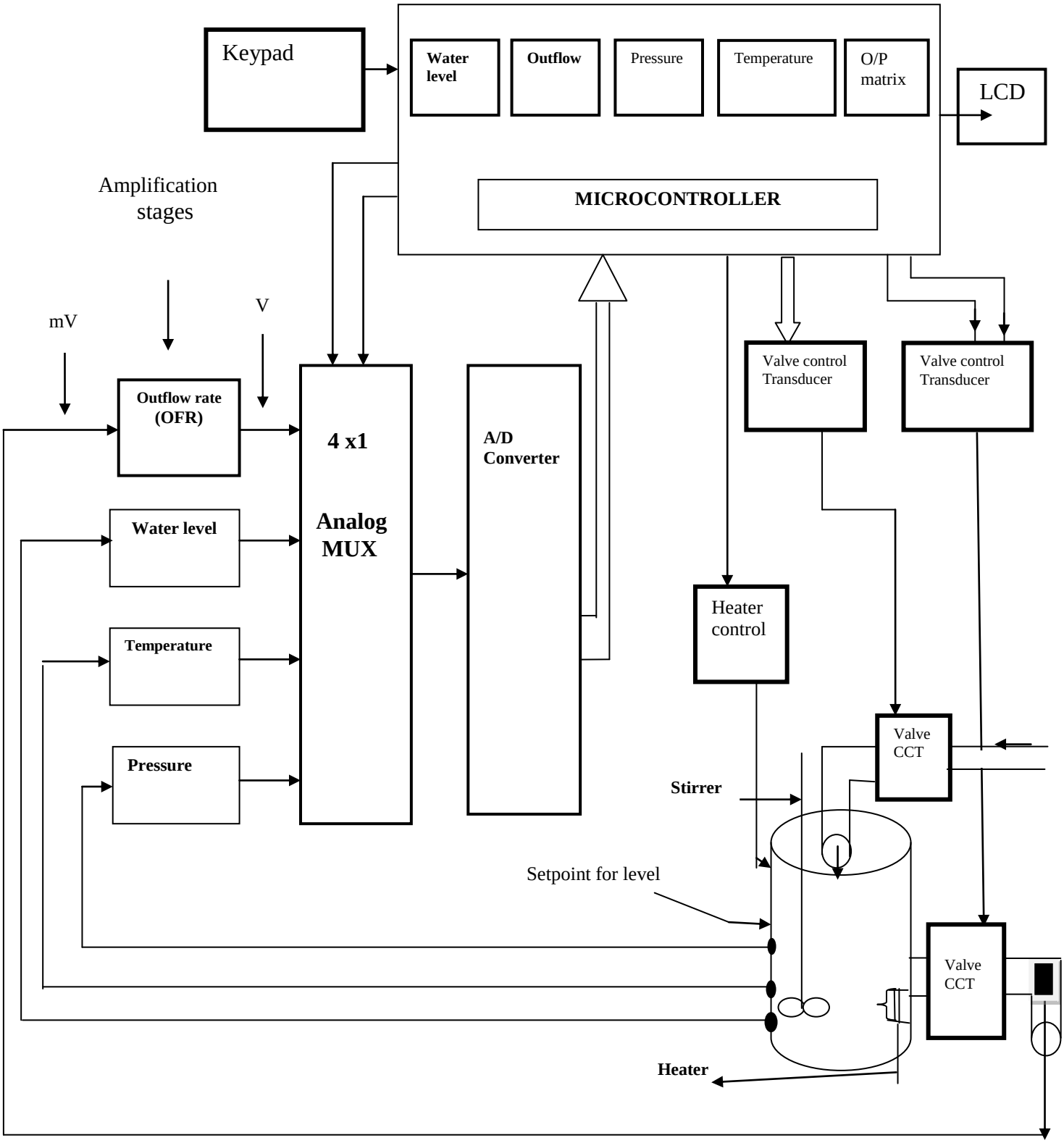


Figure 4.7: Fuzzy controller of the process under control

The keyboard serves as the input device for the controlled parameters while the Liquid Crystal Display (LCD) displays the output of the controlled variables. Outputs of the matrix are as shown in the matrix tables discussed in table 4.1, 4.2 and 4.3.

4.2 The Neural Network Design: The procedures for designing a neural network for the process under control involve the followings:

- Collection of data (same as that of fuzzy network)
- Creating and Configuring the network
- Training the network
- Validating and Using the network

4.2.1 Collection of data: The process involves collecting of data and separating them into training set and a test set. The training cases are used to adjust the weights, and the test cases are used for network validation. The data used for training and testing must include all the attributes that are useful for solving the problem. The system can only learn as much as the data can tell. The data is the same as fuzzy sets (see section 4.1.2)

4.2.2 Creating and configuring the network: Once the training and testing data sets are identified, the next step is to design the structure of the neural networks. A feed-forward four layer showing the neural controller was configured for liquid level, pressure and temperature

respectively (see figures 4.8a, 4.8b and 4.8c). The first layer represents input variables, the middle (hidden) layers represents membership functions and fuzzy rules respectively and the fourth layer represents the output variables.

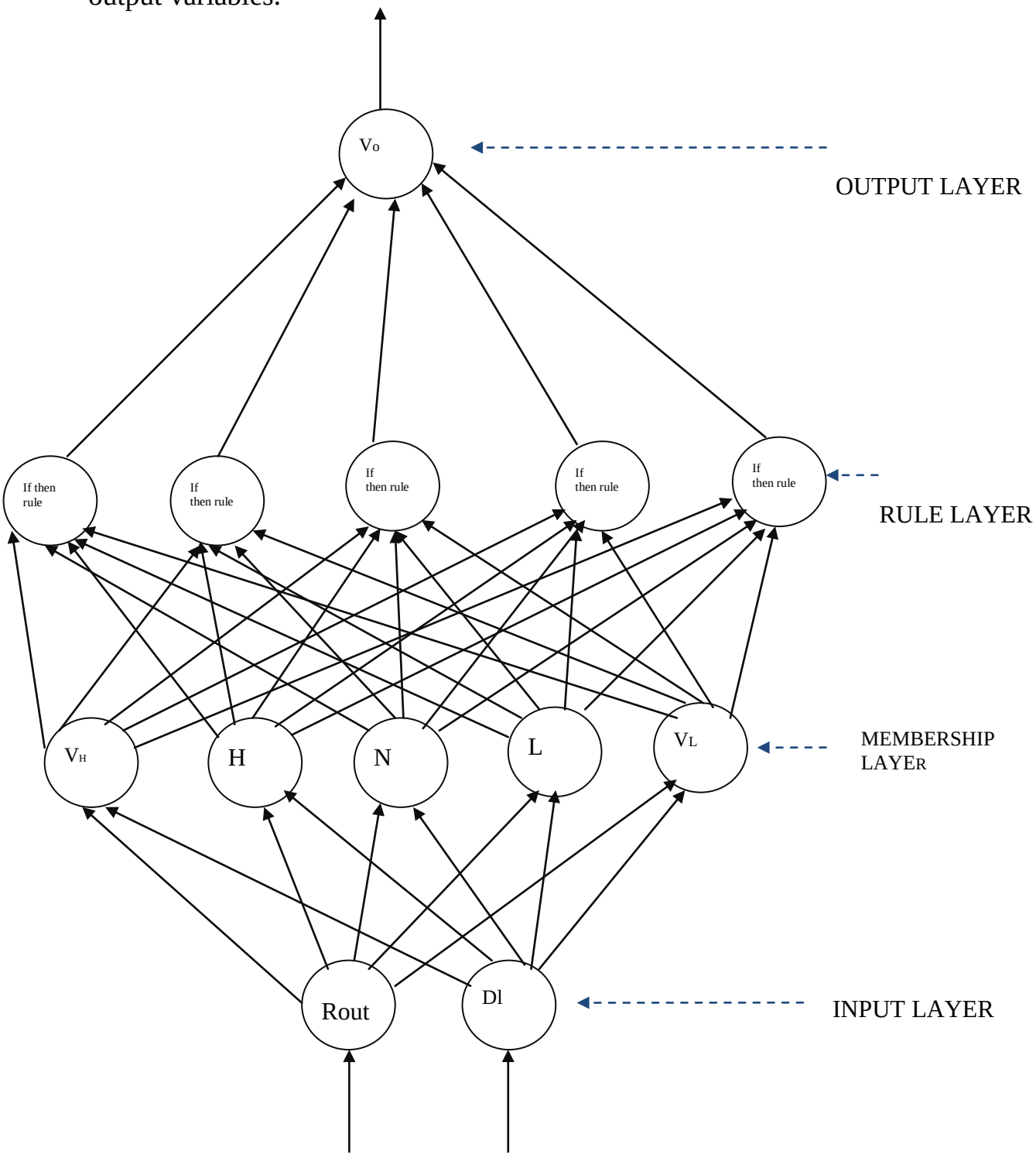


Figure 4.8a: Neural controller for the water level

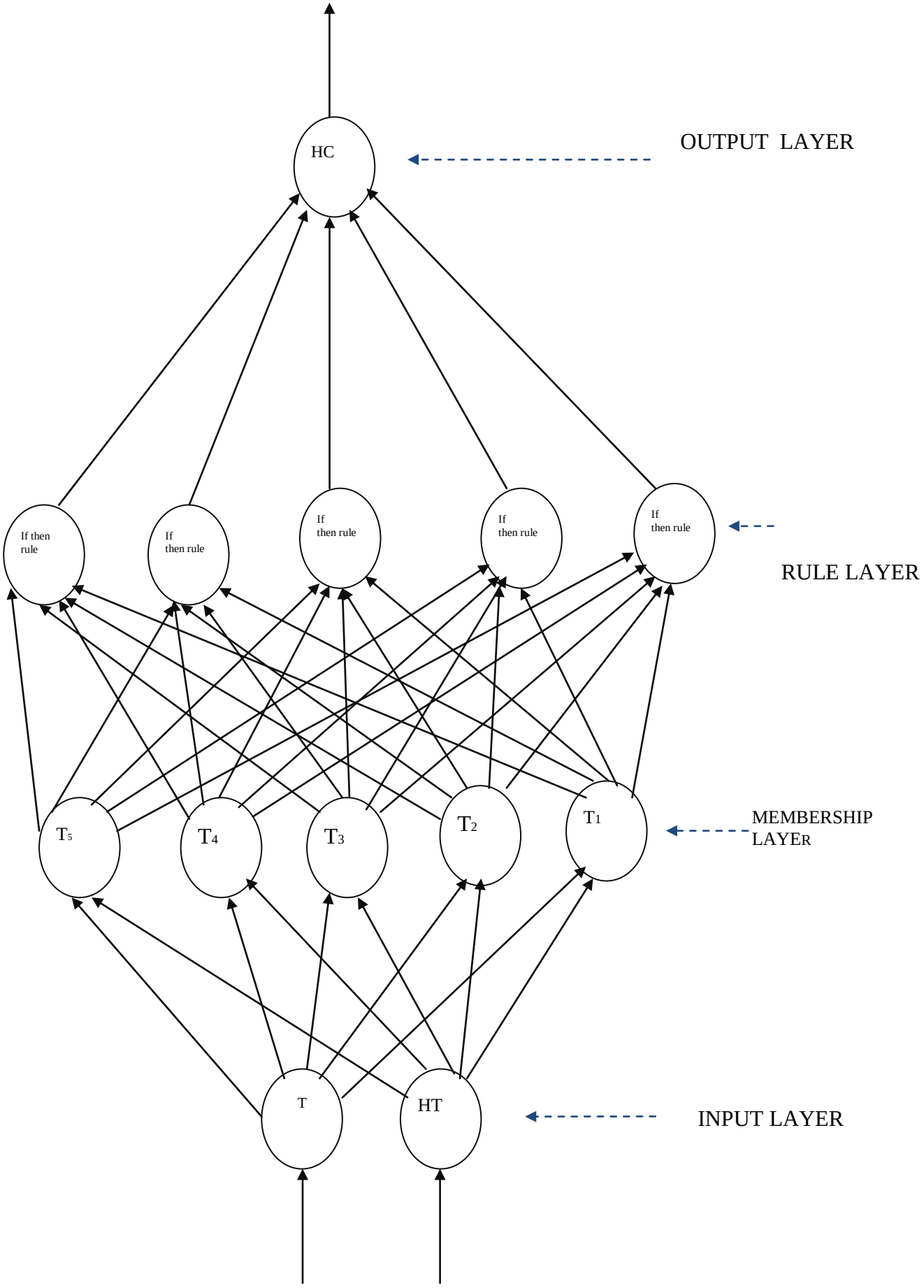


Figure 4.8b: Neural controller for the temperature

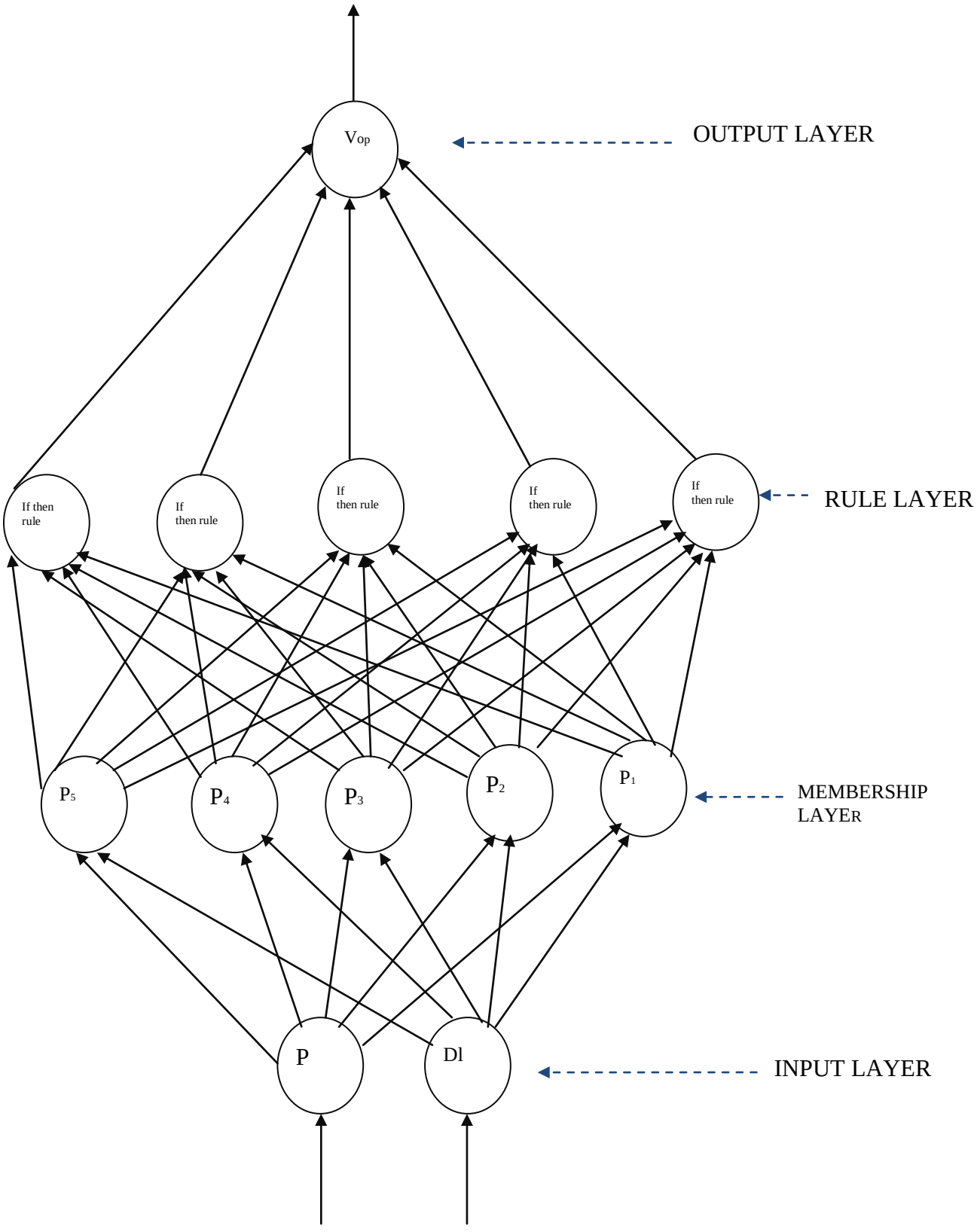


Figure 4.8c: Neural controller for pressure

4.2.3 Training the Network: Once the network structure is chosen, a learning algorithm to identify a set of connection weights that best cover the training data and have the best predictive accuracy is achieved. For the feed forward topology which was applied in this work, the back propagation algorithm was implemented. Back propagation is the most widely used supervised learning algorithm in neural computing. During supervised training, externally provided correct patterns are compared with the neural network's output and the feedback is used to adjust the weights until all the training patterns are categorized as correctly as possible by the network. Since many commercial packages are available on the market, there is no need to implement the learning algorithm instead a suitable commercial package (Neural ware) was chosen to analyze the data. Training of artificial neuron (see the flowchart in figure 4.9) is an iterative process that starts from a random set of weights and gradually enhances the fitness of the network model and the known data set .The iteration continues until the error sum is converged to below a preset acceptable level. In back propagation two parameters, learning rate and momentum was adjusted to control the speed of reaching a solution.

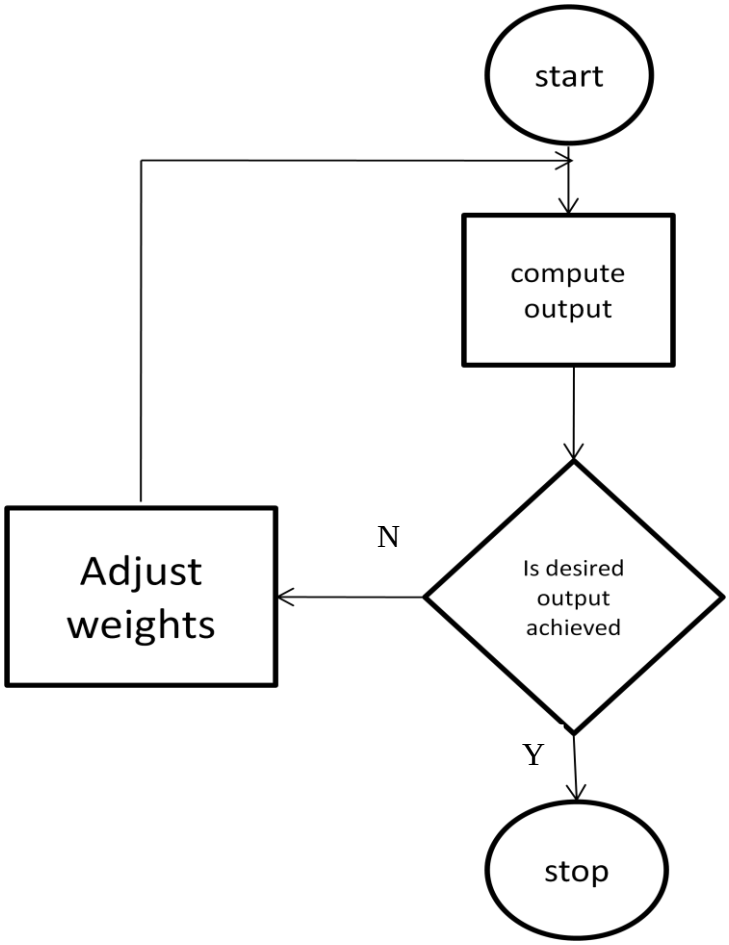


Figure 4.9: Flowchart for training a neuron

4.3 Validating and using the network

Some data conversation is necessary in the training process. This includes, changing the data format to meet the requirements of the software, normalization of the data scale to make them more comparable and removing problematic data. Once the training data set is ready, it is loaded into the package and the learning process executed. Once the training has been completed, the network is tested to examine the performance of the derived network model. The implementation of the

network requires interfaces with other computer based information systems as discussed below.

4.4 Neural Control of Tank Water Level

The neural control of the process was designed as shown in fig 4.10a

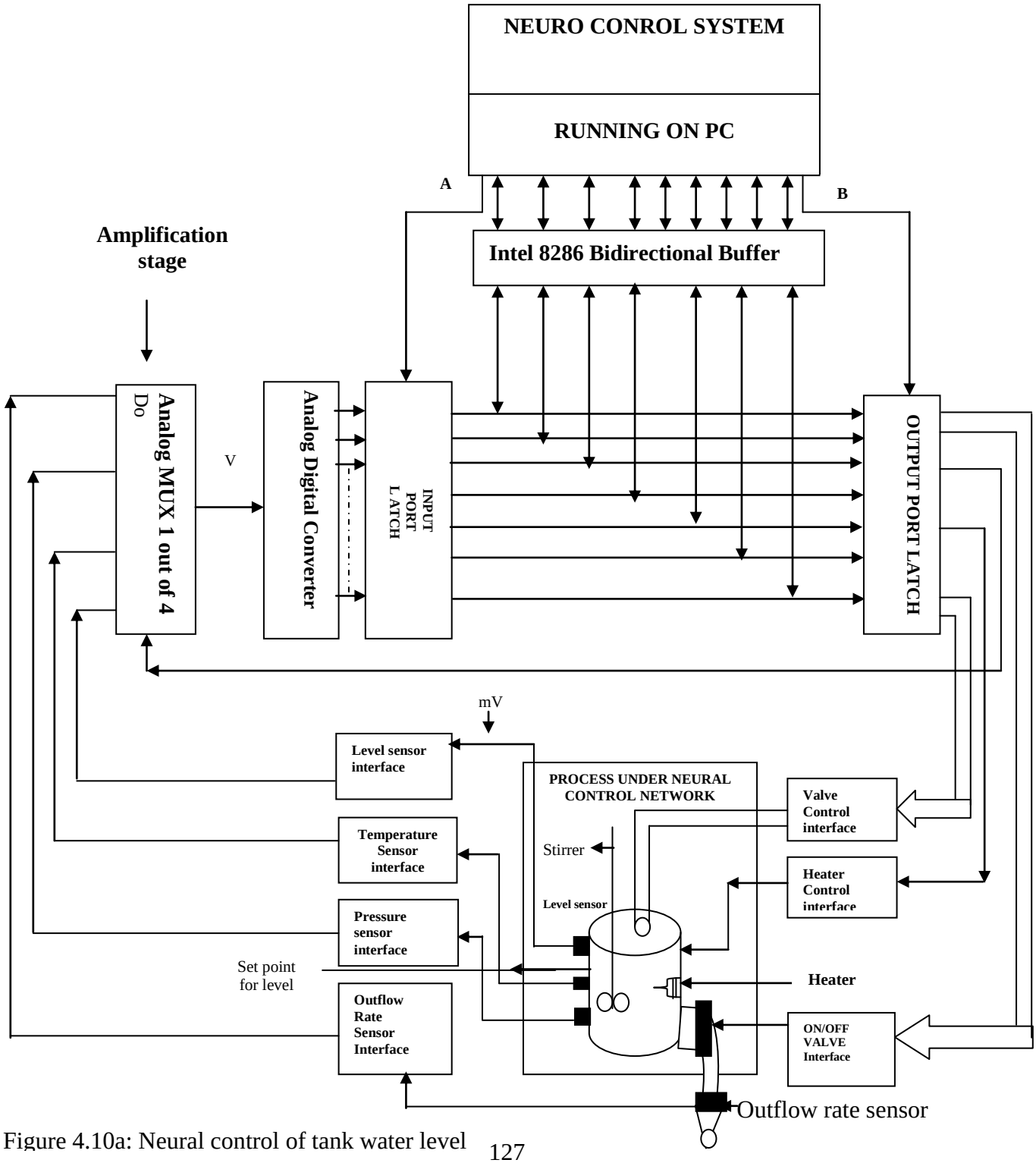


Figure 4.10a: Neural control of tank water level

The Neural control system running on PC interfaced to the process under control via an INTEL 8286 Bidirectional Buffer. When the signal line B is active, the buffer sends information to the process under control via the output port latch. Similarly, when the signal line A is active, the feedback signals from the process are input via the input port latch to the bidirectional buffer and from thence to the PC. The process under Neural control has a valve that controls the inflow of the liquid into the tank and another one that controls the outflow from the tank. The heater increases the temperature of the liquid while the stirrer is used to ensure that the liquid is of uniform temperature. The control system tries to keep the liquid level in the tank constant within the set point for level. This is achieved by neural control by adjusting the inflow rate and outflow rate dynamically as appropriate. Four sensors were used: 1) to sense the liquid level in the tank and 2) to sense the outflow rate from the tank, 3) to sense the temperature and 4) to sense the pressure of the system. The four sensor outputs are selected one at a time via 4 X 1 analog multiplexer. The selected signal is first amplified and then converted to digital pattern via an analog to digital converter. The digitalized sensor outputs are latched by the input port latch and forwarded to the neural control system via the bidirectional buffer. Figure 4.10b depicts the real time simulation of Neural control of tank water level in Proteus.

