FEED-FORWARD FUZZY INTERVENTION IN PID CONTROLLER DESIGN

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

MASTERS OF ENGINEERING

IN

CONTROL AND INSTRUMENTATION

BY

DEEPAK AGARWAL

UNDER THE GUIDANCE OF

MR RAM BHAGAT

DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI COLLEGE OF ENGINEERING

DELHI-110042

JUNE-2006

ABSTRACT

The proposed controller design aims to overcome the drawbacks of classical PID controller, such as: integrator windup due to the saturation in the actuator and the slow response to reject disturbances. The proposed controller consists of basic PID blocks: Proportional, Derivative, and Integral actions plus a fuzzy anticipation block to modify the control signal according to operating conditions. The fuzzy anticipation block provides feed-forward correction terms to speed-up the system response and to retain the desired output. This block receives all different signals in PID block and generates anti-windup action to improve the system behavior. This action also compensates rapidly the disturbance effect on the system response. The proposed control design methodology is tested on a numerical example via simulation. The simulation work is carried out using MATLAB SIMULINK environment.

CERTIFICATE

This is to certify that the dissertation entitled “Feed-forward Fuzzy Intervention in PID Controller” has been submitted by Deepak Agarwal as partial fulfillment for the degree of Master of Engineering in Control & Instrumentation in the Department of Electrical Engineering, Delhi College of Engineering, University of Delhi. This is a record of his own work carried out by him under my supervision & guidance. The matter embodied in this major thesis report hasn’t been submitted for the award of any other degree or diploma.

RAM BHAGAT
Lecturer
Department of Electrical Engineering
Delhi College of Engineering
Delhi- 110042.

ACKNOWLEDGEMENT

I feel honored in expressing my profound sense of gratitude and indebtedness to Mr. Ram Bhagat, Lecturer, Department of Electrical Engineering, Delhi college of Engineering, Delhi for giving me the opportunity to work on such a practical problem, under his expert guidance. He constantly guided and helped us throughout the project. Words cannot express the support and motivation provided by him.

I also like to extend my sincerest gratitude to Prof. Parmod Kumar, HOD Dept. of Electrical Engineering, Delhi college of Engineering, Delhi.

PLACE: DELHI

(DEEPAK AGARWAL)
O1/C&I/04
UNIVERSITY ROLL NO-8661

TABLE OF CONTENTS

LIST OF SYMBOLS

SYMBOLS		QUANTITY
b	:	Dead Zone Gain
E	:	Error
f	:	Feedback Signal
G	:	Transfer Function
H	:	Dead Zone Range
i	:	Integrator Output
K	:	Proportional Gain
N	:	Filter Factor
T	:	Temperature
Td	:	Derivative Time
Ti	:	Integral Time
Tr	:	Reference Temperature
Tt	:	Tracking Time Constant
U	:	Input to actuator
Ua	:	Output of Fuzzy Anticipation Block
Ui	:	Integrator Output
Upd	:	Output from Proportional + Derivative
Upid	:	Output of Proportional + Integral + Derivative

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Control of a dynamic system requires manipulable inputs. The manipulation is usually transmitted (or transferred) to the system via constrained actuators. In many technical systems actuators are transducers which transforms a low power signal, usually electric, into high power action. Examples are valves for flow control and power electronics for electric power control. In most cases, properly dimensioned actuators will saturate even under normal operation.

What happens if, or when, actuators saturate depends critically on the ability of control strategy (the controller) to handle a saturation event as well as on the properties of controlled system. Some systems are easier to control via constrained actuators than others. Some controllers are better suited to handle saturation events than others. The following example illustrates this.

Consider [1] three equal linear systems given by

$$G=1/(s+1)$$

controlled by the three different controllers shown in Figure1.1. The first system is controlled by a pure feed-forward controller, the second by a PI-controller and the third by a P-controller. By ignoring the saturation in the loops, the transfer functions from the reference to the output are, however, the same in the three cases namely

$$Gr=5/(s+5)$$

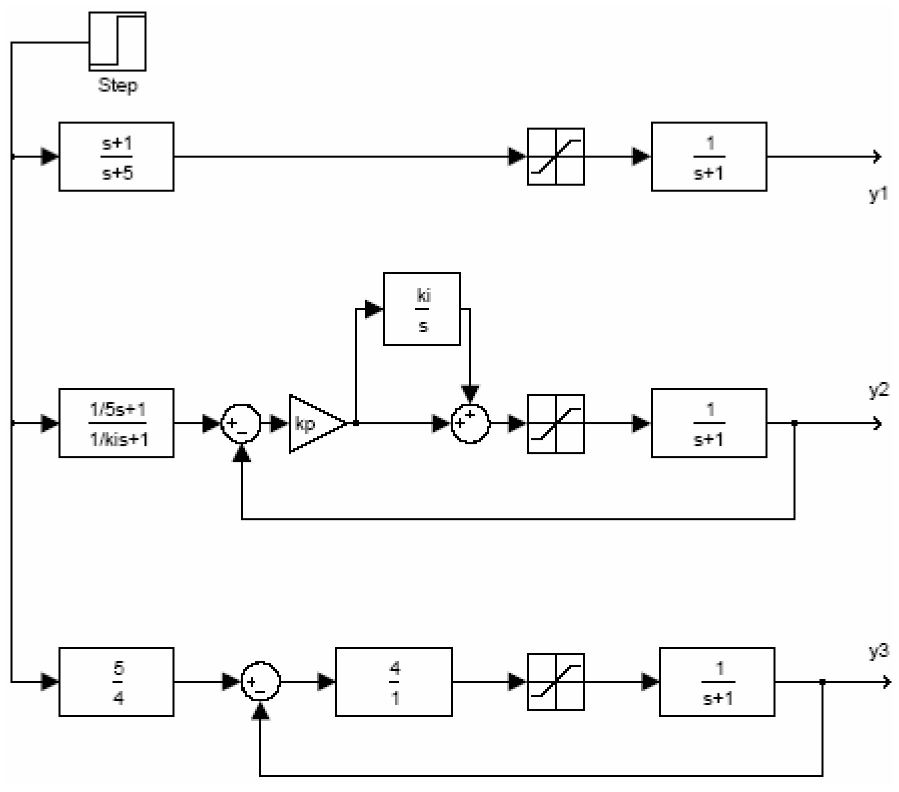


Figure1.1: Three different systems having equal linear response but different saturation effects.

The reference-step responses of the three systems are identical as shown in Figure1.2.

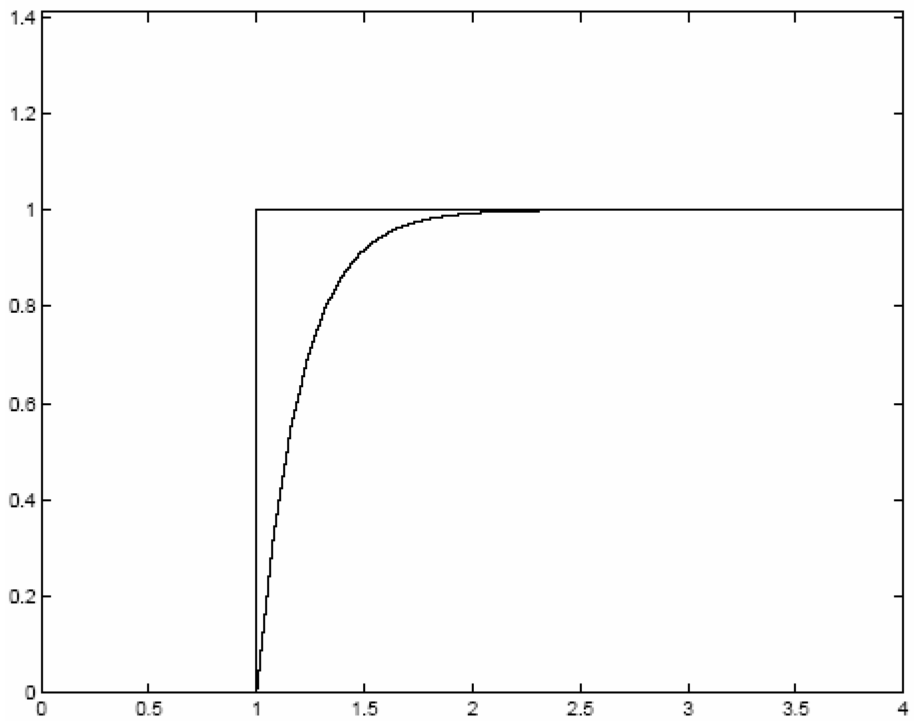


Figure1.2: Equal linear response.

But in the real case, when the inputs saturate, they behave quite differently from each other. From Figure1.3 one could say that the first system (y_1) needs more feedback the second (y_2) needs less and the third (y_3) behaves well (at least what saturation concerns). The second system suffers from integrator windup, a phenomenon which has been discussed in the literature for many decades. These undesired phenomena and what cause them, and how to overcome them, are discussed in this thesis.

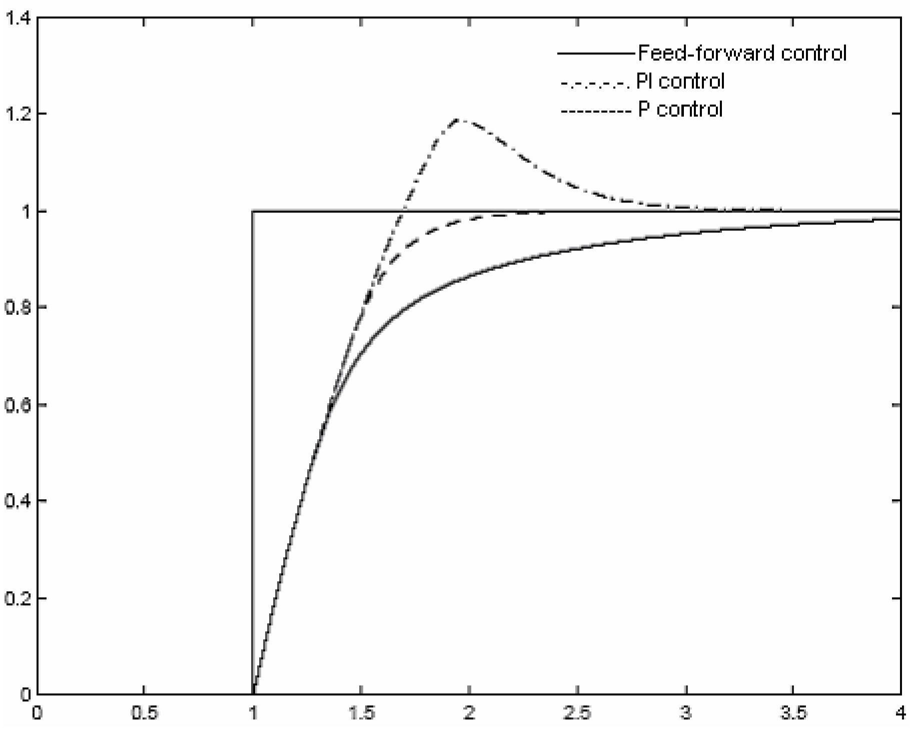


Figure 1.3: Different saturation effects.

1.2 GOAL OF THE PROJECT

The first goal of this project is to investigate the classical approaches to treat reset wind-up problem and to find out what its effect on settling time, overshoot and speed of response.

The main purpose of this project is to solve the problem encountered in classical methods to treat wind-up problems by introducing feed forward correction through the use of fuzzy logic. Beside that eliminating the effect of disturbance in minimum time. So aim is to design a controller, which act pre facto i.e. it give correction term before actuator enter in the saturating region. Secondly to act on signals rather than on parameters to improve the PID behavior using fuzzy intervention in different situation.

1.3 ORGANIZATIN OF THESIS

This project report has the following main parts:

I. The first part includes an introduction to the subject. It explains the goals of this thesis and it describes the background used to reach those goals.

II. The second part gives description of basic terms and it also illustrate reset windup phenomenon.

III. The third part discusses different classical methods to treat reset windup problem.

IV. The fourth parts of this thesis through some light on fuzzy logic .It also explain how fuzzy logic is different from conventional control methods, how does fuzzy logic work and what its advantages is.

V. The fifth part explains fuzzy intervention in PID controller design.

VI. The sixth give required description of SIMULINK and FUZZY LOGIC tool box.

VII. The seventh part shows simulation works and results.

VIII. The final part gives conclusion, further scope and references.

CHAPTER 2

BASIC TERMS

2.1 PID CONTROLLER

PID controller is a one of the earliest industrial controllers. It has many advantages: Its cost is economic, simple easy to be tuned and robust. This controller has been proven to be remarkably effective in regulating a wide range of processes. It does not require an exact model and hence, it can be used for processes whose models are considerably difficult to be driven. However, in spit of the advantages of the PID controller, there remain several drawbacks. It can not cope well in some cases such as:

- Non-linear processes (changing in operating point).
- Time-varying parameters.
- Compensation of strong and rapid disturbances.
- Supervision in multivariable control.

PID controller is simple and linear; it can give a good performance for stable linear processes. Self-tuning and adaptive PID design approaches can overcome the operating point varying parameters. However this requires a high capacity of computations and makes the PID performance not guaranteed. PID controller consists of three terms:

- Proportional action.
- Derivative action to speed up the response.
- Integral action to eliminate the steady state error.

2.1.1 Proportional Band

With proportional band, the controller output is proportional to the error or a change in measurement.

Controller output = E(t)*100/(Proportional Band)

With a proportional controller offset (deviation from set-point) is present. Increasing the controller gain will make the loop go unstable. Integral action was included in controllers to eliminate this offset.

2.1.2 Integral

With integral action, the controller output is proportional to the amount of time the error is present. Integral action eliminates offset.

Controller output = (1/Integral Time)*(Integral of E(t))

Integral action eliminates the offset. The response is somewhat oscillatory and can be stabilized some by adding derivative action. Integral action gives the controller a large gain at low frequencies that results in eliminating offset and "beating disturbances".

2.1.3 Derivative

With derivative action, the controller output is proportional to the rate of change of the measurement or error. The controller output is calculated by the rate of change of the error with time**.**

Controller output = Derivative Time*(Derivative of E(t))

Derivative action can compensate for a changing measurement. Thus derivative takes action to inhibit more rapid changes of the measurement than proportional action. When a load or set-point change occurs, the derivative action causes the controller gain to move the "wrong" way when the measurement gets near the set-point. Derivative is often used to avoid overshoot. Derivative action can stabilize loops.

