SPEED CONTROL OF DC MOTOR USING PI,PD AND PID CONTROLLER USING MATLAB

A
Report Submitted
In Partial Fulfilment of the Requirements
for the Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRICAL ENGINEERING

By

ANURAG KUMAR	**(1130433009)**
ANURAG TIWARI	**(1130433010)**
ISHAN DUBEY	**(1130433023)**
PRADIP KUMAR GAUTAM	**(1130433039)**
RISHABH KUMAR SINGH	**(1130433049)**

Under the supervision of

Mr. Shashikant

Senior lecturer

School of Engineering

BABU BANARASI DAS UNIVERSITY, LUCKNOW

May,2017

CERTIFICATE

It is certified that the work contained in this Project entitled **"Speed control of DC motor using PI, PD and PID Controller using MATLAB"** by Anurag kumar (1130433009), Anurag Tiwari (1130433010), Rishabh kumar singh (1130433049), Ishan dubey (1130433023), Pradip kumar gautam (1130433039) for the award of **Bachelor of Technology** from Babu Banarasi Das University has been carried out under my supervision.

Mr. Shashikant

Senior Lecturer
Department of Electrical Engineering
School of Engineering
Babu Banarasi Das University
Lucknow (U.P)

Mr. V. K Maurya

Associate Professor & Incharge
Department of Electrical Engineering
School of Engineering
Babu Banarasi Das University
Lucknow(U.P)

ABSTRACT

This project is a simulation and experimental investigation into the development of PID controller using MATLAB/SIMULINK software. The simulation development of the PID controller with the mathematical model of DC motor is done using Ziegler–Nichols method and trial and error method. The PID parameter is to be tested with an actual motor also with the PID controller in MATLAB/SIMULINK software. In order to implement the PID controller from the software to the actual DC motor data acquisition is used. From the simulation and the experiment, the result performance of the PID controller is compared in term of response and the assessment is presented.

ACKNOWLEDGEMENT

Whenever a module of work is completed successfully, a source of inspiration and guidance is always there for the student. I, hereby take the opportunity to thank those entire people who helped me in many different ways.

First and foremost, I am grateful to my thesis guide **Mr. Shashikant, Senior Lecturer, Department of Electrical Engineering, Babu Banarasi Das University**, for showing faith in my capability and providing able guidance and his generosity and advice extended to me throughout my thesis.

Last, but not least I would like to thank my entire faculty and my friends for helping me in all measure of life and for their kind cooperation and moral support.

ANURAG KUMAR(1130433009)

ANURAG TIWARI(1130433010)

ISHAN DUBEY(1130433021)

PRADIP KUMAR GAUTAM(1130433039)

RISHABH KUMAR SINGH(1130433049)

LIST OF FIGURES

Figure 1.6.1: Step response without any controller 3

Figure 1.6.2: Step response with P controller,Kp=10,Ki=0,Kd=0 4

Figure 1.6.3: Step response with P controller Kp=100,Ki=0,Kd=0 4

Figure 1.7.1: Step response with P:I controller,Kp=200,Ki=100,Kd=0 5

Figure 1.7.2: Step response with P:I controller Kp=200,Ki=200,Kd=10 6

Figure1.8: Step response with PID controller,Kp=200,Ki=200,Kd=10 7

Figure 2.2: Ziegler:nichols PID controller tuning method 10

Figure 2.3: Cohen coon PID tuning method 11

Figure 2.5.1: MATLAB:simulink diagram to show the effect of P control or first order plant 13

Figure 2.5.1(a): Output of the closed loop system only P control,Kp=10 14

Figure 2.5.1(b): Error of the closed loop system with only P control,Kp=10 14

Figure 2.5.1(c): Output control of the closed loop system with only P control,Kp=100 15

Figure 2.5.1(d): Error of closed loop system with only P control,Kp=100 15

Figure 2.5.2: MATLAB:simulink diagram to show the effect of P controller on second order plant 16

Figure 2.5.1(a): Output of the closed loop systemwith only P control,Kp=5 16

Figure 2.5.1(b): Error of the control loop system with only P control,Kp=5 17

Figure 2.5.1(c): Output of the closed loop system with only P control,Kp=10 18

Figure 2.5.1(d): Error of the closed loop system with only P control,Kp=10 18

Figure 2.5.1(e): Output of the closed loop system with only P control,Kp=100 19

Figure 2.5.1(f): Error of the control loop system with only P controller,Kp=100 19

Figure 2.5.2(a): Output of the closed loop system(first order) with P:D control,Kp=10,Kd=10 22

Figure 2.5.2(b): Error of the closed loop system (first order) with P:D control,Kp=10,Kd=10 22

Figure 2.5.2(c): Output of the closed loop system(second order) with P:D control,Kd=10 23

Figure 2.5.2(d): Error of the closed loop system(second order) with P:D control,Kd=10 23

Figure 2.5.2(e): MATLAB:simulink diagram to show the effect of P:I control Or first and second order plants 24

Figure 2.5.3(a): Output of the closed loop system(first order)with P:I control Kp=10,Ki=20 25

Figure 2.5.3(b): Error of the closed loop system(first order) with P:I control, Kp=10,Ki=20 25

Figure 2.5.3(c): Output of the closed loop system(second order) with control, Kp=10,Ki=20 26

Figure 2.5.3(d): Error of the closed loop system(second loop)with control, Kp=10,Ki=20 26

Figure 2.5.4: MATLAB:simulink diagram to show the effect of PID control on fiest and second order plant 27

Figure2.5.4(a): Output of the closed loop system(first order) with control, Kp=30,Ki=20,Kd=10 28

Figure2.5.4(b): Error of the closed loop system(first order)with control,Kp=30, Ki=20,Kd=10 28

Figure2.5.4(c): Output of the closed loop system(second order) with PID control, Kp=30,Ki=20,Kd=10 29

Figure2.5.4(d): Error of the closed loop system(second order) with PID control, Kp=30,Ki=20,Kd=10 29

Figure2.5.4(c): Output of the closed loop system(second order)with PID control tuning,Kp=104,Ki=106,Kd=24 30

Figure 2.6.1: Stairstep response of open loop system, sampling period Ts=0.05second 31

Figure2.6.2: Stairstep response of closed loop system with PID, sampling period Ts=0.05 seconds 32

Figure2.6.3: Stairstep response of closed loop system with PID after pole placement,sampling period Ts=0.05 seconds 32

Figure 2.6.4: Stairstep response of open loop system, sampling period Ts=4 seconds 33

Figure 3: The Block Diagram of the DC Motor Speed Control Loop 35

Figure 3.4: The Schematic of the DC Motor 36

Figure 3.6.1: The Block Diagram of the System with Proportional Controller 37

Figure 3.6.2: The MATLAB Result, Kp=100 38

Figure 3.6.3: The MATLAB Result for Ki=1, Kd=1, Kp=100 39

Figure 3.6.4: The MATLAB Result for Ki=200, Kd=1, Kp=100 40

Figure 3.6.5: The MATLAB Result for Ki=200, Kd=10, Kp=100 40

Figure 4.1.1: DC Motor Position Control 45

Figure 4.1.2: The Block Diagram of the DC Motor Position Control Loop 46

Figure 4.2.1: The Step Response of the DC Motor for Kp=1 48

Figure 4.2.2: The Step Response of the DC Motor for Kp=250 49

Figure 4.2.3: The Step Response of the DC Motor for Kp=3000 49

Figure 4.2.4:The Step Response of the DC Motor for Kp=100, Ki=200, Kp=10 50

Figure 4.3.1:The Step Response of the DC Motor with ZOH 51

Figure 4.3.2:The Root locus of the Compensated System 51

Figure 4.3.3:The RHS of the Root Locus 52

Figure 4.3.4: The LHS of the Root Locus 52

Figure 4.3.5:After the Addition of the Pole at :0.983 53

Figure 4.3.6: The Step Response of the System for gain=0.0028 53

Figure 4.3.7:The Block Diagram of Both Speed and Position Control of DC Motor 54

Figure 5.4:Concept of the Feedback Loop to Control the Dynamic Behavior of the reference 57

Figure 5.5: Closed:loop controller or feedback controller 58

Figure 6.2.1: A square wave showing the definition of $Y_{min,}$ Y_{max} and D 61

Figure 6.2.2: PWM Pulse generator from comparing sinewave and sawtooth 62

Figure 7.1.1: MATLAB default command windows 64

Figure 7.1.2: Simulink running a simulation of a thermostat controller heating system 66

Figure 7.2: Block diagram of the system 66

Figure 7.3.1: DC motor 67

Figure 7.3.2: Personal Computer 67

Figure 7.3.3:Microcontroller IC 68

Figure 7.3.4:ATmega8 IC 68

Figure 7.3.5:USB to TTL 69

Figure 7.3.6:Jumper wire 69

Figure 7.3.7:IC Base 28 pin 69

LIST OF TABLE

Table2.2: Ziegler-Nichols P-I-D control tuning method 10

Table2.3: Cohen-Coon P-I-D control tuning method,adjusting Kp,Ki and Kd 12

TABLE OF CONTENT

		Page No.
CERTIFICATE		**ii**
ABSTRACT		**iii**
ACKNOWLEDGEMENT		**iv**
LIST OF FIGURES		**v**
LIST OF TABLES		**ix**
CHAPTER 1: INTRODUCTION		**1-7**
1.1	Aim of reaction	1
1.2	P controller	1
1.3	PI controller	2
1.4	PD controller	2
1.5	PID controller	2
1.6	Simulation and results to find the constraints on loop tuning	3
1.7	Conclusions	5
1.8	Conclusions	6
1.9	Conclusions	7
CHAPTER 2 : LOOP TUNING		**8-30**
2.1	Manual tuning method	9
2.2	Ziegler-Nichols tuning method	9
	2.2.1 Advantages	10
	2.2.2 Disadvantages	11

2.3 Cohen-Coon tuning method 11

2.4 Comparison of the two method 12

2.5 Transient response of P,P-D,P-I and P-I-D controls 13

2.5.1 Transient response of P controller 13

2.5.2 Transient response of P-D controller 21

2.5.3 Transient response of P-I controller 24

2.5.4 Transient response of P-I-D controller 27

2.6 Digital P-I-D control 30

CHAPTER 3 : P-I-D CONTROLLER DESIGN FOR CONTROLLING DC MOTOR SPEED IN THE PROJECT **34-41**

3.1 Why do we need to control the speed of DC motor 34

3.2 Why to choose P-I-D as controller 34

3.3 P-I-D parameter 36

3.4 The design requirement of the system 36

3.5 The parameter of the DC motor 37

3.6 The open loop transfer function of the DC motor 37

3.7 S-domain to Z-domain with ZOH 41

CHAPTER 4 : P-I-D CONTROLLER DESIGN FOR CONTROLLING DC MOTOR POSITION IN THE PROJECT **46-54**

4.1 Why we need to position the DC motor 46

4.2 The design requirement of the system 48

4.3 The transfer function of the DC motor with zero order hold 50

CHAPTER 5 : BACKGROND OF PROJECT **55-58**

5.1 General 55

5.2 Problem statement 55

5.3 Permanent magnet DC motor 56

5.4 Control theory 57

5.5 Closed loop transfer function 58

CHAPTER 6 : PID CONTROLLER **60-62**

6.1 General 60

6.2 Pulse Width Modulation 61

CHAPTER 7 : MATLAB AND SIMULINK **63-67**

7.1 General 63

7.2 System Description 66

7.3 Hardware 67

CHAPTER 8 : MATLAB CODE **70-78**

8.1 Code for speed control of DC motor 70

CONCLUSION **79**

REFERENCE **80**

CHAPTER 1

1.Introduction

It is mainly about P, P-D, P-I and P-I-D controllers, their digital versus continuous time realizations and their characteristics including sampling period effects on the response of digital ones. Moreover, position and velocity form of P-I-D control was modeled on the‘Gate’ project.

Apart from these topics, P-I-D tuning methods such as manual tuning, Ziegler-Nichols tuning, Cohen-Coon tuning and MATLAB tuning method are discussed. Transient performances of P, P-D, P-I and P-I-D controllers were explained in detail. Modeling a discrete time P-I-D controller to control a continuous time plant was explained over a MATLAB code introducing the effect of sampling time and the choice of s*-domain to z-domain transformation method on MATLAB. It was explained how to remove poles that cause instability in discrete time by adding a new pole. Finally, it was shown how one could control the speed and position of the vehicle using discrete time P-I-D controller on the ‘Gate’ project.

1.2 Aim of the Recitation

Aim of the recitation was to introduce the concept of Discrete Time P-I-D controllers and how they can be implemented on real life projects.

It was first intended to explain the usage of continuous time P-I-D controllers. In the first part of the recitation, it was aimed to show the how P, P-I, P-I-D controllers change the steady state response of the closed loop systems. Moreover, the methods to tune P-I-D controllers were introduced. It was meant to show that how hard it could get to properly tune a P-I-D controller. Secondly, it was intended to show how P, P-D, P-I, and P-I-D controllers affect the transient response of the closed loop system. It was meant to show how one can gain a feature but lose the other. Thirdly, it was intended to show how one should estimate the dynamics of the continuous time plant and use proper sampling time for discrete time P-I-D controller. It was also meant to show how changing transformation method may cause different pole locations on the z-plane. Lastly, it was intended to show how one could control the velocity and the position of the vehicle of the ‘Gate’ project by implementing a discrete time P-I-D controller in that project

1.3 P Controller

P controller is mostly used in first order processes with single energy storage to stabilize the unstable process. The main usage of the P controller is to decrease the steady state error of the system. As the proportional gain factor K increases, the steady state error of the system decreases. However, despite the reduction, P control can never manage to eliminate the steady state error of the system. As we increase the proportional gain, it provides smaller amplitude and phase margin, faster dynamics satisfying wider frequency band and larger sensitivity to the noise. We can use this controller only when our system is tolerable to a constant steady state error. In addition, it can be easily concluded that applying P controller decreases the rise time and after a certain value of reduction on the steady state error, increasing K only leads to overshoot of the system response. P control also causes oscillation if sufficiently aggressive in the presence of lags and/or dead time. The more lags (higher order), the more problem it leads. Plus, it directly amplifies process noise.

1.3 P-I Controller

P-I controller is mainly used to eliminate the steady state error resulting from P controller. However, in terms of the speed of the response and overall stability of the system, it has a negative impact. This controller is mostly used in areas where speed of the system is not an issue. Since P-I controller has no ability to predict the future errors of the system it cannot decrease the rise time and eliminate the oscillations. If applied, any amount of I guarantees set point overshoot.

1.4 P-D Controller

The aim of using P-D controller is to increase the stability of the system by improving control since it has an ability to predict the future error of the system response. In order to avoid effects of the sudden change in the value of the error signal, the derivative is taken from the output response of the system variable instead of the error signal. Therefore, D mode is designed to be proportional to the change of the output variable to prevent the sudden changes occurring in the control output resulting from sudden changes in the error signal. In addition D directly amplifies process noise therefore D-only control is not used.

1.5 P-I-D Controller

P-I-D controller has the optimum control dynamics including zero steady state error, fast response (short rise time), no oscillations and higher stability. The necessity of using a derivative gain component in addition to the PI controller is to eliminate the overshoot and the oscillations occurring in the output response of the system. One of the main advantages of the P-I-D controller is that it can be used with higher order processes including more than single energy storage.