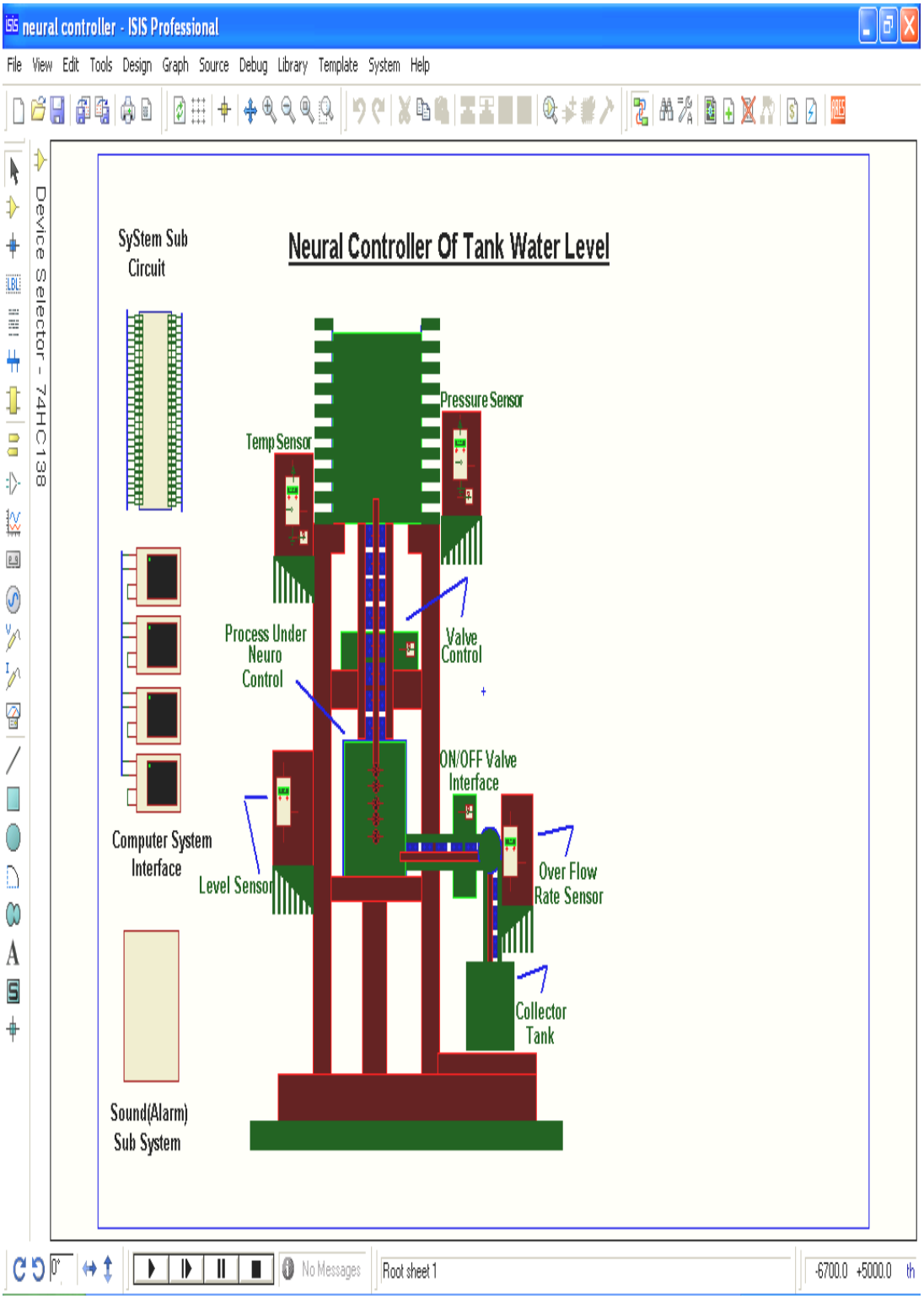


Figure 4.10b: Real time simulation of neural controller of tank water level in Proteus

The real time simulation of the control system used as a typical industrial scenario is achieved by clicking the play button. When this is done the liquid from the upper tank flows to the lower tank. The valve to the water inlet of the upper tank is being controlled based on the rate of the outflow from the lower tank. This is done to make sure that the set point of the process under control is maintained. The heater heats the liquid while the stirrer ensures that a uniform liquid temperature is maintained. The alarm sub system sounds at indication of any fault. One of the four computer system interfaces displays the possible rectification methods of the fault and the time evolution of such faults while the other three, displays show the various readings of the outflow rate/liquid level, the pressure and temperature. When Proteus software is used to implement a real-time simulation, not all the system components are placed on the layout. Some are placed in the sub sheet e.g. controller; some are not visible but are there by default e.g. reset circuit, crystal microcontroller, power, etc; while some are taken care of by the control program. In Proteus design, this is called “referencing.

4.5 Intelligent Process Control System Using NeuroFuzzy Network

Fuzzy neural networks (FNN) combine fuzzy logic with artificial neural networks. In this work, the input and output variables processed by the fuzzy logic was fed into the neural networks for learning. (This step is

called fuzzification). The neural network then takes the fuzzified input and output scales to derive a model which is converted back to the original input and output scales (defuzzification). Then the output of the "defuzified" fuzzy system became input to the process under control as shown in the block diagram in Figure 3.1. The Fuzzy Neural controller of tank water level shown in Figure 4.12a has four layers. The first layer represents the input variables for liquid level, temperature and pressure respectively, the second represents their membership sets, the third layer represents the fuzzy rules and the fourth layer represents the defuzified output for the three variables under consideration. The rule layer produces seventy-five rules altogether, twenty-five rules from each of the variables and gives three outputs at a time for each of the three variables that satisfies the condition at a particular instance.

The design specification is in two stages. The first stage has the personal computer (PC) where all the software controlling the system is installed. The PC has limited ports which are used for other computer peripheral and communication devices. Out of these ports, one is then used for the bidirectional buffer which interfaced the process control system and the PC. This helps to eliminate the use of two ports for input and output from the PC. When line A is active (Figure 4.11a), line B becomes inactive, and then feedback signal from the process via input latch moves to the bidirectional buffer and goes to the PC. The feedback signals from the

process sensors pass through a set of stages before reaching the bidirectional buffer. After amplification of the signal, the multiplexer selects one out of the four variables at a time and sends it to Analog Digital Converter which digitizes the signal and then passes it to the buffer through the input latch. When line B is active, the buffer sends information to the process under control via the output latch to the process through the actuators.

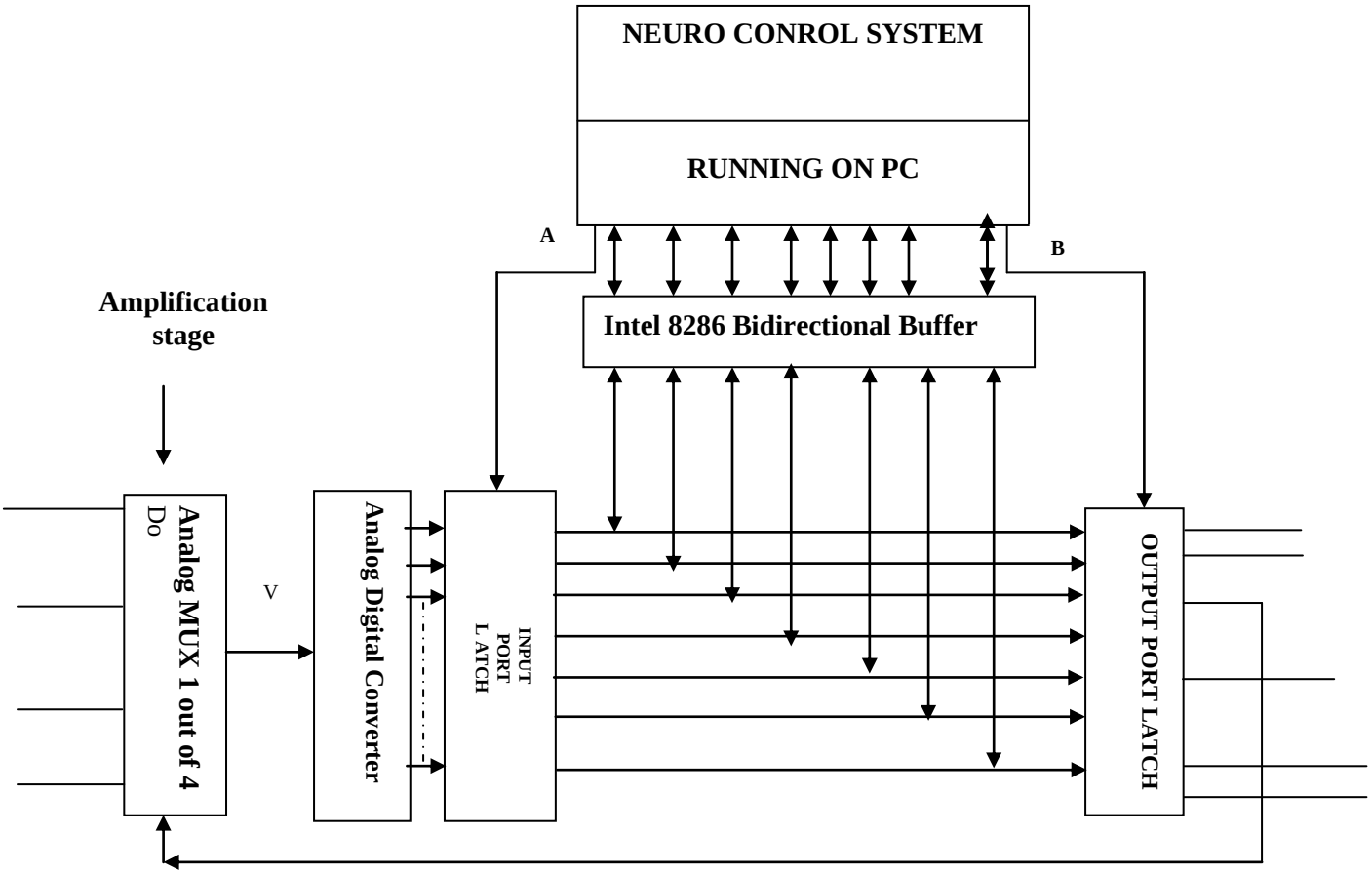


Figure 4.11a: Design specification for Fuzzy Neural control of tank water level (First stage)

The second stage as shown in figure 4.11b shows the process under fuzzy neural control, which shows a tank containing the liquid that is to be at a set level as water flows in and out of the tank, the heater heats up the

liquid and the stirrer continually stirs it to ensure a uniform temperature. The three actuators shown are the interfaces to the process under control, and they are the heater control interface, the valve control interface and the power control interface. Four transducers which sense the variables (liquid level, temperature, pressure and outflow rate) were used to convert each respective signal to electrical signal which is sent to the multiplexer where one is selected at a time and from the ADC through the input port latch back to the PC. When Figures 4.11a and 4.11b are combined, figure 4.12b, the model diagram results.

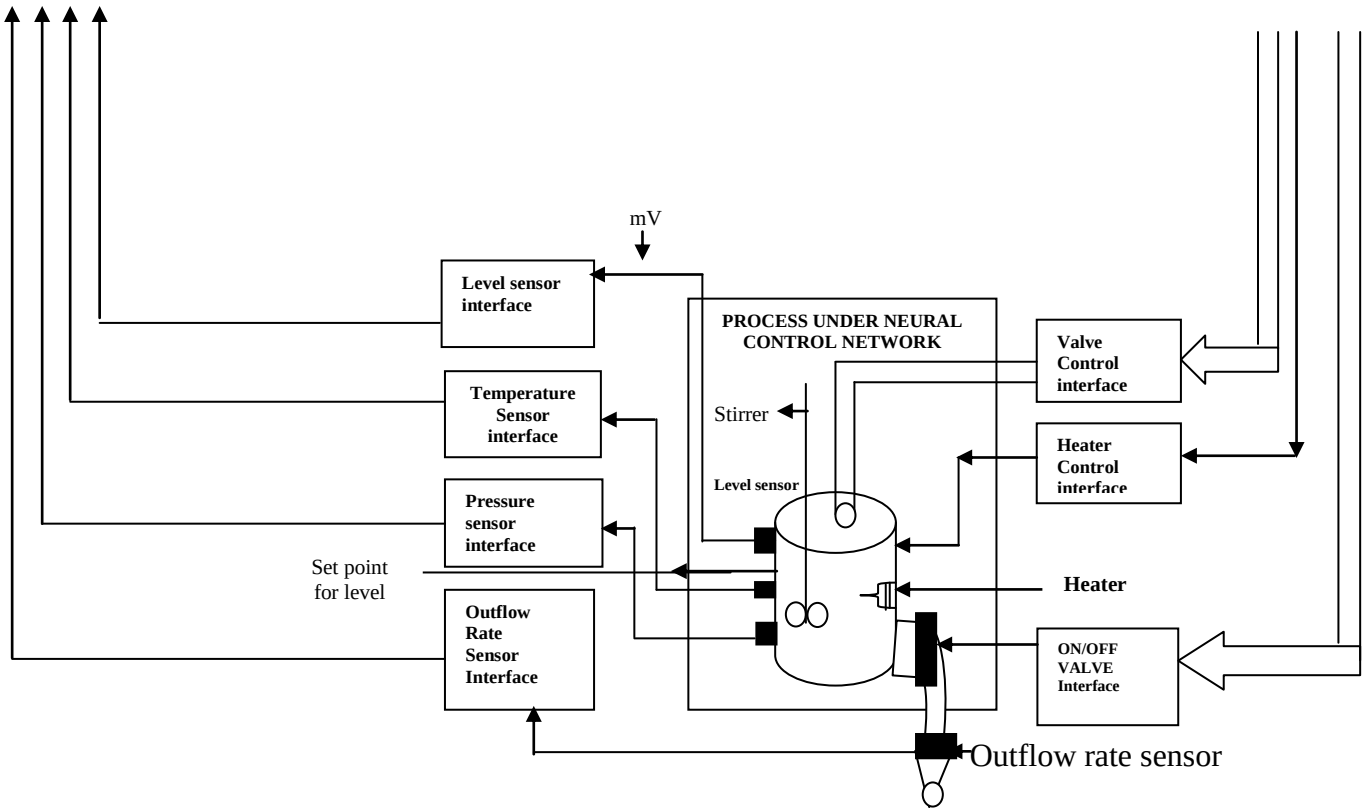


Figure 4.11b: Design specification for Fuzzy Neural control of tank water level (Second stage)

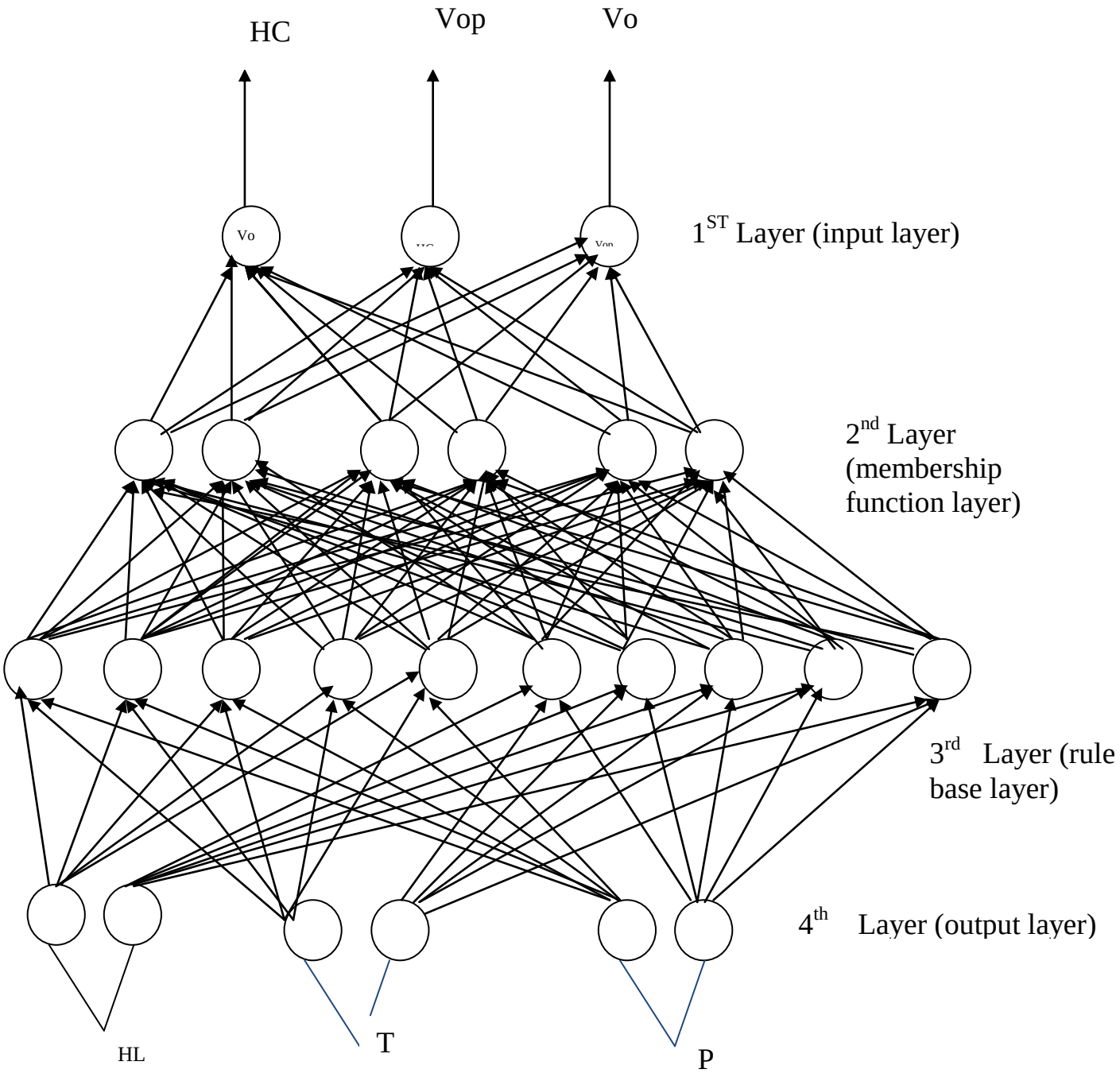


Figure 4.12a: Fuzzy Neural control of tank water level

Figure 4.12b shows the block diagram of the Intelligent Process Control System Using Fuzzy-Neural Network. Here, the Fuzzy Neural control system running on PC interfaced to the process under control via an INTEL 8286 Bidirectional Buffer. Just like in the neural controller, when the signal line B is active, the buffer sends information to the process

under control via the output port latch. Similarly, when the signal line A is active, the feedback signals from the process are input via the input port latch to the bidirectional buffer and from thence to the PC. The process under Fuzzy Neural control has a valve that controls the inflow of the liquid into the tank and another one that controls the outflow from the tank. The heater increases the temperature of the liquid while the stirrer is used to ensure that the liquid is of uniform temperature. The control system tries to keep the liquid level in the tank constant within the set-point for level. This is achieved by Fuzzy Neural control by adjusting the inflow rate and outflow rate dynamically as appropriate. Four sensors were also used: 1) to sense the liquid level in the tank and 2) to sense the outflow rate from the tank, 3) to sense the temperature and 4) to sense the pressure of the system. The four sensor outputs are selected one at a time via 4-out-of-1 analog multiplexer. The selected signal is first amplified and then converted to digital pattern via an analog to digital converter. The digitalized sensor output are latched by the input port latch and forwarded to the Fuzzy Neural control system via the bidirectional buffer.

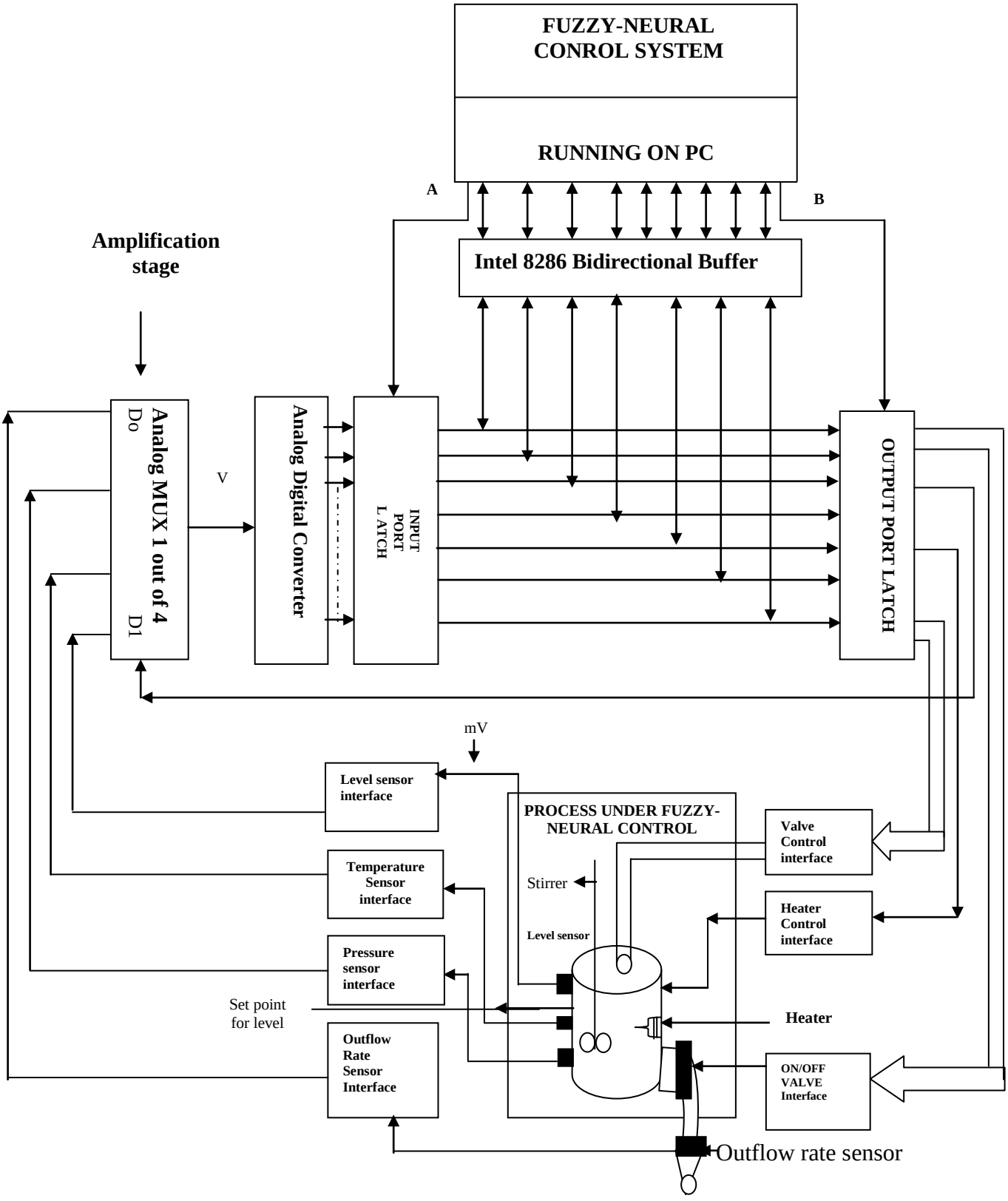


Figure 4.12b: Block diagram of The Intelligent Process Control System Using Fuzzy-Neural Network

4.6 The Industrial scenario

It is always necessary to perform a real-time simulation of any control system before the circuit is constructed, to ensure the workability of that system. The simulated prototype model of the system using Proteus software which is a good software tool for real time simulation is shown in figure 4.13. In this work, Proteus 7 software package was used for the simulation.