The PID controller output U in s-domain is given by the following equation:

$$U(s) = K\,(1 + 1/T_i s + T_d s / (1 + T_d s/N))\,E(s)$$

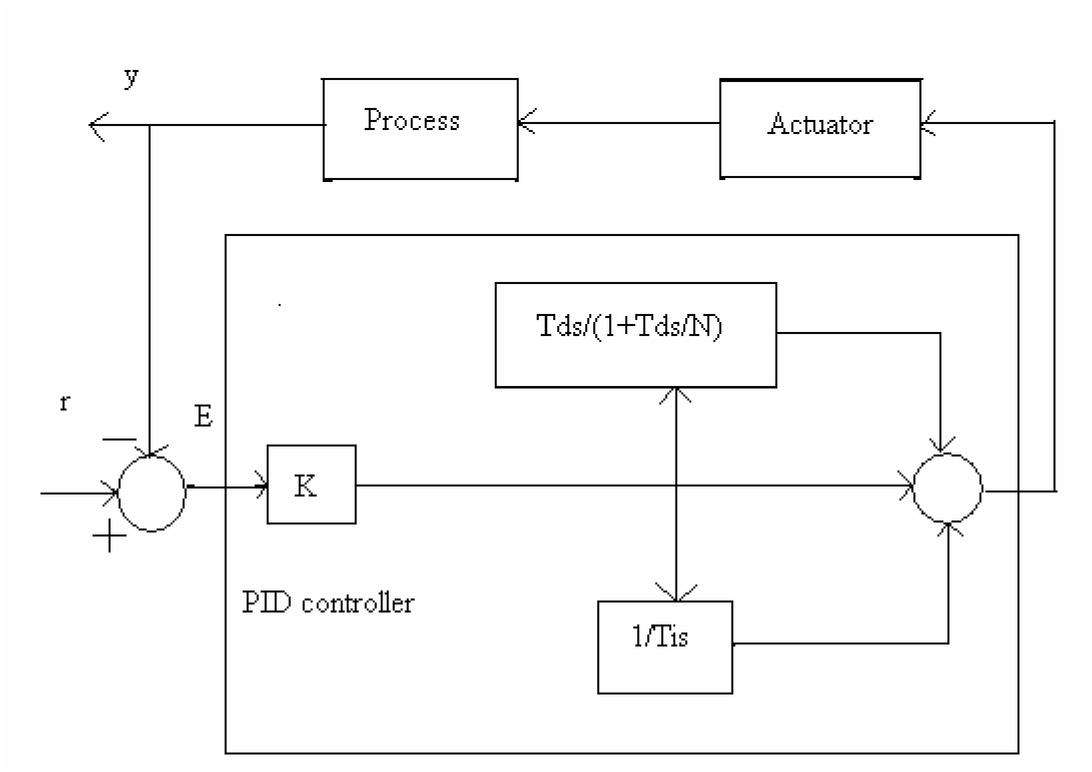


Fig2.1: General structure of PID controller

2.1.4 Tuning

The process of setting the optimal gains for P, I and D to get an ideal response from a control system is called tuning. In general there are two approaches in PID tuning:

- Model based approach, if the process model is available.
- Non-model based approach, if the process model is difficult to be driven.

In the second approach, Ziegler and Nichols method can be applied based on the step response. This approach is more practical in the industry. Others are used the relay feedback to estimate the limit cycle and then tune the PID parameters.

The gains of a PID controller can be obtained by trial and error method. In this method, I and D terms are set to zero first and the proportional gain is increased until the output of the loop oscillates. As one increases the proportional gain, the system becomes faster, but care must be taken not make the system unstable. Once P has been set to obtain a desired fast response, the integral term is increased to stop the oscillations. The integral term reduces the steady state error, but increases overshoot. Some amount of overshoot is always necessary for a fast system so that it could respond to changes immediately. The integral term is tweaked to achieve a minimal steady state error. Once the P and I have been set to get the desired fast control system with minimal steady state error, the derivative term is increased until the loop is acceptably quick to its set point. Increasing derivative term decreases overshoot and yields higher gain with stability but would cause the system to be highly sensitive to noise.

2.2 ACTUATOR

An actuator is that position of a valve that responds to applied signal and causes the motion resulting in modification of a fluid flow .Thus an actuator is any devices that causes the valve stem to move .It may be manually positioned device, such as hand wheel or lever .The manual actuator may be open-closed, or it may be manually positioned at any position between fully open and fully closed.

System actuators use hydraulic, electronic, and pneumatic signals to help activate process control equipment, including mechanical arms, robots, and other automating agents. Common actuators include electrical motors, pistons, electro-active polymers, thermal bimorphs, and more, with every actuator especially equipped to start specific processes. Important factors for actuators include the ability for precision control, operating life, and power consumption versus work ability.

From an engineering perspective, an actuator is a type of transducer, or a device that turns some input signal into physical motion. Linear actuators describe systems that result in a linear motion, while rotary actuators produce a circular result.
An actuator can be said to have two basic functions:

(1) To respond to external signal directed to it and caused and caused valve to move accordingly.

(2) To provide a convenient support for certain valve accessory item including positioner, limit switches, solenoid valve and local controllers

2.3 RESET WIND-UP

A properly tuned controller will behave well as long as its output remains in a range where it can change the manipulated flow. However it will behave poorly if, for any reason, the effect of the controller output on the manipulated flow is lost. A gap between the limit on the controller output and the operational limit *of* the control valve is the most common cause of integral (reset) wind-up. The block diagram shown in Fig2.2 is used to explain this problem.

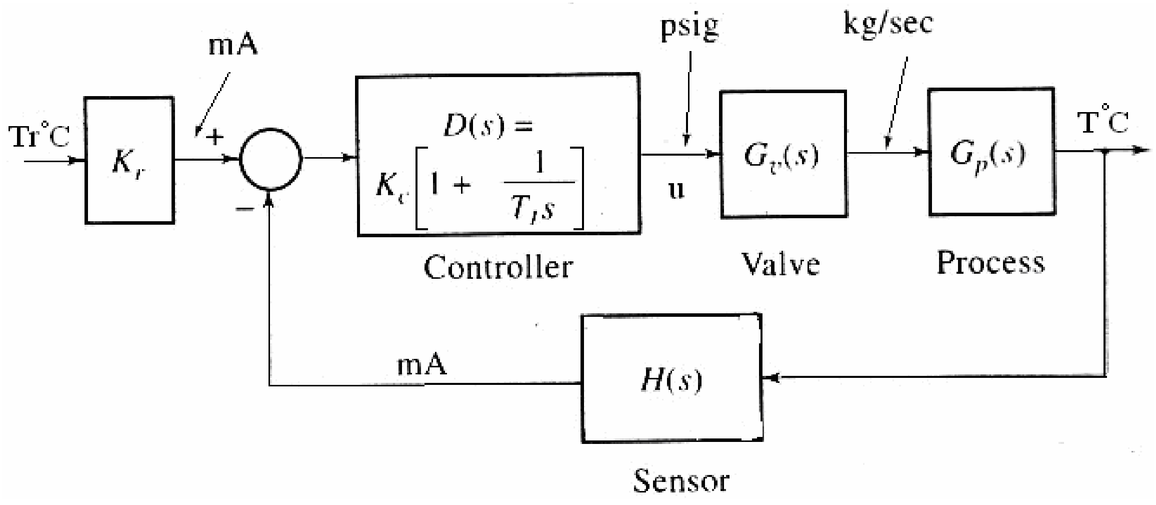


Fig2.2: A block diagram

Assume that a PI controller is used to control the temperature T by adjusting the steam control valve. The instrumentation is electro pneumatic. The controlled variable T is measured with a sensor that generates a signal (mA) that is proportional to the temperature. The measurement is sent to the controller where it is compared with the set-point (the term Kr on the set-point signal is included to indicate the conversion of the set-point scale). The controller then generates a control signal (mA) on the basis of the error between the measurement and the set-point. The controller output signal is then connected to the actuator of the steam control valve through a current-to-pressure transducer.

The nominal pressure range of the valve is 3 to 15 psig and the supply pressure is 20 psig.

Assume that a command signal for a step change Tr, in the set-point is given. Initially the temperature T is much below its set-point value; because of the large error the controller output is driven to a large value by integral action. A typical time recording of the experiment is shown in Fig2.3. At t= t1 the controller output pressure is 15 psig. The control valve (sized to be wide open at 15 psig) saturates at this pressure. At t = t1, (when the control valve is fully open), the controlled variable T has not reached its set-point. Since there is still an error, the controller will try to correct for it by further increasing (integrating the error) its output pressure, even though the valve will not open more after 15 psig. The output of the controller can, in fact,integrate up to the supply pressure 20 psig.

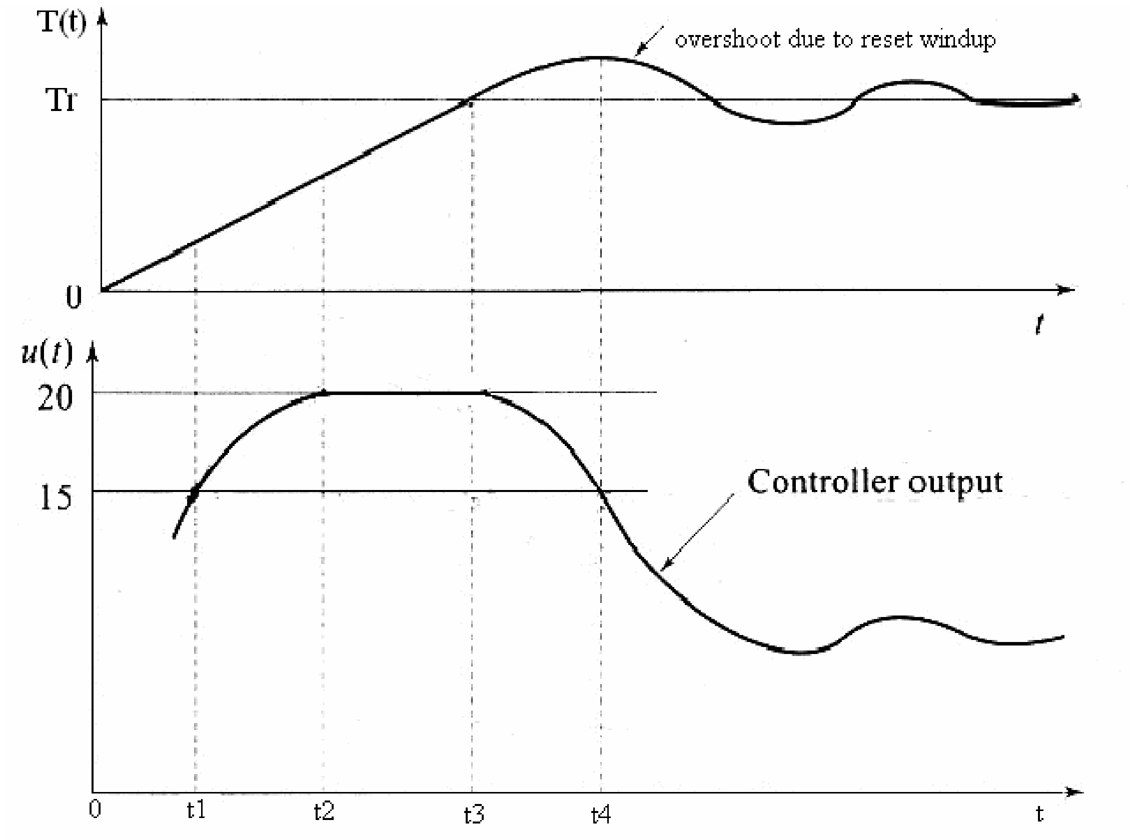


Fig2.3: Reset wind-up

The point t = t2, corresponds to this situation. The controller cannot increase its output pressure for t > t2, its output having saturated at t = t2 although the controller is saturated, it keeps the steam valve fully open. This is the correct strategy to force the temperature T to set-point value in minimum time. The point t = t_3 corresponds to this situation.

The wind-up problem begins to show up when the controlled variable T reaches its set-point. At that instant ($t = t_3$) the controller output is 20 psig. The error reverses at $t = t_3$; the valve cannot respond to this change, until the integral signal (which has 'wound-up' to 20 psig) is `unwound' back to the 15 psig level at $t = t_4$. This delayed response effect is called reset wind-up or integral wind-up. This delay is in addition to the normal lagging behavior of integral control and can thus cause excessive overshooting and stability problems.

One way to prevent the large overshoot caused by reset wind-up is to keep the controller on manual until the temperature reaches the set-point, and then switch it to automatic. In this case the steam valve is kept fully open by manually setting the controller output to 15 psig. This ensures that the control valve will start to close as soon as the controller is switched to automatic.

A second alternative is to install a limiter on the controller output to keep it from going beyond the operating range of the control valve, i.e., above 15 psig or below 3 psig. Note that this implementation, where the limiter is placed on the controller output, does not prevent the wind-up problem: the output of the integral action will still be driven beyond the controller output limits and cause wind-up. In order to prevent wind-up, the output of the integral action must some how be limited. In pneumatic and electronic controllers, this limiting is accomplished in a very ingenious way.

CHAPTER 3

CLASSICAL ANTI WINDUP STRATEGIES

3.1 Conditional Integration Method

In the method of conditional integration, integration is switched on or off depending on certain conditions, such as the size of the control signal or the control error. One of these conditions that give a good result is to use the following rule:

```
If
(Actuator output saturates) and
(Both control error and integral output have the
Same sign)
Then
Switch integral action OFF,
Else
Switch integral action ON
```

A zero steady state error always has to be guaranteed, that is, steady state must not be reached with the integrator switched off.

3.2 Limited Integration Method

This is a very simple approach to reduce the effects of integral windup. **A** feedback signal is created from the integrator output by feeding the integrator output through a dead zone with a high gain. The dead zone output is used to reduce the integrator input as shown in Fig3.1. To allow the full linear range of the actuator, the dead zone range has to be the same as the linear range of the actuator. Once the integrator value is out of the dead zone range, a feedback signal of magnitude

$$f = b(i(t) - H)$$

where H is the dead zone range, b is the dead zone gain, and *i(t)* the integrator value, is generated and acts upon the integrator input. If the dead zone gain *b* is sufficiently high (b>10), the integrator output will effectively be limited to H.

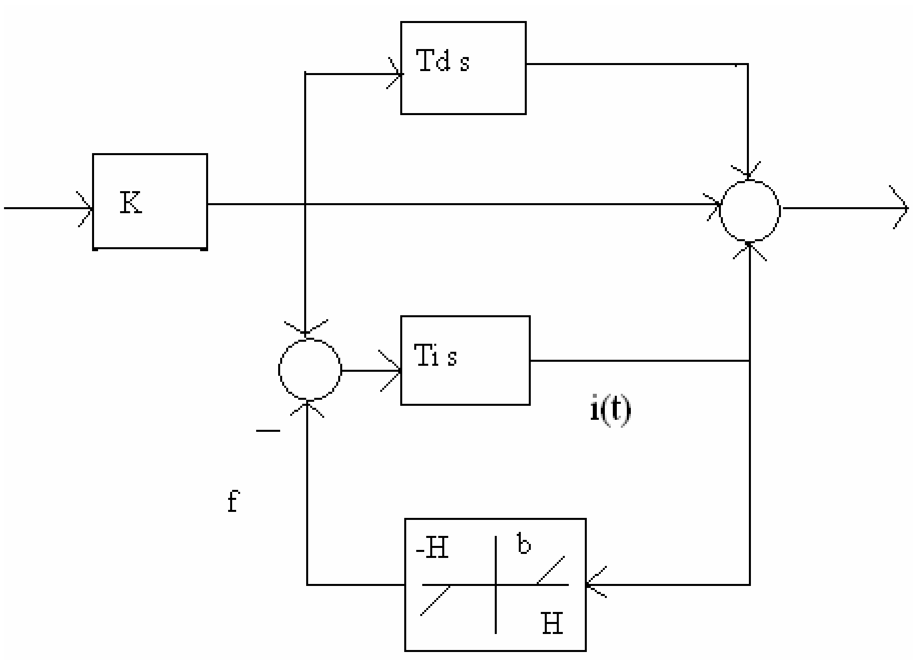


Fig3.1: PID Controller with a Limited Integrator

3.3 Tracking Anti-Windup Method

Tracking Anti- Windup is the “classical” method to prevent integral windup. The standard tracking anti-windup structure commonly described is shown below in Fig3.2. where Tt is denoted as the Tracking Time Constant. Once the controller output exceeds the actuator limits, a feedback signal is generated from the difference of the saturated and the unsaturated control signals and used to reduce the integrator input. The saturation in Fig3.2 may either be the actual saturation in the actuator, if the actuator output can be measured, or a model used in the controller. If the actuator is described by linear dynamics followed by saturation, it may be disadvantageous to limit the controller output, as this also limits the speed of the actuator response. The structure shown in Fig3.2 may therefore be replaced by the structure shown in Fig3.3, where the unrestricted control signal is applied to the process, and the feedback signal is generated using a dead zone. The dead zone range H must again represent the linear range of the actuator. And the relation between Tt and the dead zone gain *b* is given by

$$b=Ti/Tt.$$

Tt=Ti, corresponds to b=l.