In order to observe the basic impacts, described above, of the proportional, integrative and derivative gain to the system response, see the simulations below prepared on MATLAB in continuous time with a transfer function and unit step input. The results will lead to tuning methods

1.6 Simulations and Results to Find the Constraints on Loop Tuning

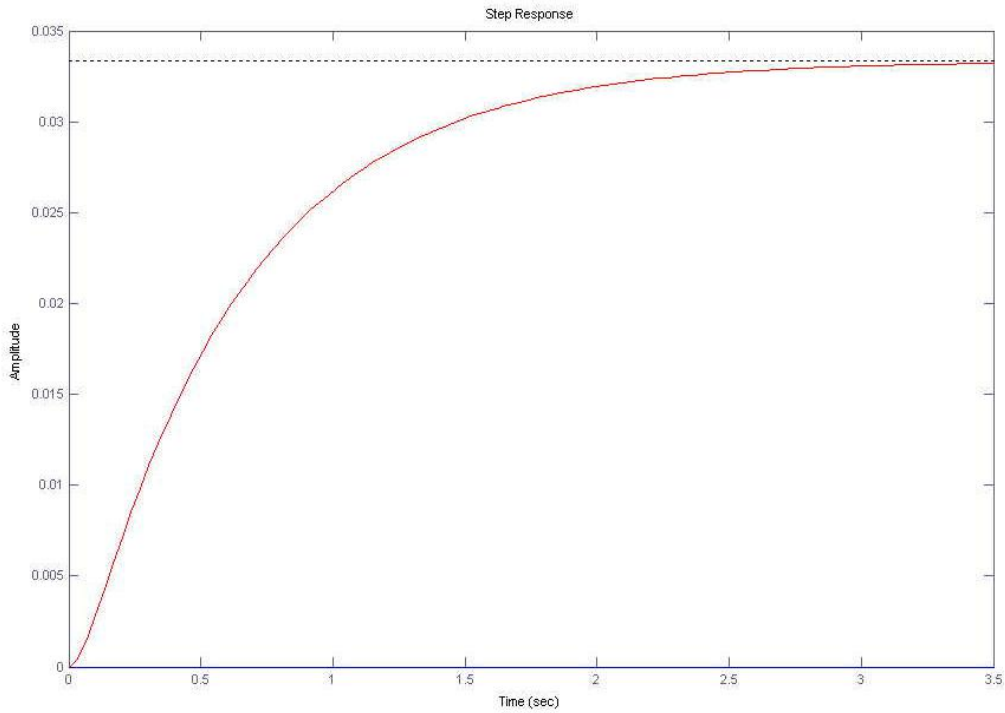

Figure 1.6.1: Step response without any controller

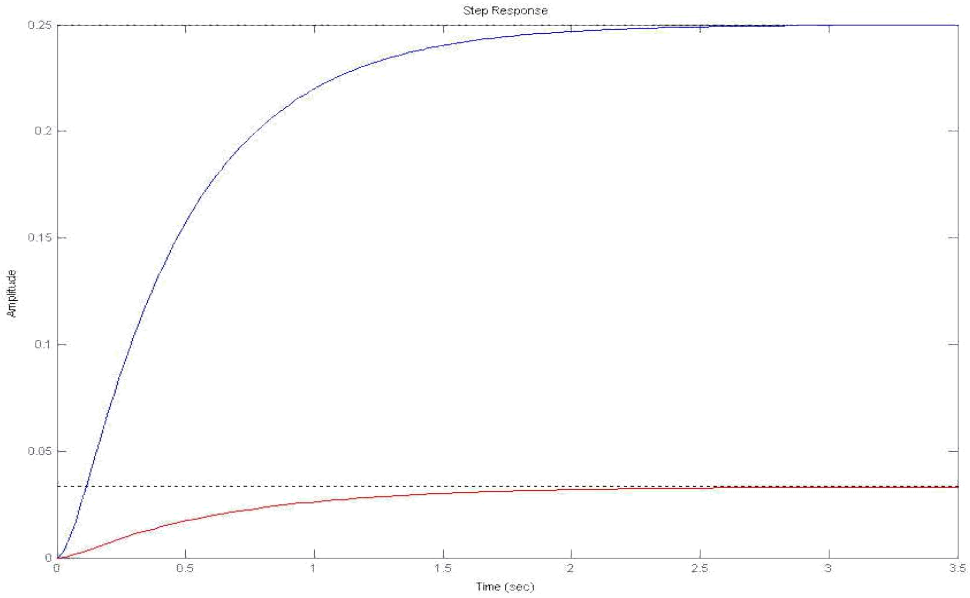

Figure 1.6.2: Step response with P controller, Kp = 10,Ki = 0, Kd = 0

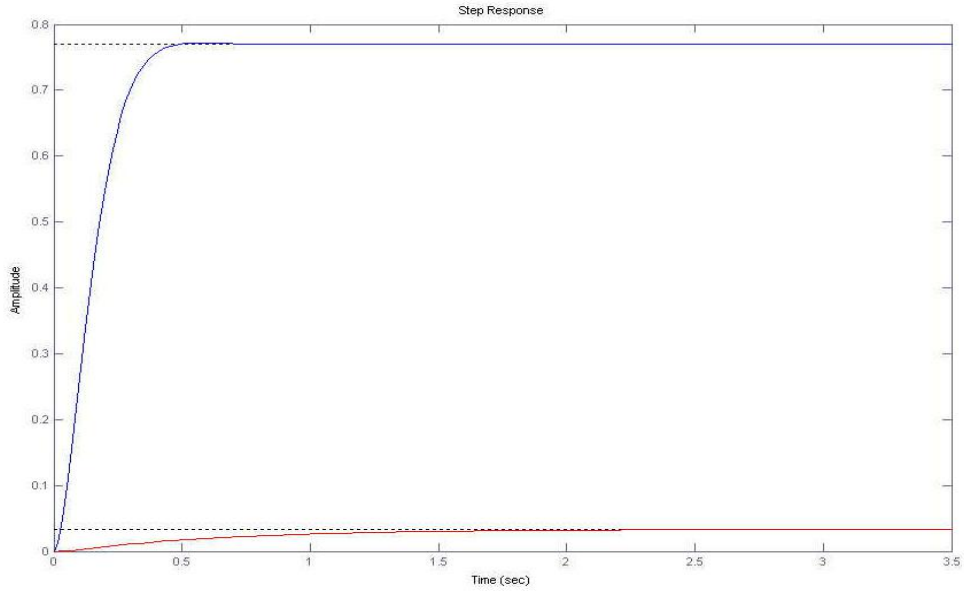

Figure 1.6.3: Step response with P controller, Kp = 100, Ki = 0, Kd = 0

1.7 Conclusion

1. Increasing K_p will reduce the steady state error.
2. After certain limit increasing K_p only causes overshoot.
3. Increasing K_p reduces the rise time.

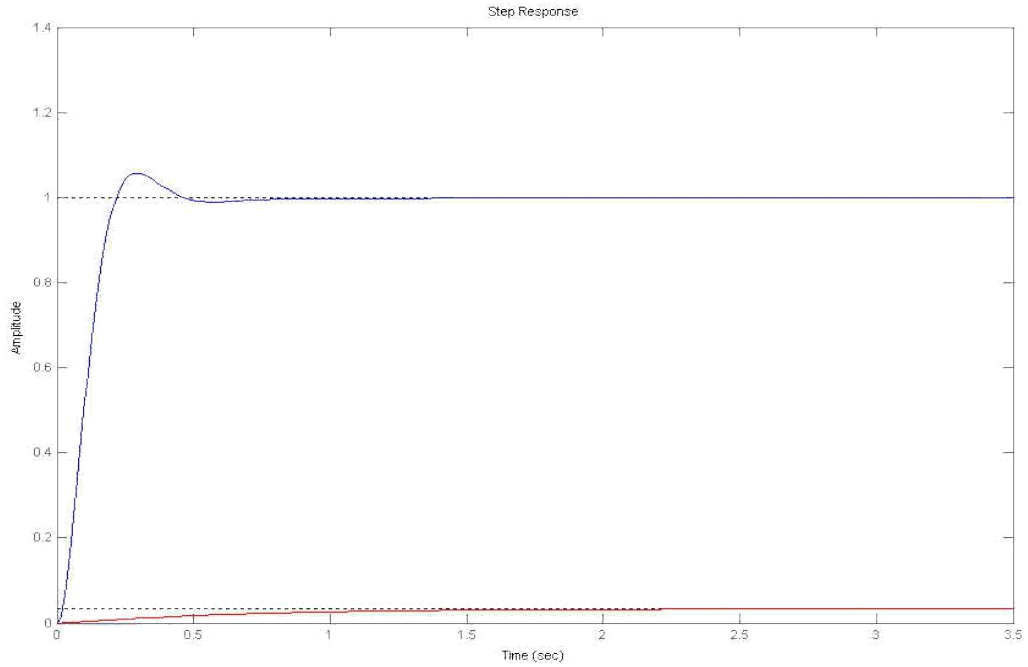

Figure 1.7.1: Step response with P-I controller, Kp = 200, Ki = 100, Kd = 0

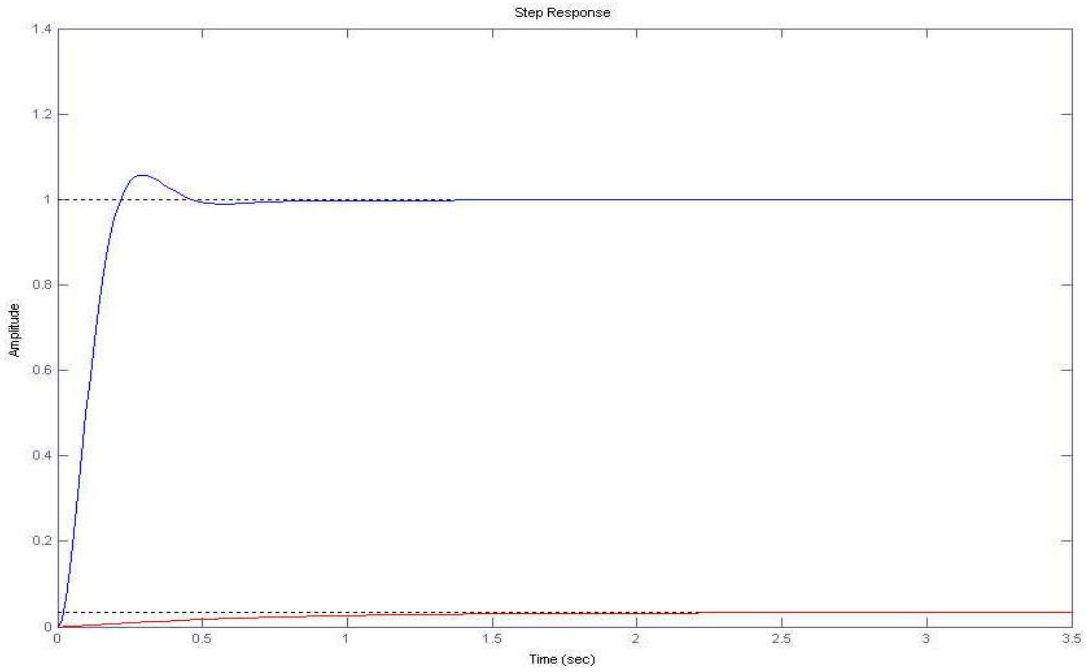

Figure 1.7.2: Step response with P-I controller, Kp = 200,Ki = 200, Kd = 0

1.8 Conclusions

1. Integral control eliminates the steady state error.
2. After certain limit, increasing K_i will only increase overshoot.
3. Increasing K_i reduces the rise time a little.

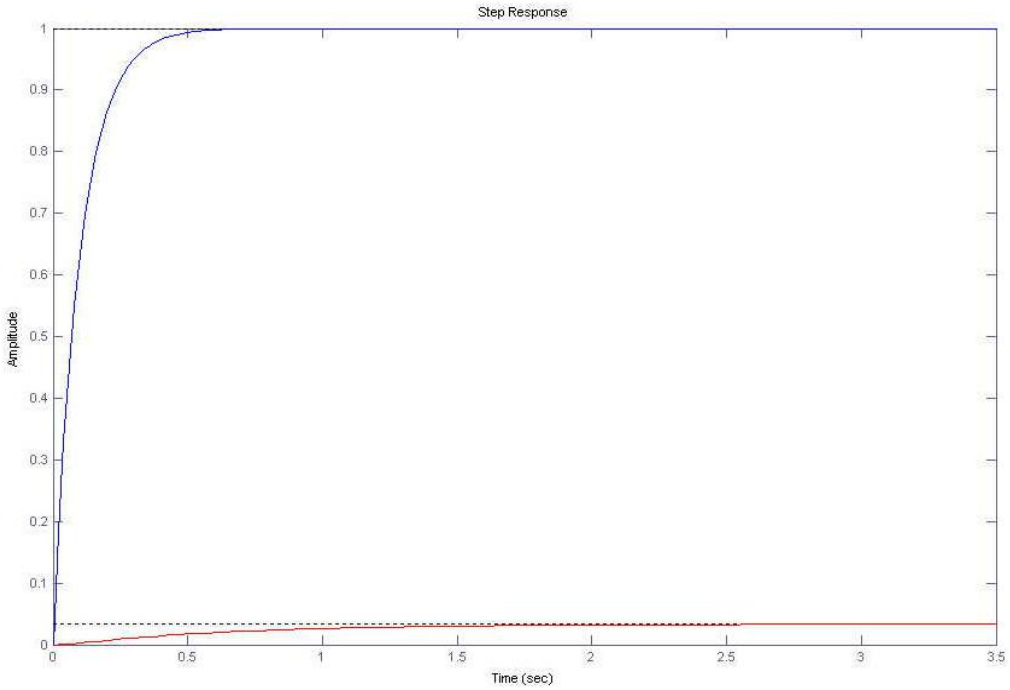

Figure 1.8: Step response with P-I-D controller,Kp = 200,Ki = 200 and Kd= 10

1.9 Conclusions

.

1. Increasing Kd decreases the overshoot.
2. Increasing Kd reduces the settling time.

CHAPTER 2

2. Loop Tuning

Tuning a control loop is arranging the control parameters to their optimum values in order to obtain desired control response. At this point, stability is the main necessity, but beyond that, different systems leads to different behaviors and requirements and these might not be compatible with each other. In principle, P-I-D tuning seems completely easy, consisting of only 3 parameters, however, in practice; it is a difficult problem because the complex criteria at the P-I-D limit should be satisfied. P-I-D tuning is mostly a heuristic concept but existence of many objectives to be met such as short transient, high stability makes this process harder. For example sometimes, systems might have nonlinearity problem which means that while the parameters works properly for full load conditions, they might not work as effective for no load conditions. Also, if the P-I-D parameters are chosen wrong, control process input might be unstable, with or without oscillation; output diverges until it reaches to saturation or mechanical breakage.

For a system to operate properly, the output should be stable, and the process should not oscillate in any condition of set point or disturbance. However, for some cases bounded oscillation condition as a marginal stability can be accepted.

As an optimum behavior, a process should satisfy the regulation and command breaking requirements. These two properties define how accurately a controlled variable reaches the desired values. The most important characteristics for command breaking are rise time and settling time. For some systems where overshoot is not acceptable, to achieve the optimum behavior requires eliminating the overshoot completely and minimizing the dissipated power in order to reach a new set point.

In today's control engineering world, P-I-D is used over 95% of the control loops. Actually if there is control, there is P-I-D, in analog or digital forms. In order to achieve optimum solutions K_p, K_i and K_d gains are arranged according to the system characteristics. There are many tuning methods, but most common methods are as follows:

1. Manual Tuning Method
2. Ziegler-Nichols Tuning Method
3. Cohen-Coon Tuning Method
4. PID Tuning Software Methods (ex. MATLAB)

2.1 Manual Tuning Method

Manual tuning is achieved by arranging the parameters according to the system response. Until the desired system response is obtained K_i, K_p and K_d are changed by observing system behavior.

Example (for no system oscillation): First lower the derivative and integral value to 0 and raise the proportional value 100. Then increase the integral value to 100 and slowly lower the integral value and observe the system's response. Since the system will be maintained around set point, change set point and verify if system corrects in an acceptable amount of time. If not acceptable or for a quick response, continue lowering the integral value. If the system begins to oscillate again, record the integral value and raise value to 100. After raising the integral value to 100, return to the proportional value and raise this value until oscillation ceases. Finally, lower the proportional value back to 100.0 and then lower the integral value slowly to a value that is 10% to 20% higher than the recorded value when oscillation started (recorded value times 1.1 or 1.2).

Although manual tuning method seems simple it requires a lot of time and experience

2.2 Ziegler-Nichols Method

More than six decades ago, P-I controllers were more widely used than P-I-D controllers. Despite the fact that P-I-D controller is faster and has no oscillation, it tends to be unstable in the condition of even small changes in the input set point or any disturbances to the process than P-I controllers. Ziegler-Nichols Method is one of the most effective methods that increase the usage of P-I-D controllers.