The real time simulation of the control system used as a typical industrial scenario is achieved by clicking the play button. When this is done the liquid from the upper tank flows to the lower tank. The valve at the water inlet of the upper tank is being controlled based on the rate of the outflow from the lower tank. This is done to make sure that the set-point of the process under control is maintained. The heater heats the liquid while the stirrer ensures that a uniform liquid temperature is maintained. The alarm sub system sounds at indication of any fault. Figure 4.13 shows the real time simulation for a typical Fuzzy Neural controller of tank water level in Proteus environment. The first three computer system interface shown displays the variables under control (liquid level/outflow rate, pressure and temperature) while the fourth one displays the fault diagnosis features (type of faults, time of occurrence and possible rectification methods).. It should be noted that when Proteus software is used to implement a real-

time simulation, not all the system components are placed on the layout. Some are placed in the sub sheet for example the controller and some are not visible but are there by default e.g. reset circuit, bidirectional buffer, power, the interfaces, analog to digital converter, the multiplexer etc; while some are taken care of by the control program. In Proteus design, this is called “referencing”.

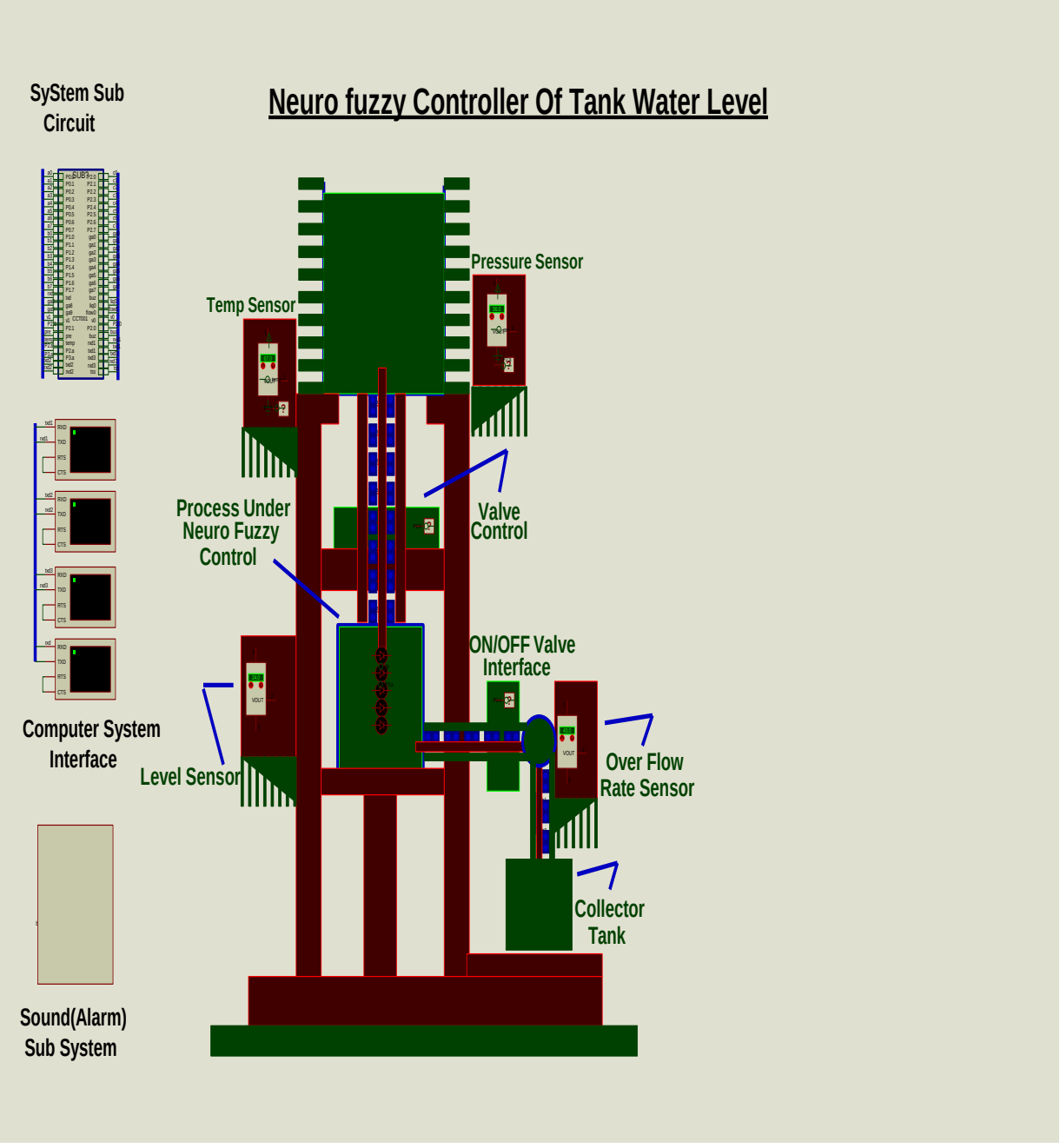


Figure 4.13 Real time simulation for a typical Fuzzy Neural controller of tank water level in Proteus

4.7 Design of the Fault Diagnosis System:

The main aim of the Fault diagnosis system is the monitoring of the process control during its normal conditions so as to detect the occurrence of failures, recognize the location and time of occurrence. Typical failure modes may include leaks, sensor failures, temperature, pressure, valve failure, etc which is the characteristics of a process failure as well as a variety of vibration induced faults that are affecting mechanical and electro-mechanical process elements. In this research fuzzy diagnostic system (FDS) was used. The FDS takes features of the process control as inputs and then outputs any indication that a failure mode has occurred. Fig 4.14 shows the FDS of tank water level.

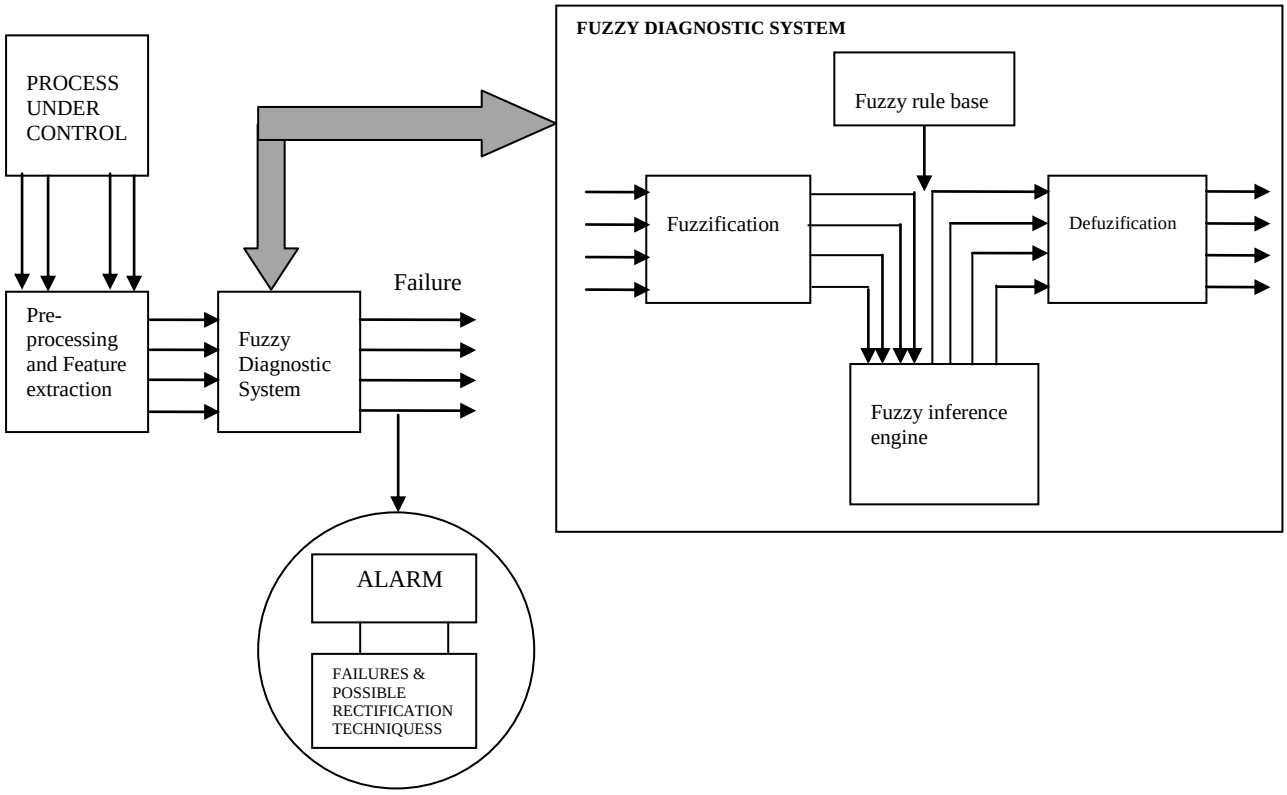


Figure 4.14: FDS of tank water level.

The fuzzification block converts the feature extraction to degree and type of failure, the fuzzy rule base is constructed from symptoms that indicate a potential failure mode (This is developed directly from user experience, simulated models or experimental data). The inference engine determines the degree of fulfillment for each rule corresponding to each failure mode while the last stage defuzzifies the resulting output to indicate a failure and this triggers an alarm and at the same time suggests a solution to the fault. Table 4.6 show the rule base for failure, the time of occurred fault and rectification techniques. This is easily produced by the inference engine.

Table 4.6: Rule base for failure, time of occurred fault and rectification techniques.

Time of Occurrence	**Observation (Rule Base)**	**Fault (Output)**	**Rectification Techniques**
12.00pm	If Water level is at constant position Then	Outflow valve /inflow valve is faulty	Verify which and change the valve
1.00am	If Outflow rate is below the lower threshold level then	The inflow valve is faulty	Change the inflow valve
....etc	... etc	... etc	... etc

4.8 The Choice of the Sensor Used

A sensor is a device that monitors a parameter and produces an output in a required form while an actuator is a device that converts an electrical signal into a mechanical signal (such as heat, light, sound or movement) and vice versa. Actuators and sensors are both transducers **(**devices that change one kind of signal into a different kind of signal).

The following sensors were considered.

4.8.1 Piezoelectric Pressure Sensor.

Piezoelectric Pressure Sensor was considered in this research. Piezoelectric elements are bi-directional transducers capable of converting stress into an electric potential and vice versa. They consist of metalized quartz or ceramic materials. One important factor to remember is that it has a dynamic effect, which is providing an output only when the input is changing. This means that these sensors can be used only for varying pressures .The piezoelectric element has a high-impedance output and care must be taken to avoid loading the output by the interface electronics. Some piezoelectric pressure sensors include an internal amplifier to provide an easy electrical interface.

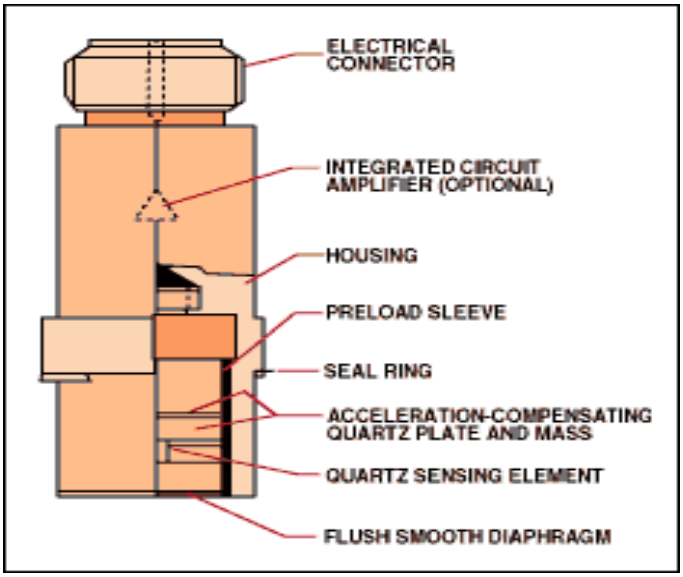


Figure 4.15a: Piezoelectric Pressure Sensor

Piezoelectric Pressure sensors (figure 4.15a) convert stress into an electric potential and vice versa. Sensors based on this technology are used to measure varying pressure.

4.8.2 Temperature sensors: Temperature sensors tend to measure heat to ensure that a process is either; staying within a certain range, providing safe use of that application, or meeting a mandatory condition when dealing with extreme heat, hazards, or inaccessible measuring points. Thermistor was used in this work. Thermistors, are also inexpensive, readily available, easy to use, and adaptable temperature sensors. They are used, however, to take simple temperature measurements rather than for high temperature applications. They are made of semiconductor material with a resistivity that is especially sensitive to temperature. The

resistance of a thermistor decreases with increasing temperature so that when temperature changes, the resistance change is predictable. They are widely used as inrush current limiters, temperature sensors, self-resetting overcurrent protectors, and self-regulating heating elements.

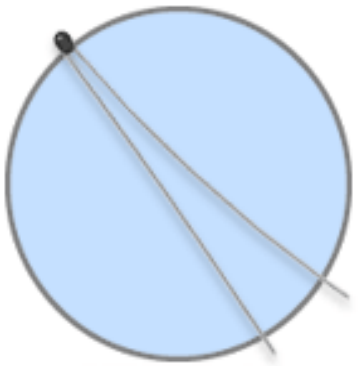


Figure 4.15b: Thermistor element

The 44000 series thermistor element shown in Figure 4.15b is the simplest form of thermistor. Because of their compact size, thermistor elements are commonly used when space is very limited. OMEGA offers a wide variety of thermistor elements which vary not only in form factor but also in their resistance versus temperature characteristics. Since thermistors are non-linear, the instrument used to read the temperature must linearize the reading.

4.8.3 Capacitance Level Sensors

Like ultrasonic sensors, capacitance sensors (Figure 4.15c) can handle point or continuous level measurement. Capacitance sensor was

implemented in this work. They use a probe to monitor liquid level changes in the tank, electronically conditioning the output to capacitive and resistive values, which are converted to analog signals. The probe and the vessel wall equate to two plates of a capacitor, the liquid to the dielectric medium. Because the signal emanates from level changes alone, material build-up on the probe has no effect. Non-conductive fluid vessels may dictate dual probes or an external conducting strip.

The probe, which can be rigid or flexible, commonly employs conducting wire insulated. Using stainless steel as the probe's base metal provides the extra sensitivity needed for measuring liquids that are non-conductive, granular, or low in dielectric properties (dielectric constant less than 4). Flexible probes must be used when there is insufficient clearance for a rigid probe, or in applications that demand very long lengths. Rigid probes offer higher stability, especially in turbulent systems, where swaying of the probe can cause signal fluctuations.

Figure 4.15c: Capacitance Measurement Probe

Capacitance Measurement Probe is able to withstand high temperatures and pressures, and impervious to many corrosives, LV3000/4000 Series probes give reliable continuous level measurements in difficult applications. Appropriate for liquids, pastes, and some solids–whether conductive or non-conductive–they have no moving parts and are easy to install. After rectifying and filtering incoming power, generating a radio frequency signal, and calculating changes in current, the electronic circuitry produces a 4 to 20 mA 2-wire output signal proportional to the process level.

4.8.4 Calorimetric Flow meter

The Calorimetric Flow meter was used in this project. Calorimetric Flow meter principle for fluid flow measurement is based on two temperature sensors in close contact with the fluid but thermal insulated from each other. See Fig 4.15d

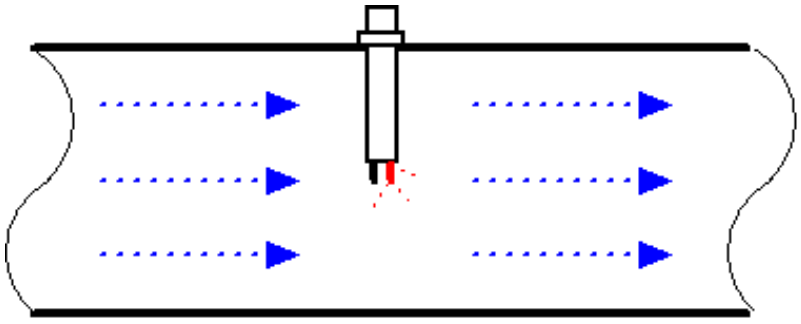


Figure 4.15d: Calorimetric Flow meter

One of the two sensors is constantly heated and the cooling effect of the flowing fluid is used to monitor the flow rate. In a stationary (no flow)

fluid condition there is a constant temperature difference between the two temperature sensors. When the fluid flow increases, heat energy is drawn from the heated sensor and the temperature difference between the sensors are reduced. The reduction is proportional to the flow rate of the fluid. Response times will vary due the thermal conductivity of the fluid. In general lower thermal conductivity requires higher velocity for proper measurement. The calorimetric flow meter can achieve relatively high accuracy at low flow rates.

CHAPTER FIVE

SYSTEM SIMULATION AND EVALUATION

5.1 Real –Time Simulation Results and Analysis

Three software tools were used to implement this work. First is the VB.net that was used to develop the matrix table. Each cell of the matrix table represents a possible output of the Fuzzy Neural control system. Secondly, Proteus was used for real time simulation. Proteus is specialized software for virtual implementation of embedded system designs, and its use in this project is to check the workability of this system in a real-life situation. The readings obtained during the testing of the process was tabulated and plotted in a graph using excel.

5.2 The Fuzzy Network Result and Analysis: After the simulation of the system control under fuzzy network, the following data were obtained and plotted against time. Table 5.1a shows the fuzzy output for outflow rate while Fig 5.1shows the graph.

Table 5.1: Data of fuzzy output for outflow rate

Rout (cm^3/s)	23.5	24.3	25.1	50.7	25.1	90.3	100.8	112.2	111.0	112.3	112.5	118.3	120.6	119.9
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

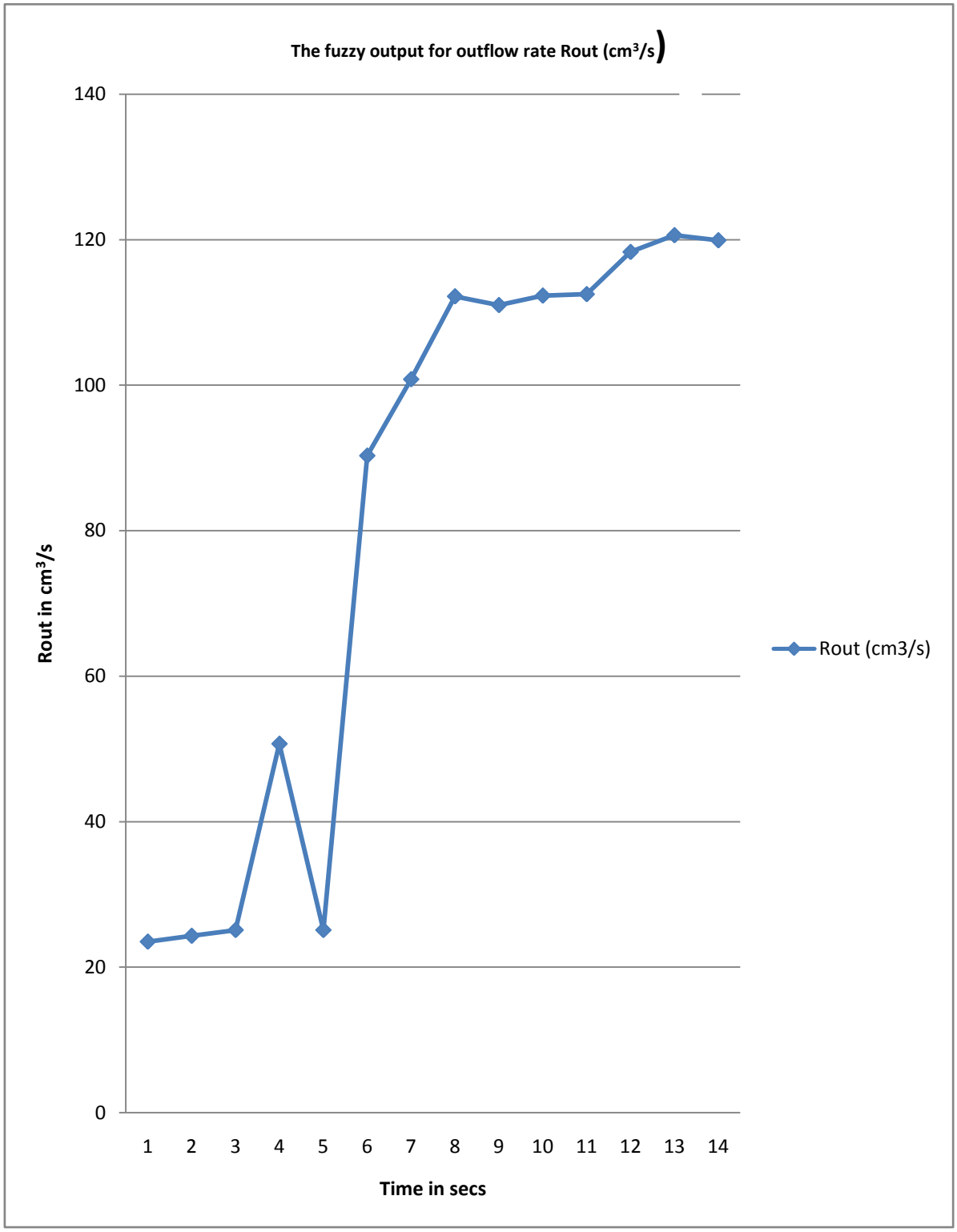


Figure 5.1: Fuzzy output for outflow rate

In Figure 5.1, it was observed that the outflow rate for fuzzy network gave a sharp rise at 5s before the settling down after 8s. The fuzzy

network operates on whole number data. When data are in fraction and decimals, it cannot deduce what should be the output which is the cause of the overshoot between 3s and 5s.

Table 5.2 shows the fuzzy output for temperature while Fig 5.2 shows the graph of fuzzy output for temperature.