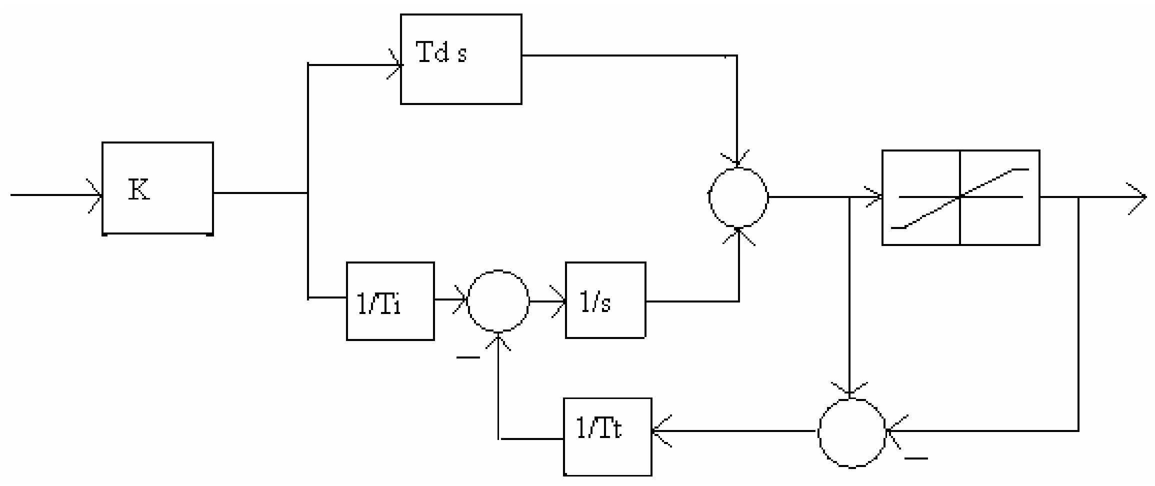


Fig3.2: Standard structure of a tracking anti-windup PID controller.

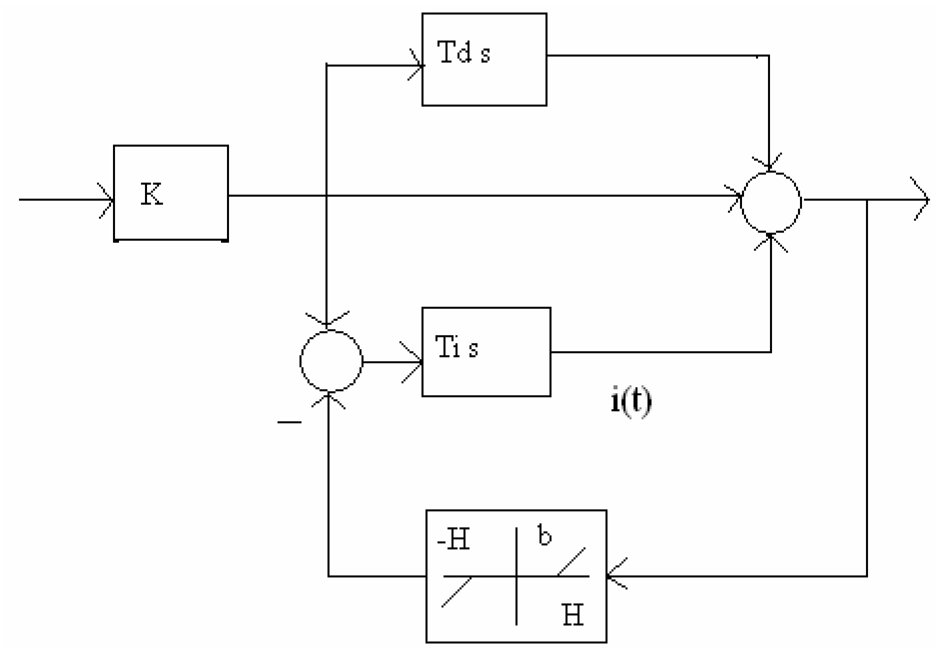


Fig3.3: Alternative structure of a tracking anti-windup PID controller.

3.4 Modified Tracking Anti-Windup Method

The method of tracking anti-windup has been found very sensitive to changes in the parameter Tt or the gain b. Too high a value for b may effectively reduce the amount of overshoot, but may also lead to a slow response of the system.

In this method an alternative structure with an additional limit on the feedback signal has been imposed. This technique is shown in Fig3.4. The slow response results from the fact that a very high initial controller output (due to a high proportional gain and the derivative action) will, in the case of a high gain b, give a large feedback signal. This feedback signal will drive the integrator to a large negative value, bringing the controller output back to the linear range. As time increases, the proportional and the derivative part of the control signal will decrease, but the integrator output will not increase fast enough to compensate this decrease. This means that the controller output will become very small, or even negative.

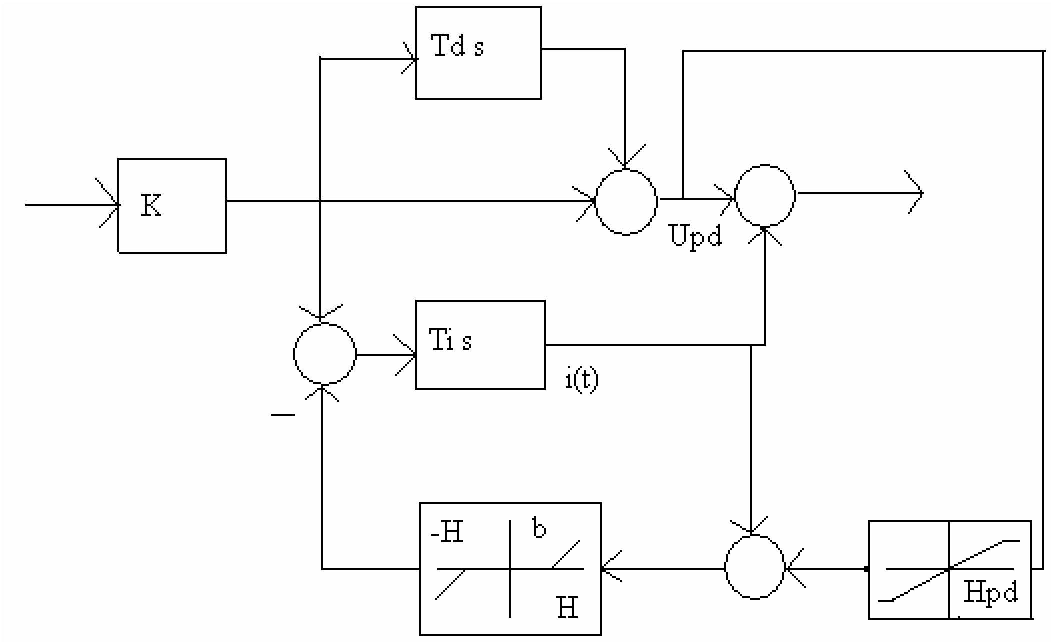


Fig3.4: Modified tracking anti-windup PID controller.

To avoid this, an additional limit on the proportional-derivative part of the control signal used to generate the anti-windup feedback signal, has been introduced. If this is done, a high gain b *(*b= 10) can be selected.

The effect of the additional saturation can be explained as follows: **A** feedback signal of magnitude

$$f = b\,(i(t)+\min\{Hpd,Upd(t)\}-H)$$

comes into effect if the integrator value i(t) exceeds H - Upd(t) or H-Hpd,, whichever is larger. This feedback signal will reduce the integrator input and thus hold the integral action. Normal integral action is performed only if

$$-H-\max\{-Hpd,Upd(t)\}<i(t)<H-\min\{Hpd,Upd(t)\}$$

Where Upd is the proportional and derivative part of the control signal. This means that the integrator value is limited dynamically to (H - Upd(t)) or (H - Hpd), whichever is larger.

Introducing the additional limits gives one more design parameter a ratio r=Hpd/H. A good choice for the parameter r is the range r =0.5…1.5. A value of r= l , that is Hpd=H, can be interpreted as holding the integral action until the control signal from the proportional and derivative action Upd returns to the linear range and then setting the integrator to run. The integrator will therefore not be driven negative and the disadvantage of the tracking method, a very slow step response for a high dead zone gain, will be avoided.

CHAPTER 4

FUZZY LOGIC

4.1Introduction

Fuzzy logic has emerged as one of the active areas of research activity particularly in control application. Fuzzy logic is a very powerful method of reasoning when mathematical models are not available and input data are imprecise. Its applications, mainly to control, are being studied throughout the world by control engineers. The results of these studies have shown that fuzzy logic is indeed a powerful control tool, when it comes to control systems or processes which are complex .Some studied have also shown that the fuzzy logic performs better when compared to conventional control mechanisms like PID .Whenever logic in the spirit of human thinking can be introduced we get a more robust summary of the information .What does this really mean? Though we are conditioned to think in precise quantities, at a subconscious level, we think and take actions that are fuzzy in nature .And that is the way we perceive the nature and react to it.

4.2 Origin of fuzzy Logic

The concept of Fuzzy Logic (FL) was conceived by Lotfi Zadeh, a professor at the University of California at Berkley, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time. Professor Zadeh reasoned that people do not require precise, numerical information input, and yet they are capable of highly adaptive control. If feedback controllers could be programmed to accept noisy, imprecise input, they would be much more effective and perhaps easier to implement. Unfortunately, U.S. manufacturers have not been so quick to embrace this technology while the Europeans and Japanese have been aggressively building real products around it.

4.3 Fuzzy logic

In this context, FL is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. FL provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy, or missing input information. FL's approach to control problems mimics how a person would make decisions, only much faster.

4.4 Difference between fuzzy logic and conventional control methods

FL incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving control problem rather than attempting to model a system mathematically. The FL model is empirically-based, relying on an operator's experience rather than their technical understanding of the system. For example, rather than dealing with temperature control in terms such as "SP =500F", "T <1000F", or "210C <TEMP <220C", terms like "IF (process is too cool) AND (process is getting colder) THEN (add heat to the process)" or "IF (process is too hot) AND (process is heating rapidly) THEN (cool the process quickly)" are used. These terms are imprecise and yet very descriptive of what must actually happen. Consider what you do in the shower if the temperature is too cold: you will make the water comfortable very quickly with little trouble. FL is capable of mimicking this type of behavior but at very high rate.

4.5 Advantages of fuzzy logic

FL offers several unique features that make it a particularly good choice for many control problems.

1) It is inherently robust since it does not require precise, noise-free inputs and can be programmed to fail safely if a feedback sensor quits or is destroyed. The output control is a smooth control function despite a wide range of input variations.

2) Since the FL controller processes user-defined rules governing the target control system, it can be modified and tweaked easily to improve or drastically alter system performance. New sensors can easily be incorporated into the system simply by generating appropriate governing rules.

3) FL is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise thus keeping the overall system cost and complexity low.

4) Because of the rule-based operation, any reasonable number of inputs can be processed (1-8 or more) and numerous outputs (1-4 or more) generated, although defining the rule base quickly becomes complex if too many inputs and outputs are chosen for a single implementation since rules defining their interrelations must also be defined. It would be better to break the control system into smaller chunks and use several smaller FL controllers distributed on the system, each with more limited responsibilities.

5) FL can control nonlinear systems that would be difficult or impossible to model mathematically. This opens doors for control systems that would normally be deemed unfeasible for automation.

4.6 Linguistic variables

Linguistic variables are objects or words, rather than numbers. The sensor input is a noun, e.g. "temperature", "displacement", "velocity", "flow", "pressure", etc. Since error is just the difference, it can be thought of the same way.

The fuzzy variables themselves are adjectives that modify the variable (e.g. "large positive" error, "small positive" error, "zero" error, "small negative" error, and "large negative" error). As a minimum, one could simply have "positive", "zero", and "negative" variables for each of the parameters. Additional ranges such as "very large" and "very small" could also be added to extend the responsiveness to exceptional or very nonlinear conditions, but aren't necessary in a basic system.

4.7 The rule matrix

The fuzzy parameters of error (command-feedback) and error-dot (rate-of-change-of-error) were modified by the adjectives "negative", "zero", and "positive". To picture this, imagine the simplest practical implementation, a 3-by-3 matrix. The columns represent "negative error", "zero error", and "positive error" inputs from left to right. The rows represent "negative", "zero", and "positive" "error-dot" input from top to bottom. This planar construct is called a rule matrix. It has two input conditions, "error" and "error-dot", and one output response conclusion (at the intersection of each row and column). In this case there are nine possible logical products (AND) output response conclusions.

Although not absolutely necessary, rule matrices usually have an odd number of rows and columns to accommodate a "zero" center row and column region. This may not be needed as long as the functions on either side of the center overlap somewhat and continuous dithering of the output is acceptable since the "zero" regions correspond to "no change" output responses the lack of this region will cause the system to continually hunt for "zero". It is also possible to have a different number of rows than columns. This occurs when numerous degrees of inputs are needed. The maximum number of possible rules is simply the product of the number of rows and columns, but definition of all of these rules may not be necessary since some input conditions may never occur in practical operation. The primary objective of this construct is to map out the universe of possible inputs while keeping the system sufficiently under control.

4.8 Membership functions

The membership function is a graphical representation of the magnitude of participation of each input. It associates a weighting with each of the inputs that are processed, define functional overlap between inputs, and ultimately determines an output response. The rules use the input membership values as weighting factors to determine their influence on the fuzzy output sets of the final output conclusion. Once the functions are inferred, scaled, and combined, they are defuzzified into a crisp output which drives the system. There are different memberships functions associated with each input and output response. Some features to note are:

SHAPE - triangular is common, but bell, trapezoidal, haversine and, exponential have been used. More complex functions are possible but require greater computing overhead to implement. HEIGHT or magnitude (usually normalized to 1) WIDTH (of the base of function), SHOULDERING (locks height at maximum if an outer function. Shouldered functions evaluate as 1.0 past their center) CENTER points (center of the member function shape) OVERLAP (N&Z, Z&P, typically about 50% of width but can be).

The degree of membership (DOM) is determined by plugging the selected input parameter (error or error-dot) into the horizontal axis and projecting vertically to the upper boundary of the membership function(s).