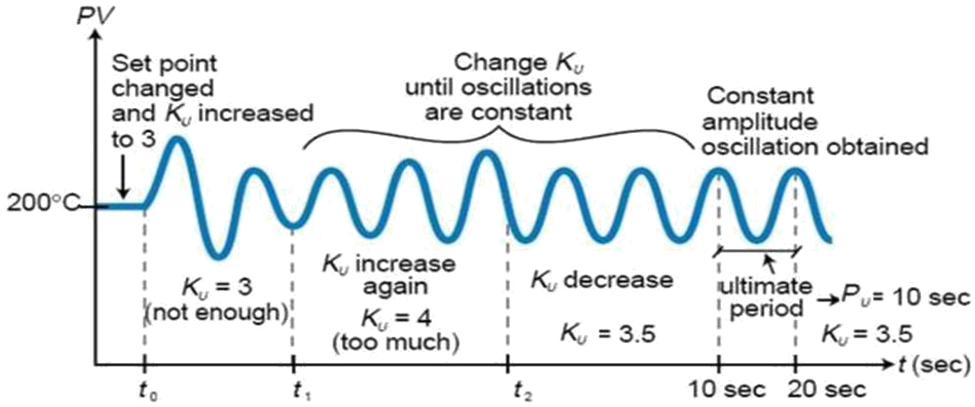

Figure 2.2: Ziegler-Nichols P-I-D controller tuning method

The logic comes from the neutral heuristic principle. Firstly, it is checked that whether the desired proportional control gain is positive or negative. For this, step input is manually increased a little, if the steady state output increases as well it is positive, otherwise; it is negative. Then, K_i and K_d are set to zero and only K_p value is increased until it creates a periodic oscillation at the output response. This critical K_p value is attained to be "ultimate gain", K_c and the period where the oscillation occurs is named as P_c "ultimate period". As a result, the whole process depends on two variables and the other control parameters are calculated according to the table in the Figure 9.

Ziegler-Nichols method giving K'values (loop times considered to be constant and equal to dT)			
Control type	Kp	Ki	Kd
P	0.50Kc	0	0
PI	0.45Kc	1.2KpdT/Pc	0
PID	0.60Kc	2KpdT/(8dT)	KpPc/(8dT)

Table2.2: Ziegler-Nichols P-I-D controller tuning method, adjusting Kp, Ki and Kd

2.2.1 Advantages

1. It is an easy experiment; only need to change the P controller.
2. Includes dynamics of whole process, which gives a more accurate picture of how the system is behaving

2.2.2 Disadvantages

1. Experiment can be time consuming.
2. It can venture into unstable regions while testing the P controller,which could Cause the system to become out of control.
3. For some cases it might result in aggressive gain and overshoot.

2.3 Cohen-Coon Tuning Method

This tuning method has been discovered almost after a decade than the Ziegler-Nichols method. Cohen-Coon tuning requires three parameters which are obtained from the reaction curve as in the Figure 2.3.

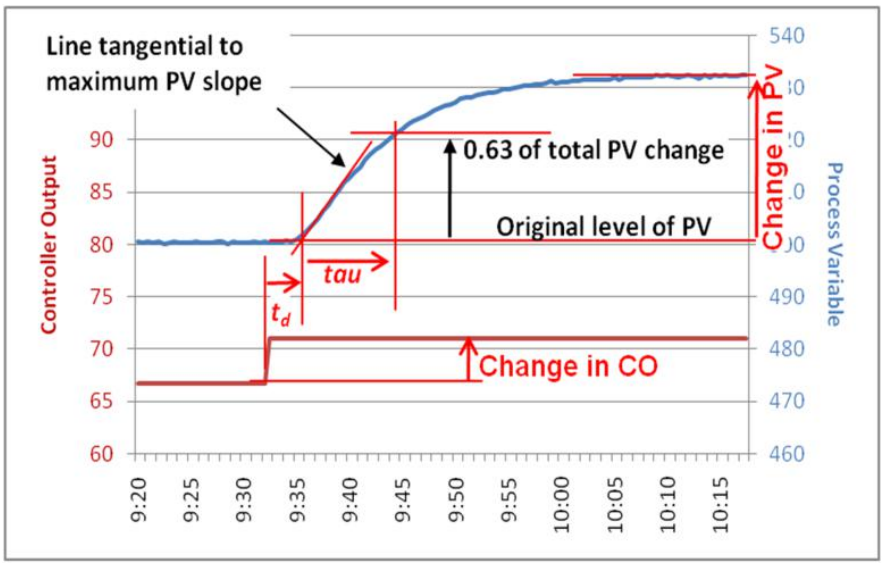

Figure 2.3: Cohen-Coon P-I-D Tuning Method

The controller is manually placed and after the process settled out a few percent of the change is made in the controller output (CO) and waited for the process variable (PV) to settle out at a new value. As observed from the graph, process gain (gp) is calculated as follow:

gp =(in %)

The maximum slope at the inflection point on the PV response curve is found and drawn a tangential line. t_d (dead time) is measured as taking the time difference between the change in CO and the intersection of the tangential line and the original PV level.As a final parameter τ(time constant) as the time difference between intersection at the end of thedead time and thePV reaching 63% of its total change. After converting the time

variables into the same units and applying couple of tests until to find similar result, these three variables are used to define new control parameters using the table in the Figure below.

	Controller gain	Integral time	Derivative Time
P Controller	$K_c = \frac{1.03}{G_p}\left(\frac{\tau}{T_d}+0.34\right)$		
P-I Controller	$K_c = \frac{0.9}{G_p}\left(\frac{\tau}{T_d}+0.092\right)$	$T_I = 0.27T_d\left(\frac{\tau-0.342T_d}{\tau+0.129T_d}\right)$	
P-D Controller	$K_c = \frac{1.24}{G_p}\left(\frac{\tau}{T_d}+0.129\right)$		$T_d = 0.27T_d\left(\frac{\tau-0.342T_d}{\tau+0.129T_d}\right)$
P-I-D Controller	$K_c = \frac{1.35}{G_p}\left(\frac{\tau}{T_d}+0.185\right)$	$T_I = 2.5T_d\left(\frac{\tau+0.185T_d}{\tau+0.611T_d}\right)$	$T_d = 0.31T_d\left(\frac{\tau}{\tau+0.185T_d}\right)$

Table 2.3: Cohen-Coon P-I-D Tuning Method, adjusting Kp, Ki and Kd

2.4 Comparison of the two methods

If we want to compare these two methods, Ziegler-Nichols can be used for any order of the systems, especially for the higher ones, while Cohen-Coon can only be used for first order systems. Therefore, Ziegler-Nichols tuning method is more widely used. However, for the first order systems Cohen-Coon is more flexible since as Ziegler-Nichols is only applicable when the dead time is less than of the time constant, Cohen-Coon is tolerable until of this value and it can be even extended. Therefore for systems having time delay this tuning method is more convenient. All in all, despite the fact that tuning a system seems easy to apply, in practice, it is really hard to analyze and pick a tuning method satisfying all system requirements. Using the logic of arranging the control parameters described above, some PID tuning software methods are developed which are easier to apply and saves time to get an optimum solution.

2.5 Transient Responses of P, P-D, P-I and P-I-D controllers

In this part, transient performances of P, P-D, P-I and P-I-D controllers are explained. Their steady state error performances are also discussed.

2.5.1Transient Response of P Controller

As a general rule, increasing proportional gain decreases the steady state error. However, the actual performance of P controller depends on the order of the plant.

If P controller is used to control a second order plant, it has following properties:

1. Increasing gain decreases rise time (Advantage)
2. Increasing gain increases percent overshoot and number of oscillations (Disadvantage)
3. Increasing gain decreases steady state error (Advantage)
4. Steady state is never zero if only-P type controller is used (Disadvantage)
5. In order to have zero steady state error gain should be infinity (Physically impossible)

The discussion above shows that only-P control is not enough to control second order plants. In fact, only-P control is usually used to control first order plants, because there are no natural oscillations in first order plants and P control is easy to implement. The following simulations were done on MATLAB-Simulink to illustrate the performance of P control on first and second order plants.

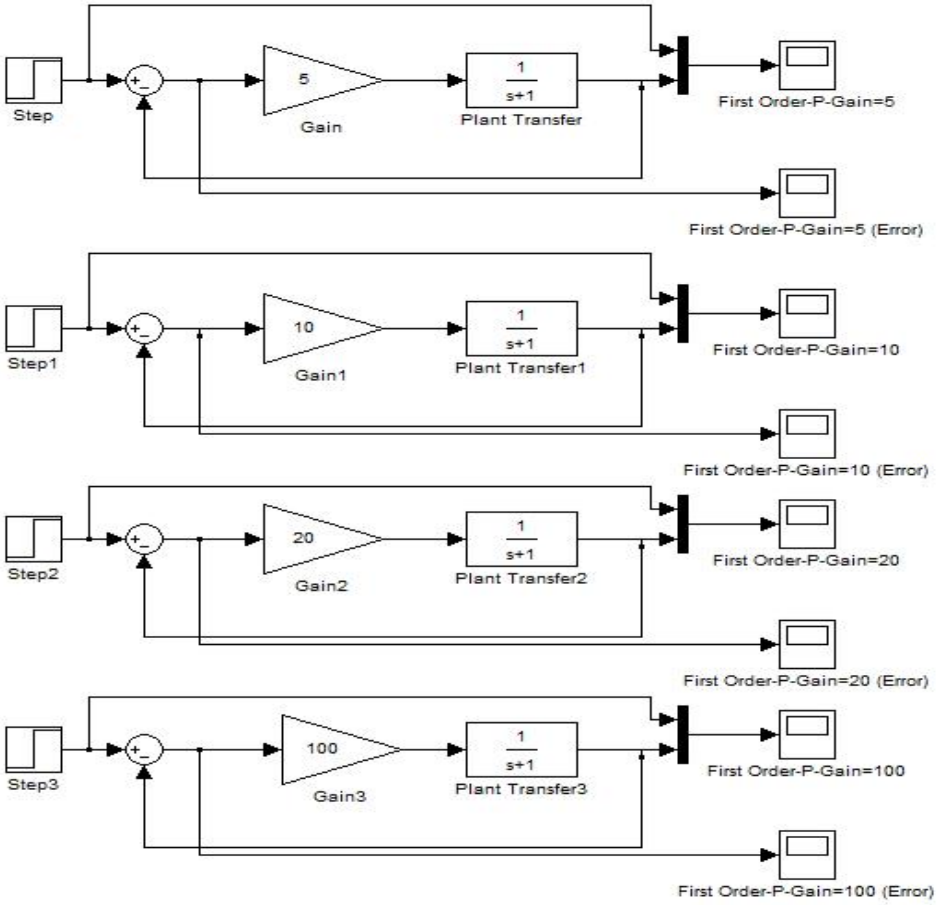

Figure 2.5.1: MATLAB-Simulink Diagram to show the effect of P control on first order Plant.