Table 5.2: Data of fuzzy output for temperature

T(oC)	29.5	30.1	34.6	35.2	40.0	50.6	50.9	57.8	59.5	60.4	70.8	80.1	80.3	79.4
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

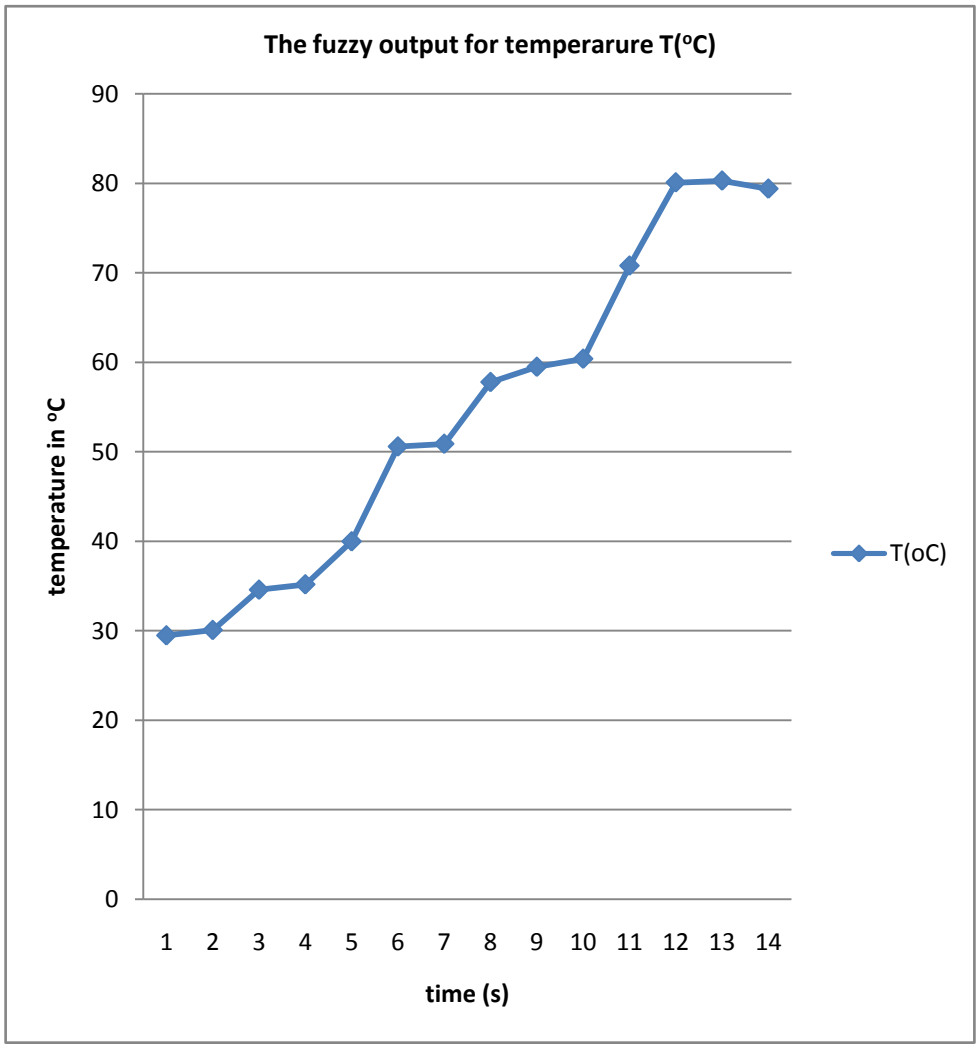


Figure 5.2: Graph of fuzzy output for temperature

In Figure 5.2, the temperature increased gradually from 30°C for 11s to get stabilized at 80°C between 12s and 14s. The temperature increase was a function of the heater which is being controlled by the heater control as to keep the liquid at a set temperature.

Table 5.3 shows the fuzzy output for pressure while Figure 5.3 shows the graph.

Table 5.3: Data of fuzzy output for pressure

P(KN/m^2)	3.5	6.0	7.3	5.3	7.0	5.8	4.9	5.7	6.2	7.5	11.5	10.5	6.8	8.5
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

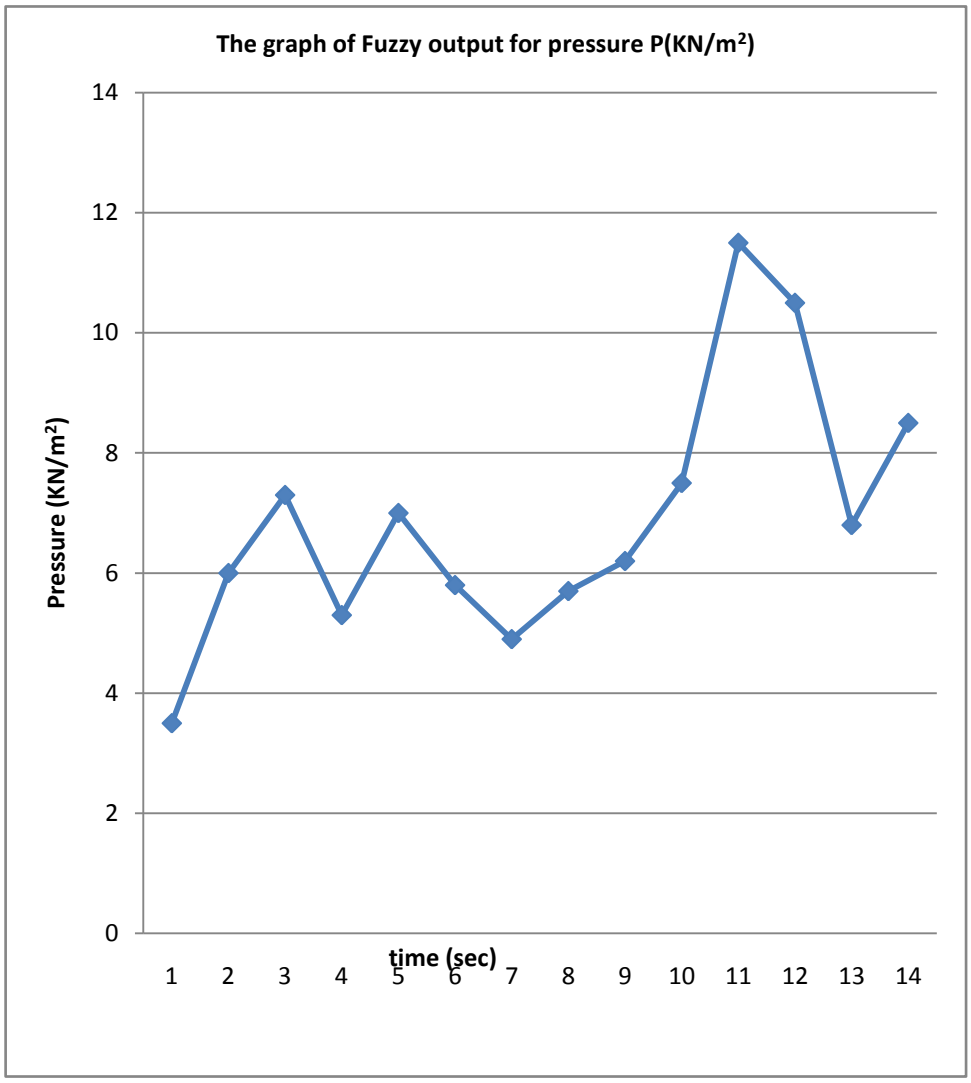


Figure 5.3: Graph of fuzzy output for pressure

From figure 5.3, the graph shows that the pressure in fuzzy logic network was not steady but got at its peak at 11s. The pressure of the liquid during

the systems operation rose to 7.2KN/m^2 at 3s, went up and down and got at its peak at 12s.

5.3 The Neural Network Result and Analysis:

Sometimes the data generated by the fuzzy are not whole numbers and the fuzzy network is not intelligent to approximate it and deduce an output. Hence the application of Neural Network helps to fine tune the data and deduce an action. Table 5.2 gave rise to Table 5.4 which is the data for outflow rate which the Neural Network fine-tunes while Figure 5.4 shows the graph.

Table 5.4: Data of Neural Network for outflow rate

Rout (cm^3/s)	24	24	25	30	51	90	101	112	111	112	113	118	121	120
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

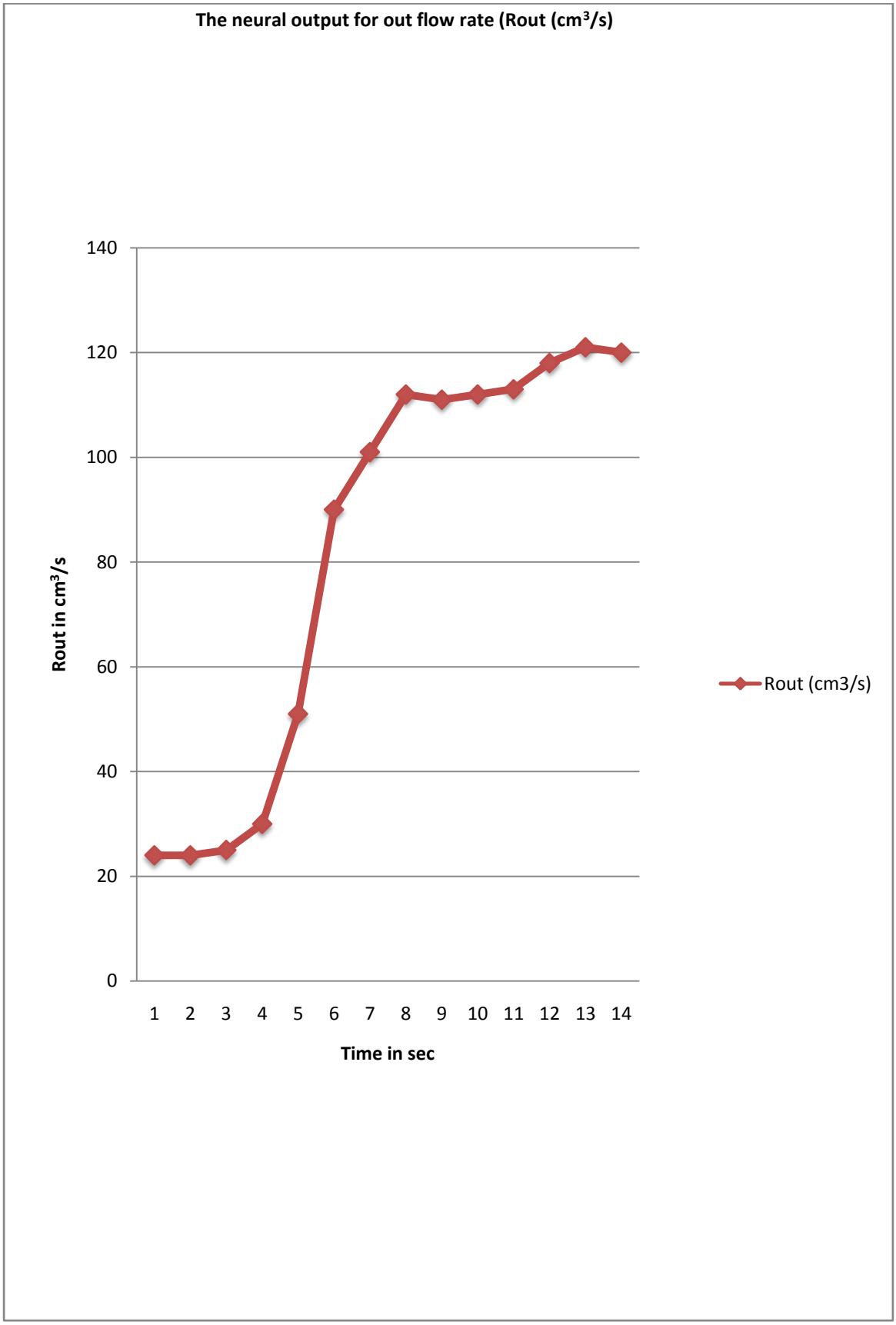


Figure 5.4: Neural output for outflow rate

Figure 5.4 shows the output of outflow rate performed in neural network. It rose gradually from 3s and gave a settling time of 13s. This performed much better than that of fuzzy network because the neural network approximates data in fractions and deduces an output.

Table 5.5 shows the data for temperature after fine tuning by the neural network while Figure 5.5 graph.

Table 5.5: Data of neural output for temperature

T(oC)	30	30	35	35	40	51	51	58	60	60	71	79	80	79
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

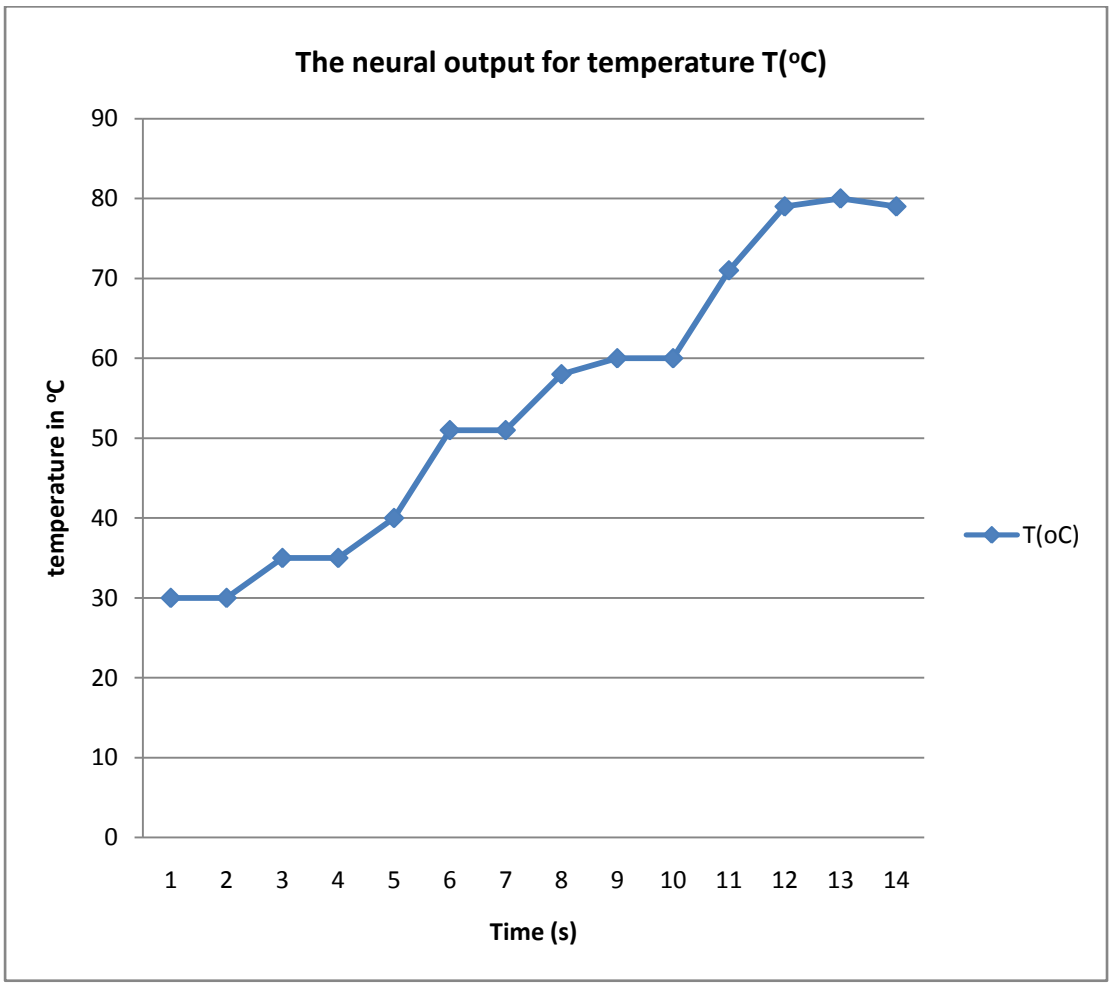


Figure 5.5: Graph of the Neural output for temperature

In figure 5.5, the rise time for the temperature was 9s before undershoot of 1s and a further rise time of 2s. The neural network performed better than the fuzzy logic. There is no much change temperature during the systems operation when compared with that of the fuzzy network.

Table 5.6 shows the data of pressure for the Neural Network after fine tuning while Figure 5.6 the graph.

Table 5.6: Data of pressure for the neural network

P(KN/m^2)	4	6	7	5	7	6	5	6	6	8	12	11	7	9
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

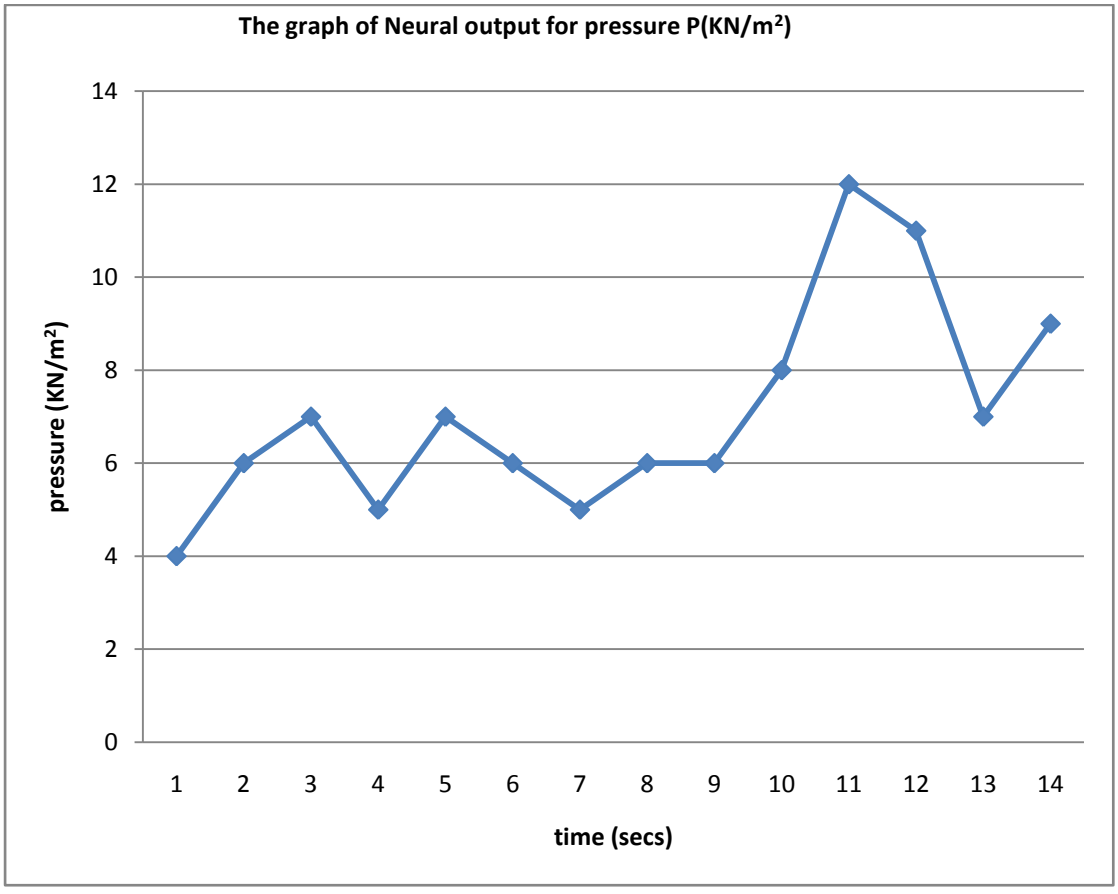


Figure 5.6: Graph of Neural output for pressure

In figure 5.6, the pressure of the liquid was going up and down and became very high at 13s .This is the same when compared with the fuzzy output.

5.4 The Fuzzy Neural Network Result and Analysis:

However, the combination of the Fuzzy Network and Neural Network directly fine-tunes the data generated by the fuzzy and deduce an action. Table 5.7 shows the data of Fuzzy Neural for outflow rate while Figure 5.7shows the graph.

Table 5.7: Data of Fuzzy Neural for outflow rate

$R_{out}(cm^3/s)$	23	24	24	24	110	110	111	111	112	112	119	119	120	120
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

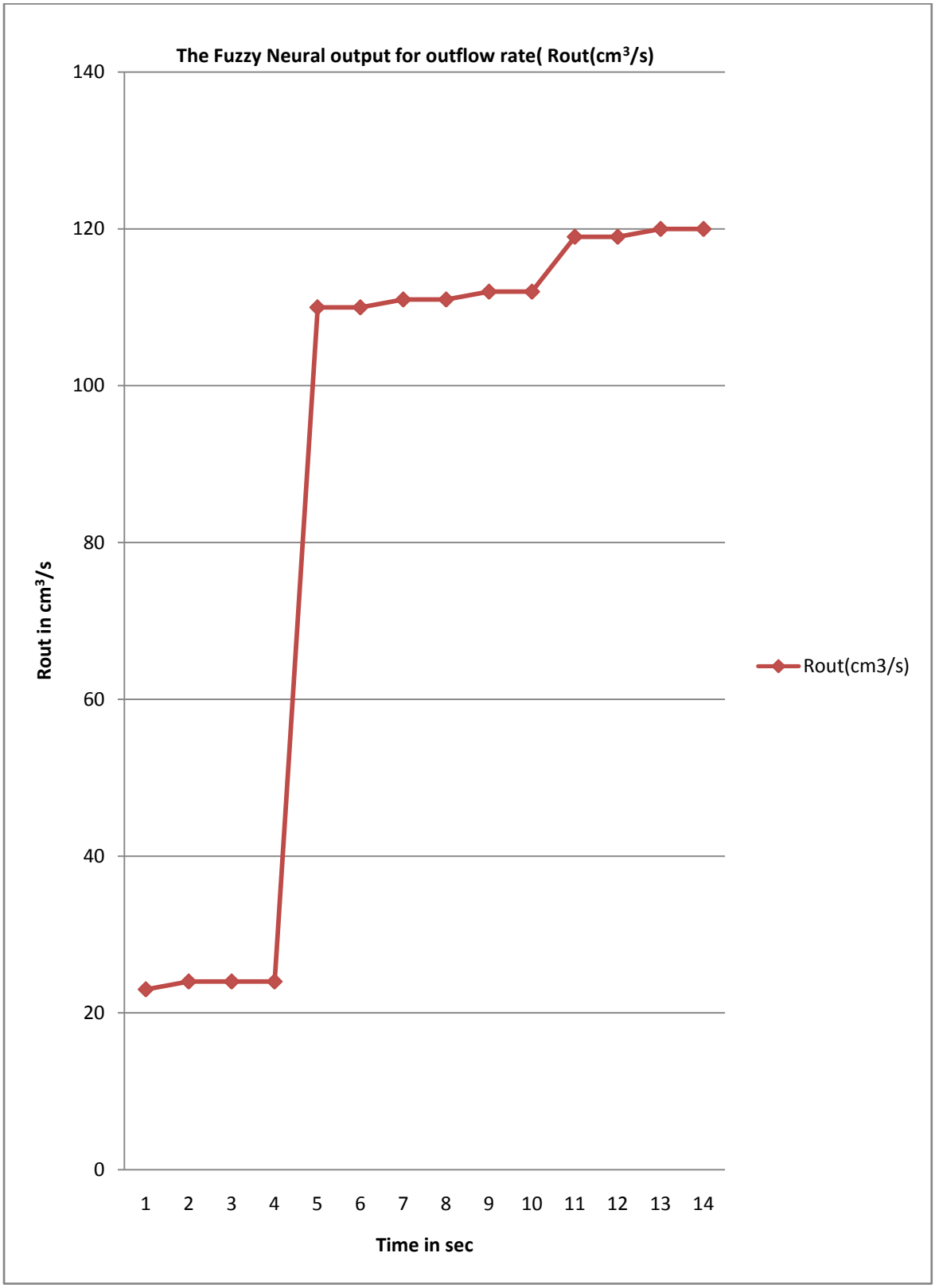


Figure 5.7: Fuzzy Neural output for outflow rate

The outflow rate of Fuzzy Neural plotted in graph (see figure 5.7) shows that the Fuzzy Neural Network performed better than Neural Network

Table 5.8: Data of Fuzzy Neural for temperature

$T(^0C)$	30	40	45	50	50	50	50	59	60	58	57	58	59	59
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

alone. There was a sharp rise after 4s, this increased gradually and stabilized after 6s.Table 5.8 shows the data of Fuzzy Neural for temperature.