4.9 Inferencing

The logical products for each rule must be combined or inferred (max-min'd, max-dot'd, averaged, root-sum-squared, etc.) before being passed on to the defuzzification process for crisp output generation. Several inference methods exist. The MAX-MIN method tests the magnitudes of each rule and selects the highest one. The horizontal coordinate of the "fuzzy centroid" of the area under that function is taken as the output. This method does not combine the effects of all applicable rules but does produce a continuous output function and is easy to implement.

The MAX-DOT or MAX-PRODUCT method scales each member function to fit under its respective peak value and takes the horizontal coordinate of the "fuzzy" centroid of the composite area under the function(s) as the output. Essentially, the member function(s) are shrunk so that their peak equals the magnitude of their respective function ("negative", "zero", and "positive"). This method combines the influence of all active rules and produces a smooth, continuous output.

The AVERAGING method is another approach that works but fails to give increased weighting to more rule votes per output member function. For example, if three "negative “rules fire, but only one” zero rules does, averaging will not reflect this difference since both averages will equal 0.5. Each function is clipped at the average and the "fuzzy" centroid of the composite area is computed.

The ROOT-SUM-SQUARE (RSS) method combines the effects of all applicable rules, scales the functions at their respective magnitudes, and computes the "fuzzy" centroid of the composite area. This method is more complicated mathematically than other methods, but was selected for this example since it seemed to give the best weighted influence to all firing rules.

4.10 Defuzzication-Getting back to crisp numbers

The defuzzification of the data into a crisp output is accomplished by combining the results of the inference process and then computing the "fuzzy centroid" of the area. The weighted strengths of each output member function are multiplied by their respective output membership function center points and summed. Finally, this area is divided by the sum of the weighted member function strengths and the result is taken as the crisp output. One feature to note is that since the zero center is at zero, any zero strength will automatically compute to zero. If the center of the zero function happened to be offset from zero (which is likely in a real system where heating and cooling effects are not perfectly equal), then this factor would have an influence.

4.11 Tuning

Tuning the system can be done by changing the rule antecedents or conclusions, changing the centers of the input and/or output membership functions, or adding additional degrees to the input and/or output functions such as "low", "medium", and "high" levels. These new levels would generate additional rules and membership functions which would overlap with adjacent functions forming longer "mountain ranges" of functions and responses. The techniques for doing this systematically are a subject unto itself.

4.12 Designing a fuzzy logic controller

Figure 4.1 shows the basic configuration of a fuzzy logic controller (FLC),which comprises four principal components :a rule base, a fuzzy inference system input fuzzification interface and an output defuzzification interface .The rule base holds a set of IF-THEN rules that quantify the knowledge that human experts have amassed about solving particular problems. It acts as a resource to the fuzzy inference system, which makes successive decisions about which rules are most relevant to the current situations and applies the actions indicated by these rules. The input fuzzifier takes the crisp numeric inputs and, as its name implies convert them into the fuzzy form needed by the fuzzy inference system. At the output the defuzzification interface combines the conclusions reach by the fuzzy inference system and converts them into crisp numeric value as control actions.

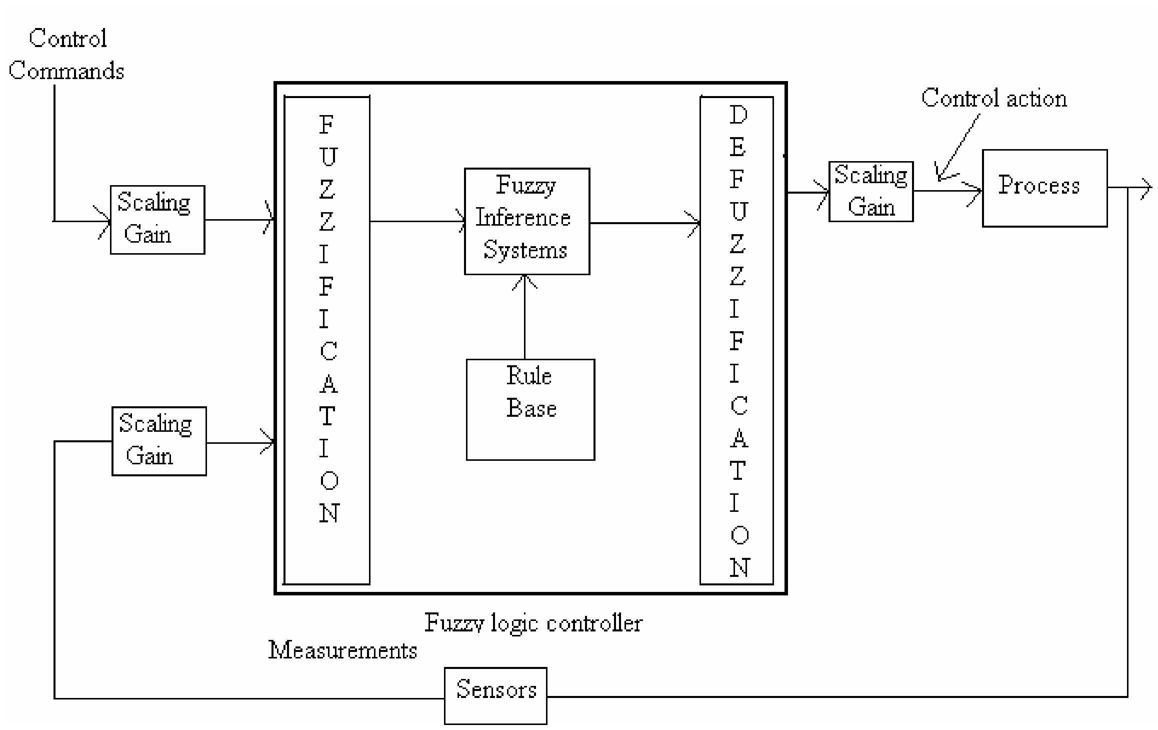


Fig4.1: Fuzzy logic control system block diagram.

The fuzzy logic controller methodology, step by step:

Step One: Define inputs and outputs for the FLC.

Step Two: Define frames for fuzzy variables.

Step Three: Assign membership values to fuzzy variables.

Step Four: Create a rule base.

Step Five: Choose scaling gains for the variables.

Step Six: Fuzzify inputs to the FLC.

Step Seven: determine which rules fire.

Step Eight: Infer the output Recommended by each rule.

Step Nine: Aggregate the fuzzy outputs recommended by each rule.

Step Ten: Defuzzify the aggregated fuzzy set to form crisp output from the FLC.

CHAPTER 5

PID CONTROLLER WITH FUZZY ANTICIPATION

In the case of time varying processes, fuzzy logic can be employed to adapt the parameters of PID controller. But, here interest is to act on signals rather than on parameters to improve the PID behavior using fuzzy intervention in different situations. Figure 5.1 illustrates its employing

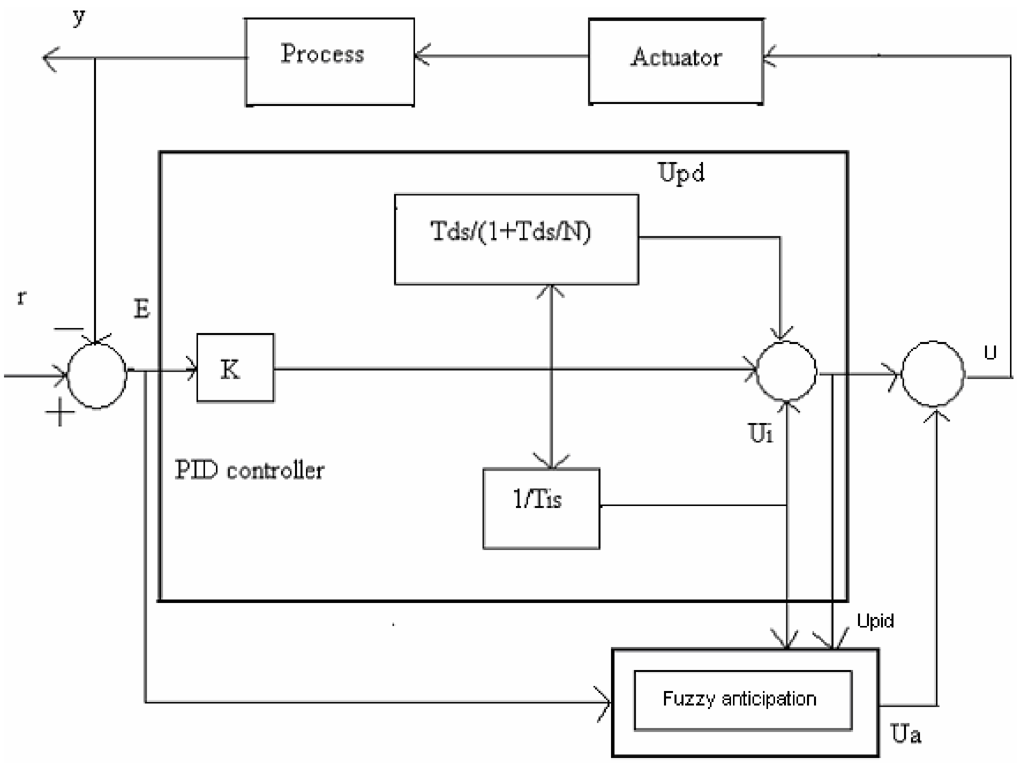


Fig5.1: PID controller with fuzzy anticipation.

The fuzzy anticipation receives all different signals in PID block and generates anti-windup action Ua to improve the system behavior. This action also will compensate rapidly the disturbance effect and improve the system response. Finally, the fuzzy correction term will be added to the Upid signal to generate the applied control signal U to the actuator. Fuzzy anticipation can be considered as a signal processor. It accelerates the system response (feed forward action) and adds selfautonomy behavior to the PID controller. The added fuzzy anticipation is employed to treat two important problems in PID controller.

-Windup in integrator due to saturation in actuator.

-Slow response to reject strong disturbances.

Here feed-forward compensation to treat the integrator windup problem is applied. This can lead to fast response without large overshoots. The anti-windup block receives the input signals:

- control output signal Upid,
- integral action Ui, and
- control error signal E.

If the Upid is not saturated, the block output is set to be zero; otherwise a generated action should be taken as function of Ui and the control error signal E. The following fuzzy rules are applied where P, Z, and N are Positive, Zero, and Negative fuzzy sets respectively.

Table 5.1.Fuzzy sets

Ui E	N	Z	P
N	Z	Z	N
Z	P	Z	N
P	P	Z	Z

CHAPTER 6

MATLAB MANUAL

6.1 SIMULINK

6.1.1 Introduction

Simulink is a software package that enables you to model, simulate, and analyze systems whose outputs change over time. Such systems are often referred to as dynamic systems. Simulink can be used to explore the behavior of a wide range of real-world dynamic systems, including electrical circuits, shock absorbers, braking systems, and many other electrical, mechanical, and thermodynamic systems.

Simulating a dynamic system is a two-step process with Simulink. First, a user creates a block diagram, using the Simulink model editor that graphically depicts time-dependent mathematical relationships among the system's inputs, states, and outputs. The user then commands Simulink to simulate the system represented by the model from a specified start time to a specified stop time.

6.1.2 Modeling Dynamic Systems

A Simulink block diagram model is a graphical representation of a mathematical model of a dynamic system. A mathematical model of a dynamic system is described by a set of equations. The mathematical equations described by a block diagram model are known as algebraic, differential, and/or difference equations.

6.1.3 Block Diagram Semantics

A classic block diagram model of a dynamic system graphically consists of blocks and lines (signals). The history of these block diagram model is derived from engineering areas such as Feedback Control Theory and Signal Processing. A block within a block diagram defines a dynamic system in itself. The relationships between each elementary dynamic system in a block diagram are illustrated by the use of signals connecting the blocks. Collectively the blocks and lines in a block diagram describe an overall dynamic system.

Simulink extends these classic block diagram models by introducing the notion of two classes of blocks, nonvirtual block and virtual blocks. Nonvirtual blocks represent elementary systems. A virtual block is provided for graphical organizational convenience and plays no role in the definition of the system of equations described by the block diagram model. Examples of virtual blocks are the Bus Creator and Bus Selector which are used to reduce block diagram clutter by managing groups of signals as a "bundle.".

In general, block and lines can be used to describe many "models of computations." One example would be a flow chart. A flow chart consists of blocks and lines, but one cannot describe general dynamic systems using flow chart semantics.

The term ‘time-based block diagram' is used to distinguish block diagrams that describe dynamic systems from that of other forms of block diagrams. In Simulink, we use the term block diagram (or model) to refer to a time-based block diagram unless the context requires explicit distinction.
To summarize the meaning of time-based block diagrams:

- Simulink block diagrams define time-based relationships between signals and state variables. The solution of a block diagram is obtained by evaluating these relationships over time, where time starts at a user specified "start time" and ends at a user specified "stop time." Each evaluation of these relationships is referred to as a time step.
- Signals represent quantities that change over time and are defined for all points in time between the block diagram's start and stop time.
- The relationships between signals and state variables are defined by a set of equations represented by blocks. Each block consists of a set of equations

(block methods). These equations define a relationship between the input signals, output signals and the state variables. Inherent in the definition of a equation is the notion of parameters, which are the coefficients found within the equation.

6.1.4 Creating Models

Simulink provides a graphical editor that allows you to create and connect instances of block types selected from libraries of block types via a library browser. Simulink provides libraries of blocks representing elementary systems that can be used building blocks. The blocks supplied with Simulink are called built-in blocks. Simulink users can also create their own block types and use the Simulink editor to create instances of them in a diagram. Customer-defined blocks are called custom blocks.

6.1.5 Time

Time is an inherit component of block diagrams in that the results of a block diagram simulation change with time. Put another way, a block diagram represents the instantaneous behavior of a dynamic system. Determining a system's behavior over time thus entails repeatedly executing the model at intervals, called time steps, from the start of the time span to the end of the time span. Simulink refers to the repeated execution of a model at successive time steps as simulating the system that the model represents. It is possible to simulate a system manually, i.e., to execute its model manually. However, this is unnecessary as the Simulink engine performs this task automatically on command from the user.

6.1.6 States

Typically the current values of some system, and hence model, outputs are functions of the previous values of temporal variables. Such variables are called states. Computing a model's outputs from a block diagram hence entails saving the value of states at the current time step for use in computing the outputs at a subsequent time step. Simulink performs this task during simulation for models that define states.

Two types of states can occur in a Simulink model: discrete and continuous states. A continuous state changes continuously. Examples of continuous states are the position and speed of a car. A discrete state is an approximation of a continuous state where the state is updated (recomputed) using finite (periodic or a periodic) intervals. An example of a discrete state would be the position of a car shown on a digital odometer where it is updated every second as opposed to continuously. In the limit, as the discrete state time interval approaches zero, a discrete state becomes equivalent to a continuous state.

Blocks implicitly define a model's states. In particular, a block that needs some or all of its previous outputs to compute its current outputs implicitly defines a set of states that need to be saved between time steps. Such a block is said to have states.

Blocks that define continuous states include the following standard Simulink blocks:

- Integrator
- State-Space
- Transfer Function
- Zero-Pole

The total number of a model's states is the sum of all the states defined by all its blocks. Determining the number of states in a diagram requires parsing the diagram to determine the types of blocks that it contains and then aggregating the number of states defined by each instance of a block type that defines states. Simulink performs this task during the Compilation phase of a simulation.