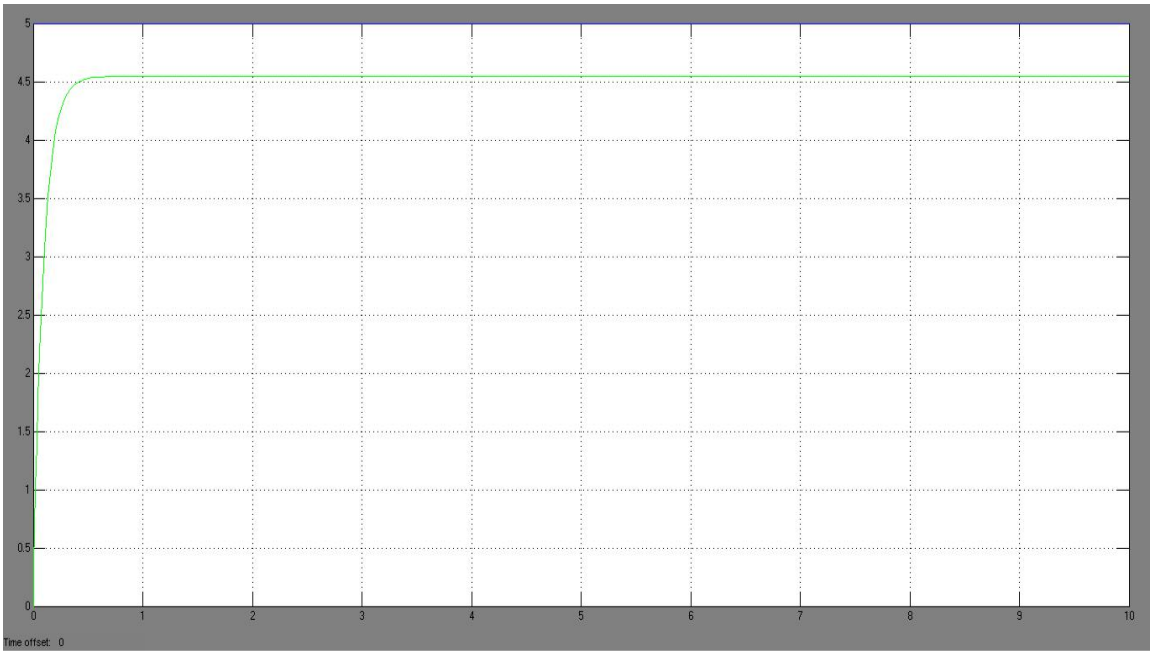

Figure 2.5.1(a): Output of the closed loop system with only P control,Kp=10

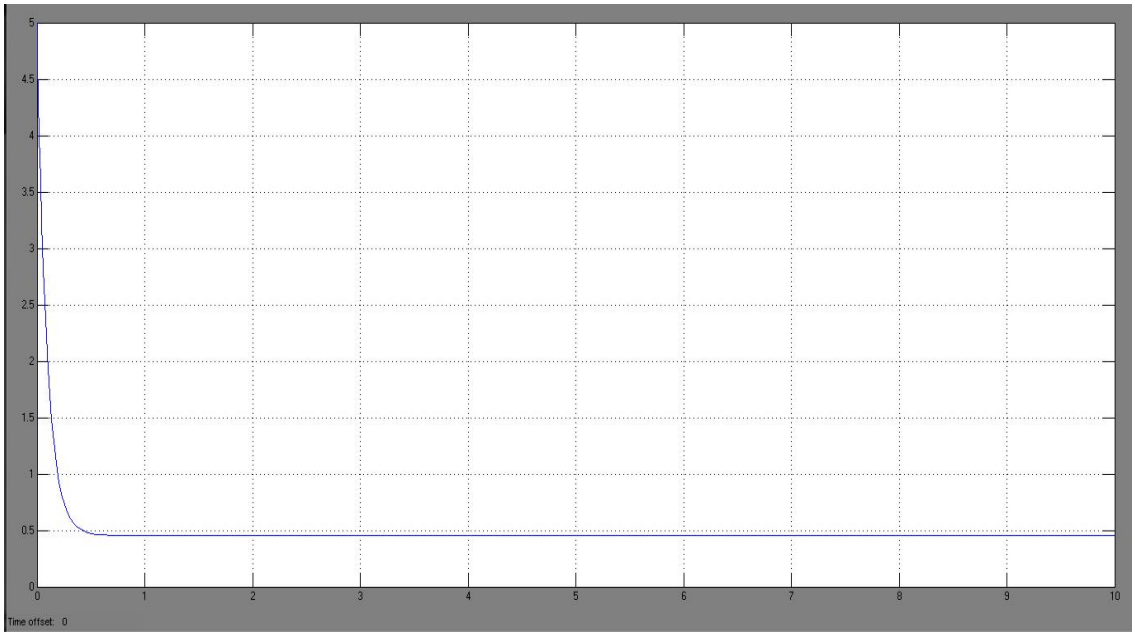

Figure 2.5.1(b): Error of the closed loop system with only P control, Kp=10

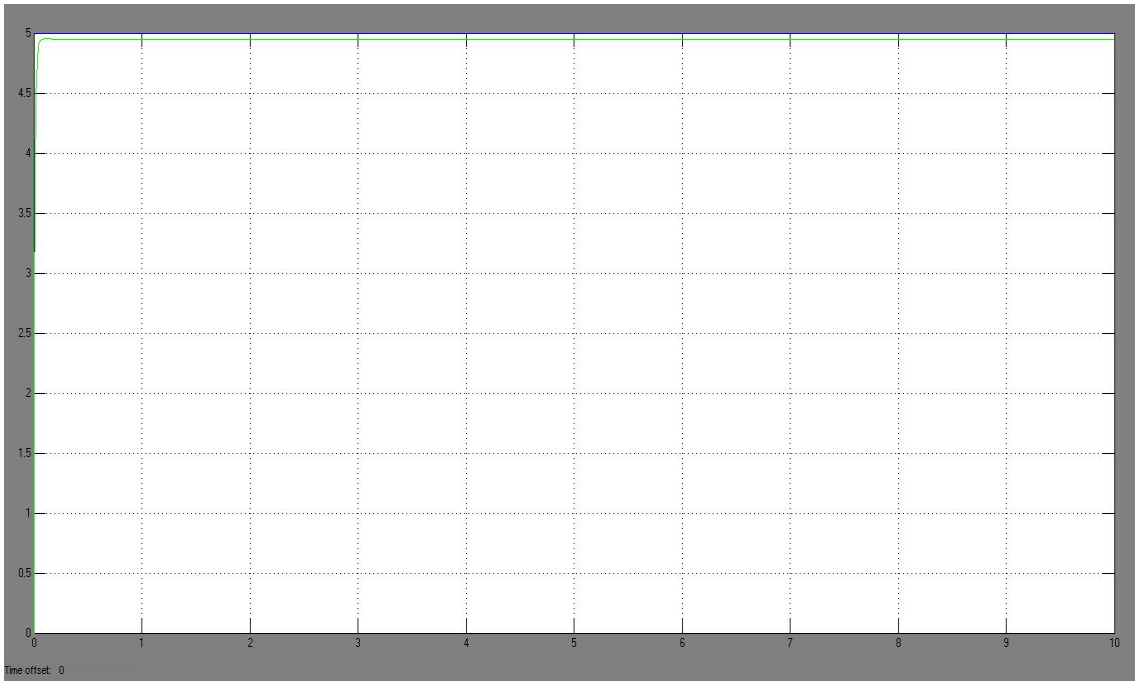

Figure 2.5.1(c): Output of the closed loop system with only P control, Kp=100

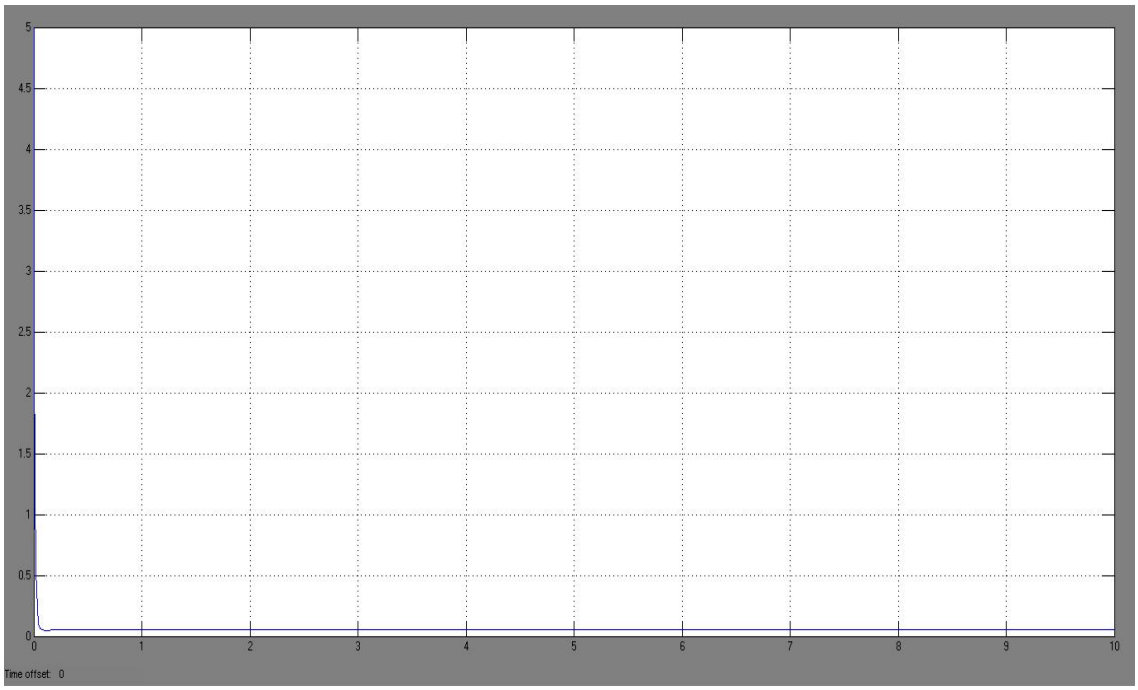

Figure 2.5.1(d): Error of the closed loop system with only P control, Kp=100

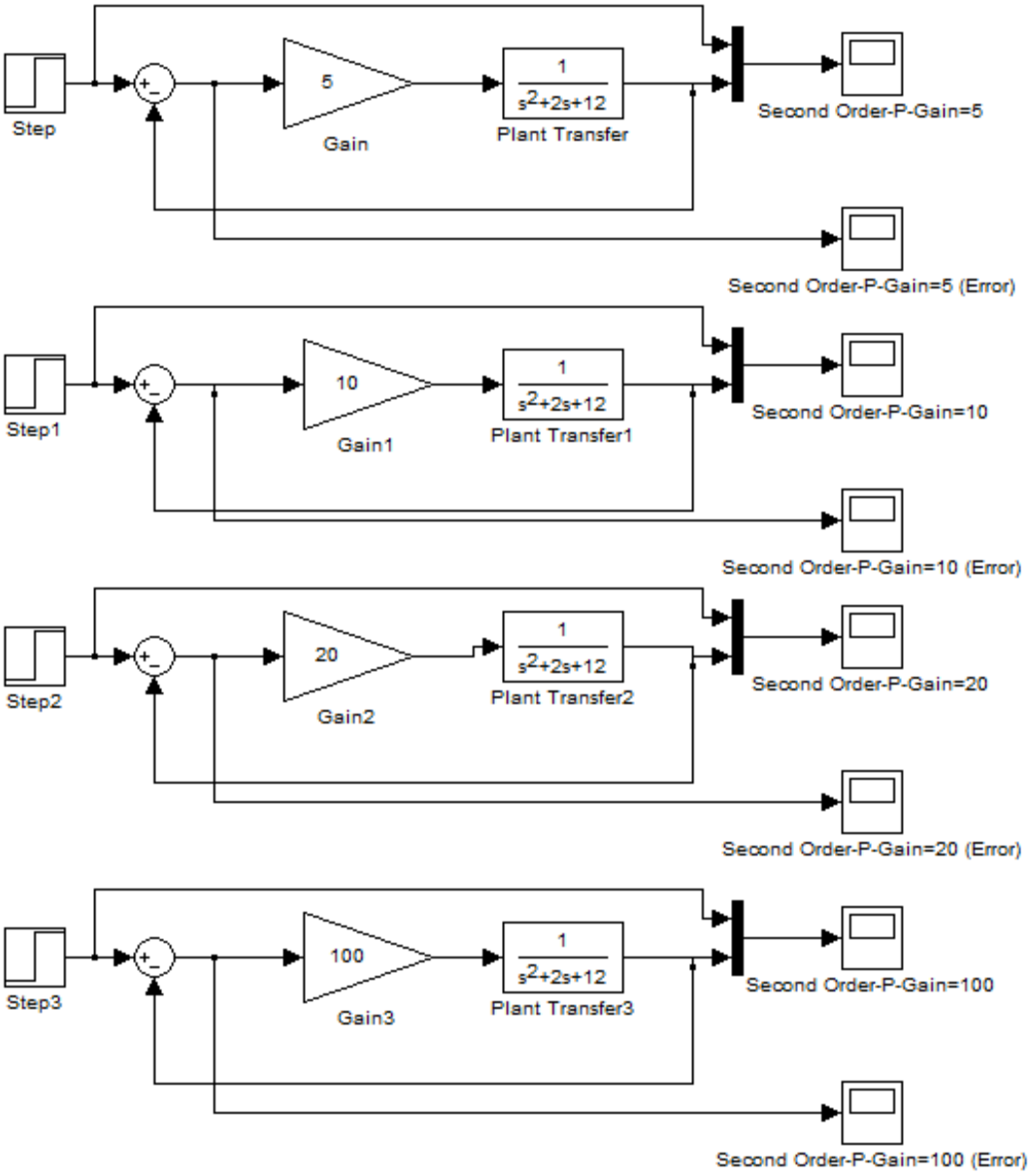

Figure 2.5.2: MATLAB-Simulink Diagram to show the effect of P control on second orderplant