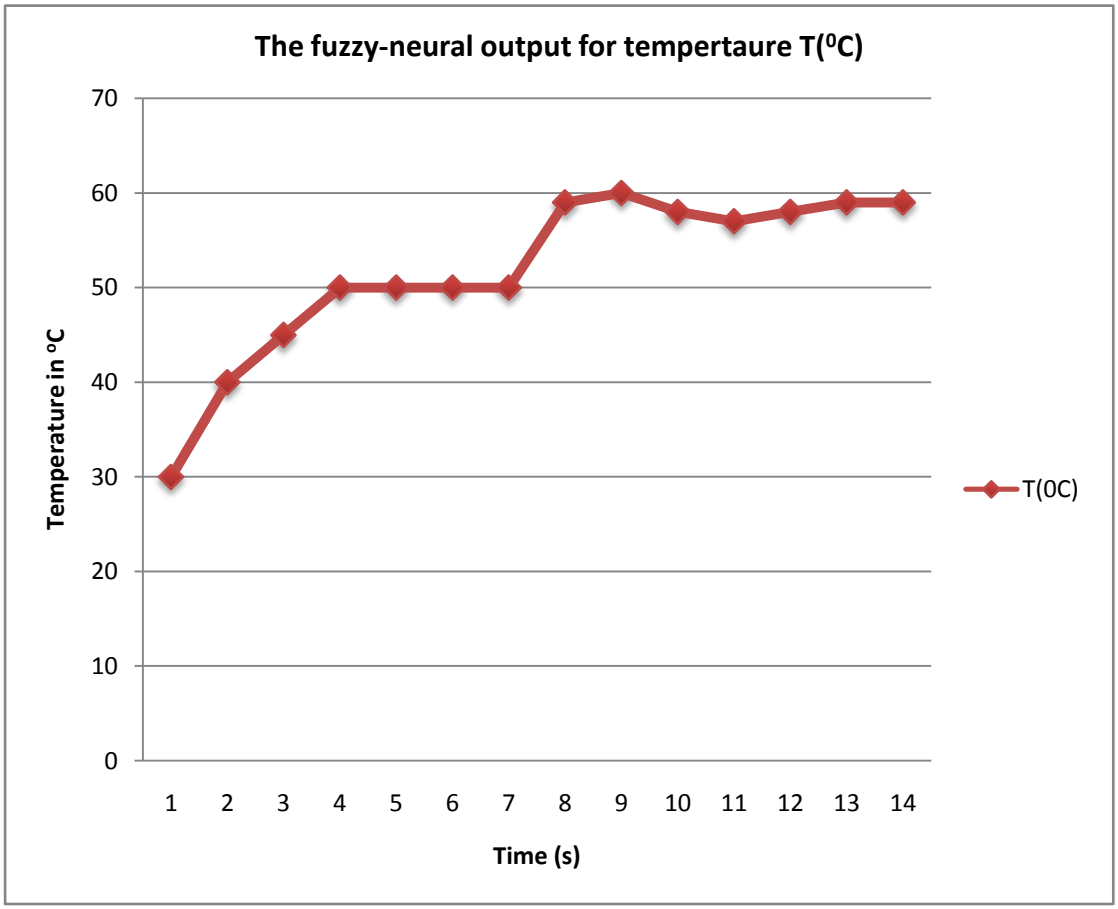


Fig 5.8: Graph of Fuzzy Neural output for temperature

Figure 5.8 shows the graph of temperature in fuzzy-neural network respectively. Here the temperature rose from (30 to 50) ^{O}C for 3s stabilized for 3s, rose to 60^{o}C for 1s and finally stabilized. This is much better than the fuzzy network and neural network alone.

Table 5.9 shows the data of Fuzzy Neural for pressure while Figure 5.9 shows the graph.

Table 5.9: Data of Fuzzy Neural for pressure

P(KN/m^2)	3	6	7	5	5	5	5	5.5	6	6	6.5	5	5	5
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

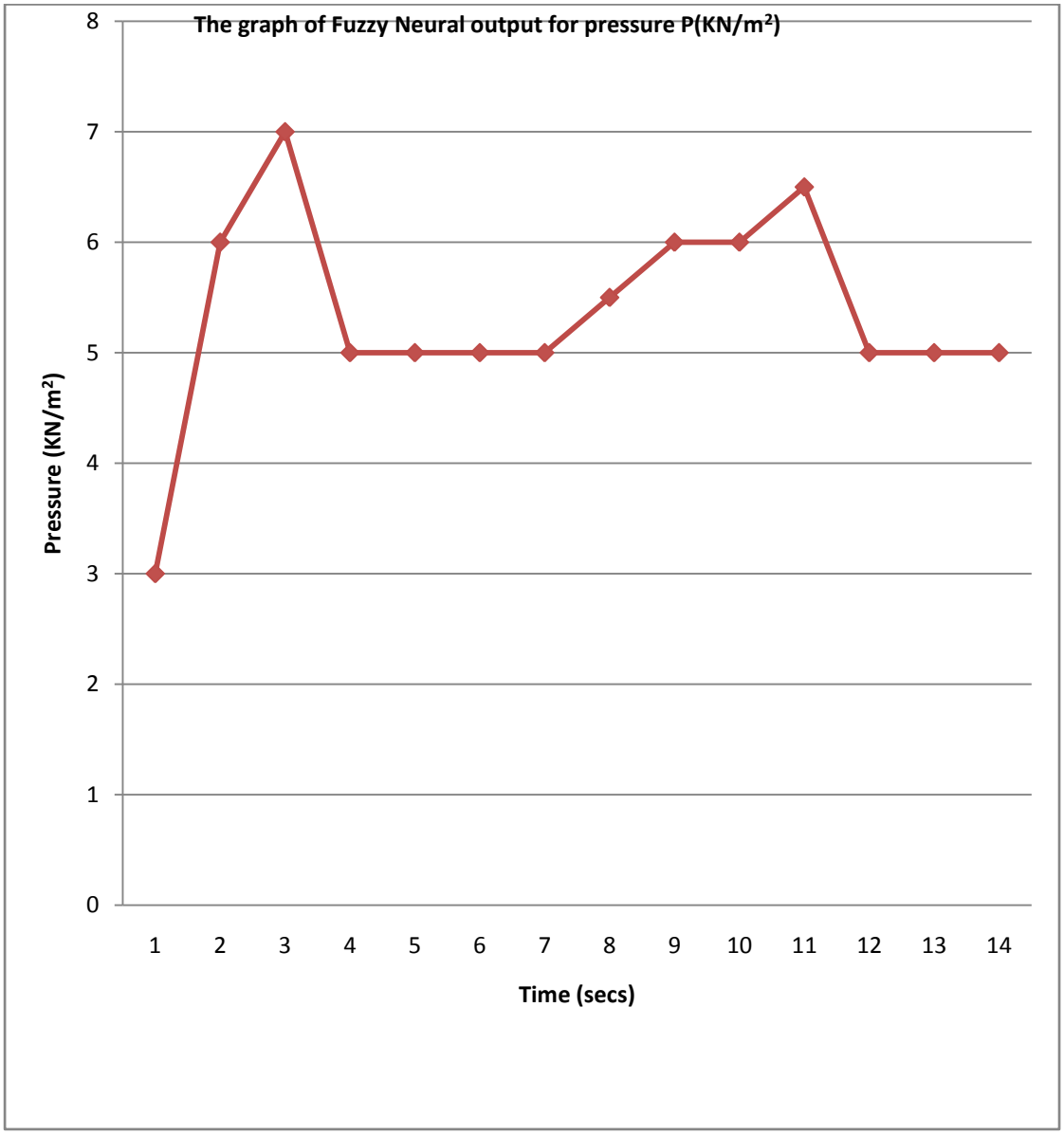


Figure 5.9: Graph of Fuzzy Neural output for pressure

In Figure 5.9 shows the graph of fuzzy neural output for pressure. Here, there was an overshoot of pressure at 3s which game to normal at 5s to 7s, increases again within 4s and finally normalizes at 2s.

5.5 Performance Evaluation

It was observed that the Intelligent Process Control Systems Using Fuzzy Neural Network gave a steep rise time of 4s and stabilized after 6s. The Intelligent Process Control Systems using Fuzzy network rose sluggishly

after an undershoot at 5s and stabilized after 9s while the Intelligent Process Control Systems using Neural Network kicked off its rise time at 3s and stabilized after 9s. The Fuzzy Neural model performed better than the fuzzy and the neural network respectively considering the rising time and stabilizing time. When compared with classical PID whose rise time was 3s and took 2s to stabilize, the Fuzzy Neural model performed best with a rise time of 1s and showed a smoother overall performance (see Table 5.10 and Figure 5.10).

Similarly, comparison was also shown for temperature output for Fuzzy, Neural and Fuzzy Neural for the PID controller. It was observed that for the Fuzzy Neural, the temperature started rising and got stabilized at a higher temperature when compared with Fuzzy Logic and Neural Network output alone. Table 5.10 shows the data while Figure 5.11 shows the graph.

Table 5.10: data for outflow rate for Fuzzy, Neural, and Fuzzy Neural network

R_{out}(cm^3/s) for FNN	23	24	24	24	110	110	111	111	112	112	119	119	120	120
Rout (cm^3/s) for fuzzy	23.5	24.3	25.1	50.7	25.1	90.3	100.8	112.2	111	112.3	112.5	118.3	120.6	119.9
Rout (cm^3/s) for neural	24	24	25	30	51	90	101	112	111	112	113	118	121	120
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

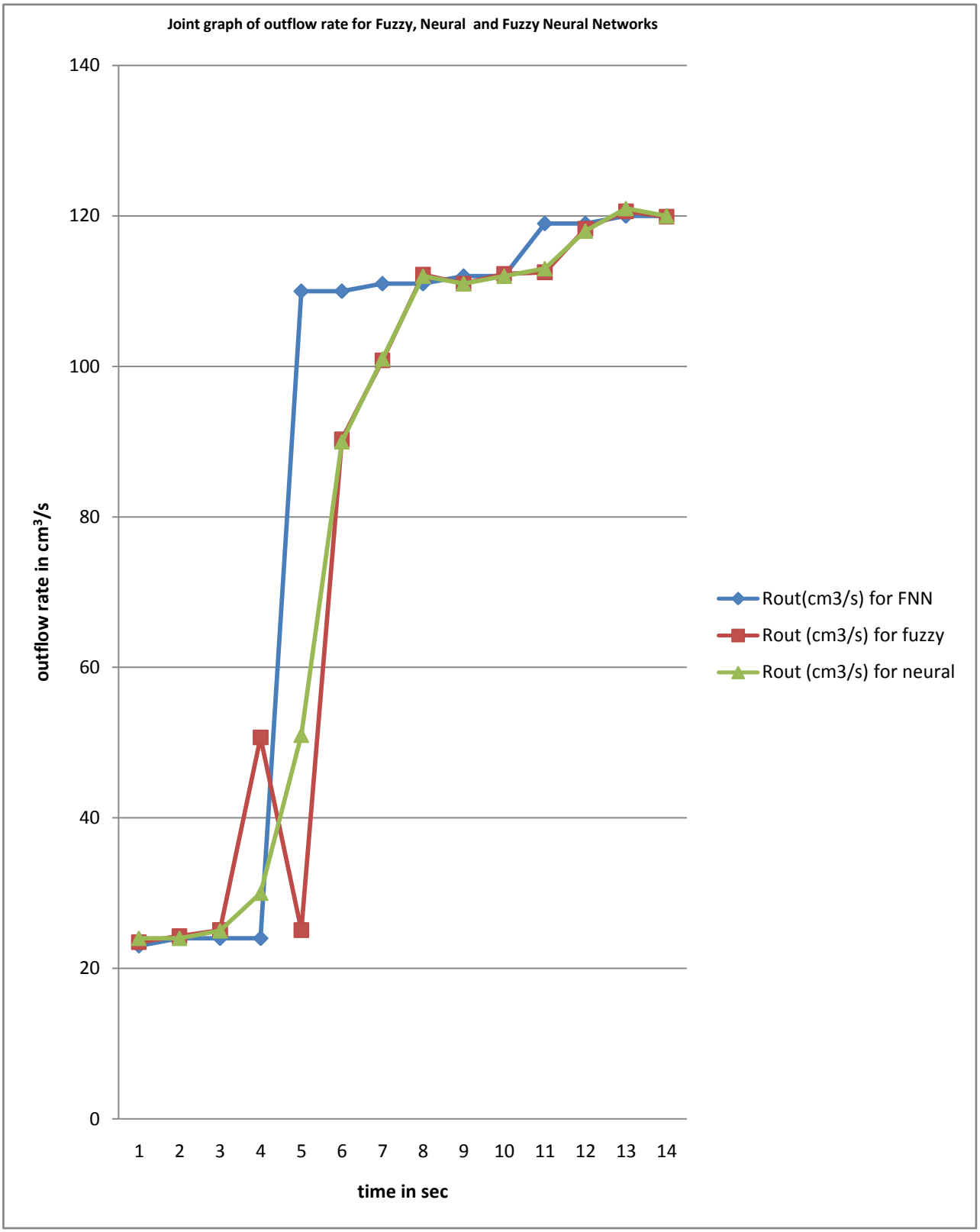


Figure 5.10: The combination of graph of Fuzzy, Neural and Fuzzy Neural for pressure

Table 5.11: data of temperature for Fuzzy, Neural, and Fuzzy Neural Network

T(^{o}C) for fuzzy	29.5	30.1	34.6	35.2	40	50.6	50.9	57.8	59.5	60.4	70.8	80.1	80.3	79.4
T(^{o}C) for neural	30	30	35	35	40	51	51	58	60	60	71	79	80	79
T(^{0}C) for fuzzy-neural	30	40	45	50	50	50	50	59	60	58	57	58	59	59
T(s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

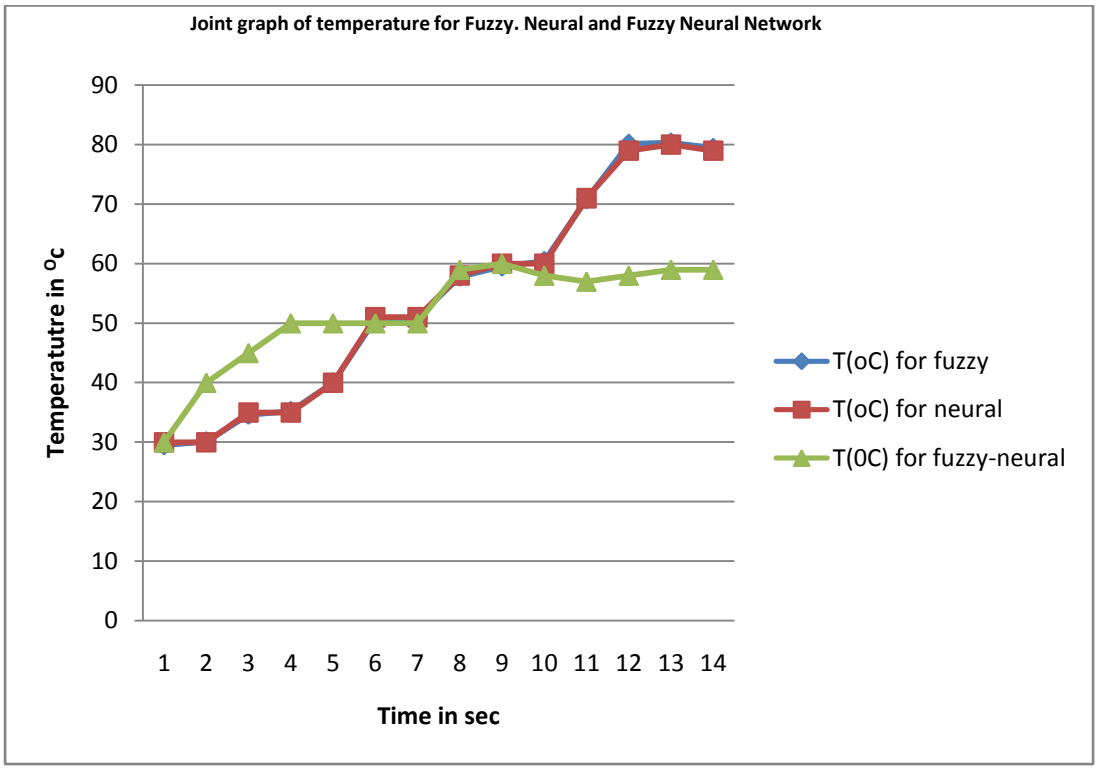


Figure 5.11: Joint graph of temperature for Fuzzy, Neural, and Fuzzy Neural

In Figure 5.11, the graphs of temperature for Fuzzy, Neural, and Fuzzy Neural Network were compared. It was observed that there was uniformity for change in temperature of the liquid for both Fuzzy and Neural Network. The change in temperature for Fuzzy Neural Network was minimal and better.

CHAPTER SIX

SUMMARY AND CONCLUSION

6.1 Summary of Achievement: Though very tedious and time consuming, yet the joy was that a system has been developed to monitor, detect, diagnose and indicate faults in control systems using Fuzzy Neural logic principles.

6.2 Problems Encountered: So many problems which include the initial sourcing of research material and majorly financing were encountered. The process of development of VB.net software, the training of the neurons and the design of the fuzzy controller was a big challenge. The hard disk got crashed in the process and all the data were lost so the work has to be re-started all over again.

6.3 Contribution to Knowledge:

1) In this research work, Fuzzy Logic PID controller is used to model a control system and it has been shown that it is possible without passing through the rigorous mathematics needed to establish a transfer function for the system under control.

2). An Artificial Neural Network (ANN) has been used to optimize the performance of a Fuzzy Logic PID controller given rise to what is known as Fuzzy Neural PID controller. This optimization led to a faster response time of 4s compared to 5s for the Fuzzy based PID alone.

3) Error detection and isolation has been implemented in a Fuzzy Neural PID controller thereby obviating the need of trial and error in fault diagnosis.

6.4 Suggestion for Further Improvement: Any important document gathered should be saved over the internet through the e-mail. Genetic Algorithm techniques should be applied to see if there will be any improvement over the Fuzzy Neural. The research was software based therefore a hardware implementation can be worked on.

6.5 Conclusion:

A Fuzzy Neural PID controller has been successfully developed without passing through the rigorous mathematics needed to establish a transfer function for the system under control. The Fuzzy PID control system was developed and optimized using Neural Network which gave rise to a faster response time. The Fuzzy Neural PID control system has also been successfully simulated and compared with classical PID controller to highlight the performance superiority of the Fuzzy Neural PID control system. Fuzzy logic based PID controller is easier to implement than classical PID controllers and Neural Network fine-tunes it to obtain a Fuzzy Neural system which produces a better response time in the output. Also, the developed Fuzzy Neural system features error detection and presents detailed rectification steps. This is much better than the trial and error method used in repair and maintenance of process control systems.

6.6 Recommendation:

It is strongly recommended that this technique be applied in every process control system because, it will drastically reduce the trial and error method in repairs and maintenance. Furthermore, it is easy to implement and use, and does not require much mathematical rigor even for complex control systems.

REFERENCES

Abraham, A. (1997) . Adaptation of Fuzzy Inference System Using Neural Learning, Fuzzy System Engineering: Theory and Practice.

Abraham A., Nath B.(2001) A Neuro-Fuzzy Approach for Forecasting Electricity Demand InVictoria, Applied Soft Computing Journal, Elsevier Science, 1&2, pp. 127-138.

Ahlawat, Nishant, Ashu Gautam, and Nidhi Sharma (2014) "Use of Logic Gates to Make Edge Avoider Robot." International Journal of Information & Computation Technology (Volume 4, Issue 6; page 630) ISSN 0974-2239 (Retrieved 27 April 2014)

Ajanagadde V, Shastri L.(1991). Rules and variables in neural networks. Neural Computing 3:121-134.

Aleksander I (ed). (1989) Neural Computing Architectures. The Design of Brain-like Machines.Cambridge, Mass, MIT Press. Aleksander I, Morton H. 1990. An Introduction to Neural Computing.London, Chapman & Hall.

Amit D. (1989). Modelling brain function: The world of attractor neural networks. Cambridge, England, Cambridge University Press.

Anderson J. (1995). An Introduction to Neural Networks. Cambridge, Mass, MIT Press.

Arbib M (1995). The Handbook of Brain Theory and Neural Networks. MIT Press.

Abiyev, R.H., O. Kaynak, T. Alshanableh, F. Mamedov, (2011). A type-2 neuro-fuzzy system

Ang, K. K, Quek, C, and Pasquier, M. (2003). POPFNN-CRI(S): Pseudo outer product based fuzzy neural network using the compositional rule of inference and singleton fuzzifier. IEEE Transactions on Systems, Man and Cybernetics, Part B, 33(6): 838-849.

Ang, K.H., Chong, G.C.Y., and Li, Y. (2005). PID control system analysis, design, and technology, IEEE Trans Control Systems Tech, 13(4): 559-576.

Anthonio, M.C. (2002). Intelligent Car Parking Using Fuzzy Neural Networks, Emerging Technologies, Paper No:008 pg.1-6.

Bart, K. (1991) Neural Networks and Fuzzy Systems: a dynamical systems approach to machine intelligence, Prentice-Hall, Inc., Upper Saddle River, NJ.

Barlett P. 1993. The sample size necessary for learning in multi-layer networks. In Proceedings of theFourth Australian Conference on Neural Networks, Melbourne, Australia, Sydney University Electrical Engineering, 14-17.

Beale R, Jackson T. (1990). Neural Computing—Introduction. Bristol, England, Adam Hilger.

Bezdek J, Pal. S. (ed) (1992). Fuzzy Models for Pattern Recognition. New York, IEEE Press.

Binaghi E. (1992). Empirical learning for fuzzy knowledge acquisition. In Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, pp 245-251.

.

Borisyuk R, Holden A, Kryukov V (1991). Interacting neural oscillators can imitate selective attention. In Neurocomputers and Attention. Neurobiology, Synchronization and Chaos. Manchester,England, Manchester University Press, pp 189-200.

Bulsara A, Jacobs E, Zhou T, (1991). Stochastic resonance in a single neuron model: Theory and analog simulation. Journal of Theoretical Biology 152:531-555.

Cheng-J, L., Lee, C-Y and Cheng-C, C, (2006). Temperature control using neuro-fuzzy controllers with m compensatory operations and wavelet neural networks, Journal of Intelligent and Fuzzy Systems, 17: 145–157.

Chin-Teng, L. and Lee, C. S. G (1991). Neural Network-Based Fuzzy Logic Control and Decision System, IEEE Transactions on Computers, pp 1320-1336.

Chen S-M. (1988). A new approach to handling fuzzy decision making problems. IEEE Transactions on Systems, Man and Cybernetics 18:1012-1016.

Dorf R.C and Bishop R.H (2004), Modern Control Systems: Robust Control pp 688-698

Dubois D, Prade H.(1988). Possibility Theory. An Approach to Computerized Processing of Uncertainty. New York, Plenum Press.

Eckmiller R, Napp-Zinn H.(1993). Information processing in biology-Inspired pulse coded neural networks in Proceedings of the International Joint Conference on Neural Networks, Nagoya, Japan, IEEE, pp 643-648.

Ellis G. (2012) Control System Design Guide Using Your Computer to Understand and Diagnose Feedback Controllers 4th Edition published by Butterworth Heineman, July 2014, ISBN 9780123859204

Feigenbaum M. (1989). Artificial Intelligence, A Knowledge-Based Approach. Boston, PWS.

Feldkamp A, Puskorius G, Yuan F (1992). Architecture and training of a hybrid neural-fuzzy system. In Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks, lizuka, Japan, pp 131-134.

Fletcher D, Goss E. (1993). Forecasting with neural networks. An application using bankruptcy data. Journal of Information and Management 24:159-167.

Freeman J, Skapura D. (1992). Neural Networks Algorithms, Applications and programming techniques, Addison-Wesley Publ. Comp., Reeding, Massachusetts.

Fogelman F, Lamy B, Viennet E. (1993). Multimodular Neural Network architectures for pattern recognition. Internationl Journal of Pattern Recognition and Artificial Intelligence,7(4).

Fu K.S (1970) Learning Control Systems ---Review and Outlook, IEEE Transactions on Automatic Control

Fukushima K, Miyake S, Ito T. (1983). Neocognition: A neural network model for a mechanism of visual pattern recognition. IEEE Transactions on Systems, Man and Cybernetics 13:826-834.

Funahashi K. (1989). On the approximate realization of continuous mappings by neural networks. Neural Networks 2:183-192.

Fuzzy CLIPS, (1994). User's Manual. NRC (National Research Council) Canada.