6.1.7 Continuous States

Computing a continuous state entails knowing its rate of change, or derivative. Since the rate of change of a continuous state typically itself changes continuously (i.e., is itself a state), computing the value of a continuous state at the current time step entails integration of its derivative from the start of a simulation. Thus modeling a continuous state entails representing the operation of integration and the process of computing the state's derivative at each point in time. Simulink block diagrams use Integrator blocks to indicate integration and a chain of operator blocks connected to the integrator block to represent the method for computing the state's derivative. The chain of blocks connected to the Integrator's is the graphical counterpart to an ordinary differential equation (ODE).

In general, excluding simple dynamic systems, analytical methods do not exist for integrating the states of real-world dynamic systems represented by ordinary differential equations. Integrating the states requires the use of numerical methods called ODE solvers. These various methods trade computational accuracy for computational workload. Simulink comes with computerized implementations of the most common ODE integration methods and allows a user to determine which it uses to integrate states represented by Integrator blocks when simulating a system.

Computing the value of a continuous state at the current time step entails integrating its values from the start of the simulation. The accuracy of numerical integration in turn depends on the size of the intervals between time steps. In general, the smaller the time step, the more accurate the simulation. Some ODE solvers, called variable time step solvers, can automatically vary the size of the time step, based on the rate of change of the state, to achieve a specified level of accuracy over the course of a simulation. Simulink allows the user to specify the size of the time step in the case of fixed-step solvers or allow the solver to determine the step size in the case of variable-step solvers. To minimize the computation workload, the variable-step solver chooses the largest step size consistent with achieving an overall level of precision specified by the user for the most rapidly changing model state. This ensures that all model states are computed to the accuracy specified by the user.

6.1.8 Discrete States

Computing a discrete state requires knowing the relationship between the current time and its value at the time at which it previously changed value. Simulink refers to this relationship as the state's update function. A discrete state depends not only on its value at the previous time step but also on the values of a model's inputs. Modeling a discrete state thus entails modeling the state's dependency on the systems' inputs at the previous time step. Simulink block diagrams use specific types of blocks, called discrete blocks, to specify update functions and chains of blocks connected to the inputs of the block's to model the state's dependency on system inputs.

As with continuous states, discrete states set a constraint on the simulation time step size. Specifically a step size must be chosen that ensure that all the sample times of the model's states are hit. Simulink assigns this task to a component of the Simulink system called a discrete solver. Simulink provides two discrete solvers: a fixed-step discrete solver and a variable-step discrete solver. The fixed-step discrete solver determines a fixed step size that hits all the sample times of all the model's discrete states, regardless of whether the states actually change value at the sample time hits. By contrast, the variable-step discrete solver varies the step size to ensure that sample time hits occur only at times when the states change value.

6.1.9 Modeling Hybrid Systems

A hybrid system is a a system that has both discrete and continuous states Strictly speaking a hybrid model is identified as having continuous and discrete sample times from which it follows that the model will have continuous and discrete states. Solving a model of such a system entails choosing a step size that satisfies both the precision constraint on the continuous state integration and the sample time hit constraint on the discrete states. Simulink meets this requirement by passing the next sample time hit as determined by the discrete solver as an additional constraint on the continuous solver. The continuous solver must choose a step size that advances the simulation up to but not beyond the time of the next sample time hit. The continuous solver can take a time step short of the next sample time hit to meet its accuracy constraint but it cannot take a step beyond the next sample time hit even if its accuracy constraint allows it to.

6.1.10 Block Parameters

Key properties of many standard blocks are parameterized. For example, the Constant value of the Simulink Constant block is a parameter. Each parameterized block has a block dialog that lets you set the values of the parameters. You can use MATLAB expressions to specify parameter values. Simulink evaluates the expressions before running a simulation. You can change the values of parameters during a simulation. This allows you to determine interactively the most suitable value for a parameter.

A parameterized block effectively represents a family of similar blocks. For example, when creating a model, you can set the Constant value parameter of each instance of the Constant block separately so that each instance behaves differently. Because it allows each standard block to represent a family of blocks, block parameterization greatly increases the modeling power of the standard Simulink libraries.

Each time you change parameters, you change the meaning of the model. Simulink lets you modify the parameter values during execution of your model. For example, you can pause simulation, change parameter values, and continue simulation. It should be pointed out that parameter changes do not immediately occur, but are queued up and then applied at the start of the next time step during model execution. Returning to our example of the constant block, the function it defines as

$$\text{Signal (t) =Constant Value}$$

for all time. If we were to allow the constant value to be changed immediately, then the solution at the point in time at which the change occurred would be invalid, thus we must queue the change for processing on the next time step.

6.1.11 Systems and Subsystems

A Simulink block diagram can consist of layers. Each layer is defined by a subsystem. A subsystem is part of the overall block diagram and ideally has no impact on the meaning of the block diagram. Subsystems are provided primarily to help in the organization aspects a block diagram. Subsystem do not define a separate block diagram. Simulink differentiates between two different types of subsystems virtual and nonvirtual subsystems. The main difference is that nonvirtual subsystems provide the ability to control when the contents of the subsystem are evaluated.

6.1.12 Conditionally Executed Subsystems

You can create conditionally executed subsystems that are executed only when a transition occurs on a triggering, function-call, action, or enabling input. Conditionally executed subsystems are atomic. Unconditionally executed subsystems are virtual by default. You can, however, designate an unconditionally executed subsystem as atomic. This is useful if you need to ensure that the equations defined by a subsystem are evaluated "together" as a unit.

6.1.13 Signals

Simulink uses the term signal to refer to a time varying quantity that has values at all points in time. Simulink allows you to specify a wide range of signal attributes, including signal name, data type (e.g., 8-bit, 16-bit, or 32-bit integer), numeric type (real or complex), and dimensionality (one-dimensional or two-dimensional array). Many blocks can accept or output signals of any data or numeric type and dimensionality. Others impose restrictions on the attributes of the signals they can handle.

On the block diagram, you will find that the signals are represented with lines that have an arrow head. The source of the signal corresponds to the block that writes to the signal during evaluation of its block methods (equations). The destinations of the signal are blocks that read the signal during the evaluation of its block methods (equations). A good analogy of the meaning of a signal is to consider a classroom. The teacher is the one responsible for writing on the white board and the students read what is written on the white board when they choose to. This is also true of Simulink signals, a reader of the signal (a block method) can choose to read the signal as frequently or infrequently as so desired.

6.1.14 Simulating Dynamic Systems

Simulating a dynamic system refers to the process of computing a system's states and outputs over a span of time, using information provided by the system's model. Simulink simulates a system when you choose Start from the model editor's Simulation menu, with the system's model open.

6.1.15 Model Compilation

First, the Simulink engine invokes the model compiler. The model compiler converts the model to an executable form, a process called compilation. In particular, the compiler

- Evaluates the model's block parameter expressions to determine their values.
- Determines signal attributes, e.g., name, data type, numeric type, and dimensionality, not explicitly specified by the model and checks that each block can accept the signals connected to its inputs.
- Simulink uses a process called attribute propagation to determine unspecified attributes. This process entails propagating the attributes of a source signal to the inputs of the blocks that it drives.
- Performs block reduction optimizations. Flattens the model hierarchy by replacing virtual subsystems with the blocks that they contain (see Solvers). Sorts the blocks into the order in which they need to be executed during the execution phase (see Solvers). Determines the sample times of all blocks in the model whose sample times you did not explicitly specify

6.2 FUZZY LOGIC TOOLBOX

6.2.1 Fuzzy Logic Toolbox

The Fuzzy Logic Toolbox is a collection of functions built on the MATLAB numeric computing environment. It provides tools for you to create and edit fuzzy inference systems within the framework of MATLAB, or if you prefer, you can integrate your fuzzy systems into simulations with Simulink. You can even build stand-alone C programs that call on fuzzy systems you build with MATLAB. This toolbox relies heavily on graphical user interface (GUI) tools to help you accomplish your work, although you can work entirely from the command line if you prefer.

The toolbox provides three categories of tools:

- Command line functions
- Graphical interactive tools
- Simulink blocks and examples

The first category of tools is made up of functions that you can call from the command line or from your own applications. Many of these functions are MATLAB M-files, series of MATLAB statements that implement specialized fuzzy logic algorithms. You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files.

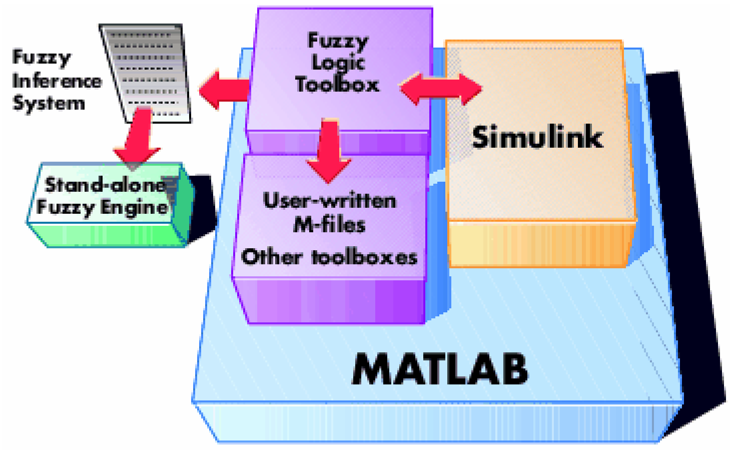
Secondly, the toolbox provides a number of interactive tools that let you access many of the functions through a GUI. Together, the GUI- based tools provide an environment for fuzzy inference system design, analysis, and implementation.

The third category of tools is a set of blocks for use with the Simulink simulation software. These are specifically designed for high speed fuzzy logic inference in the Simulink environment.

6.2.2 Function of Fuzzy Logic Toolbox

The Fuzzy Logic Toolbox allows you to do several things, but the most important thing it lets you do is create and edit fuzzy inference systems. You can create these systems using graphical tools or command-line functions, or you can generate them automatically using either clustering or adaptive neuro-fuzzy techniques.

If you have access to Simulink, you can easily test your fuzzy system in a block diagram simulation environment. The toolbox also lets you run your own stand-alone C programs directly, without the need for Simulink. This is made possible by a stand-alone Fuzzy Inference Engine that reads the fuzzy systems saved from a MATLAB session. You can customize the stand-alone engine to build fuzzy inference into your own code. All provided code is ANSI compliant.



Because of the integrated nature of the MATLAB environment, you can create your own tools to customize the Fuzzy Logic Toolbox or harness it with another toolbox, such as the Control System Toolbox, Neural Network Toolbox, or Optimization Toolbox, to mention only a few of the possibilities.

6.2.3 Building Systems with the Fuzzy Logic Toolbox

Now we're going to work through a similar tipping example, only we'll be building it using the graphical user interface (GUI) tools provided by the Fuzzy Logic Toolbox. Although it is possible to use the Fuzzy Logic Toolbox by working strictly from the command line, in general it is much easier to build a system graphically. There are five primary GUI tools for building, editing, and observing fuzzy inference systems in the Fuzzy Logic Toolbox: the Fuzzy Inference System or FIS Editor, the Membership Function Editor, the Rule Editor, the Rule Viewer, and the Surface Viewer. These GUIs are dynamically linked, in that changes you make to the FIS using one of them, can affect what you see on any of the other open GUIs. You can have any or all of them open for any given system.

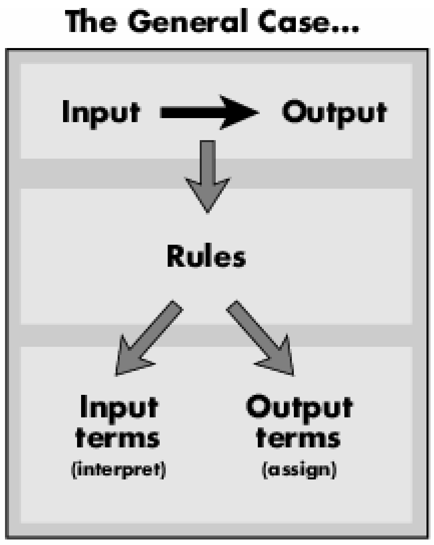
In addition to these five primary GUIs, the toolbox includes the graphical ANFIS Editor GUI, which is used for building and analyzing Sugeno-type adaptive neural fuzzy inference systems.

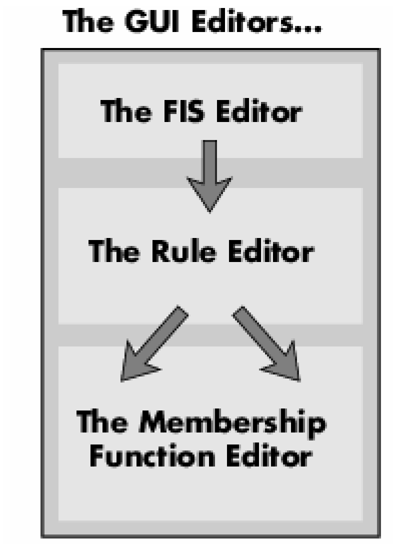
The FIS Editor handles the high-level issues for the system: How many inputs and output variables? What are their names? The Fuzzy Logic Toolbox doesn't limit the number of inputs. However, the number of inputs may be limited by the available memory of your machine. If the number of inputs is too large, or the number of membership functions is too big, then it may also be difficult to analyze the FIS using the other GUI tools.

The Membership Function Editor is used to define the shapes of all the membership functions associated with each variable. The Rule Editor is for editing the list of rules that defines the behavior of the system.

The Rule Viewer and the Surface Viewer are used for looking at, as opposed to editing, the FIS. They are strictly read-only tools. The Rule Viewer is a MATLAB based display of the fuzzy inference diagram shown at the end of the last section. Used as a diagnostic, it can show (for example) which rules are active, or how individual membership function shapes are influencing the results. The Surface Viewer is used to display the dependency of one of the outputs on any one or two of the inputs — that is, it generates and plots an output surface map for the system.

This section began with an illustration similar to the one below describing the main parts of a fuzzy inference system, only the one below shows how the three editors fit together. The two viewers examine the behavior of the entire system.





The five primary GUIs can all interact and exchange information. Any one of them can read and write both to the workspace and to the disk (the read-only viewers can still exchange plots with the workspace and/or the disk). For any fuzzy inference system, any or all of these five GUIs may be open. If more than one of these editors is open for a single system, the various GUI windows are aware of the existence of the others, and will, if necessary, update related windows. Thus if the names of the membership functions are changed using the Membership Function Editor, those changes are reflected in the rules shown in the Rule Editor. The editors for any number of different FIS systems may be open simultaneously. The FIS Editor, the Membership Function Editor, and the Rule Editor can all read and modify the FIS data, but the Rule Viewer and the Surface Viewer do not modify the FIS data in any way.

CHAPTER 7

SIMULATION WORK AND RESULTS

The simulation work is carried out using Simulink in Matlab environment software. The numeric example in [2] is used to investigate the potential of the controller. The process and actuator transfer functions are given as:

$$G_p(s) = \frac{e^{-0.2s}}{s+1} \qquad\qquad G_a(s) = \frac{e^{-0.1}}{0.2s+1}$$

The controller parameters are chosen: K=2.01, Ti=0.92, Td = 0.23**,** and N=10 (filter factor)..The actuator output was limited to 1.Model of numeric example considered is shown in the figure 7.1.Model of internal structure of actuator with saturation and plant is respectively shown in figure 7.2 and figure 7.3.