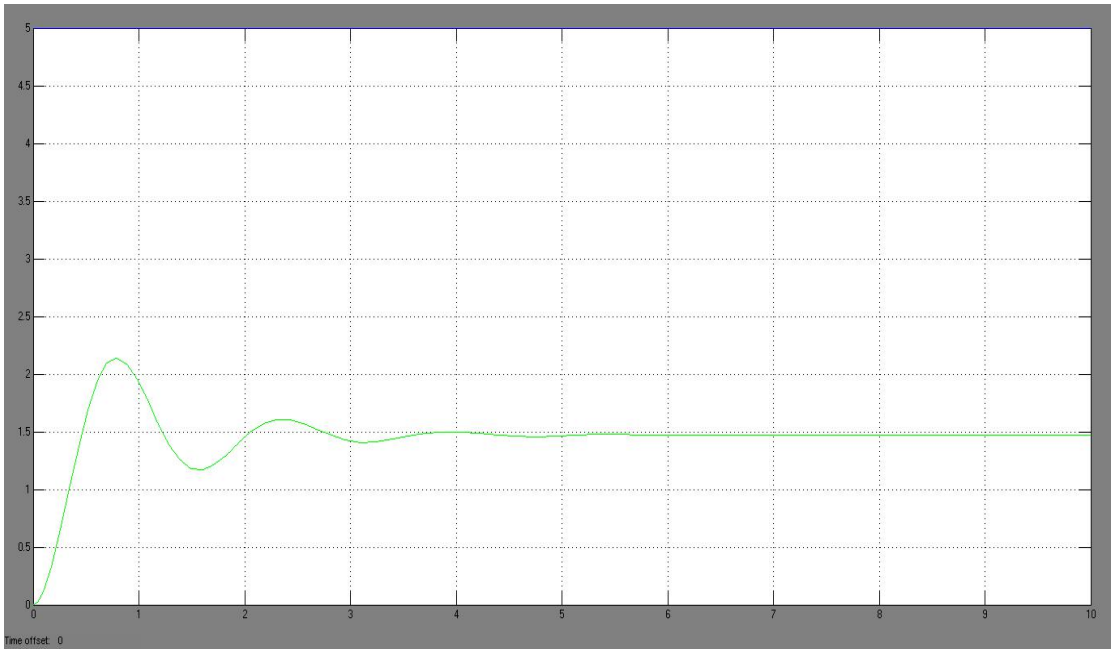

Figure 2.5.2(a): Output of the closed loop system with only P control, Kp=5

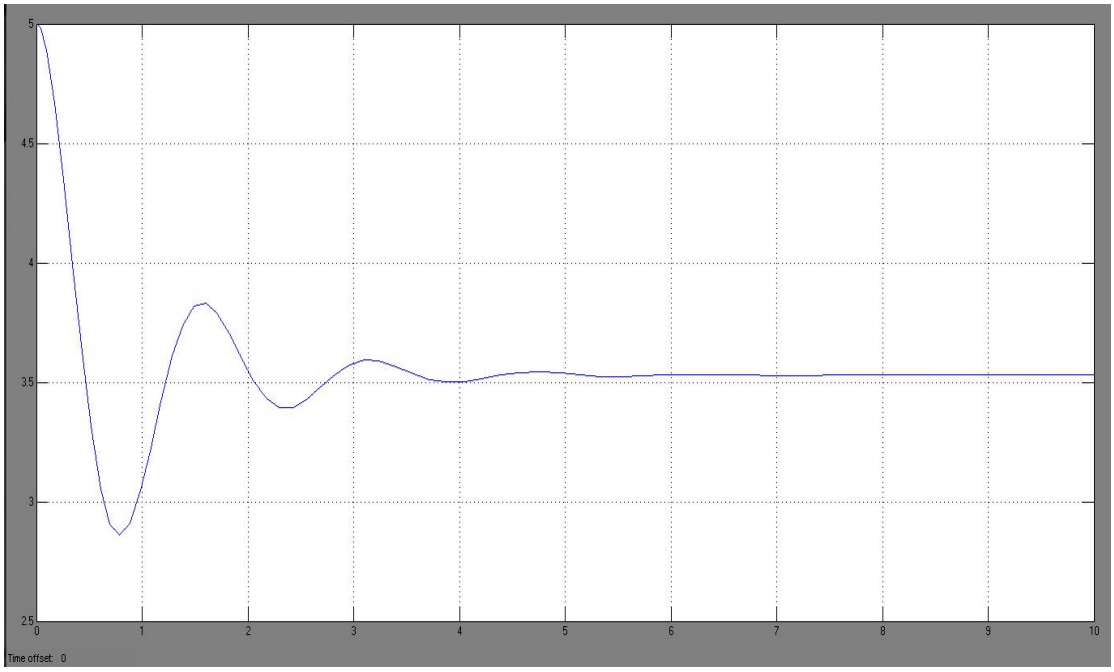

Figure 2.5.2(b): Error of the closed loop system with only P control,Kp=5

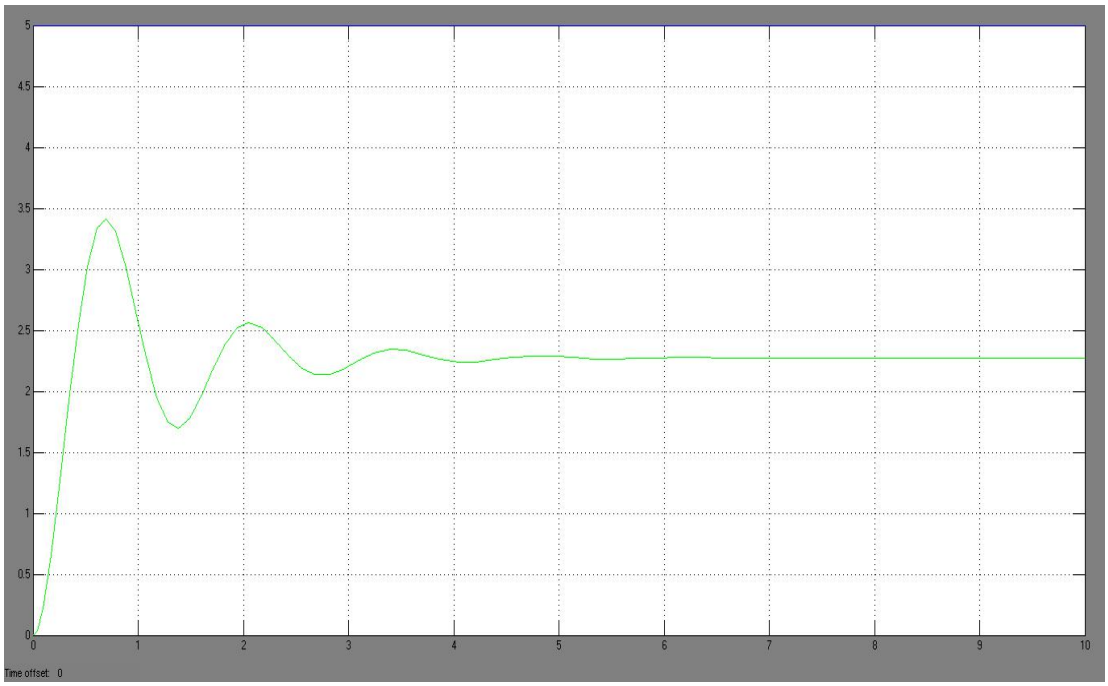

Figure 2.5.2(c): Output of the closed loop system with only P control, Kp=10

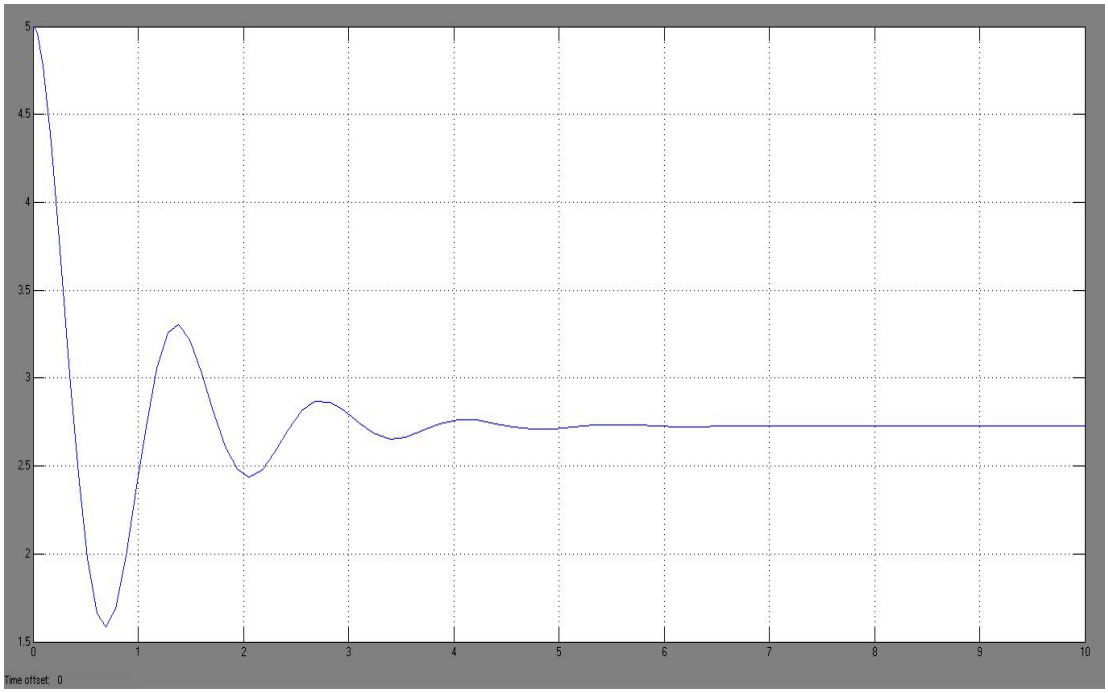

Figure 2.5.2(d): Error of the closed loop system withonly P control,Kp=10

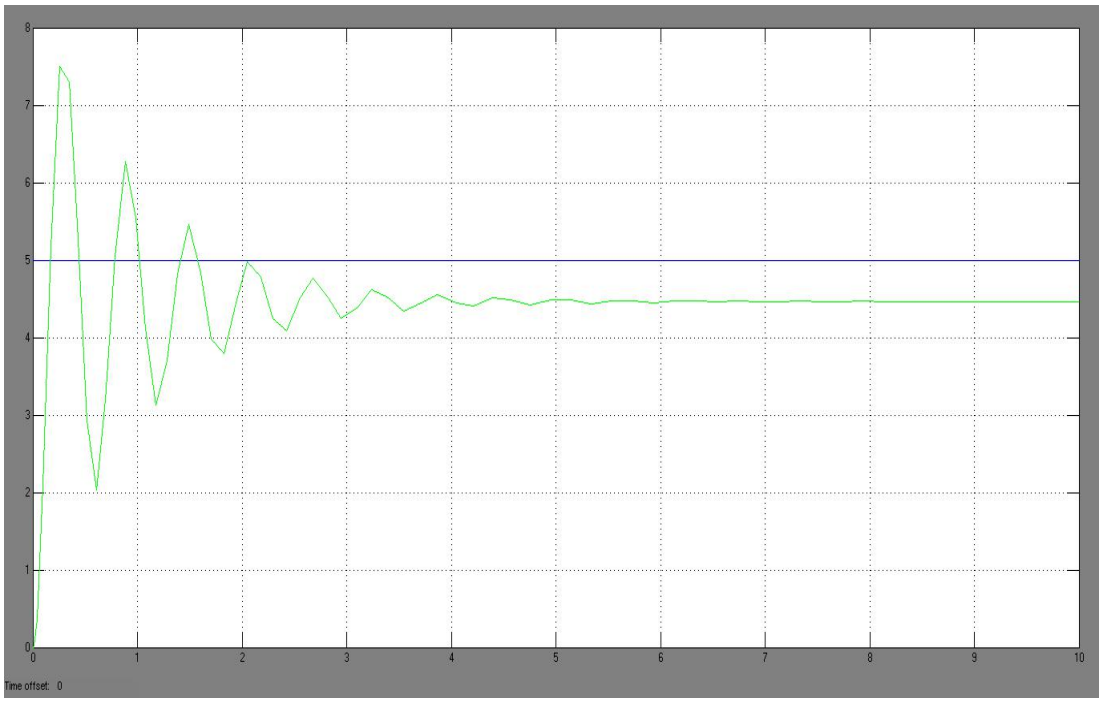

Figure 2.5.2(e): Output of the closed loop system with only P control,Kp=100

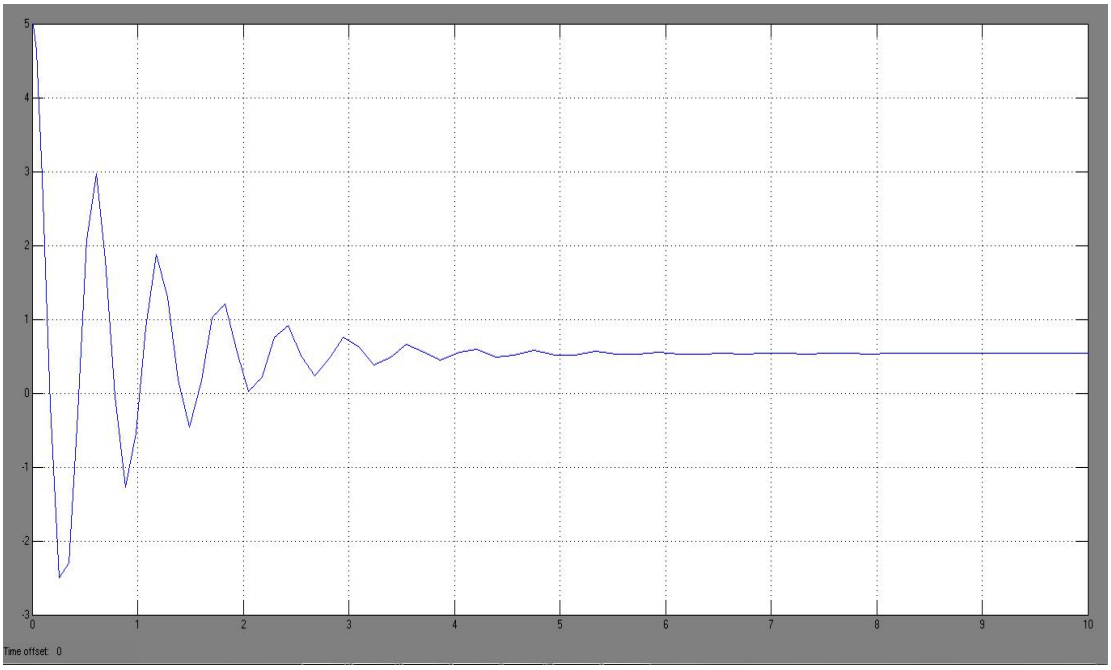

Figure 2.5.2(f): Error of the closed loop system with only P control, Kp=100

2.5.2.Transient Response of P-D Controller

Derivative action is usually used to improve transient response of the closed loop system. Only D control is not used because it amplifies high frequency noise which is never desired. Derivative action decreases rise time and oscillations. However, it does not have any effect on steady state performance of the closed loop.

The discussion above indicates that with P-D control, steady state error is still non-zero. Derivative control is usually used to decrease oscillations in closed loop system outputs. The following simulations were done on MATLAB-Simulink to illustrate the performance of P-D control on first and second order plants.

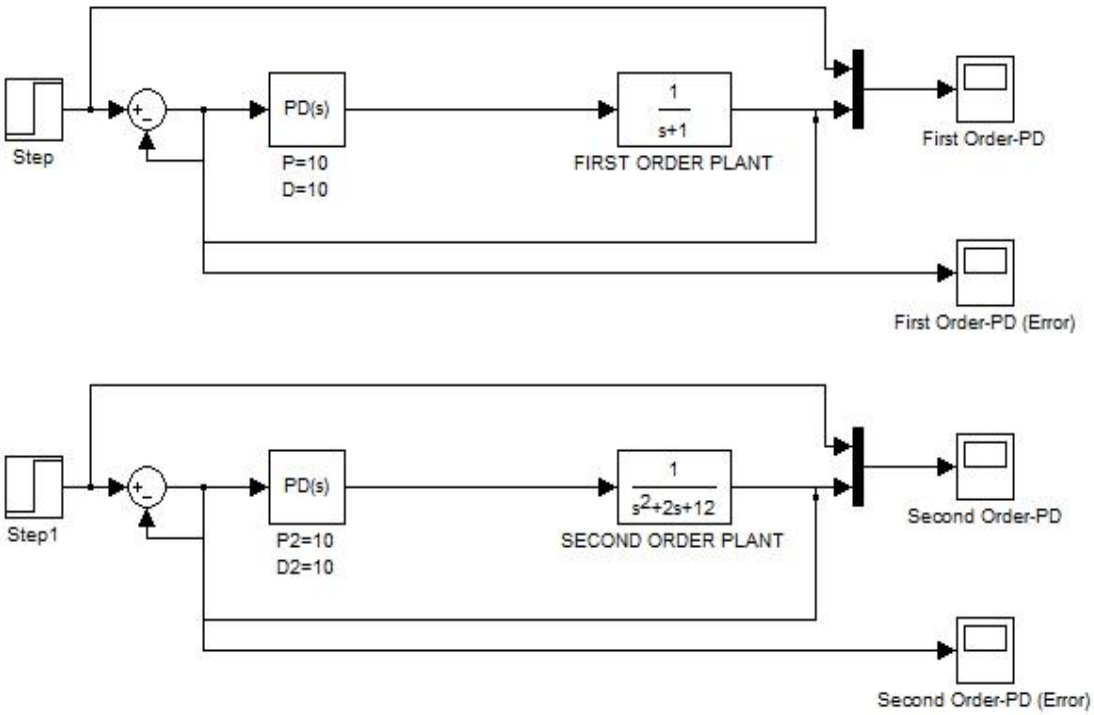

Figure 2.5.2: MATLAB-Simulink Diagram to show the effect of P-D control on first and second order plants

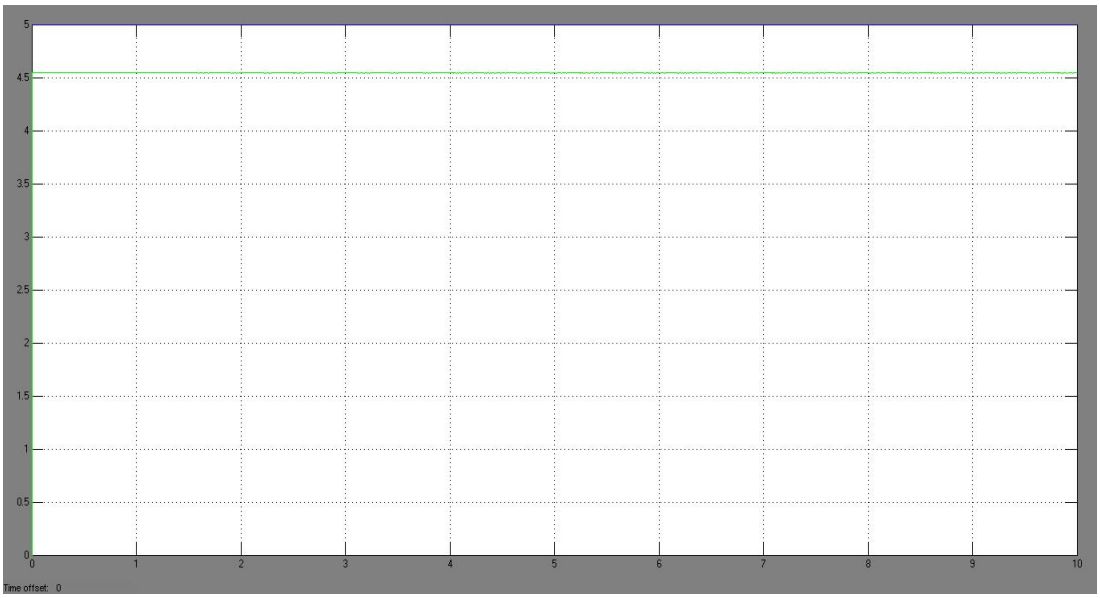

Figure 2.5.2(a): Output of the closed loop system (first order) with P-D Control,Kp=10,Kd=10

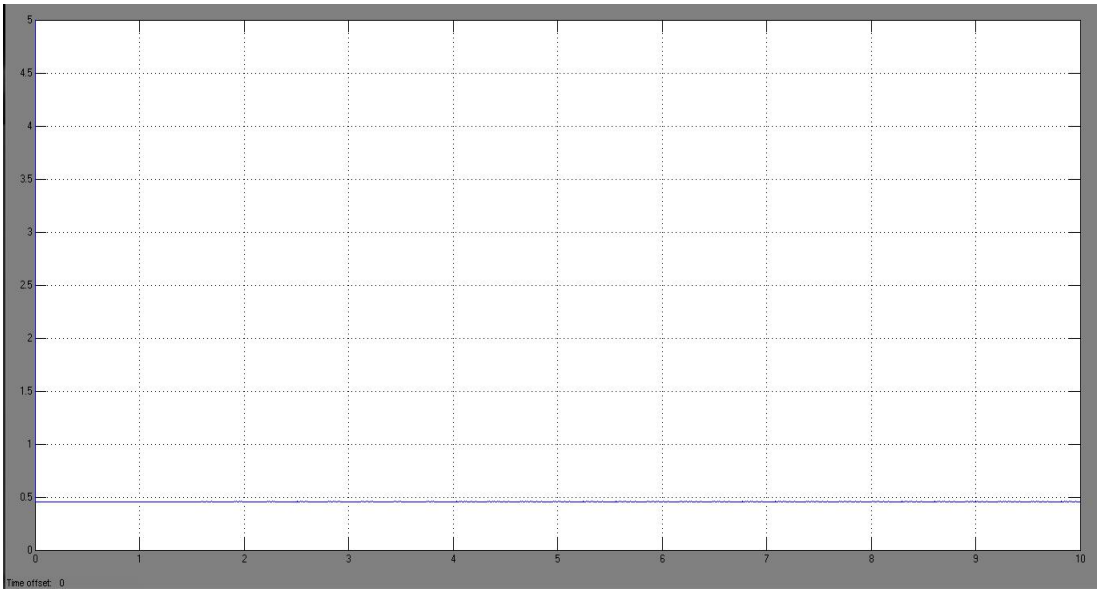

Figure 2.5.2(b): Error of the closed loop system (first order) with P-D control, Kp=10, Kd=10

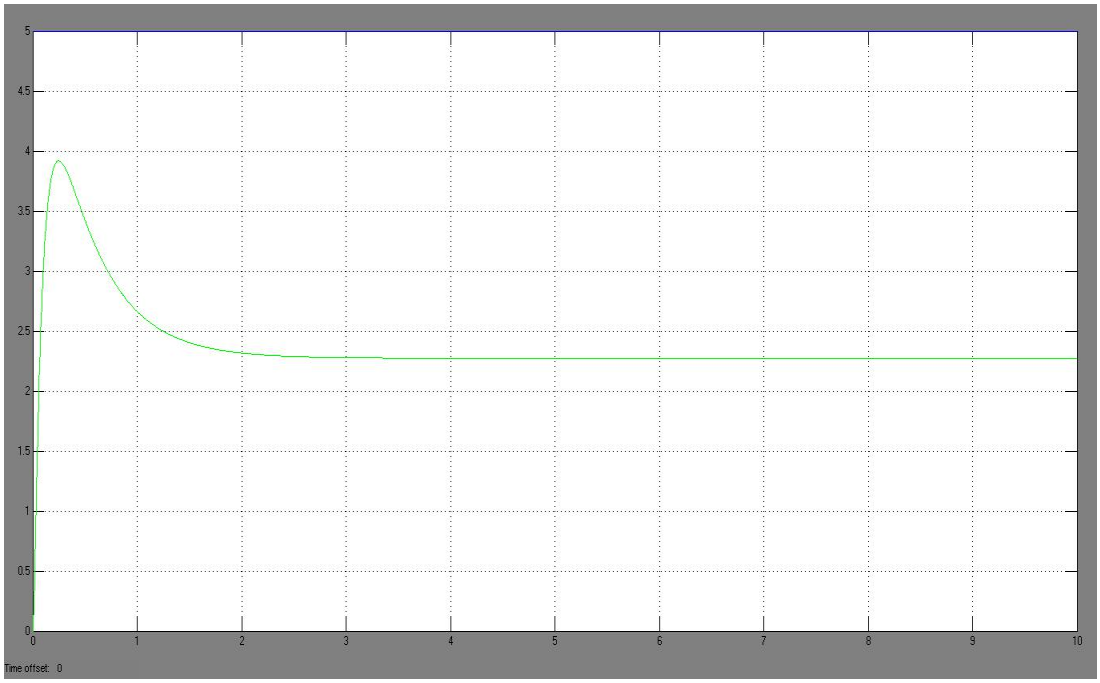

Figure 2.5.2(c): Output of the closed loop system (second order) with P-D control, Kp=10,Kd=10

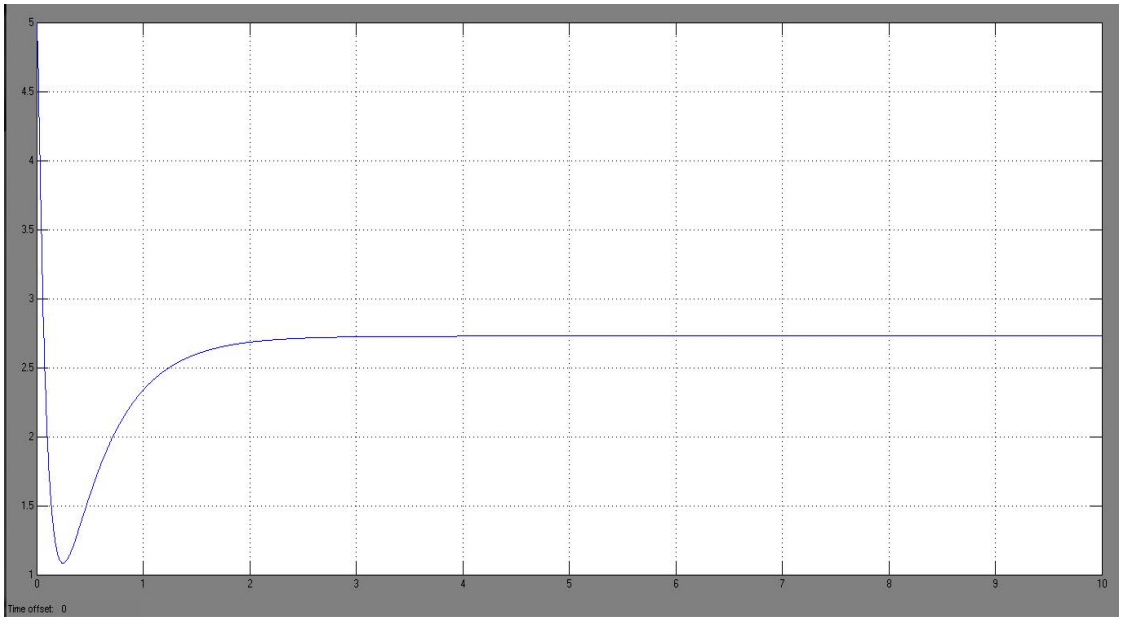

Figure 2.5.2(d): Error of the closed loop system (second order) with P-D control, Kp=10,Kd=10

2.5.3 Transient Response of P-I Controller

Integral action eliminates steady state error. However, it has very poor transient response. Using integral action increases the oscillations in the output of the closed loop systems.