Gallinari P, Thinia S, Fogelman-Soulie F. (1988). Multilayer perceptrons and data analysis in Proceedings of IEEE International Conference on Neural Networks, vol 2, July 24-27, 1988, pp 1391-1399.

Goonatilake S, Campbell J. (1994). Genetic fuzzy hybrid systems for decision making. In Proceedings of the 1994 IEEE/Nagoya University World Wisemen/women Workshop, Nagoya, Japan, pp 143-155.

Gupta M. (1992). Fuzzy logic and neural networks. In Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, pp 157-160.

Hertz , A., Krogh, R and Palmer G (1991), Introduction to the theory of neural computation.

Hayashi Y. (1991). A neural expert system with automated extraction of fuzzy if-then rules and its application to medical diagnosis. In Lippman RP, Moody JE, Touretzky DS (eds), Advances in Neural Information Processing Systems, ed 3. San Mateo, Calif, Morgan Kaufmann, pp 578-584.

Hech-Nielsen R. (1988). Application of counter propagation networks. Neural Networks 1:131-139.

Holland J. (1975). Adaptation in Natural and Artificial Systems. The University of Michigan.

Hornik K. (1991). Approximation capabilities of multilayer feedforward networks. Neural Networks 4:251-257.

Hoskins JC, Himmelbaum DM. (1990). Fault detection and diagnosing using artificial neural networks. In Mavrovouniotis ML (ed), Artificial Intelligence in Process Engineering Orlando, Fla, Academic Press, pp 123-160.

Huifang, K.; Ren Guoqing; Jizhu, H. and Benxian, X. (2011)The Application of Fuzzy Neural Network in Fault Self-diagnosis System of Automatic Transmission.Journal of Software, Vol.6No. , pp.209-212

Huynh, H. T. and Won, Y. (2011,Artificial Neural Network Theory and Simulation Mechanical Industry Press, Beijing, pp.44-68).

Iniya Raghavi S., Lalitha P.(2014) Effective F-Score Feature Selection (KFFS) and Fuzzy Neural Network (FNN) to Classify Congestive Heart Failure Patients in International Journal of Engineering Research & Technology Vol. 3 - Issue 11 (November - 2014) e-ISSN: 2278-0181

Jinghua, Z (2006). PID Controller Tuning: A Short Tutorial Retrieved

Kandel A (ed). (1991). Fuzzy Expert Systems. Boca Raton, Fla, CRC Press.

Kar, S.; Das, S.; Ghosh, P.K. (2014): Applications of Neuro-Fuzzy Systems: A Brief Review and Future Outline, Applied Soft Computing, Vol.15, pp. 243-259

Kataria S.K and Sonns (2008) Automatic Control Systems: State Space Analysis of Control Systems pp 350-407

.

Kazuo, T and Hua, O. W (2001). Fuzzy control systems design and analysis: a linear matrix inequality approach. John Wiley and Sons: ISBN 9780471323242.

Kasabov N. (1995). Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. Fuzzy Sets and Systems, special issue "Hybrid conn. systems".

.

Keller J, Chen Z. (1992). Learning in fuzzy neural networks utilising additive hybrid operators. In Proceedings of the Second International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, July 17-22, 1992, pp 85-87.

King, M (2010). Process Control: A practical Approach. Chinchester, UK: John Wiley and sons Ltd ISBN 978-0-470-97587-9.

Kim D. H., Hong W. P., Park J. I. (2002). Auto-tuning of reference model based PID controller using immune algorithm, IEEE international conference on evolutionary computation, Hawaii, pp. 483-488.

Kim D. H.(2002). Intelligent tuning of a PID controller using an immune algorithm, Transactions of KIEE, 51-D, 1 pp. 78-91.

Kim D. H.(2004). PID Controller Tuning of a Boiler Control System Using Immune AlgorithmTyped Neural Network, 4th International Conference Computational Science, Lecture Notes in Computer Science, pp. 695-698.

Kim D. H., Cho J. H.(2004).Robust PID Controller Tuning Using Multi objective Optimization Based on Clonal Selection of Immune Algorithm, 8th International Conference Knowledge-Based Intelligent Information and Engineering Systems, Lecture Notes in Computer Science, Springer, pp.50-56.

Kohers G. (1992). The use of modular neural networks on time series forecasting. In Proceedings of the 23rd Annual Meeting of the Decision Sciences Institute, San Francisco, pp 759-761.

.

Kosko B. (1991). Neural Networks for Signal Processing. Englewood Cliffs, NJ, Prentice Hall.

Kosko, B (1992). Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence. Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-611435-0.

Langavi, R. and Berenji, H.R.(1992) Fuzzy logic, Fuzzy logic controller. Part I book of Intelligent Control. Van Nostrand, New York.

Lee, C-H., Yu, T-M, and Chien, J-C (2011) Adaptive Neural Network Controller Design for a Class of Nonlinear Systems Using Simultaneous Perturbation Stochastic Approximation (SPSA), Proceedings of the International Multi conference of Engineers and Computer Scientists (Vol II).

Lee, C.C. (1990), Fuzzy logic in control systems: Fuzzy logic controller. Part I and Part II. IEEE. Trans.System Cybernet. pp 20-26.

Lima A. M. N., Jacobina C. B., Souza Filho E. B. (1997). Nonlinear parameter estimation of steady-state induction machine models, IEEE Trans. Industrial Electronics.

Lim M, Rahardja S, Gwee B. (1995). A GA paradigm for learning fuzzy rules. Fuzzy Sets and Systems, Special Issue "Hybrid Connectionist Systems".

Limkens D, Nie J. (1992). Rule extraction for BNN neural network-based fuzzy control system by self learning. In Aleksander I, Taylor J (eds), Artificial Neural Networks, vol 2. Amsterdam, Elsevier Science, pp 459- 466.

Lippman R. (1987). An introduction to computing with neural nets. IEEE ASSP, April, pp 4-21.

Lin, C.T. and Lee, C. S. G. (1996). Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems. Upper Saddle River, NJ: Prentice Hall.

Lobbrecht A.H and Solomatine D.P (1999). Intelligent control schemes for reproducing optimal control actions for polder water level, Water Industry Systems, Modeling and Optimization Applications pp.1-10.

Menychtas, A. Konstanteli, K. (2012), Fault Detection and Recovery Mechanisms and Techniques for Service Oriented Infrastructures, Achieving Real-Time in Distributed Computing: From Grids to Clouds, IGI Global, pp. 259–274

Mitchell, M .A., Lopes, P,, J.A., Fidalgo, J.N and McCalley, J.D (2000). Neural Network to predict the Dynamic Frequency Response of a power system to an Under-frequency Load Shedding Scenario, I.E.E.E Journal pp. 346-351.

Mohamed, A. H. (2014) A Fuzzy Neural Network Fault Diagnostic System International Journal of Computer Applications (0975 –8887)Volume 94–No 1, May 2014,

Onkar P.N (2010), Principles of Process Control pp 252-284

Park B. J.(2002). Fuzzy polynomial neural networks: Hybrid architectures of fuzzy modeling, IEEE Trans. on Fuzzy systems,10, 5, pp. 607-621.

Peymanfar, A., A. Khoei, Kh. Hadidi, (2010). Design of a general propose neuro-fuzzy controller by using modified adaptive-network-based fuzzy inference system, AEU–International Journal of Electronics and Communications, 64: 433-442.

Pislaru, M., Trandabat, A and Olariu, M (2003) Neurofuzzy System for Industrial Processes Fault Diagnosis.

Prabakaran M. P., Kannan G. R., Thirupathi K. , Hari Prakash A.(2014) Optimization Turning Process Parameters of Aluminum Alloy 5083 using Response Surface Methodology in International Journal of Engineering Research & Technology Vol. 3 - Issue 4 (April- 2014) e-ISSN: 2278-01

Quek, C. and Zhou, R. W. (1999). "POPFNN-AAR(S): a pseudo outer-product based fuzzy neural network." IEEE Transactions on Systems, Man and Cybernetics, Part B, 29(6), 859-870.

Rakesh Gautam, Rajesh Ingle, Milin Nagpure (2014) Modeling And Control of 3 Linkbiped Leg using PID Controller in International Journal of Engineering Research & Technology Vol. 3 - Issue 4 (April- 2014) e-ISSN: 2278-0181

Ross, T.J (1995). Fuzzy Logic with Engineering Applications, McGraw-Hill, Hightstown, NJ.

Seema C., Mitra, R. and Vijay K. (2007) Neural Network Tuned fuzzy controller for Multiple Input Multiple output (MIMO),World Academy of Science, Engineering and Technology pp.485-491.

Sijimol A. S, Pooja V, Soji K. (2014) Cardiovascular Disease Diagnosis Using Fuzzy Petri Net in International Journal of Engineering Research & Technology Vol. 3 - Issue 4 (April- 2014) e-ISSN: 2278-0181

Sklansky J. (1966) Learning Systems for automatic Control. IEEE Transactions on Automatic Control

Stathacopoulou R.; Magoulas G.D.; Grigoriadou, M. and Samarakou, M.(2011): Neural-Fuzzy Knowledge Processing in Intelligent Learning Interactive Neuro-Fuzzy Expert System for Diagnosis of Leukemiam Global,Journals Inc. pp.112-130.

Tan, K. K., Wang, Qing-G and Hang Chang, C. (1999). Advances in PID Control, London, UK: Springer-Verlag. ISBN 1-85233-138- 0.

Tanomaru, J. and S. Omatu, (1992). Process Control by On-Line Trained Neural Controllers , IEEETrans. Ind. Electron., 39: 511-521.

Valiant, Leslie, (2013) Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World New York: Basic Books. ISBN 978-0465032716

Verma P, Verma N, Bhandari S (2014), Modeling of Adaptive Artificial Neural Networks using VHDL is More Appropriate using Bipolar Inputs in International Journal of Engineering Research & Technology Vol. 3 - Issue 4 (April- 2014) e-ISSN: 2278-0181

Vijay M.S, Popat R.A, Khot Sachin B, Burle K. J (2014) Paper on Recent Development in Artificial Neural Control in Roboticsin International Journal of Engineering Research & Technology Vol. 3 - Issue 11 (November - 2014) e-ISSN: 2278-0181

Wayne Bequette B. (2003). Process Control: Modeling, Design, and Simulation. Prentice Hall Professional. p. 5. ISBN 9780133536409.

Wei Peng and Da-fa Zhang, (2010) Research on Fuzzy Control for Steam Generator Water Level.

Wu, D,, Karray, F. and Song, I (2003) Water Level Control by Fuzzy Logic and Neural Network.

Yager R, Zadeh L (eds). (1992). An Introduction to Fuzzy Logic Applications in Intelligent Systems,Boston, Kluwer Academic.

Yi H-J, Oh KW. (1992). Neural network-based fuzzy production rule generation and its application to approximate reasoning. In Proceedings of the Second International Conference of Fuzzy Logic andNeural Networks. Iizuka, Japan, , pp 333-336.

Zadeh L. (1984). Making computers think like people. IEEE Spectrum, August, pp 26-32.

Zimmermann H. (1987). Fuzzy Sets, Decision Making, and Expert Systems. Boston, Kluwer Academic.

Zurada J. (1992). Introduction to Artificial Neural Systems. St Paul, Minn, West.

Zhou, R. W and Quek, C. (1996). POPFNN: A Pseudo Outer-product Based Fuzzy Neural Network; Neural Networks, 9(9), 1569-1581.

APPENDIX A

The Software for the Real Time Simulation of the neural network

```
adc_a      bit p2.0
adc_b      bit p2.1
adc_c      bit p2.2
adc_start  bit p2.3
adc_ale    bit p2.4
adc_clk    bit P2.5
valve_1    bit p2.6
valve_2    bit p2.7
latch_1    bit p3.2
latch_2    bit p3.3
latch_3    bit p3.4
latch_4    bit p3.5
buzzer     bit p3.6
phase_1    equ 30h
phase_2    equ 31h
phase_3    equ 32h
ph1_hund   equ 33h
ph1_ten    equ 34h
ph1_unit   equ 35h
```

```
ph2_hund   equ 36h

ph2_ten    equ 37h

ph2_unit   equ 38h

ph3_hund   equ 39h

ph3_ten    equ 3ah

ph3_unit   equ 3bh

level_ref  equ 3ch

input      equ p0

keep_data  equ 51h


Org 0000h

clr valve_1

clr valve_2

clr c

clr buzzer

mov p1,#0ffh

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV TMOD,#20H ;timer 1, mode 2

MOV TH1,#-3H ;4800 baud rate

MOV SCON,#50H ;8-bit, 1 stop, REN enabled

SETB TR1 ;sta

mov dptr ,# humid_display
```

```
        CALL TRANS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:
;call read_sensor_0
call adc_process
  ;mov phase_1,a
;call convert1


  ;call  display1
  ;call delay
jmp start
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
read_sensor_0:
            clr adc_a                    ;
            clr adc_b                    ;Select Channel 0
            clr adc_c
            ret


;;;;;;;;;;;;;;;;;;;;;;;;;;;
read_sensor_1:
            setb adc_a                     ;
            clr adc_b                    ;Select Channel 0
```

```
        clr adc_c
        ret
;;;;;;;;;;;;;;;;;;;;;;;;

adc_process:
mov p1,#0ffh
 clr latch_1          ;;;;;;;;;;;;;;;;;;;;;;;;   this for temperature
setb latch_2          ;;;;;;;;;;;;;;;;;;;;;;;;

    call read_sensor_0
    clr adc_ale
    clr adc_start

                        ;
    call delay_small
    setb adc_ale                    ;ale pin high
    call delay_small
    setb adc_start                  ;start pin high
    call delay_small
    clr adc_ale                     ;ale pin low
    call delay_small
```

```
clr adc_start                    ;start pin low
call delay_long
mov a,P1
;add a,#66     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;lllll
mov keep_data,a
call delay_small
mov dptr ,#level_0
call trans
mov a,keep_data
call sensor_1
mov dptr ,#level
call trans
call level_check ;;;;check level
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
call read_sensor_1
clr adc_ale
clr adc_start
                        ;
call delay_small
setb adc_ale                     ;ale pin high
call delay_small
```

```
setb adc_start                  ;start pin high
call delay_small
clr adc_ale                     ;ale pin low
call delay_small
clr adc_start                   ;start pin low
call delay_long
mov a,P1
; add a,#66
mov keep_data,a
call delay_small
mov dptr ,#flow_0
call trans
mov a,keep_data
call sensor_2
mov dptr ,#flow
call trans
call flow_check ;;;;check temperature
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
mov dptr,#set_point
call trans
mov a,input
```

```
        call delay
        mov keep_data,a
        mov level_ref,a
        call sensor_3

        jmp adc_process
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_1:call convert1
          mov a,ph1_hund
          call send
          mov a,ph1_ten
          call send
          mov a,ph1_unit
          call send
          ;mov a,#0dh
          ;call send
          ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_2:call convert2
          mov a,ph2_hund
          call send
          mov a,ph2_ten
```

```
        call send
        mov a,ph2_unit
        call send
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_3:call convert3
        mov a,ph3_hund
        call send
        mov a,ph3_ten
        call send
        mov a,ph3_unit
        call send
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


delay_small:
     mov r0,#70
l1_delay_small:
```

```
    cpl adc_clk
    nop
    nop
    nop
    nop
    nop
    nop
djnz r0,l1_delay_small
    ret



delay_long:
    mov r0,#140
l1_delay_long:
    cpl adc_clk
    nop
    nop
    nop
    nop
    nop
    djnz r0,l1_delay_long
    ret
```

```
delay:
    mov r2,#255
    mov r1,#255
delay1:
    djnz r1,delay1
    djnz r2,delay1
ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
convert1:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph1_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph1_ten,a
        mov a,b
        call decode
        mov ph1_unit,a
```

```
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;
convert2:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph2_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph2_ten,a
        mov a,b
        call decode
        mov ph2_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;
convert3:;mov a,phase_1
        mov b,#100
        div ab
```

```
        call decode
        mov  ph3_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph3_ten,a
        mov a,b
        call decode
        mov ph3_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        decode :orl a,#30h
       ret

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
display1: mov dptr,#humid_display2
        call trans
        mov a,ph1_hund
        call send
        mov a,ph1_ten
```

```
        call send
        mov a,ph1_unit
        call send
        mov a,#0dh
        call send
        ret

;;;;;;;;;;;;;;;;;;;;;;

humid_display  : DB "        SIMULATION OF THE     " ,0dh
               DB "       NEURAL  CONTROLLER  " ,0dh
               DB "        OF TANK WATER LEVEL    " ,0dh
               DB "     ooooooooooooooooooooooooo" ,0dh
               DB "     ooooooooooooooooooooooooo"
               db 0dh,0
humid_display2  : DB "Relative Humidity Value : "
               db 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
level_0 :  db 0dh,0dh
          db "Tank Water Level : ",0
flow_0 :   db "Water Flow Rate  : ",0
set_point: db "Maximum Water Level point: ",0
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  flow : db " Cm^3/s" ,0dh,0
level : db " Cm" ,0dh,0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TRANS:
 H_1: CLR A
      movc a,@a+dptr;get the character
      inc dptr
      call send
      jnz h_1
      ;cjne a,#"@",h_1
      RET



SEND: MOV SBUF,A ;load the data
 H_2: JNB TI,H_2 ;stay here until last bit
      CLR TI ;get ready for next char
      RET ;return to caller

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
level_check:
        clr c
        mov a, keep_data
        cjne a,#2,l_go
        clr valve_1
        clr buzzer
        call delay
        call delay
        ret
l_go:
        subb a,#2
        jnc l_grt_tan0
        jc l_less_tan0

l_less_tan0:
      setb buzzer
      setb valve_1
      call delay
      call delay
      ret
l_grt_tan0:
```

```
        mov a,keep_data
            cjne a,#43,l_go2
            setb buzzer
            setb valve_1
            call delay
            call delay
            ret
l_go2:      subb a,#43
        jnc l_grt_tan1
        jc l_less_tan1
l_grt_tan1:  setb buzzer
        setb valve_1
            call delay
            call delay
        ret
l_less_tan1:  clr buzzer
        clr valve_1
            call delay
            call delay
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
flow_check:
        clr c
        mov a, keep_data
        cjne a,#1,f_go
        setb buzzer
        setb valve_2
        call delay
        call delay
        ret
f_go:
        subb a,#1
        jnc f_grt_tan0
        jc f_less_tan0
f_less_tan0:
    setb buzzer
    setb valve_2
    call delay

    ret
f_grt_tan0:
        mov a,keep_data
          cjne a,level_ref,f_go2
```

```
                setb buzzer
                setb valve_2
                call delay
                call delay
                ret
f_go2:          subb a,#100
            jnc f_grt_tan1
            jc f_less_tan1
            f_grt_tan1 :    call delay
            setb buzzer
            setb valve_2
                call delay
                call delay
            ret
f_less_tan1:    clr buzzer
            clr valve_2
                call delay
                call delay
            ret


    end
adc_a     bit p2.0
```