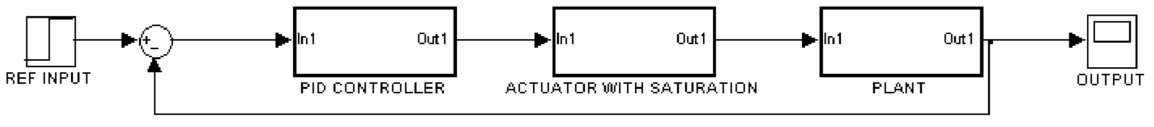


Figure7.1: General structure of a control system subject to actuator saturation.

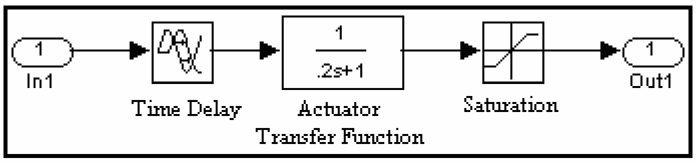


Figure7.2: Model of actuator with saturation.

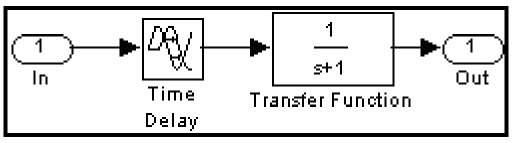


Figure7.3: Model of plant considered.

Simulation results with different PID controllers have shown below one by one. Graph with color blue, green and red indicate step input of 0.9, 0.7 and 0.5 respectively.

7.1 Conventional PID controller

Simulink model of conventional PID controller is shown in figure7.4 and step response without compensation the integrator windup is shown in figure7.5. The obtained results represent the effects of integrator windup for different step inputs (unit step, 0.9, 0.7, and 0.5).The effect of windup is clear when the step input is closed to the actuator limit value (0.9 in this case).

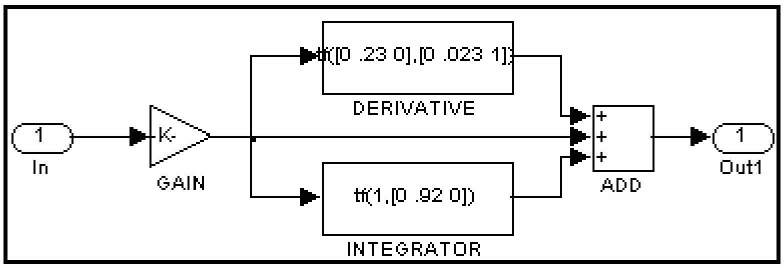


Figure7.4: Model of conventional PID controller.

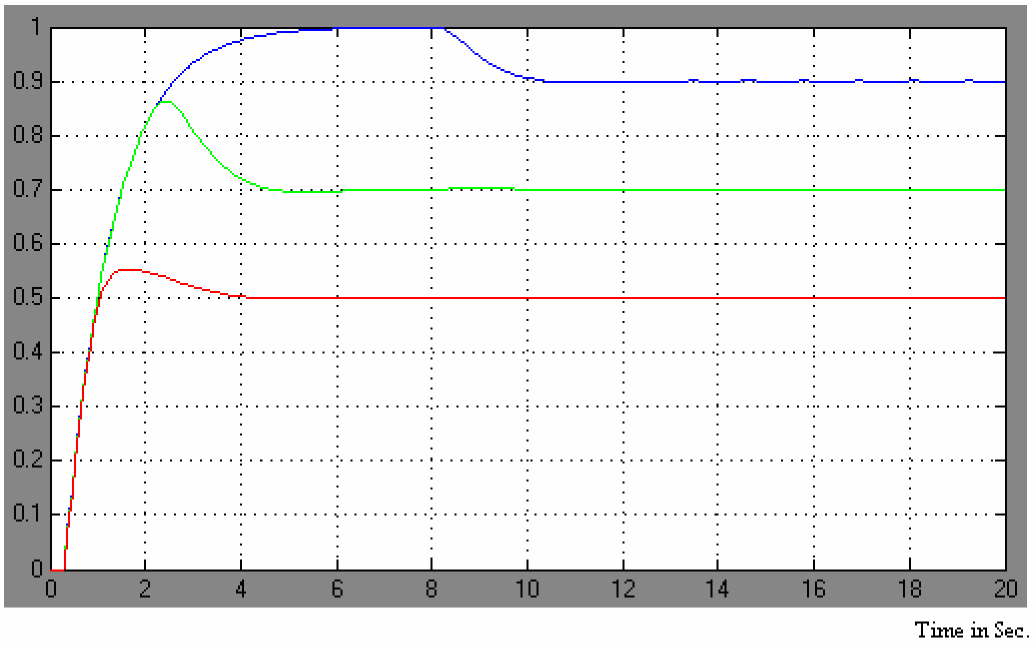


Figure7.5: Step response (Integrator windup)

7.2 Conditional integration

Simulink model of PID controller with conditional integration is shown in figure7.6 and step response using it is shown in figure7.7.

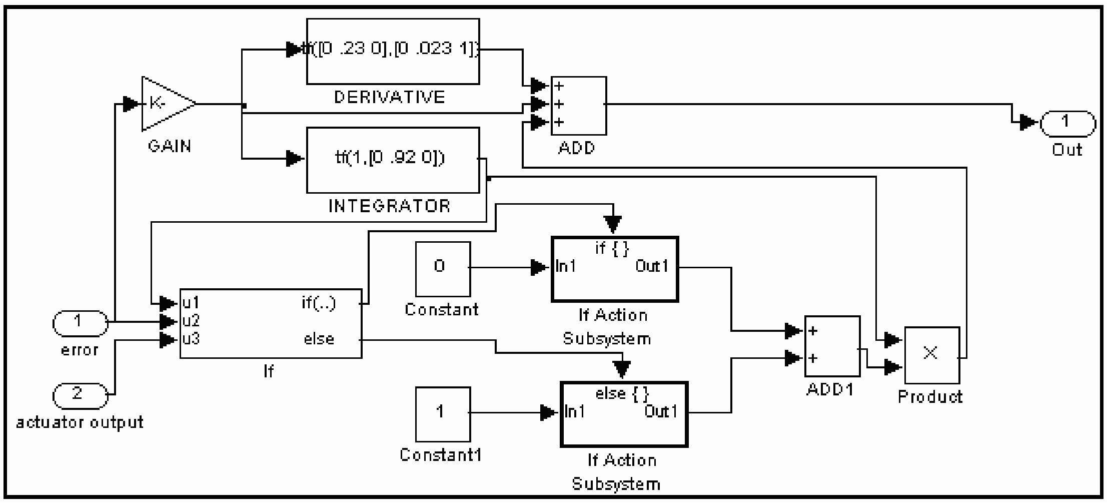


Figure7.6: Model of PID controller with conditional integration.

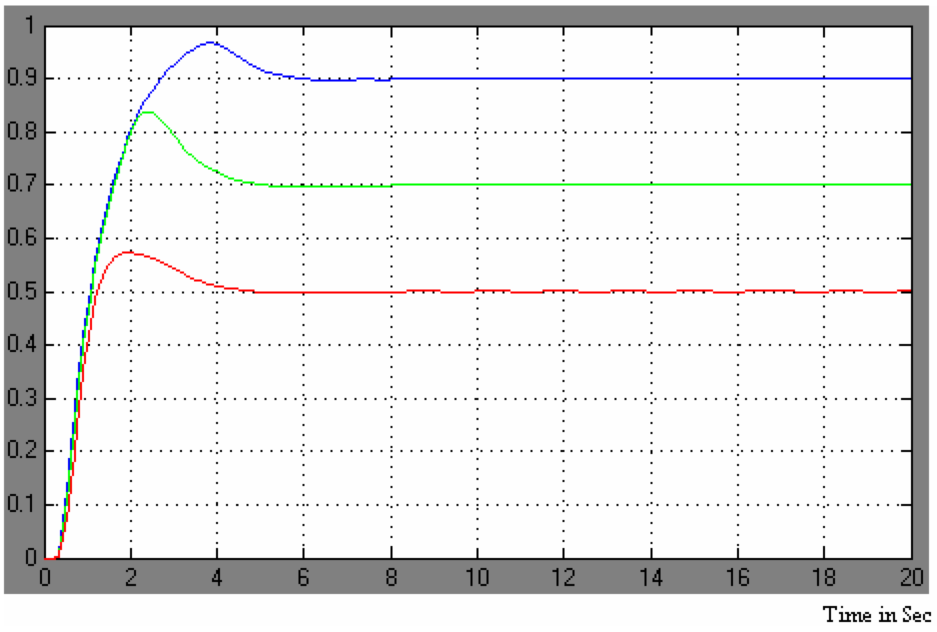


Figure7.7: Step response using conditional integration

7.3 Limited integration

Simulink model of PID controller with limited integration is shown in figure7.8 and step response using it is shown in figure7.9. The limited integrator method gives higher overshoots and that’s why it is the least desirable method.

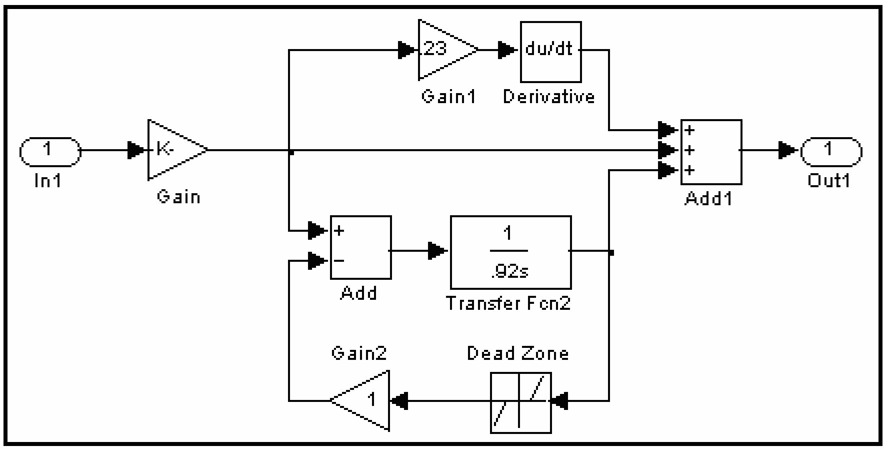


Figure7.8: Model of PID controller with a limited integration.

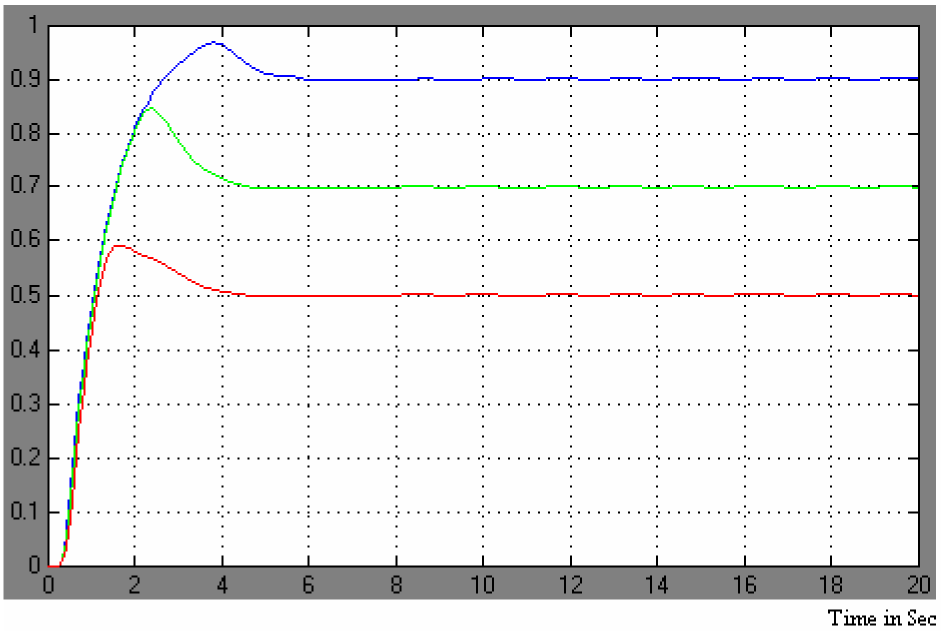


Fig7.9: Step response using limited integrator.

7.4 Tracking anti-windup PID controller

Simulink model of tracking anti-windup PID controller is shown in figure7.10 and step response using it is shown in figure7.11.In the tracking tuning is possible by adjusting the dead zone gain b. From figure7.12 it is observed that tracking anti-windup PID controller is very sensitive to changes in the gain b. Too high a value for b effectively reduce the amount of overshoot, but may also lead to a slow response of the system.