The discussion above indicates that with P-I control, steady state error is non-zero. However, Integral control causes too many oscillations in closed loop system outputs. The following simulations were done on MATLAB-Simulink to illustrate the performance of P-I control on first and second order plants.

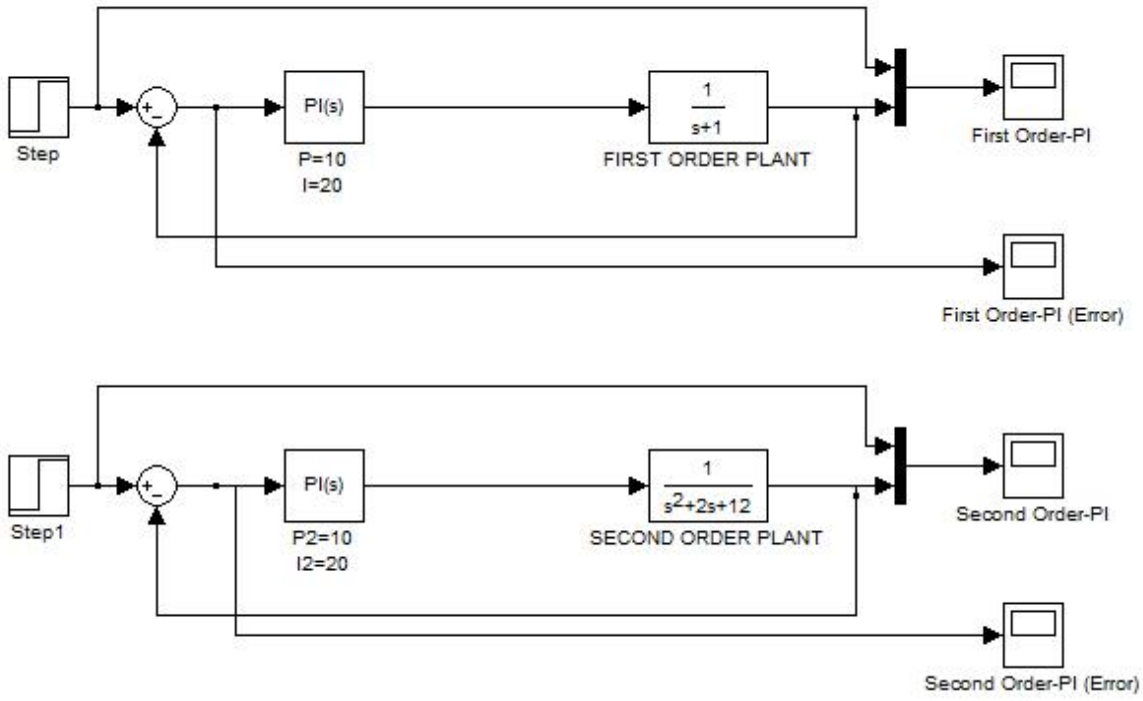

Figure 2.5.3: MATLAB-Simulink Diagram to show the effect of P-I control on first and second order plants

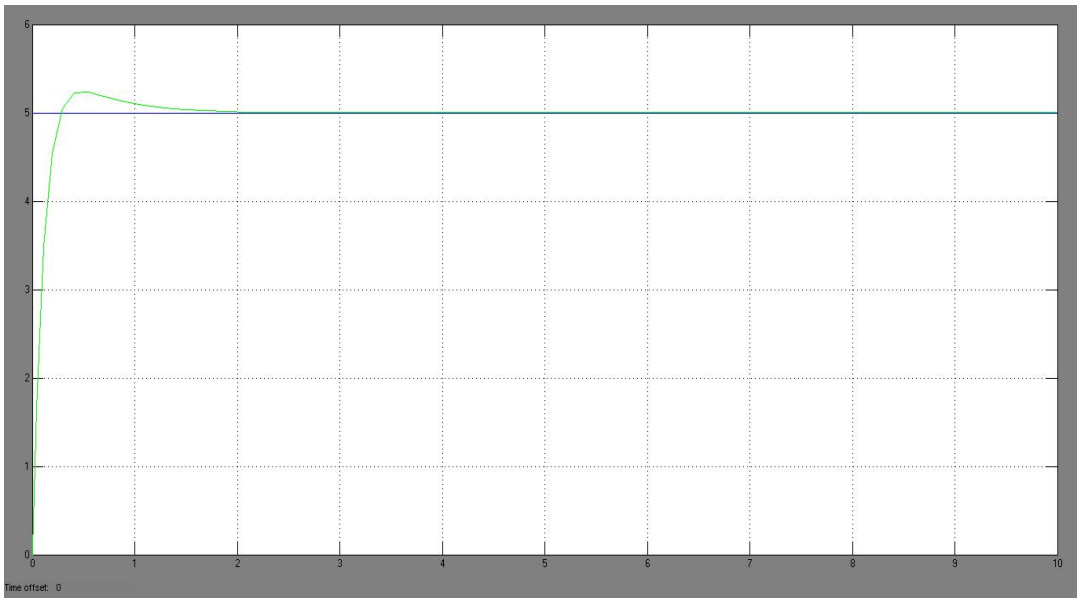

Figure 2.5.3(a): Output of the closed loop system (first order) with P-I control, Kp=10, Ki=20

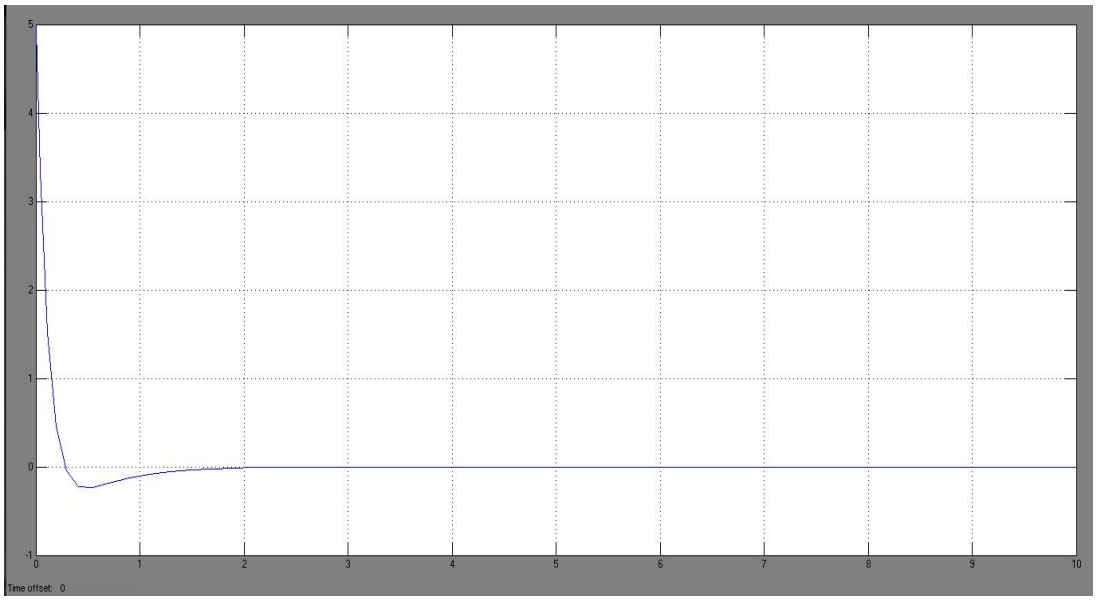

Figure 2.5.3(b): Error of the closed loop system (first order) with P-I control, Kp=10, Ki=20

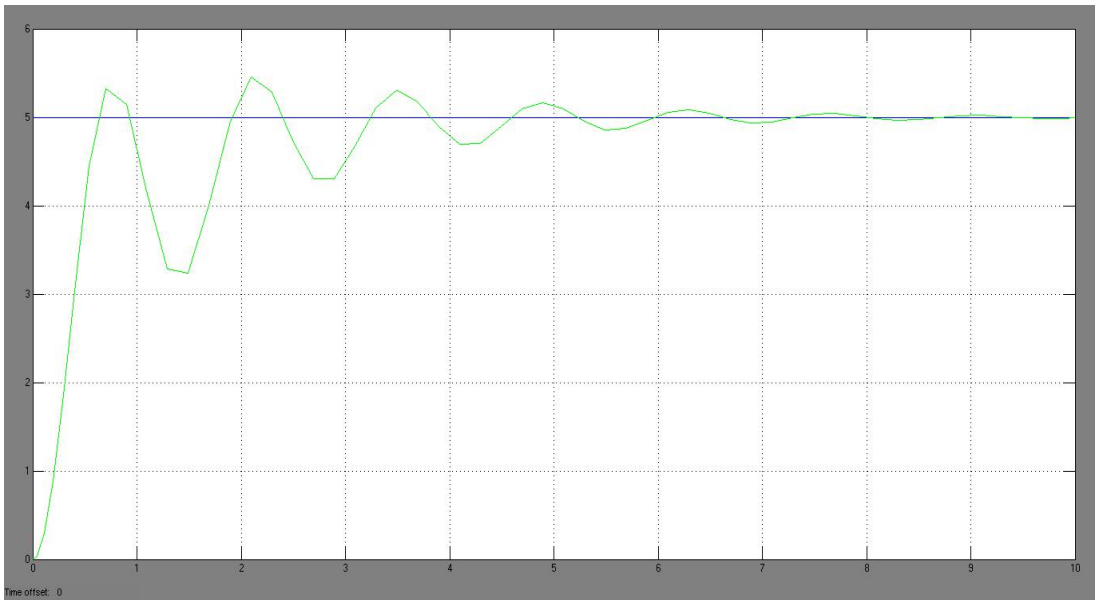

Figure 2.5.3(c): Output of the closed loop system (second order) with P-I control, Kp=10,Ki=20

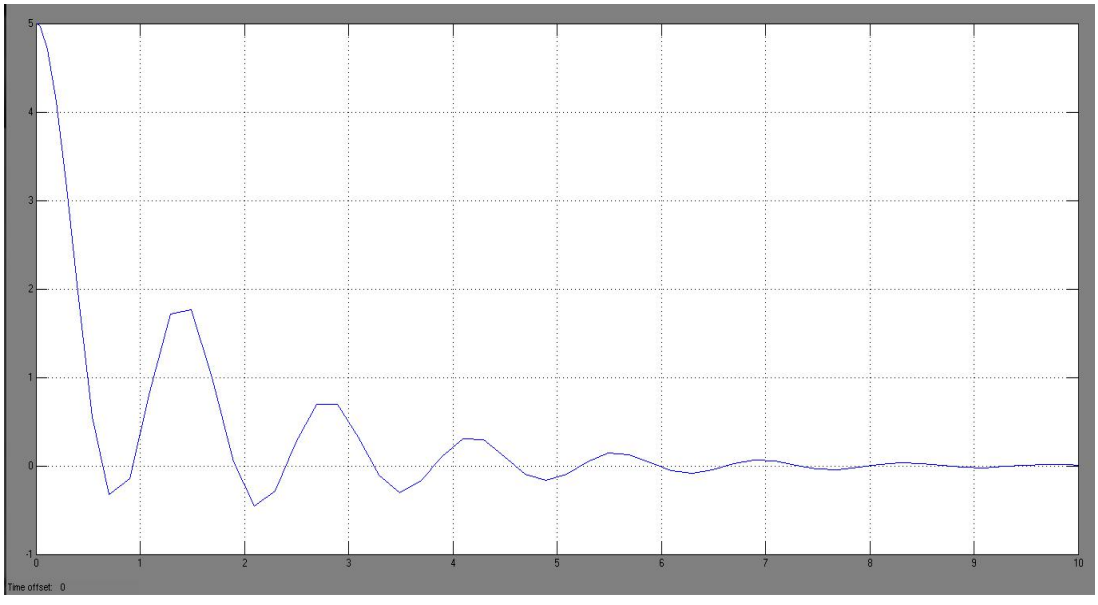

Figure 2.5.3(d): Error of the closed loop system (second order) with P-I control, Kp=10, Ki=20

2.5.4 Transient Response of P-I-D Controller

P-I-D controller is the optimal controller for high order plants. It has zero steady state error together with acceptable transient response. The only problem with P-I-D control is tuning. Fortunately, MATLAB has automatic tuning option. However, automatic tuning does not usually provide the best results, it only provides optimal results. P-I-D tuning is an engineering art and should be manually done by control engineers.

The following simulations were done on MATLAB-Simulink to illustrate the performance of P-I-D control on first and second order plants.