APPENDIX B

The Software for the Real Time Simulation of the fuzzy-neural network

```
adc_b      bit p2.1
adc_c      bit p2.2
adc_start  bit p2.3
adc_ale    bit p2.4
adc_clk    bit P2.5
valve_1    bit p2.6
valve_2    bit p2.7
valve_3    bit p2.2
valve_4    bit p3.7
heat       bit p3.2
buf1       bit p3.3
buf2       bit p3.4
buf3       bit p3.5
buzzer     bit p3.6
phase_1    equ 30h
phase_2    equ 31h
phase_3    equ 32h
```

```
ph1_hund   equ 33h
ph1_ten    equ 34h
ph1_unit   equ 35h
ph2_hund   equ 36h
ph2_ten    equ 37h
ph2_unit   equ 38h
ph3_hund   equ 39h
ph3_ten    equ 3ah
ph3_unit   equ 3bh
level_ref  equ 3ch
ph4_hund   equ 3eh
ph4_ten    equ 3fh
ph4_unit   equ 40h
ph5_hund   equ 41h
ph5_ten    equ 42h
ph5_unit   equ 43h
pre_ref    equ 44h
temp_ref   equ 45h
ph6_hund   equ 46h
ph6_ten    equ 47h
ph6_unit   equ 48h
ph7_hund   equ 49h
```

```
ph7_ten    equ 4ah

ph7_unit   equ 4bh

input      equ p0

keep_data  equ 51h


Org 0000h

clr valve_1

clr valve_2

clr valve_3

clr valve_4

clr heat

clr c

clr buzzer

mov p1,#0ffh

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV TMOD,#20H ;timer 1, mode 2

MOV TH1,#-3H ;4800 baud rate

MOV SCON,#50H ;8-bit, 1 stop, REN enabled

SETB TR1 ;sta

mov dptr ,# humid_display

        CALL TRANS
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:
;call read_sensor_0
call adc_process
  ;mov phase_1,a
;call convert1

  ;call  display1
  ;call delay
jmp start
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



read_sensor_0:
            clr adc_a                          ;
            clr adc_b                          ;Select Channel 0
            ret


;;;;;;;;;;;;;;;;;;;;;;;;;;;
read_sensor_1:
            setb adc_a                         ;
```

```
        clr adc_b                   ;Select Channel 1
        ret
;;;;;;;;;;;;;;;;;;;;;;
read_sensor_2:
        clr adc_a                   ;
        setb adc_b                  ;Select Channel 2
        ret
;;;;;;;;;;;;;;;;;;;;;;
read_sensor_3:
        setb adc_a                  ;
        setb adc_b                  ;Select Channel 3
        ret
;;;;;;;;;;;;;;;;;;;;;;


adc_process:
mov p1,#0ffh


    call read_sensor_0
    clr adc_ale
    clr adc_start
```

```
;
call delay_small
setb adc_ale                ;ale pin high
call delay_small
setb adc_start              ;start pin high
call delay_small
clr adc_ale                 ;ale pin low
call delay_small
clr adc_start               ;start pin low
call delay_long
mov a,P1
;add a,#66    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;lllll
mov keep_data,a
call delay_small
mov dptr ,#level_0
call trans
mov a,keep_data
call sensor_1
mov dptr ,#level
call trans
call level_check ;;;;check level
```

```
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
call read_sensor_1
clr adc_ale
clr adc_start

                        ;
call delay_small
setb adc_ale                    ;ale pin high
call delay_small
setb adc_start                  ;start pin high
call delay_small
clr adc_ale                     ;ale pin low
call delay_small
clr adc_start                   ;start pin low
call delay_long
mov a,P1
; add a,#66
mov keep_data,a
call delay_small
mov dptr ,#flow_0
call trans
```

```
mov a,keep_data
call sensor_2
mov dptr ,#flow
call trans
call flow_check ;;;;check temperature
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 call read_sensor_2
clr adc_ale
clr adc_start

                    ;
call delay_small
setb adc_ale                    ;ale pin high
call delay_small
setb adc_start                  ;start pin high
call delay_small
clr adc_ale                     ;ale pin low
call delay_small
clr adc_start                   ;start pin low
call delay_long
```

```
mov a,P1
; add a,#66
mov keep_data,a
call delay_small
mov dptr ,#pres_0
call trans
mov a,keep_data
call sensor_3
mov dptr ,#pres
call trans
call pres_check ;;;;check temperature
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  call read_sensor_3
clr adc_ale
clr adc_start

                        ;
call delay_small
setb adc_ale                    ;ale pin high
call delay_small
```

```
setb adc_start                  ;start pin high
call delay_small
clr adc_ale                     ;ale pin low
call delay_small
clr adc_start                   ;start pin low
call delay_long
mov a,P1
; add a,#66
mov keep_data,a
call delay_small
mov dptr ,#temp_0
call trans
mov a,keep_data
call sensor_4
mov dptr ,#temp
call trans
call temp_check ;;;;check temperature
clr c
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mov dptr,#set_water
call trans
```

```
clr buf1
setb buf2
setb buf3
call delay
mov a,input
call delay
mov keep_data,a
mov level_ref,a
call sensor_water

;;;;;;;;;;;;;;;;;;;
mov dptr,#set_pres
call trans
setb buf1
clr buf2
setb buf3
call delay
mov a,input
call delay
mov keep_data,a
mov pre_ref,a
call sensor_pres
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
mov dptr,#set_temp
call trans
setb buf1
setb buf2
clr buf3
call delay
mov a,input
call delay
mov keep_data,a
mov temp_ref,a
call sensor_temp

jmp adc_process
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_1:call convert1
    mov a,ph1_hund
    call send
    mov a,ph1_ten
    call send
    mov a,ph1_unit
    call send
```

```
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_2:call convert2
        mov a,ph2_hund
        call send
        mov a,ph2_ten
        call send
        mov a,ph2_unit
        call send
        ;mov a,#0dh
        ;call send
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_3:call convert3
        mov a,ph3_hund
        call send
        mov a,ph3_ten
        call send
        mov a,ph3_unit
```

```
        call send
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_4:call convert4
        mov a,ph4_hund
        call send
        mov a,ph4_ten
        call send
        mov a,ph4_unit
        call send
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_water:call convert5
        mov a,ph5_hund
        call send
        mov a,ph5_ten
        call send
        mov a,ph5_unit
```

```
        call send
        ;mov a,#0dh
        ;call send
        ret
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_pres:call convert6
        mov a,ph6_hund
        call send
        mov a,ph6_ten
        call send
        mov a,ph6_unit
        call send
        ;mov a,#0dh
        ;call send
        ret
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sensor_temp:call convert7
        mov a,ph7_hund
        call send
        mov a,ph7_ten
        call send
        mov a,ph7_unit
```

```
        call send
        ;mov a,#0dh
        ;call send
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

delay_small:
    mov r0,#70
l1_delay_small:
    cpl adc_clk
    nop
    nop
    nop
    nop
    nop
    nop
djnz r0,l1_delay_small
    ret



delay_long:
    mov r0,#140
```

```
l1_delay_long:
      cpl adc_clk
      nop
      nop
      nop
      nop
      nop
      djnz r0,l1_delay_long
      ret
delay:
     mov r2,#255
     mov r1,#255
delay1:
     djnz r1,delay1
     djnz r2,delay1
ret


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
convert1:;mov a,phase_1
          mov b,#100
          div ab
          call decode
```

```
        mov  ph1_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph1_ten,a
        mov a,b
        call decode
        mov ph1_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;
convert2:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph2_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph2_ten,a
```

```
        mov a,b
        call decode
        mov ph2_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;
convert3:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph3_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph3_ten,a
        mov a,b
        call decode
        mov ph3_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
convert4:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph4_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph4_ten,a
        mov a,b
        call decode
        mov ph4_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
convert5:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph5_hund,a
        mov a,b
```

```
        mov b,#10
        div ab
        call decode
        mov ph5_ten,a
        mov a,b
        call decode
        mov ph5_unit,a
        ;call display1
        ret
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
convert6:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph6_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph6_ten,a
        mov a,b
        call decode
```

```
        mov ph6_unit,a
        ;call display1
        ret
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
convert7:;mov a,phase_1
        mov b,#100
        div ab
        call decode
        mov  ph7_hund,a
        mov a,b
        mov b,#10
        div ab
        call decode
        mov ph7_ten,a
        mov a,b
        call decode
        mov ph7_unit,a
        ;call display1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        decode :orl a,#30h
       ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
display1: mov dptr,#humid_display2
        call trans
        mov a,ph1_hund
        call send
        mov a,ph1_ten
        call send
        mov a,ph1_unit
        call send
        mov a,#0dh
        call send
        ret

;;;;;;;;;;;;;;;;;;;;;;;

humid_display  : DB "         SIMULATION OF THE     " ,0dh
                 DB "       NEURO FUZZY CONTROLLER  " ,0dh
                 DB "        OF TANK WATER LEVEL    " ,0dh
                 DB "     ooooooooooooooooooooooooo" ,0dh
                 DB "     ooooooooooooooooooooooooo"
                db 0dh,0
```

```
humid_display2  : DB "Relative Humidity Value : "
                db 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
level_0 :  db 0dh,0dh
           db "Tank Water Level : ",0
flow_0 :   db "Water Flow Rate  : ",0
pres_0 :   db "Pressure Value   : ",0
temp_0 :   db "Temperature      : ",0
set_water: db 0dh,"Maximum Water Level point: ",0
set_pres:  db 0dh,"Maximum Pressure point   : ",0
set_temp:  db 0dh,"Maximum Temperature Point: ",0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;




flow  : db " Cm^3/s" ,0dh,0
level : db " Cm" ,0dh,0
pres  : db " Pascal" ,0dh,0
temp  : db " 'C" ,0dh,0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TRANS:
 H_1: CLR A
```

```
      movc a,@a+dptr;get the character
      inc dptr
      call send
      jnz h_1
      ;cjne a,#"@",h_1
      RET



SEND: MOV SBUF,A ;load the data
 H_2: JNB TI,H_2 ;stay here until last bit
      CLR TI ;get ready for next char
      RET ;return to caller

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
level_check:
              clr c
              mov a, keep_data
              cjne a,#2,l_go
              clr valve_1
```

```
            clr buzzer
            call delay
            call delay
            ret
l_go:
            subb a,#2
            jnc l_grt_tan0
            jc l_less_tan0

l_less_tan0:
        setb buzzer
        setb valve_1
        call delay
        call delay
        ret
l_grt_tan0:
            mov a,keep_data
               cjne a,#43,l_go2
               setb buzzer
               setb valve_1
               call delay
               call delay
```

```
            ret
l_go2:      subb a,#43
          jnc l_grt_tan1
          jc l_less_tan1
l_grt_tan1:  setb buzzer
          setb valve_1
            call delay
            call delay
          ret
l_less_tan1:  clr buzzer
          clr valve_1
            call delay
            call delay
          ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
flow_check:
          clr c
          mov a, keep_data
          cjne a,#1,f_go
          setb buzzer
          setb valve_2
```

```
        call delay
        call delay
        ret
f_go:
        subb a,#1
        jnc f_grt_tan0
        jc f_less_tan0

f_less_tan0:
    setb buzzer
    setb valve_2
    call delay

    ret
f_grt_tan0:
        mov a,keep_data
          cjne a,level_ref,f_go2
          setb buzzer
          setb valve_2
          call delay
          call delay
          ret
```

```
f_go2:              subb a,#100
            jnc f_grt_tan1
            jc f_less_tan1


f_grt_tan1 :    call delay
            setb buzzer
            setb valve_2
                call delay
                call delay
            ret
f_less_tan1:    clr buzzer
            clr valve_2
                call delay
                call delay
            ret


 ;;;;;;;;;;;;;;;;;;;;;;;;;;
 pres_check:
            clr c
            mov a, keep_data
            cjne a,#1,p_go
            setb buzzer
```

```
        setb valve_3
        call delay
        call delay
        ret
p_go:
        subb a,#1
        jnc p_grt_tan0
        jc p_less_tan0

p_less_tan0:
    setb buzzer
    setb valve_3
    call delay

    ret
p_grt_tan0:
        mov a,keep_data
          cjne a,pre_ref,p_go2
          setb buzzer
          setb valve_3
          call delay
          call delay
```

```
                ret
p_go2:          subb a,#100
            jnc p_grt_tan1
            jc p_less_tan1


p_grt_tan1 :    call delay
            setb buzzer
            setb valve_3
                call delay
                call delay
            ret
p_less_tan1:    clr buzzer
            clr valve_3
                call delay
                call delay
            ret
 ;;;;;;;;;;;;;;;;;;;;;;;;;;
 temp_check:
            clr c
            mov a, keep_data
            cjne a,#1,t_go
            setb buzzer
```

```
        setb valve_4
        call delay
        call delay
        ret
t_go:
        subb a,#1
        jnc t_grt_tan0
        jc t_less_tan0

t_less_tan0:
    setb buzzer
    setb valve_4
    call delay

    ret
t_grt_tan0:
        mov a,keep_data
          cjne a,temp_ref,t_go2
          setb buzzer
          setb valve_4
          call delay
          call delay
```

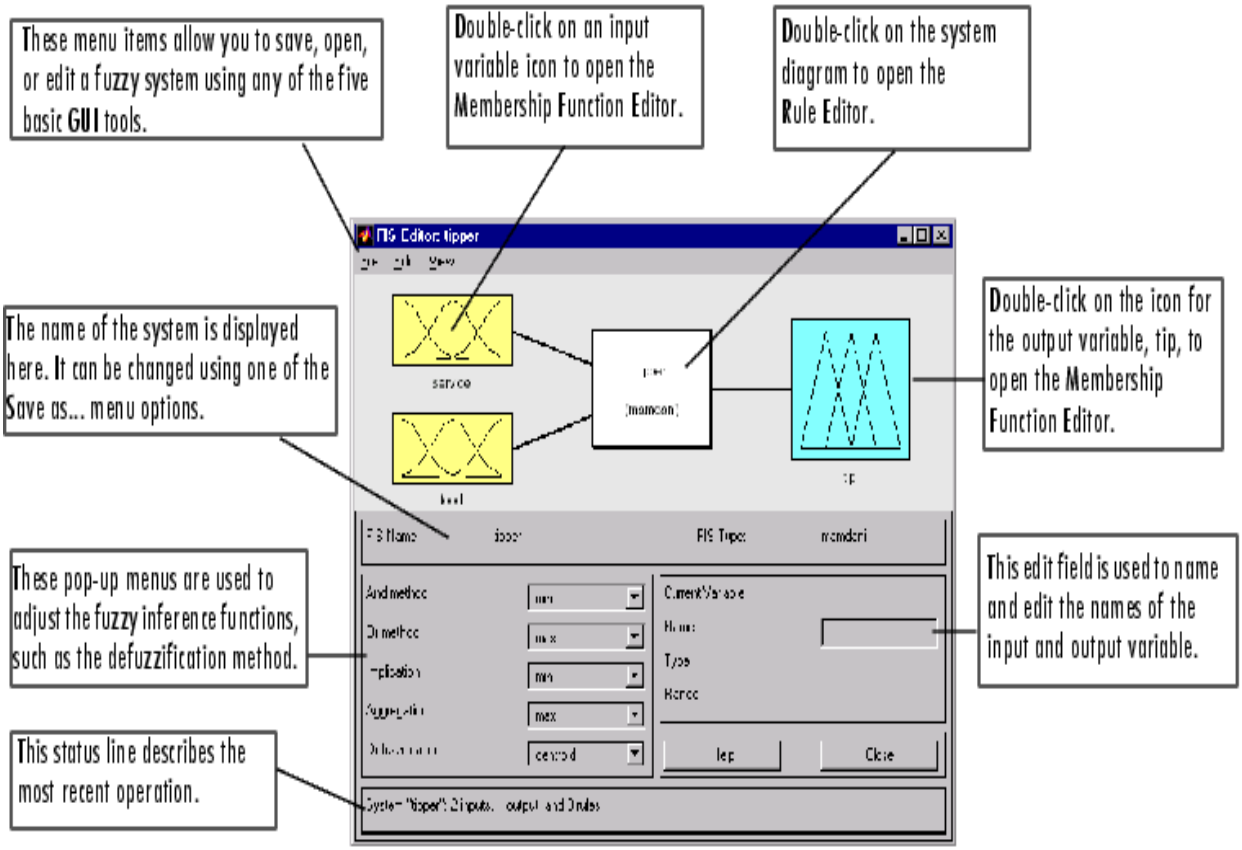
```
                ret
t_go2:          subb a,#100
              jnc t_grt_tan1
              jc t_less_tan1


t_grt_tan1 :    call delay
              setb buzzer
              setb valve_4
                call delay
                call delay
              ret
t_less_tan1:    clr buzzer
              clr valve_4
                call delay
                call delay
              ret
      end
```

APPENDIX C

Fuzzy Logic in MATLAB Environment

The FIS (Fuzzy Inference System) Editor



The following discussion walks you through building a new fuzzy inference system from scratch. If you want to save time and follow along quickly, you can load the already built system by typing

```
fuzzy tipper
```

This will load the FIS associated with the file tipper.fis (the .fis is implied) and launch the FIS Editor. However, if you load the pre-built system, you will not be building rules and constructing membership functions.

The FIS Editor displays general information about a fuzzy inference system. There's a simple diagram at the top that shows the names of each input variable on the left, and those of each output variable on the right. The sample membership functions shown in the boxes are just icons and do not depict the actual shapes of the membership functions.

Below the diagram is the name of the system and the type of inference used. The default, Mamdani-type inference, is what we've been describing so far and what we'll continue to use for this example. Another slightly different type of inference, called Sugeno-type inference, is also available. This method is explained in Sugeno-Type Fuzzy Inference. Below the name of the fuzzy inference system, on the left side of the figure, are the pop-up menus that allow you to modify the various pieces of the inference process. On the right side at the bottom of the figure is the area that displays the name of either an input or output variable, its associated membership function type, and its range. The latter two fields are specified only after the membership functions have been. Below that region are the **Help** and **Close** buttons that call up online help and close the window, respectively. At the bottom is a status line that relays information about the system.

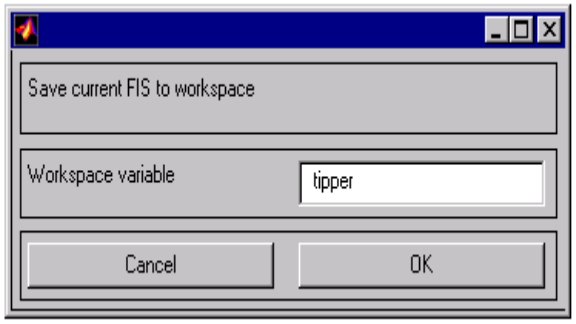
To start this system from scratch, type

```
fuzzy
```

at the MATLAB prompt. The generic untitled FIS Editor opens, with one input, labeled **input1**, and one output, labeled **output1**. For this example, we will construct a two-input, one output system, so go to the **Edit** menu and select **Add input**. A second yellow box labeled **input2** will appear. The two inputs we will have in our example are **service** and **food**. Our

one output is **tip**. We'd like to change the variable names to reflect that, though:

1. Click once on the left-hand (yellow) box marked **input1** (the box will be highlighted in red).
2. In the white edit field on the right, change `input1` to `service` and press **Return.**
3. Click once on the left-hand (yellow) box marked **input2** (the box will be highlighted in red).
4. In the white edit field on the right, change `input2` to `food` and press **Return**.
5. Click once on the right-hand (blue) box marked **output1**.
6. In the white edit field on the right, change `output1` to `tip`.
7. From the **File** menu select **Save to workspace as...**

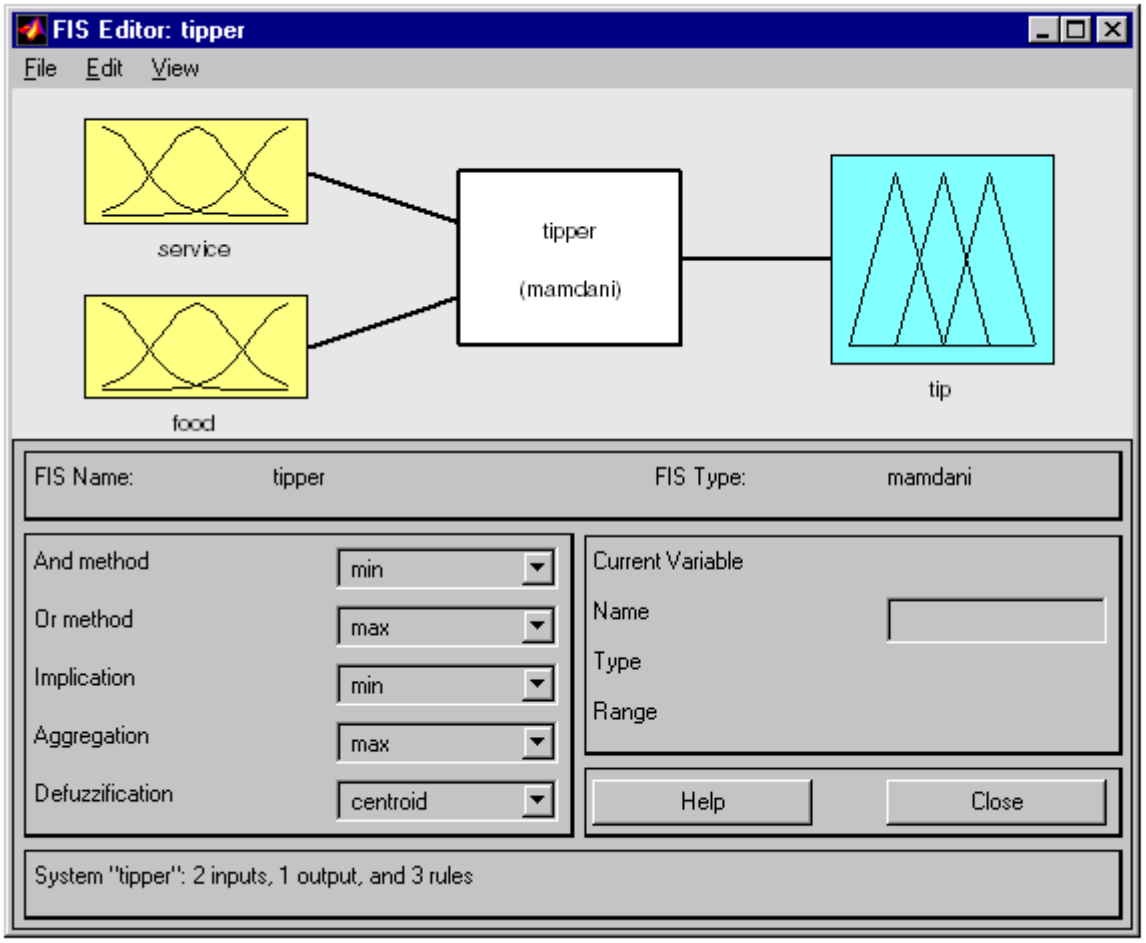


.

8. Enter the variable name `tipper` and click on **OK.**

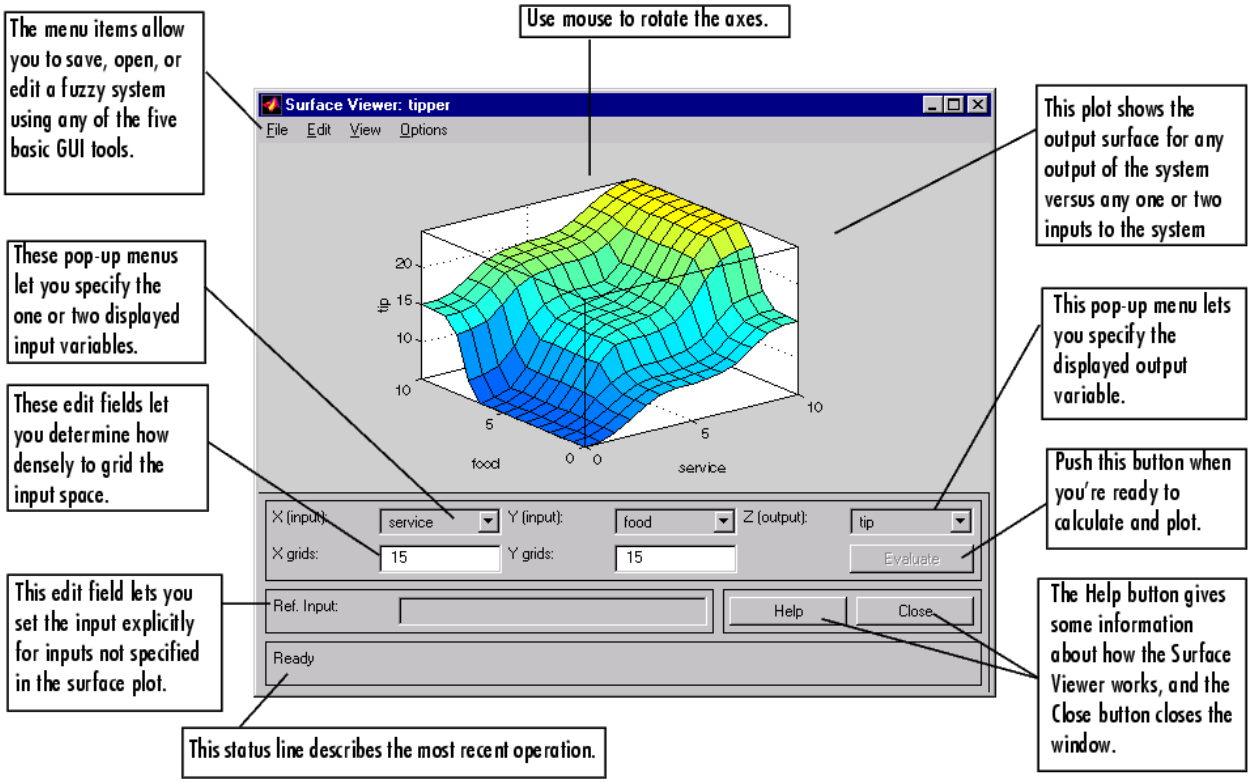
You will see the diagram updated to reflect the new names of the input and output variables. There is now a new variable in the workspace called tipper that contains all the information about this system. By saving to the workspace with a new name, you also rename the entire system. Your

window will look something like this.



Leave the inference options in the lower left in their default positions for now. You've entered all the information you need for this particular GUI. Next define the membership functions associated with each of the variables. To do this, open the Membership Function Editor. You can open the Membership Function Editor in one of three ways:

- Pull down the **View** menu item and select **Edit Membership Functions...**.
- Double-click on the icon for the output variable, **tip**.
- Type mfedit at the command line.

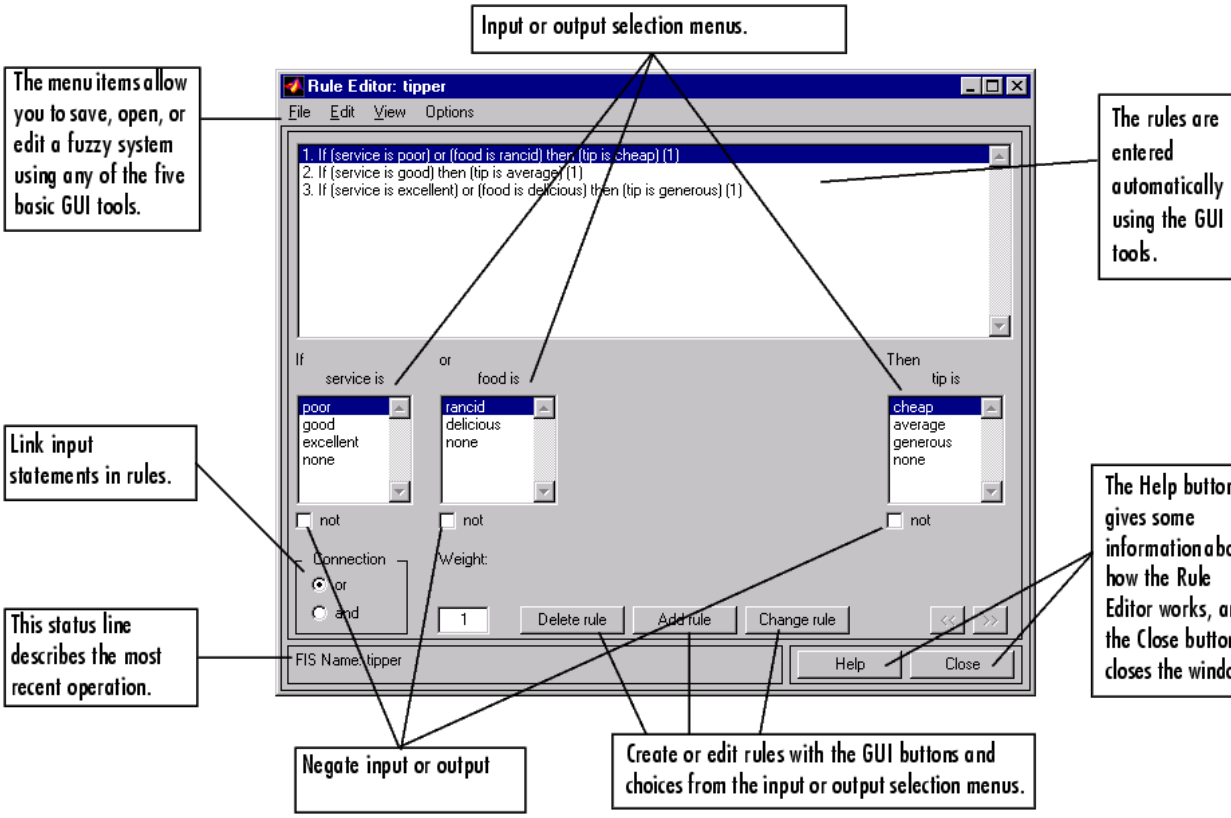


Upon opening the Surface Viewer, we are presented with a two-dimensional curve that represents the mapping from service quality to tip amount. Since this is a one-input one-output case, we can see the entire mapping in one plot. Two-input one-output systems also work well, as they generate three-dimensional plots that MATLAB can adeptly manage. When we move beyond three dimensions overall, we start to encounter trouble displaying the results. Accordingly, the Surface Viewer is equipped with pop-up menus that let you select any two inputs and any one output for plotting. Just below the pop-up menus are two text input fields that let you determine how many *x-axis* and *y-axis* grid lines you

want to include. This allows you to keep the calculation time reasonable for complex problems. Pushing the **Evaluate** button initiates the calculation, and the plot comes up soon after the calculation is complete. To change the *x-axis* or *y-axis* grid after the surface is in view, simply change the appropriate text field, and click on either **X-grids** or **Y-grids**, according to which text field you changed, to redraw the plot.