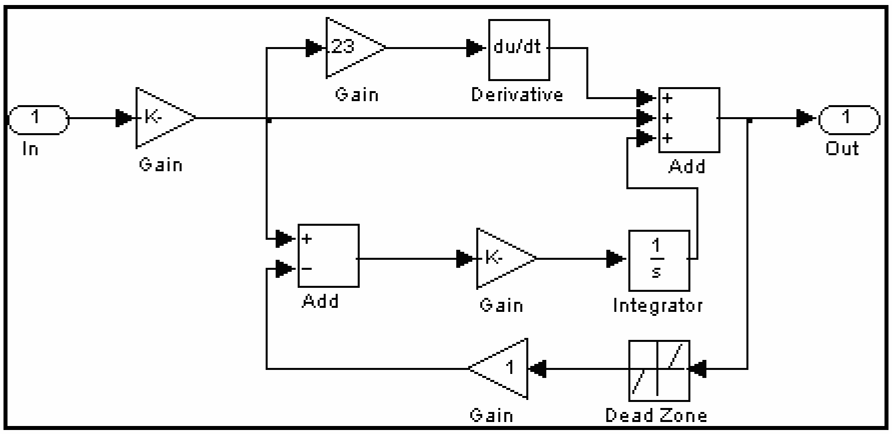


Figure7.10: Model of tracking anti-windup PID controller

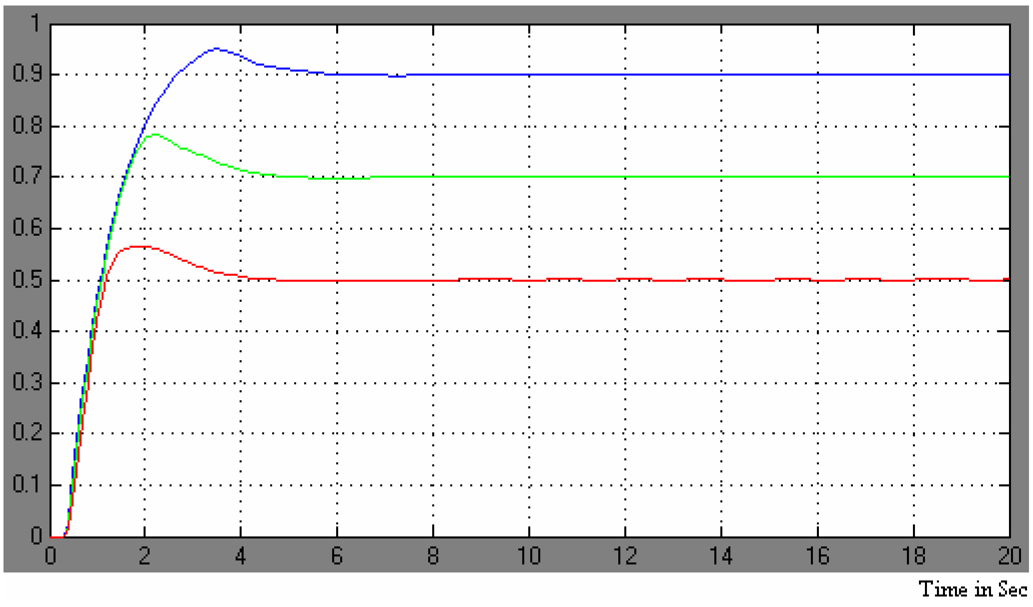


Fig7.11: Step response using tracking with b=1

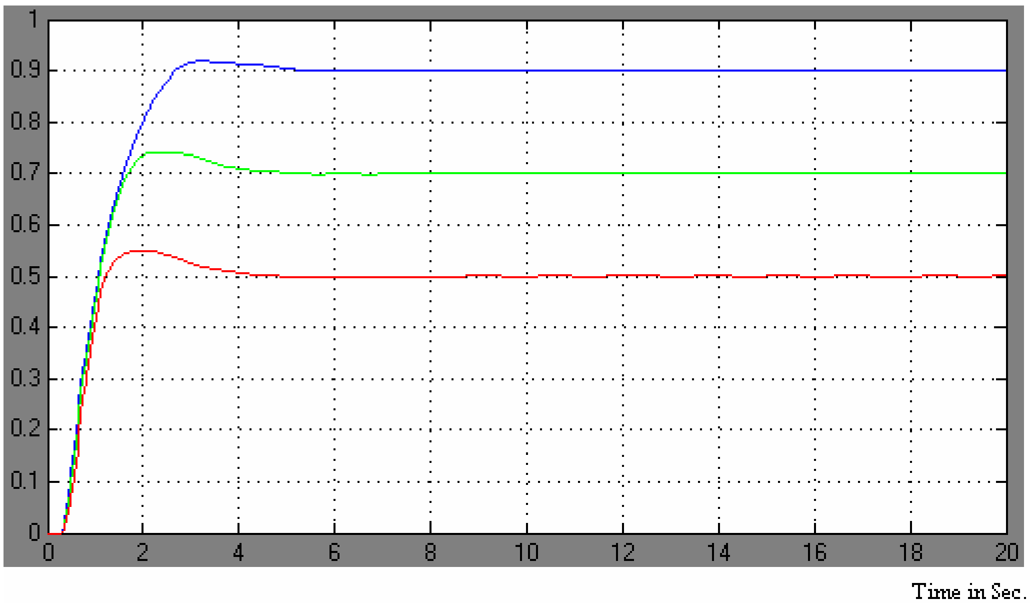


Fig7.12: Step response using tracking with b=2

7.5 Modified tracking anti-windup PID controller

Simulink model of modified tracking anti-windup PID controller is shown in figure7.13 and step response using it is shown in figure7.14. In modified tracking methods, tuning is possible by adjusting the dead zone gain *b*, and saturation Hpd. The modified tracking method is found less sensitive to changes in the system parameters, so tuning is easier and there is a smaller risk of obtaining a slow response due to mistuning. It also allows more flexible tuning as this method has two free design parameters. Reduced values for r**,** (r=0.5…1) give a faster response, but also a higher overshoot. By selecting a higher value for r **(**r=1…1.5), reduces the overshoot, but we get a slower response. A value of r = 0 corresponds to a pure limited integrator.

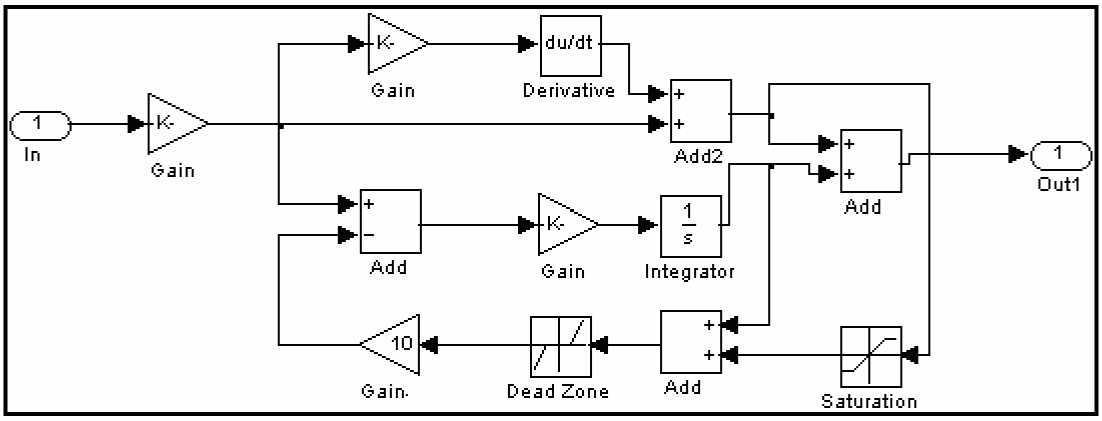


Figure7.13: Model of modified tracking anti-windup PID controller.

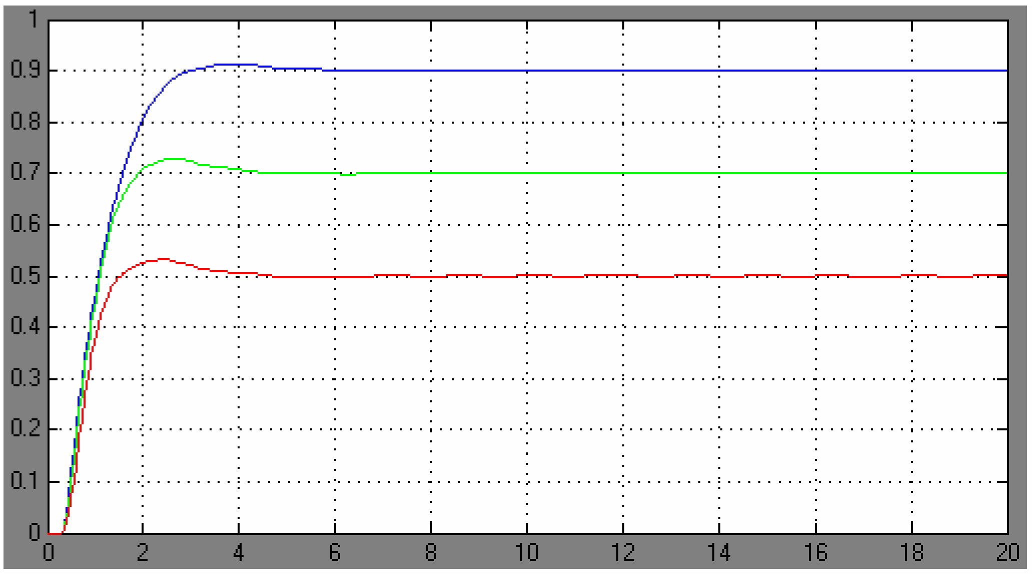


Fig7.14: Step response using modified tracking with b=10, r=1

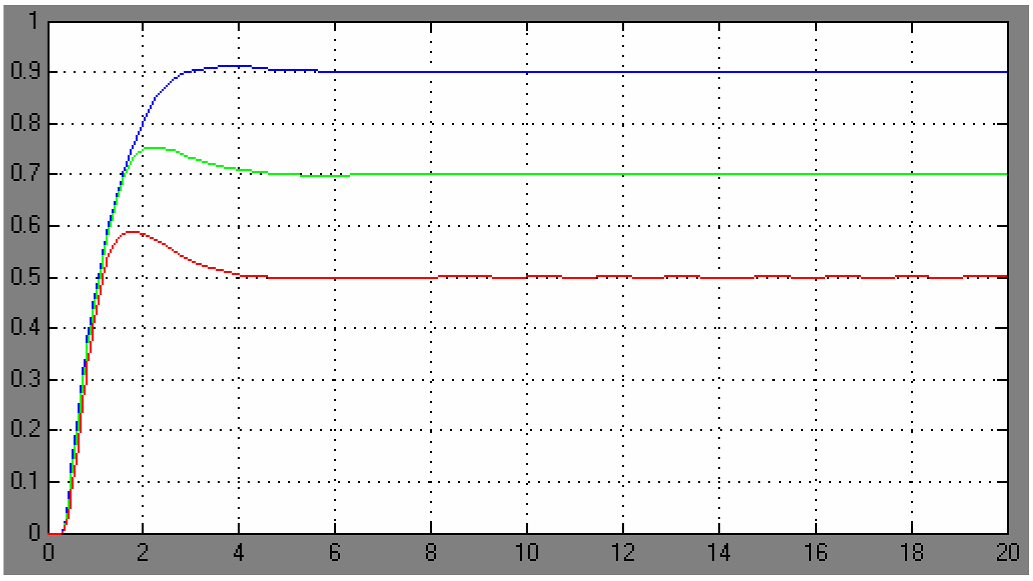


Fig7.15: Step response using modified tracking with b=10, r=0.5

7.6 PID plus fuzzy anticipation

In fuzzy it is so easy to define the inputs, outputs, membership functions and rules using the fuzzy editor. The membership functions are chosen triangular in form. Inputs and outputs are normalized in the range (-1 to 1).Fuzzy anticipation system, Inputs, outputs, membership functions, rules viewer and surface viewer is shown in figure from fig7.16 to fig7.20.

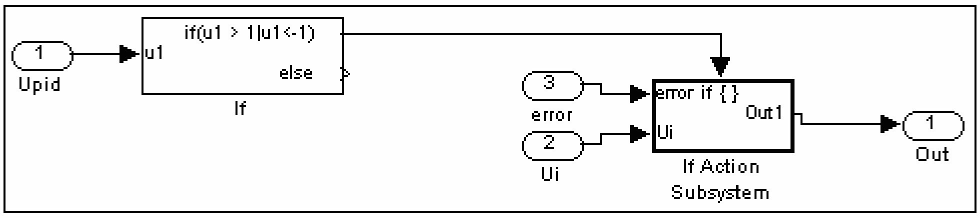


Fig7.16: Fuzzy anticipation system

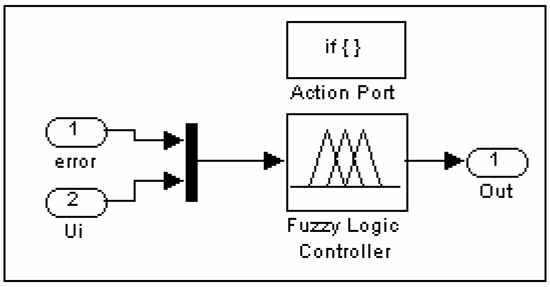
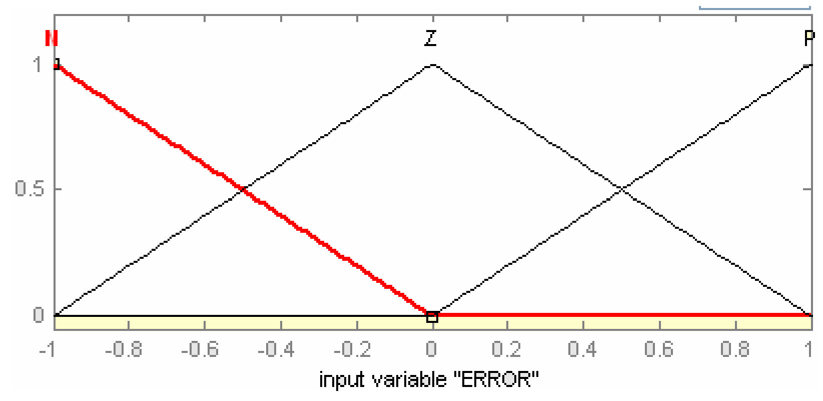
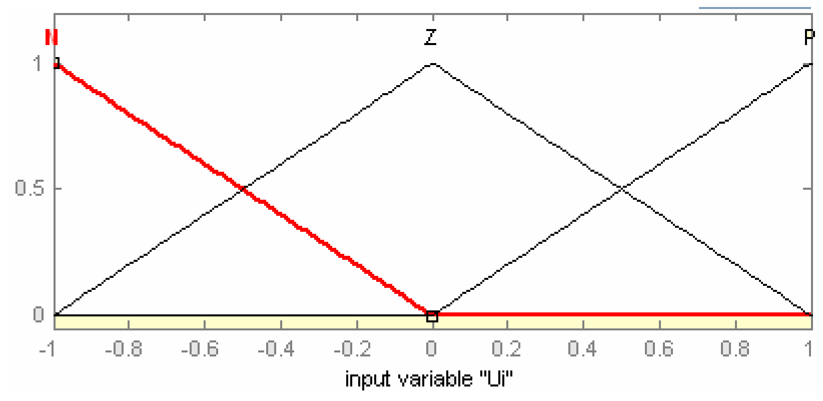