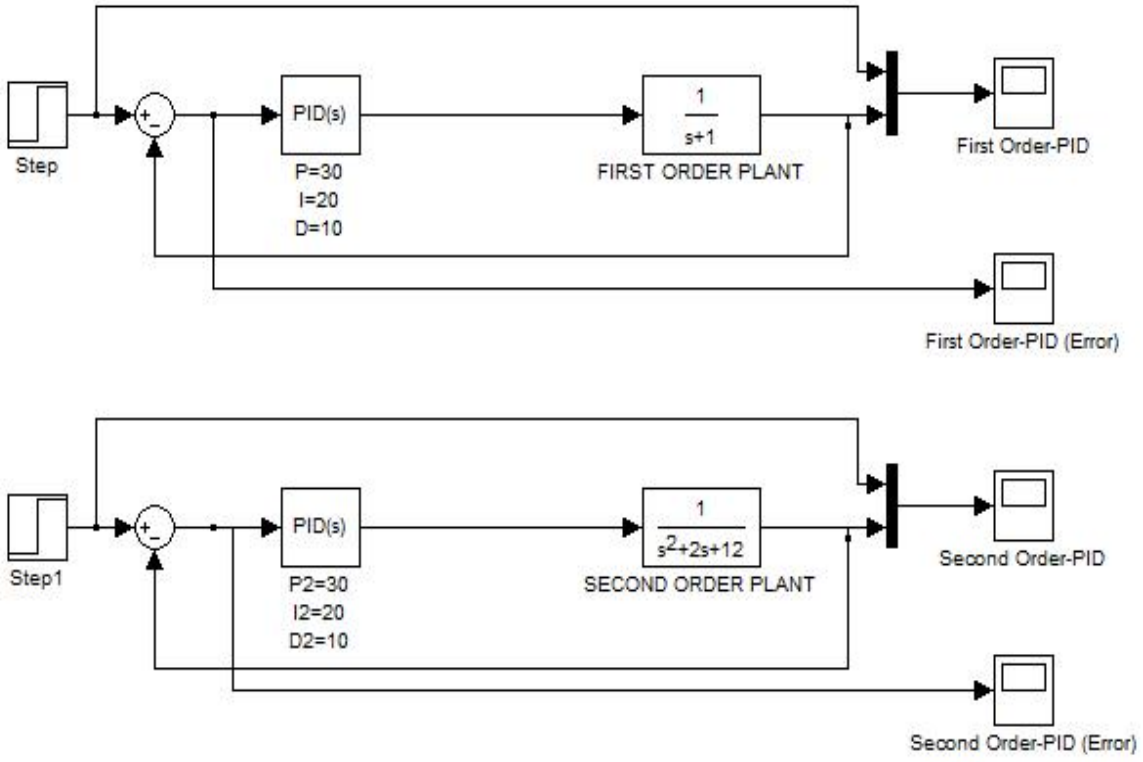

Figure 2.5.4: MATLAB-Simulink Diagram to show the effect of P-I-D control on first and second order plant

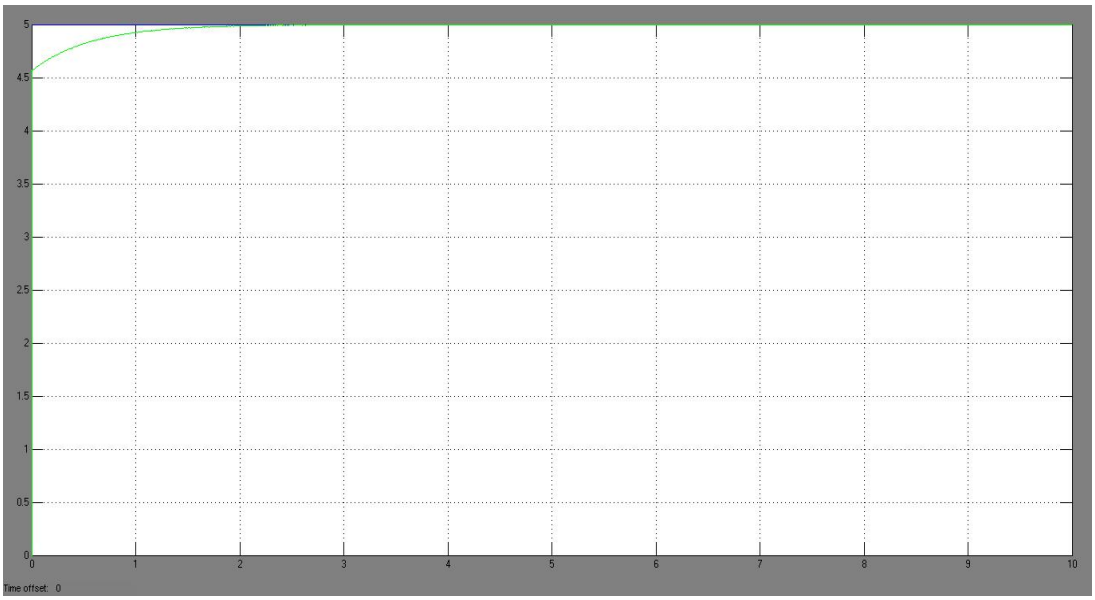

Figure 2.5.4(a): Output of the closed loop system (first order) with P-I-D control, Kp=30,Ki=20, Kd=10

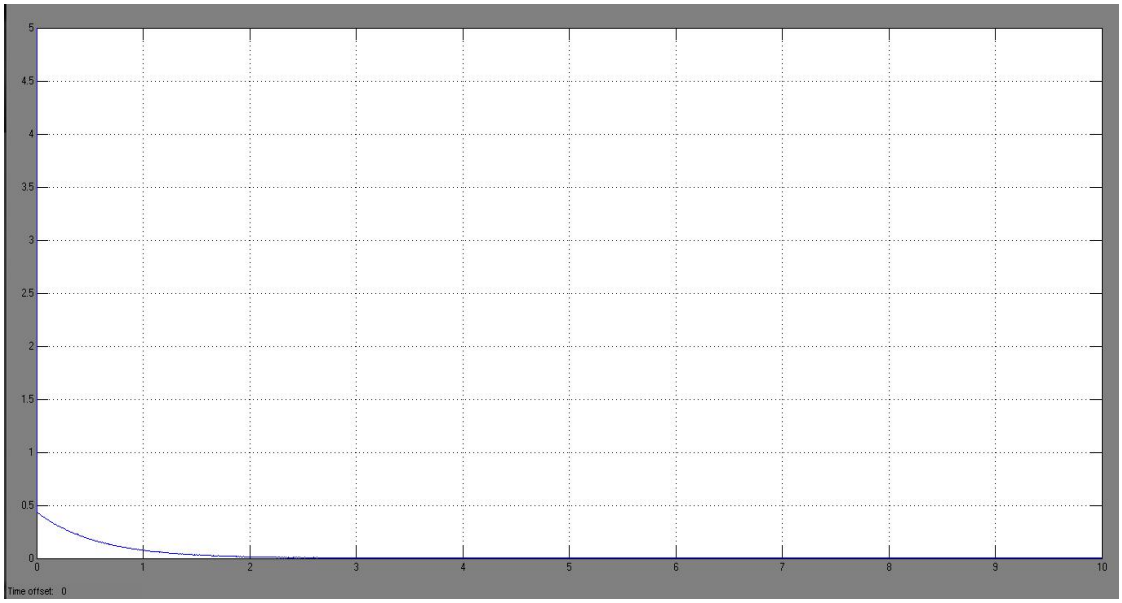

Figure 2.5.4(b): Error of the closed loop system (first order) with P-I-D control, Kp=30,Ki=20, Kd=10

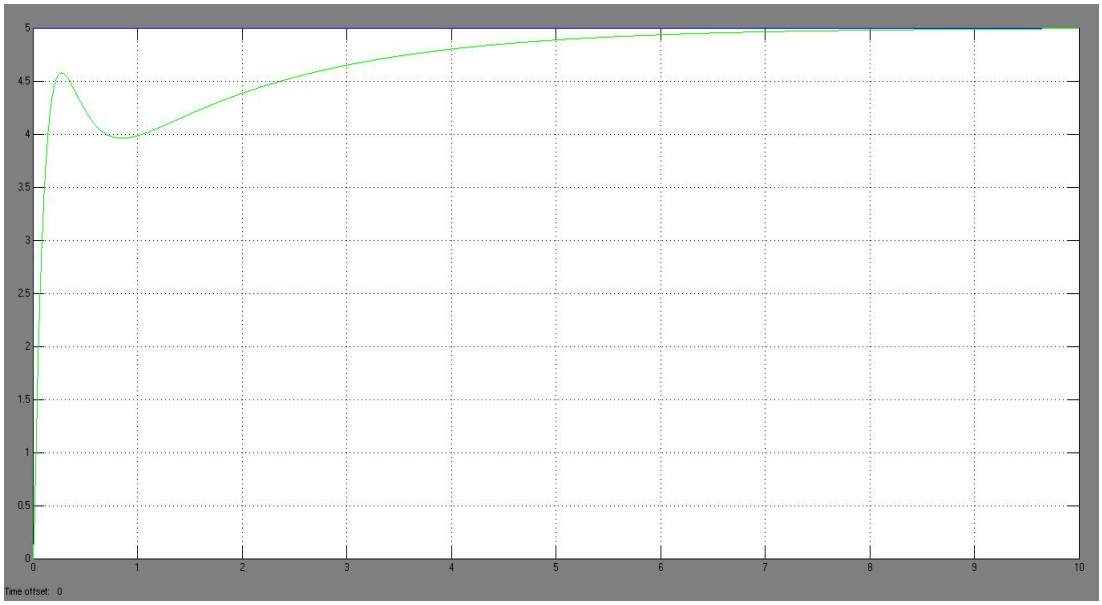

Figure 2.5.4(c): Output of the closed loop system (second order) with P-I-D control, Kp=30, Ki=20, Kd=10

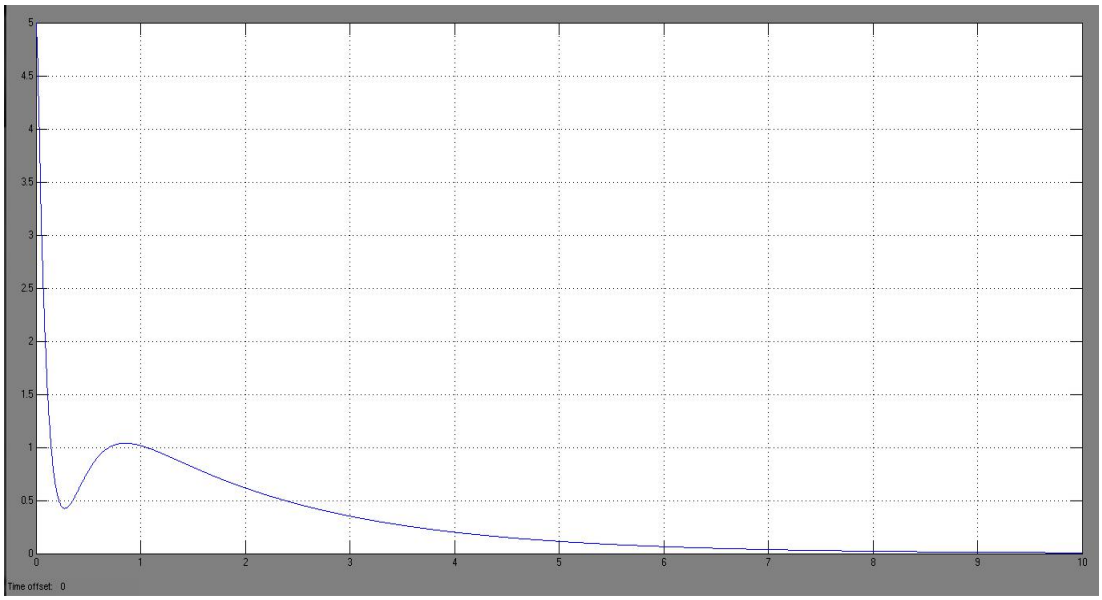

Figure 2.5.4(d): Error of the closed loop system (second order) with P-I-D control, Kp=30,Ki=20, Kd=10

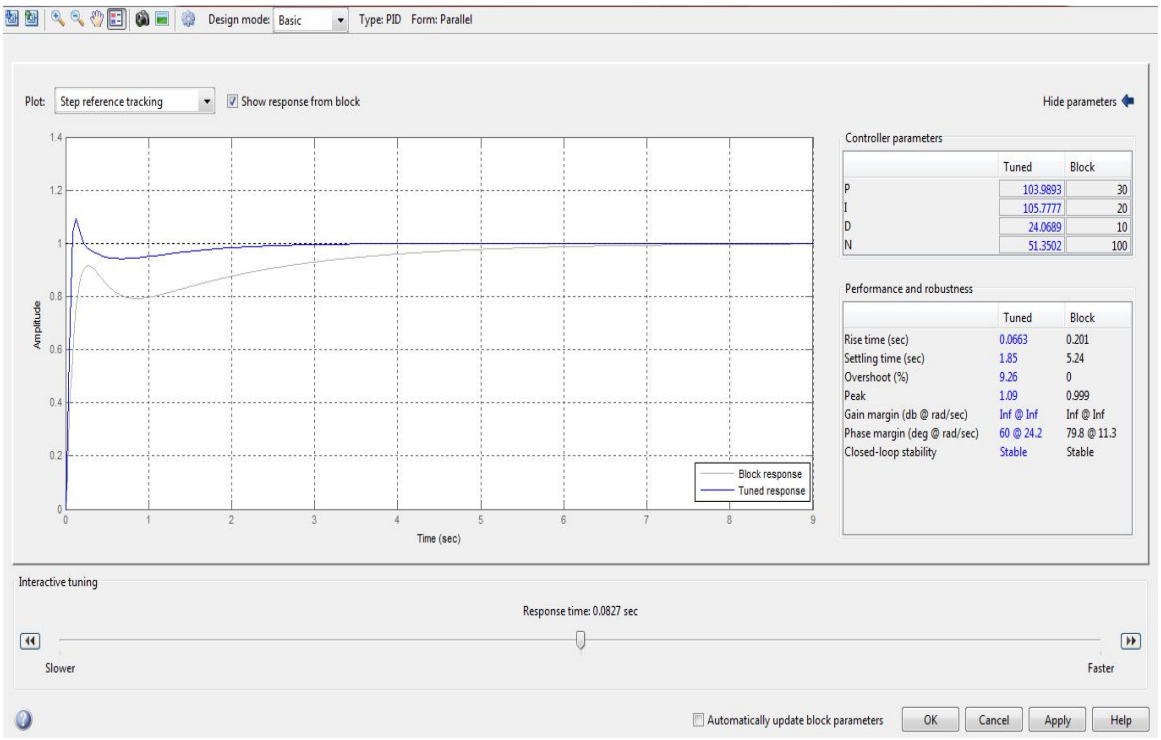

Figure 2.5.4(e): Output of the closed loop system (second order) with P-I-D control after tuning, Kp=104, Ki=106, Kd=24

2.6 Digital P-I-D Control

Digital P-I-D control is commonly used nowadays because of its ease of implementation. However, there are critical points that a designer should pay attention.

Most critical step is the choice of the sampling period. Since the nature consists of analog signals, most plant transfer functions are modeled in continuous time. In order to implement a digital P-I-D controller the designer should take samples from the continuous time error signal. However, these samples should be taken frequently enough in order not to miss system dynamics.

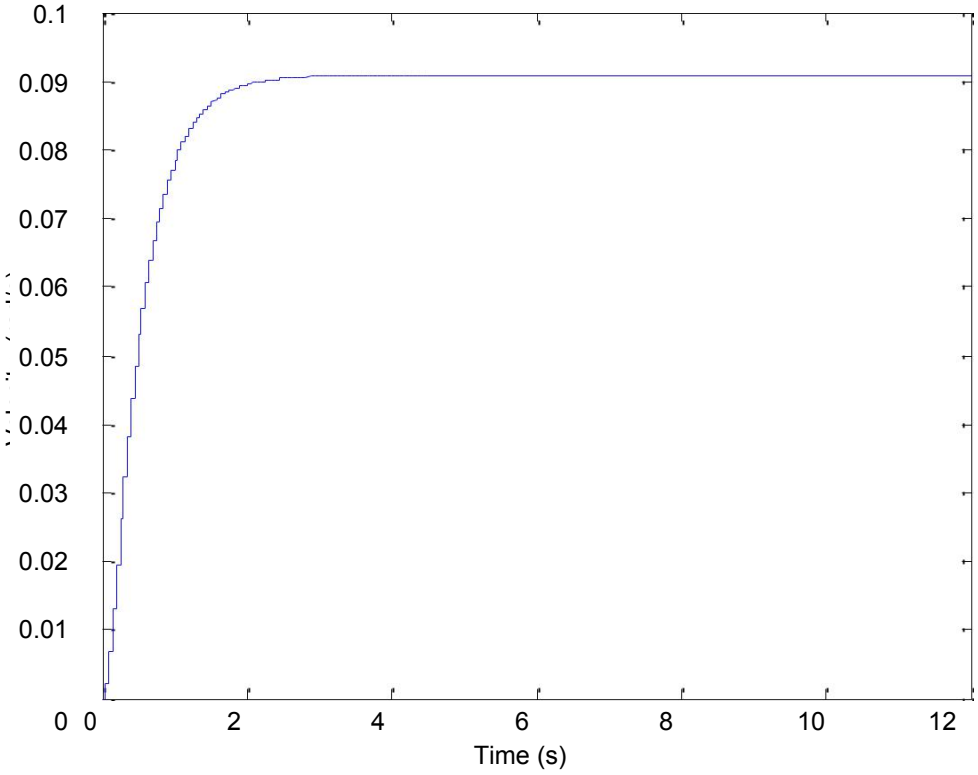

Figure 2.6.1: Stairstep response of open loop system, sampling period Ts=0.05 seconds