The Surface Viewer has a special capability that is very helpful in cases with two (or more) inputs and one output: you can actually grab the axes and reposition them to get a different three-dimensional view on the data. The **Ref. Input** field is used in situations when there are more inputs required by the system than the surface is mapping. Suppose you have a four-input one-output system and would like to see the output surface. The Surface Viewer can generate a three-dimensional output surface where any two of the inputs vary, but two of the inputs must be held constant since computer monitors cannot display a five-dimensional shape. In such a case the input would be a four-dimensional vector with `NaN`s holding the place of the varying inputs while numerical values would indicate those values that remain fixed. An `NaN` is the IEEE symbol for "not a number."

This concludes the quick walk-through of each of the main GUI tools. Notice that for the tipping problem, the output of the fuzzy system matches our original idea of the shape of the fuzzy mapping from service to tip fairly well. In hindsight, you might say, "Why bother? I could have just drawn a quick lookup table and been done an hour ago!" However, if you are interested in solving an entire class of similar decision-making problems, fuzzy logic may provide an appropriate tool for the solution, given its ease with which a system can be quickly modified.



Constructing rules using the graphical Rule Editor interface is fairly self-evident. Based on the descriptions of the input and output variables defined with the FIS Editor, the Rule Editor allows you to construct the rule statements automatically, by clicking on and selecting one item in each input variable box, one item in each output box, and one connection item. Choosing **none** as one of the variable qualities will exclude that variable from a given rule. Choosing **not** under any variable name will negate the associated quality. Rules may be changed, deleted, or added, by clicking on the appropriate button.

The Rule Editor also has some familiar landmarks, similar to those in the FIS Editor and the Membership Function Editor, including the menu bar

and the status line. The **Format** pop-up menu is available from the **Options** pull-down menu from the top menu bar -- this is used to set the format for the display. Similarly, **Language** can be set from under **Options** as well. The **Help** button will bring up a MATLAB Help window.

To insert the first rule in the Rule Editor, select the following:

- **poor** under the variable **service**
- **rancid** under the variable **food**
- The radio button, **or**, in the **Connection** block
- **cheap**, under the output variable, **tip**.

The resulting rule is

```
1. If (service is poor) or (food is rancid)
then (tip is cheap) (1)
```

The numbers in the parentheses represent weights that can be applied to each rule if desired. You can specify the weights by typing in a desired number between zero and one under the **Weight** setting. If you do not specify them, the weights are assumed to be unity `(1)`.

Follow a similar procedure to insert the second and third rules in the Rule Editor to get

1. If (service is poor) or (food is rancid) then (tip is cheap) (1)

2. If (service is good) then (tip is average) (1)

3. If (service is excellent) or (food is delicious) then (tip is generous) (1)

To change a rule, first click on the rule to be changed. Next make the desired changes to that rule, and then click on **Change rule**. For example, to change the first rule to

```
1. If (service not poor) or (food not rancid)
then (tip is not cheap) (1)
```

click **not** under each variable, and then click **Change rule**.

The **Format** pop-up menu from the **Options** menu indicates that you're looking at the verbose form of the rules. Try changing it to **symbolic**. You will see

1. (service==poor) => (tip=cheap) (1)

2. (service==good) => (tip=average) (1)

3. (service==excellent) => (tip=generous) (1)

There is not much difference in the display really, but it's slightly more language neutral, since it doesn't depend on terms like "if" and "then." If you change the format to indexed, you'll see an extremely compressed version of the rules that has squeezed all the language out.

1, 1 (1) : 1

2, 2 (1) : 1

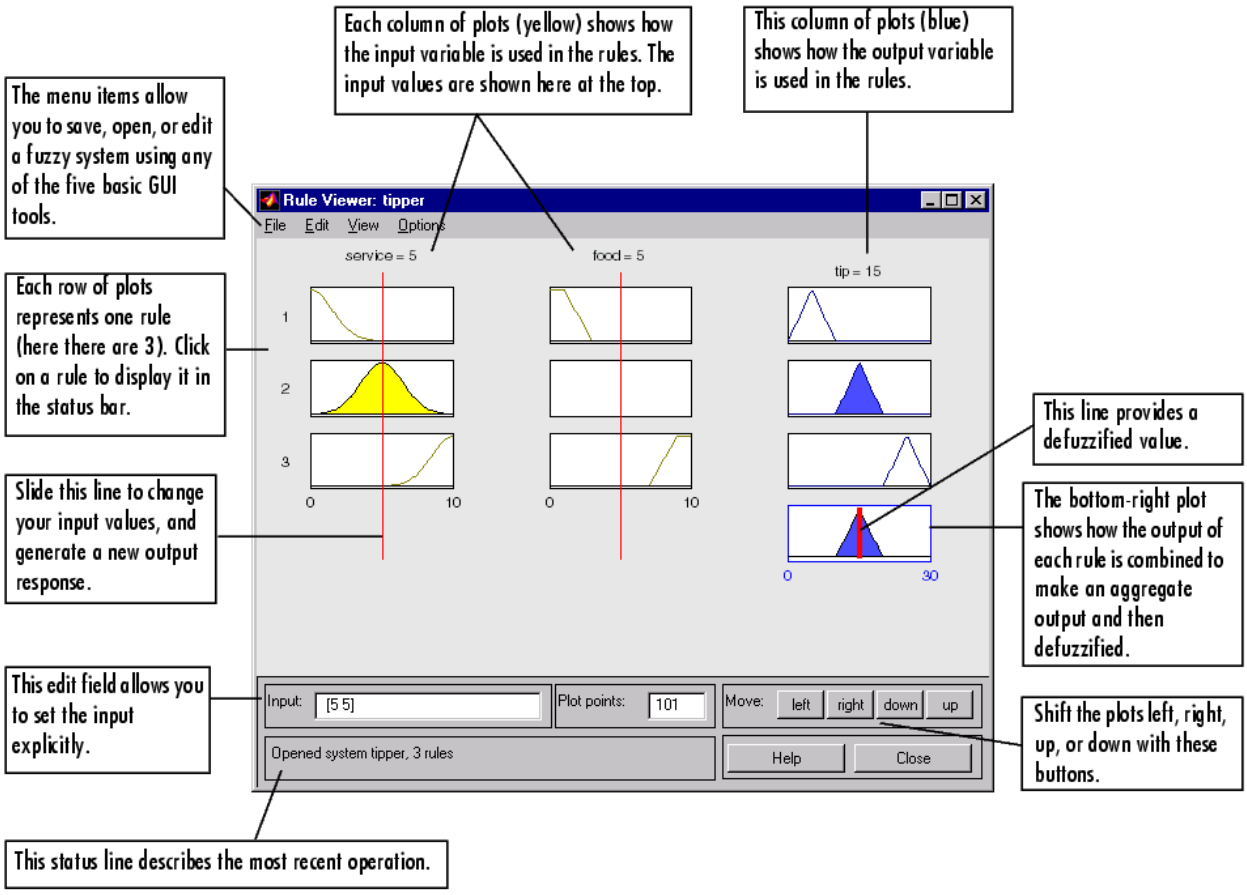
3, 3 (1) : 1

This is the version that the machine deals with. The first column in this structure corresponds to the input variable, the second column corresponds to the output variable, the third column displays the weight applied to each rule, and the fourth column is shorthand that indicates

whether this is an OR (2) rule or an AND (1) rule. The numbers in the first two columns refer to the index number of the membership function. A literal interpretation of rule 1 is: "if input 1 is MF1 (the first membership function associated with input 1) then output 1 should be MF1 (the first membership function associated with output 1) with the weight 1." Since there is only one input for this system, the AND connective implied by the 1 in the last column is of no consequence.

The symbolic format doesn't bother with the terms, *if*, *then*, and so on. The indexed format doesn't even bother with the names of your variables. Obviously the functionality of your system doesn't depend on how well you have named your variables and membership functions. The whole point of naming variables descriptively is, as always, making the system easier for you to interpret. Thus, unless you have some special purpose in mind, it will probably be easier for you to stick with the **verbose** format.

At this point, the fuzzy inference system has been completely defined, in that the variables, membership functions, and the rules necessary to calculate tips are in place. It would be nice, at this point, to look at a fuzzy inference diagram like the one presented at the end of the previous section and verify that everything is behaving the way we think it should. This is exactly the purpose of the Rule Viewer, the next of the GUI tools we'll look at. From the **View** menu, select **View rules...**.



The Rule Viewer displays a roadmap of the whole fuzzy inference process. It's based on the fuzzy inference diagram described in the previous section. You see a single figure window with 10 small plots nested in it. The three small plots across the top of the figure represent the antecedent and consequent of the first rule. Each rule is a row of plots, and each column is a variable. The first two columns of plots (the six yellow plots) show the membership functions referenced by the antecedent, or the if-part of each rule. The third column of plots (the three blue plots) shows the membership functions referenced by the consequent, or the then-part of each rule. If you click once on a rule number, the corresponding rule will be displayed at the bottom of the figure. Notice that under **food**, there is a plot which is blank. This

corresponds to the characterization of **none** for the variable **food** in the second rule. The fourth plot in the third column of plots represents the aggregate weighted decision for the given inference system. This decision will depend on the input values for the system.

There are also the now familiar items like the status line and the menu bar. In the lower right there is a text field into which you can enter specific input values. For the two-input system, you will enter an input vector, `[9 8]`, for example, and then click on **input**. You can also adjust these input values by clicking anywhere on any of the three plots for each input. This will move the red index line horizontally, to the point where you have clicked. You can also just click and drag this line in order to change the input values. When you release the line, (or after manually specifying the input), a new calculation is performed, and you can see the whole fuzzy inference process take place. Where the index line representing service crosses the membership function line "service is poor" in the upper left plot will determine the degree to which rule one is activated. A yellow patch of color under the actual membership function curve is used to make the fuzzy membership value visually apparent. Each of the characterizations of each of the variables is specified with respect to the input index line in this manner. If we follow rule 1 across the top of the diagram, we can see the consequent "tip is cheap" has been truncated to exactly the same degree as the (composite*)* antecedent--this is the implication process in action. The aggregation occurs down the third column, and the resultant aggregate plot is shown in the single plot to be found in the lower right corner of the plot field. The defuzzified output value is shown by the thick line passing through the aggregate fuzzy set.

The Rule Viewer allows you to interpret the entire fuzzy inference process at once. The Rule Viewer also shows how the shape of certain membership functions influences the overall result. Since it plots every part of every rule, it can become unwieldy for particularly large systems, but, for a relatively small number of inputs and outputs, it performs well (depending on how much screen space you devote to it) with up to 30 rules and as many as 6 or 7 variables.

The Rule Viewer shows one calculation at a time and in great detail. In this sense, it presents a sort of micro view of the fuzzy inference system. If you want to see the entire output surface of your system, that is, the entire span of the output set based on the entire span of the input set, you need to open up the Surface Viewer. This is the last of our five basic GUI tools in the Fuzzy Logic Toolbox, and you open it by selecting **View surface...** from the **View** menu.

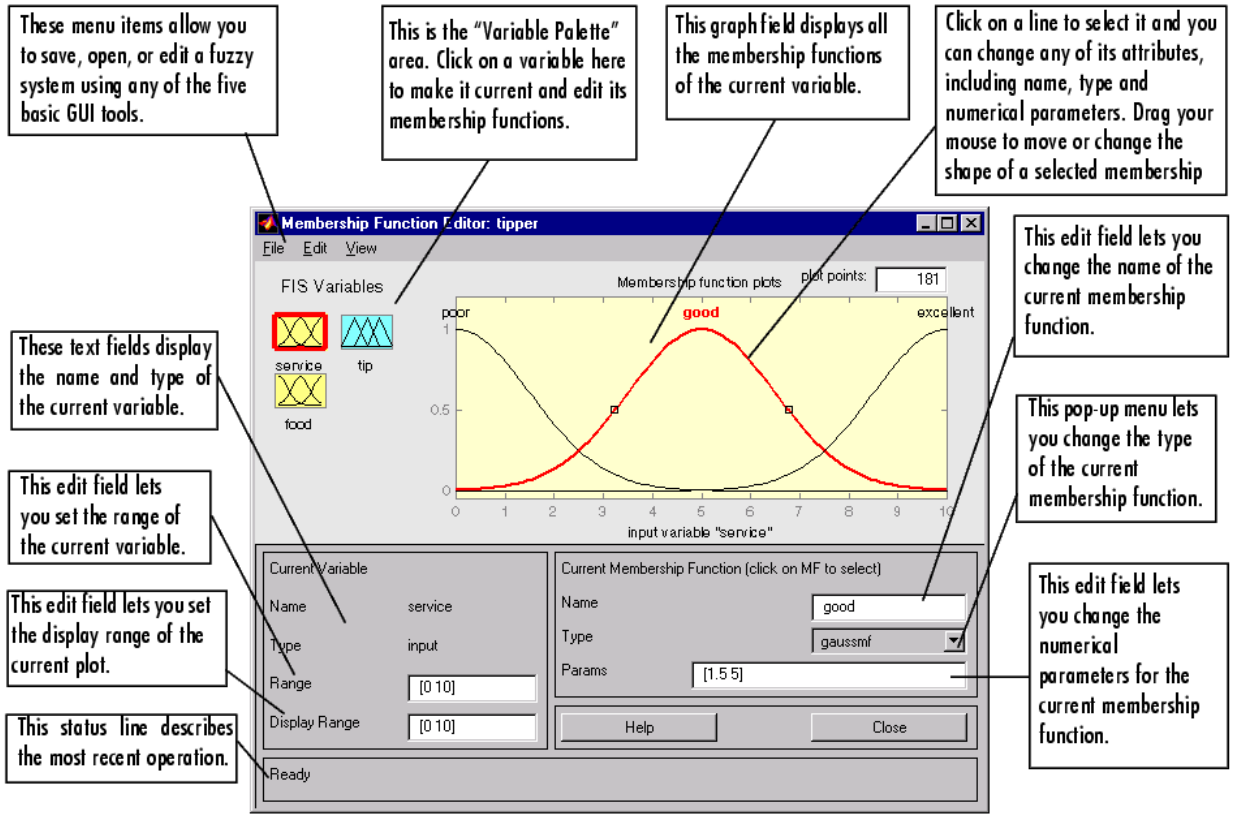
The Membership Function Editor shares some features with the FIS Editor. In fact, all of the five basic GUI tools have similar menu options, status lines, and **Help** and **Close** buttons. The Membership Function Editor is the tool that lets you display and edit all of the membership functions associated with all of the input and output variables for the entire fuzzy inference system.

When you open the Membership Function Editor to work on a fuzzy inference system that does not already exist in the workspace, there are not yet any membership functions associated with the variables that you have just defined with the FIS Editor.

On the upper left side of the graph area in the Membership Function Editor is a "Variable Palette" that lets you set the membership functions for a given variable.To set up your membership functions associated with

an input or an output variable for the FIS, select an FIS variable in this region by clicking on it.

Next select the **Edit** pull-down menu, and choose **Add MFs...**. A new window will appear, which allows you to select both the membership function type and the number of membership functions associated with the selected variable. In the lower right corner of the window are the controls that let you change the name, type, and parameters (shape), of the membership function, once it has been selected.

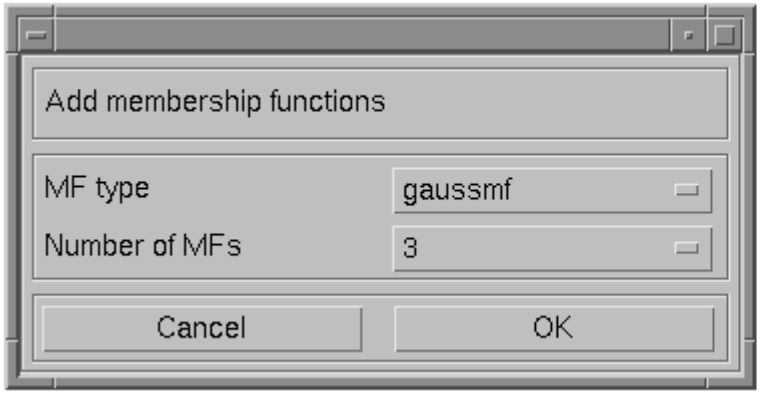


The membership functions from the current variable are displayed in the main graph. These membership functions can be manipulated in two ways. You can first use the mouse to select a particular membership function associated with a given variable quality, (such as poor, for the variable, service), and then drag the membership function from side to side. This will affect the mathematical description of the quality associated with that membership function for a given variable. The selected membership function can also be tagged for dilation or contraction by clicking on the small square drag points on the membership function, and then dragging the function with the mouse toward the *outside,* for dilation, or toward the *inside*, for contraction. This will change the parameters associated with that membership function.

Below the Variable Palette is some information about the type and name of the current variable. There is a text field in this region that lets you change the limits of the current variable's range (universe of discourse) and another that lets you set the limits of the current plot (which has no real effect on the system).

The process of specifying the input membership functions for this two input tipper problem is as follows:

1. Select the input variable, **service**, by double-clicking on it. Set both the **Range** and the **Display Range** to the vector [0 10].
2. Select **Add MFs...** from the **Edit** menu. The window below pops open



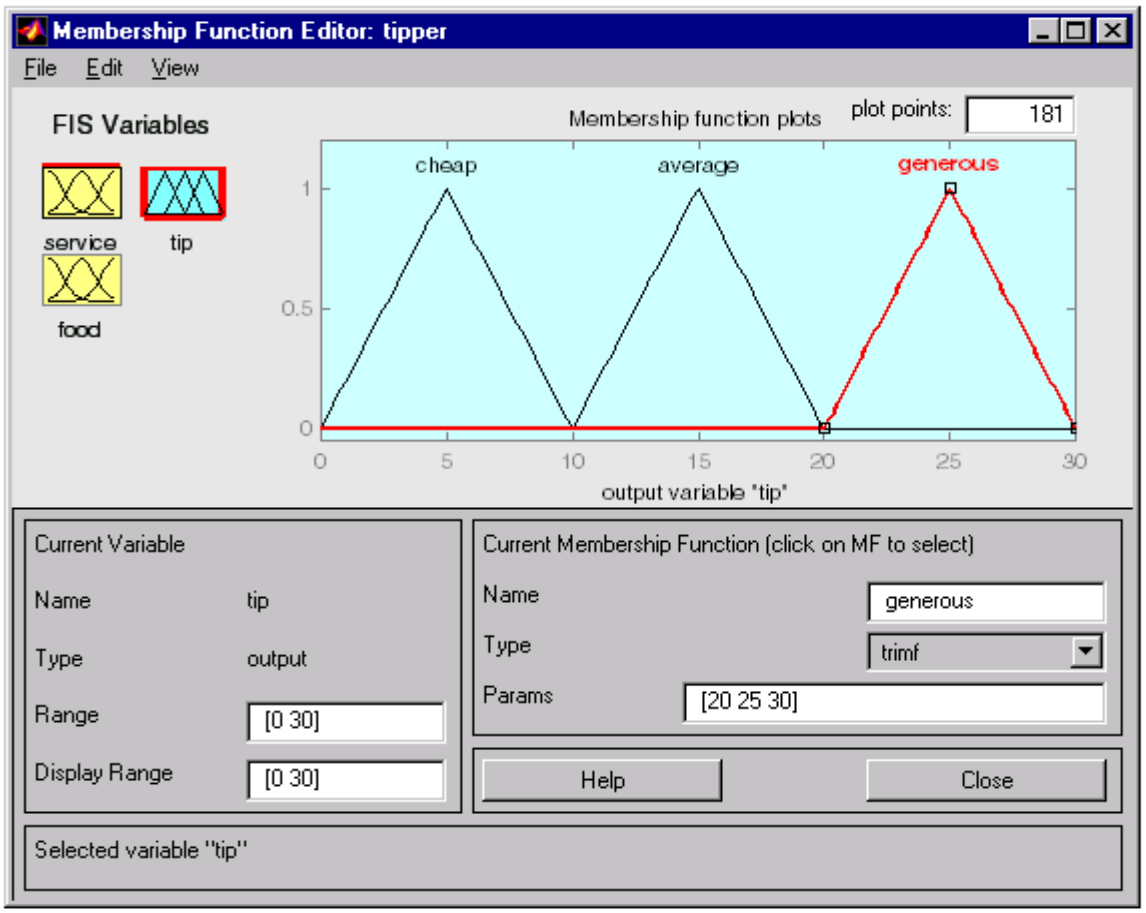
.

3. Use the pull-down tab to choose **gaussmf** for **MF Type** and **3** for **Number of MFs**. This adds three Gaussian curves to the input variable service.
4. Click once on the curve with the leftmost *hump*. Change the name of the curve to `poor`. To adjust the shape of the membership function, either use the mouse, as described above, or type in a desired parameter change, and then click on the membership function. The default parameter listing for this curve is `[1.5 0]`.
5. Name the curve with the middle hump, `good`, and the curve with the rightmost hump, `excellent`. Reset the associated parameters if desired.
6. Select the input variable, **food**, by clicking on it. Set both the **Range** and the **Display Range** to the vector `[0 10]`.
7. Select **Add MFs...** from the **Edit** menu and add two `trapmf` curves to the input variable `food`.
8. Click once directly on the curve with the leftmost trapezoid. Change the name of the curve to `rancid`. To adjust the shape of the membership function, either use the mouse, as described above, or type in a desired parameter change, and then click on the membership function. The default parameter listing for this curve is `[0 0 1 3]`.

9. Name the curve with the rightmost trapezoid, `delicious`, and reset the associated parameters if desired.

Next you need to create the membership functions for the output variable, **tip**. To create the output variable membership functions, use the Variable Palette on the left, selecting the output variable, **tip**. The inputs ranged from 0 to 10, but the output scale is going to be a tip between 5 and 25 percent.

Use triangular membership function types for the output. First, set the **Range** (and the **Display Range**) to `[0 30]`, to cover the output range. Initially, the *cheap* membership function will have the parameters `[0 5 10]`, the *average* membership function will be `[10 15 20]`, and the *generous* membership function will be `[20 25 30]`. Your system should look something like this.



Now that the variables have been named, and the membership functions have appropriate shapes and names, you're ready to write down the rules. To call up the Rule Editor, go to the **View** menu and select **Edit rules...**, or type `ruleedit` at the command line.