Fig7.17: If action subsystem





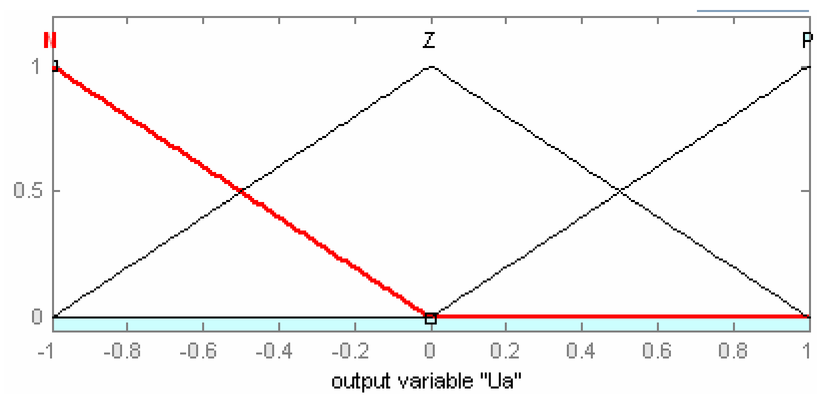


Fig7.18: Membership Functions

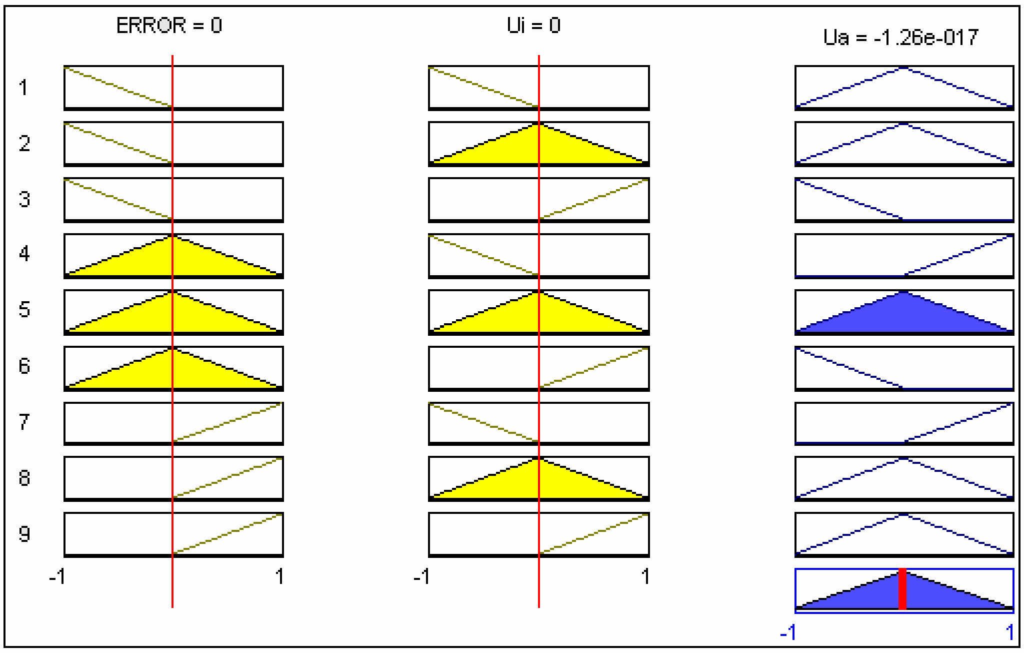


Fig7.19: Rule Viewer

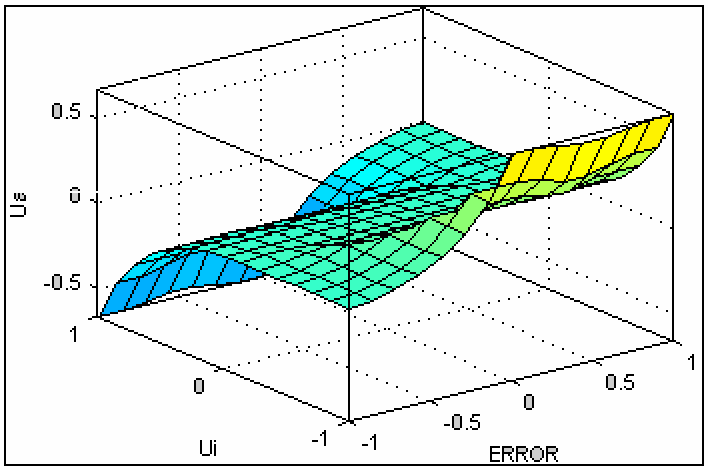


Fig7.20: Surface Viewer

The next figure shows the step response using fuzzy anticipation for the same above conditions. The obtained results show that the overshoot and settling time is reduced and affirm the potential of the proposed methodology.

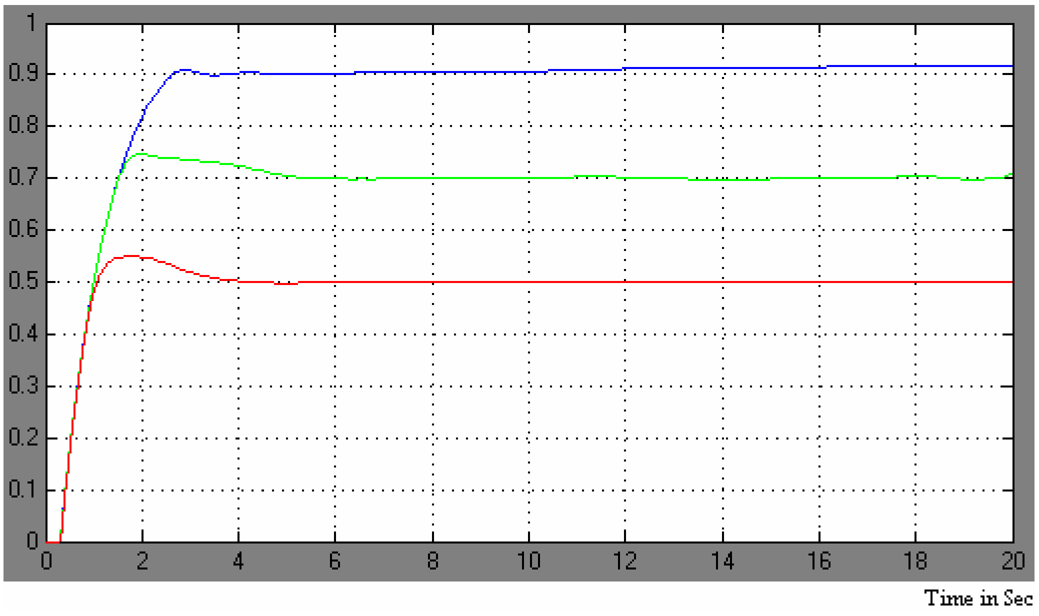


Fig7.21: Step response (PID + Fuzzy Anticipation)

The simulation work is extended to test the system behavior to reject rapidly the disturbance signal. The next figure shows the reaction of the PID controller only. The disturbance signal is applied at the instant 6 sec. Its value is constant and equal to (-0.1). The controller takes approx 2 seconds to reject the disturbance.

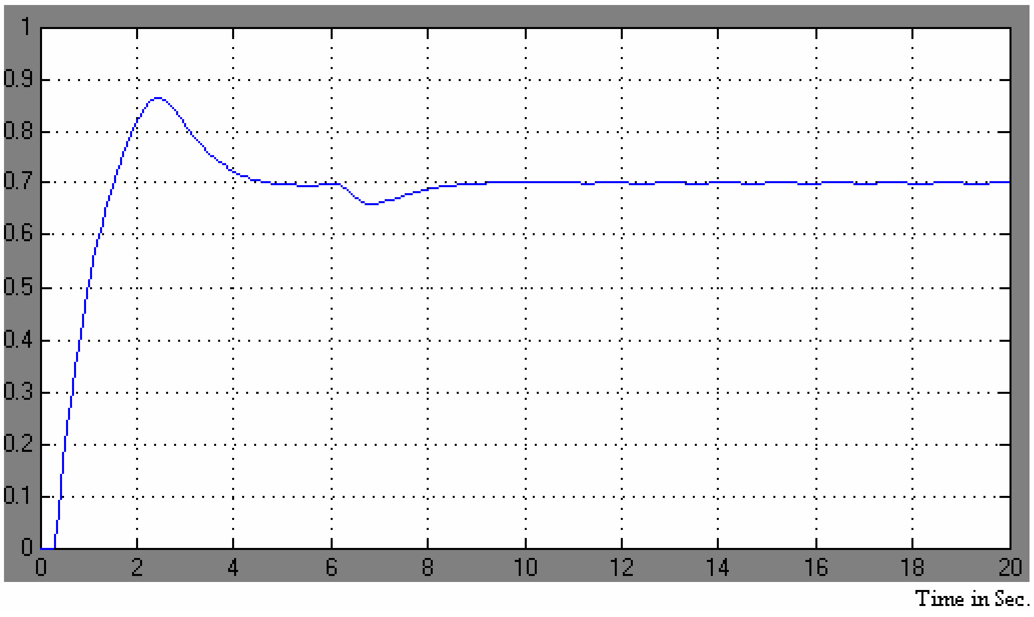


Fig7.22: Disturbance rejection (PID)

Figure7.23 shows the system response using PID with fuzzy anticipation to reject the same disturbance signal. The controller takes less time to retain the desired output. This result affirms the potential of the proposed methodology to reject rapidly the strong and fast disturbances.

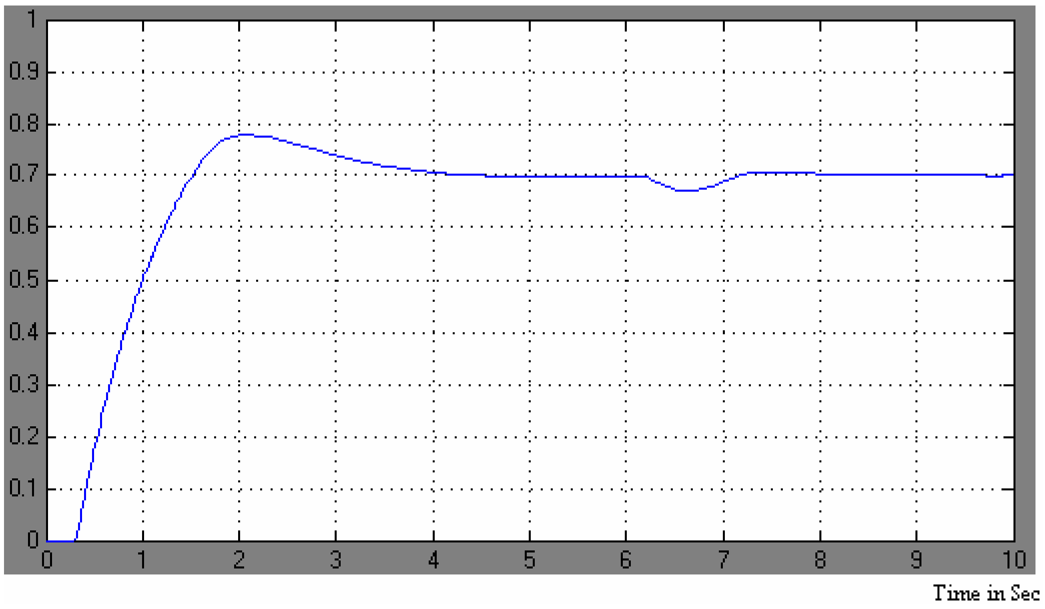


Fig7.23: Disturbance rejection (PID + Anticipation)

CHAPTER 8

CONCLUSION AND FURTHER SCOPE

All the classical approaches urge to decrease the integral action when the actuator saturates. But, the correction is taken through a feedback. It delays its effect. If strong and rapid disturbances act on the process, the anti-windup will slow down the compensation reaction that is not practical. Therefore, a fuzzy anticipation to improve the PID behavior is presented.

The controller consists of two blocks: the former is a conventional PID controller, while the second one is a feed-forward fuzzy anticipation block. The fuzzy anticipation is used to treat the windup problem when the actuator saturates and also to reject rapidly the disturbance signals. Therefore, the overall system behavior is improved by acting on signal processing of PID outputs rather than to adapt its parameters. The proposed controller is verified through simulation on a numeric example. The simulation is carried out using SIMULINK in MATLAB 7.01.The obtained results affirmed the potential of the controller.

Here in Fuzzy Anticipation system there are two inputs and one output. For normalizing the inputs and output in the range of 1 to -1, one has to multiply it by scaling gain. For error input it is not required, because error is simply divided by the reference input for bringing it in the desired range .But for other input and output it requires scaling gain, to get best response. Here one scaling gain factor is kept constant and other is varied manually till desired response achieved. Here one can also apply genetic algorithm to find out the optimal value of scaling gain

LIST OF REFERENCES

I. Jonas Ohr, "Anti-Windup and Control of Systems with Multiple Input Saturations: Tools, Solutions and Case Studies", Dissertation for the Degree of Philosophy, Uppsala University, 2003. ISBN 91-506-1691-9.

II. C. Bohn and D. P. Atherton, "An Analysis Package Comparing PID Anti-windup Strategies", IEEE Control Systems, Vol. 15, no 2, PP.34-40, April 1995.

III. S. Dutta, "Fuzzy Logic Applications", Technological and Strategic Issues, IEEE Trans. On Engineering Management, Vol. 40, no 3, PP. 237-254, 1993.

IV. L. A. Zadeh, "Fuzzy logic", Trans. IEEE Computer, pp. 83-93,April. 1988.

V. E.H Mamdani, "Application of fuzzy algorithm for control of simple dynamic plant",Proc. IEE, part ,Vol.121,pp.1585-1588,1974.

VI. Stephanopoulos. G, Chemical process Control-An introduction to Theory and practice, New Delhi: Prentice-Hall of India Private Limited, 2004.

VII. I. J Nagrath and M. Gopal, Systems: Modeling and analysis, New Delhi: Tata McGraw-Hill, 1982.

VIII. M. Gopal, Digital Control and State Variable Methods, New Delhi: Tata McGraw-Hill, 2003.

IX. M. Gopal, Control Systems Principles and Design, New Delhi: Tata McGraw-Hill, 2002.

X. John Yen and Reza Langari, Fuzzy logic, Singapore: Pearson Education, 2003.

XI. Riza C.Berkan and Sheldon L.Trubatch, Fuzzy Systems Design Principles, New Delhi: Standard publishers' distributors, 2000.

APPENDIX

This is a command line code for PID +Fuzzy anticipation.

```
clear all;
close all;
%Create a new FIS with filename "flcdemo.fis"
sys=newfis('flcdemo');
%Define membership functions for the
%input variable "ERROR"
sys=addvar(sys,'input','ERROR',[-1 1]);
sys=addmf(sys,'input',1,'N','trimf',[-2 -1 0]);
sys=addmf(sys,'input',1,'Z','trimf',[-1 0 1]);
sys=addmf(sys,'input',1,'P','trimf',[0 1 2]);
%Define membership functions for the
%input variable "Ui"
sys=addvar(sys,'input','Ui',[-1 1]);
sys=addmf(sys,'input',2,'N','trimf',[-2 -1 0]);
sys=addmf(sys,'input',2,'Z','trimf',[-1 0 1]);
sys=addmf(sys,'input',2,'P','trimf',[0 1 2]);
%Define membership functions for the
%output variable "Ua"
sys=addvar(sys,'output','Ua',[-1 1]);
sys=addmf(sys,'output',1,'N','trimf',[-2 -1 0]);
sys=addmf(sys,'output',1,'Z','trimf',[-1 0 1]);
sys=addmf(sys,'output',1,'P','trimf',[0 1 2]);
%Define fuzzy rules
rule=[1 1 2 1 1
      1 2 2 1 1
      1 3 1 1 1
      2 1 3 1 1
      2 2 2 1 1
```

```
          2 3 1 1 1
          3 1 3 1 1
          3 2 2 1 1
          3 3 2 1 1];
  sys=addrule(sys ,rule);
  %Define gain constants
  GUi=.350;
  GE=1;
  GO=4.2;
  %Define reference signal
   for k=2:700
if(k>2)
x(k)=.9;
else
x(k)=0;
end
%COMMAND LINE SIMULATION
y(2)=0;
e(k)=x(k)-y(k);
Ui(1)=0;e(1)=0;
Ui(k)=Ui(k-1)+.2185*e(k-1);
Upd(1)=0;
Upd(k)=3.759*e(k)-3.759*e(k-1)+.187*Upd(k-1);
Upid(1)=0;
Upid(k)=Upd(k)+Ui(k)+2.01*e(k);
if(Upid<-1&Upid>1)
UF(k)=0;
else
E(k)=e(k)*GE;
UI(k)=Ui(k)*GUi;
Uf(k)=evalfis([E(k) UI(k)],sys);
UF(k)=Uf(k)*GO;
end
Ux(k)=Upid(k)+UF(k);
```

```
Ua(1)=0;
Ua(k)=0.66667*Ua(k-1)+.33*Ux(k);
  if(Ua(k)<-1)
      Ub(k)=-1;
  end
  if(Ua(k)>1)
      Ub(k)=1;
  else Ub(k)=Ua(k);
  end
  Ub(1)=0;
      y(k+1)=.909*y(k)+.0909*Ub(k);

end
plot(y);
% The fuzzy system created entirely from command line may be embedded
% directly into simulink to test it out in a simulation environment.
```