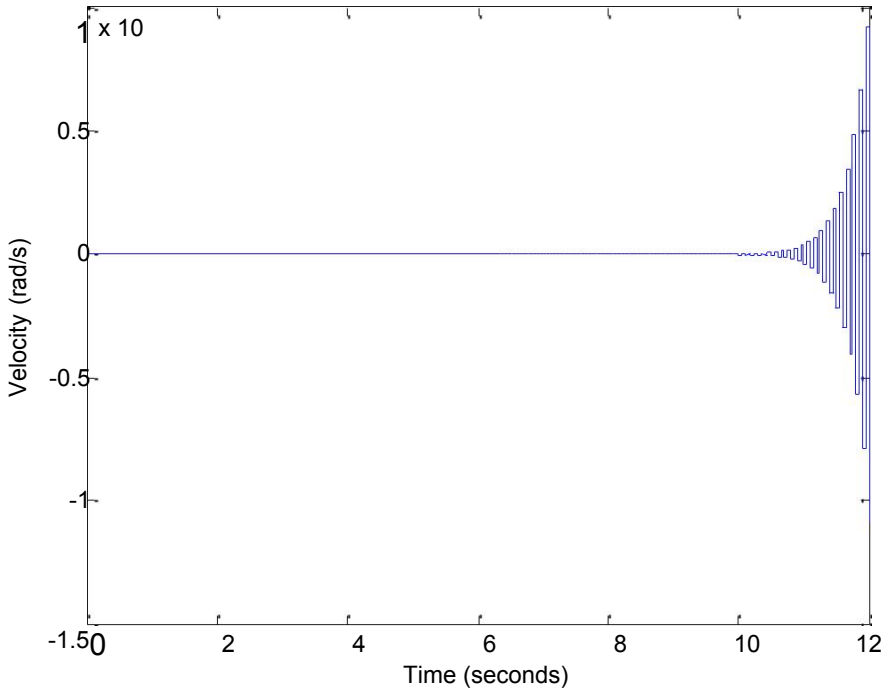

Figure 2.6.2: Stairstep response of closed loop system with PID, sampling period Ts=0.05 seconds

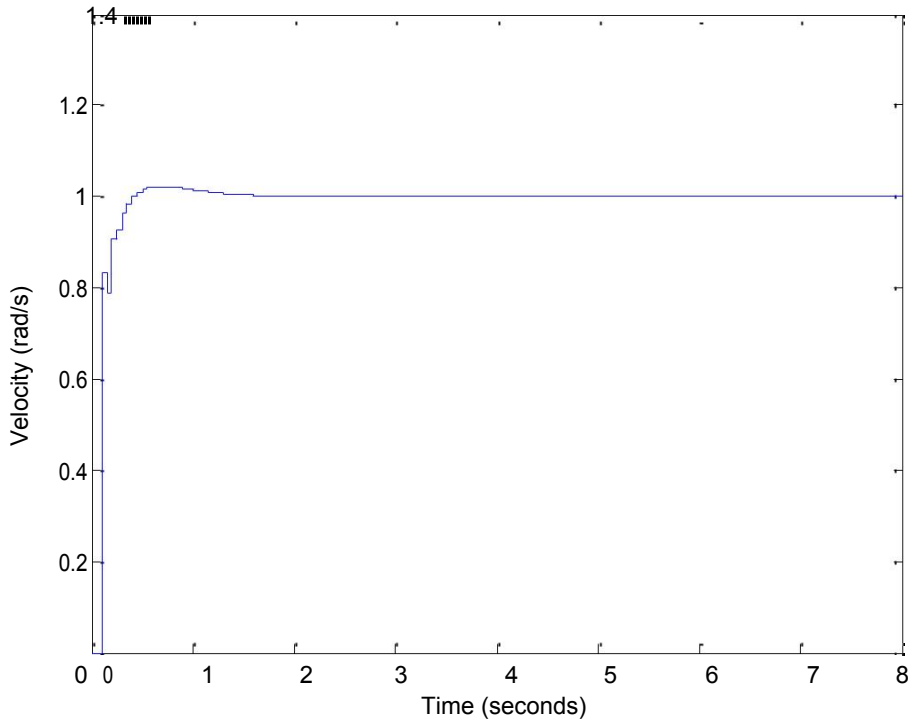

Figure 2.6.3: Stairstep response of closed loop system with PID after pole placement, sampling period Ts=0.05 seconds

Actually, the system is unstable. However, sampling period is too big to observe that. Hence, the response becomes misleading if the sampling period is not appropriately chosen.

Finally, the effect of choosing very high sampling period is shown below.

Sampling period,

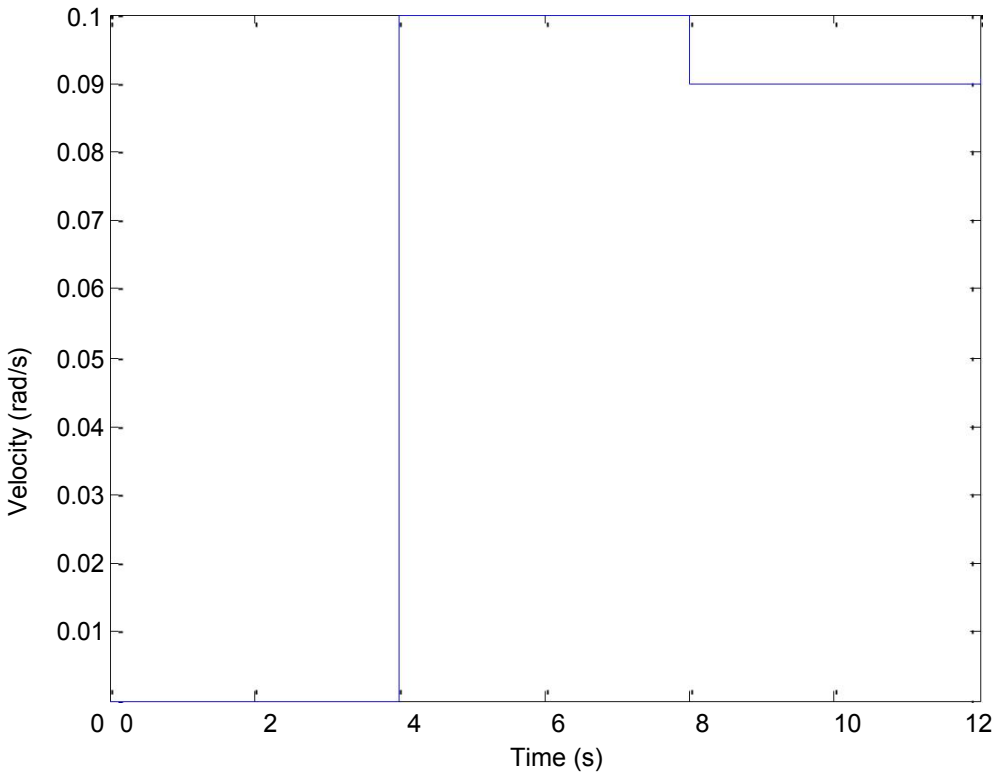

Figure 2.6.4: Stairstep response of open loop system, sampling period Ts=4 seconds :

Figure 51 is supposed to show the actual open loop characteristics of the system. However, the open loop system dynamics is faster that the sampling period. This should not be the case for a good digital P-I-D controller design.

Last but not least, a designer should place poles if they are required. Pole placement should be done with care by keeping in mind the system dynamics and the s*-domain to z-domain conversion method. The full MATLAB-code used in this part of the report can be found in Appendix.

Chapter 3

3 PID Controller Design for Controlling DC Motor Speed in the Project

3.1 Why do we need to control the speed of DC motor?

For many cases, we cannot obtain the same desired results in terms of theoretical and practical cases. For that project, we have to make theoretical power calculations for DC motors to obtain the desired DC motor speed. However in practice, we could not obtain the same results as it is calculated theoretically. For that purpose we have to use controllers to minimize the error between actual and theoretical results.

3.2 Why to choose P-I-D as controller?

The aim in using the P-I-D controller is to make the actual motor speed match the desired motor speed. P-I-D algorithm will calculate necessary power changes to get the actual speed. This creates a cycle where the motor' speed is constantly being checked against the desired speed. The power level is always set based on what is needed to achieve the correct results.

By using P-I-D controller, we can make the steady state error zero with integral control. We can also obtain fast response time by changing the P-I-D parameters. P-I-D is also very feasible when it is compared with other controllers.

In our project, first of all we have obtained the P-I-D parameters for our system. Then we have constituted our own P-I-D algorithm with coding. The P-I-D algorithm and the whole code segments can be seen in Appendix.

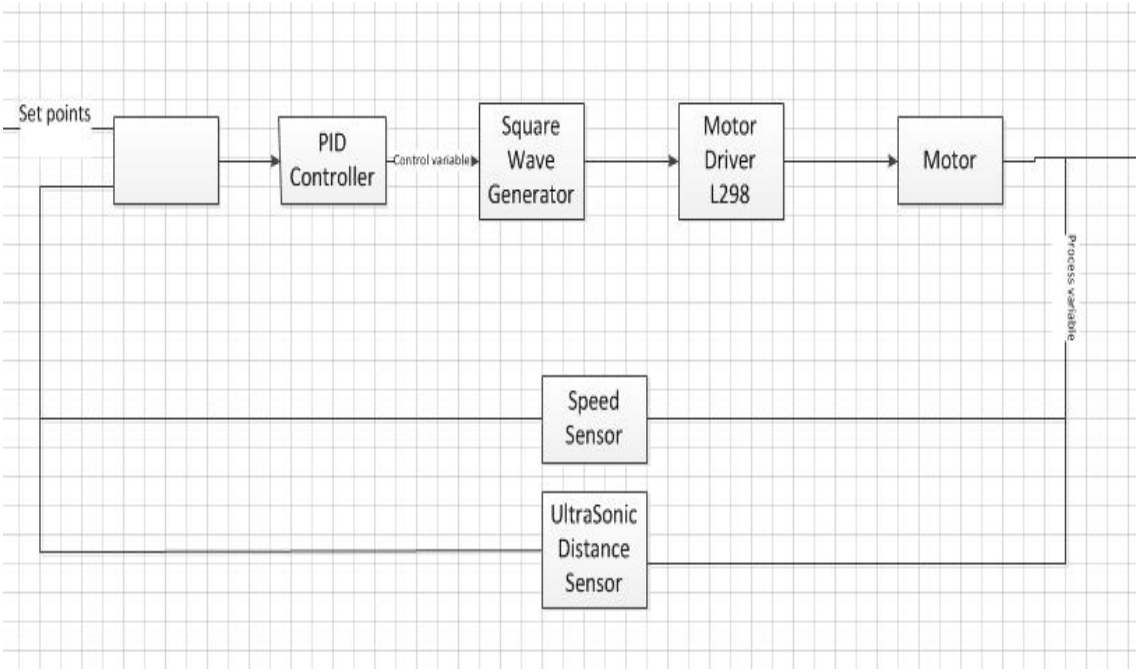

Figure 3: The Block Diagram of the DC Motor Speed Control Loop

As it is seen from the block diagram of the DC motor control loop, the speed sensor (encoder) measure the speed of the DC motor. We also have another feedback loop which measures the distance of the vehicle to the gate. The measurement of the distance is important, since we want to have different speed values of the vehicle at different distance values of the vehicle to the gate. By using Arduino microcontroller, we have constructed condition loops for different distance values. In each of these loops we have compared the actual speed of the DC motor with the desired one. The DC speed measurement gives the actual speed value. The error between theoretical and practical values is corrected with PID controller. The parameters of the PID controller are determined with MATLAB results which will be explained in the following sections. The output of the PID controller gives the duty cycle of the square wave generator. Data acquisition cards can be used as square wave generators. As a second option Arduino can also be used as square wave generator. For that purpose we have used Arduino as our square wave generator. The output of the square wave generator is motor driver. We have used L298 as motor driver which can supply current up to 2A to the DC motor.

3.3 PID Parameters

1. PID controller can be investigated under 3 main categories. Each controller has different properties in terms of controlling the whole system.
2. In proportional control, adjustments are based on the current difference between the actual and desired speed.
3. In ıntegral control, adjustments are based on recent errors.
4. In derivative control, adjustments are based on the rate of change of errors.

3.4 The Design Requirements of the System

The design requirements of the systems may vary from one system to another. For our case, we want a fast response of the system to an error. The overshoot of the system should not be higher than 5% and the settling time should be smaller than 2 seconds.

The main design requirements are as follows;

1. Settling time should be less than 2 seconds;
2. Overshoot of the system should be less than 5%;
3. Steady state error should be less than 1%

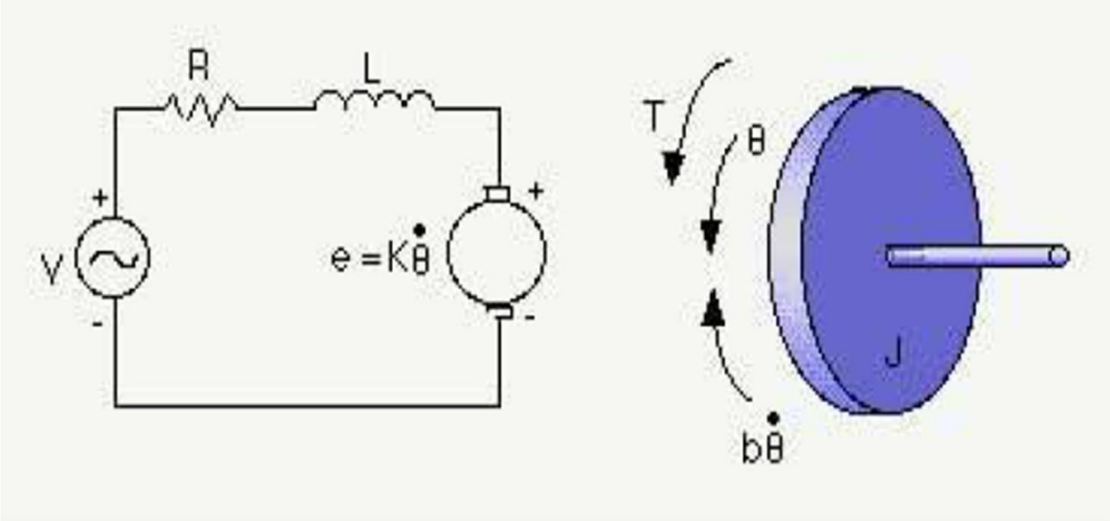

Figure 3.4:The Schematic of the DC Motor

3.5 The Parameters of the DC Motor

The parameters of the DC motors may change according to different torque and rpm values of the DC motors. For 1000 rpm DC motor that we have used in our project.

1. Rotor moment of inertia(Jm)=0.01kg*m^2/s^2
2. Resistance=1Ω
3. Inductor=0.5H
4. Electromotive Force Constant Kt=0.01Nm/Amp
5. Motor Viscous Friction Constant(Beq)=0.1Nms

3.6 The open loop transfer function of the DC motor:

The transfer function of the DC motor can be found from the schematic of the DC motor in Figure 3.6.1.From that point, we have to find the PID parameters for our PID control algorithm. To find the parameters of PID, we should start from proportional constant.By using only proportional controller, the block diagram of the overall system would be as follows;

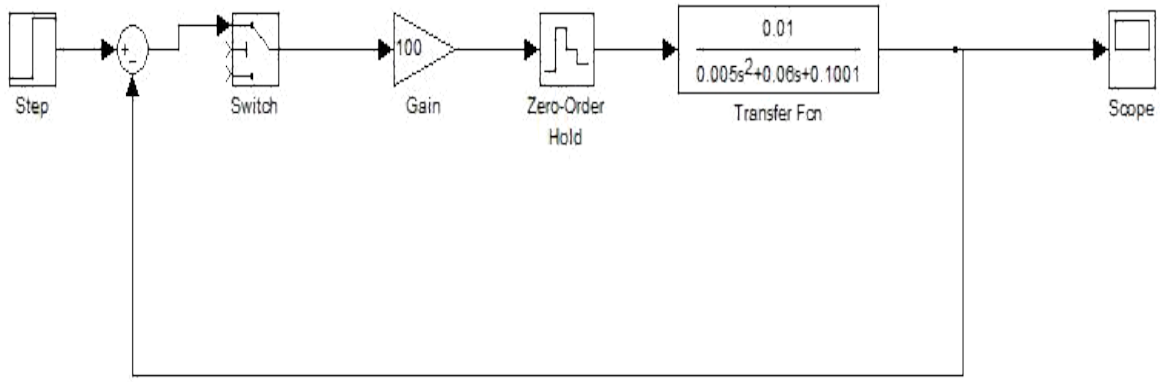

Figure 3.6.1:The Block Diagram of the System with Proportional Controller

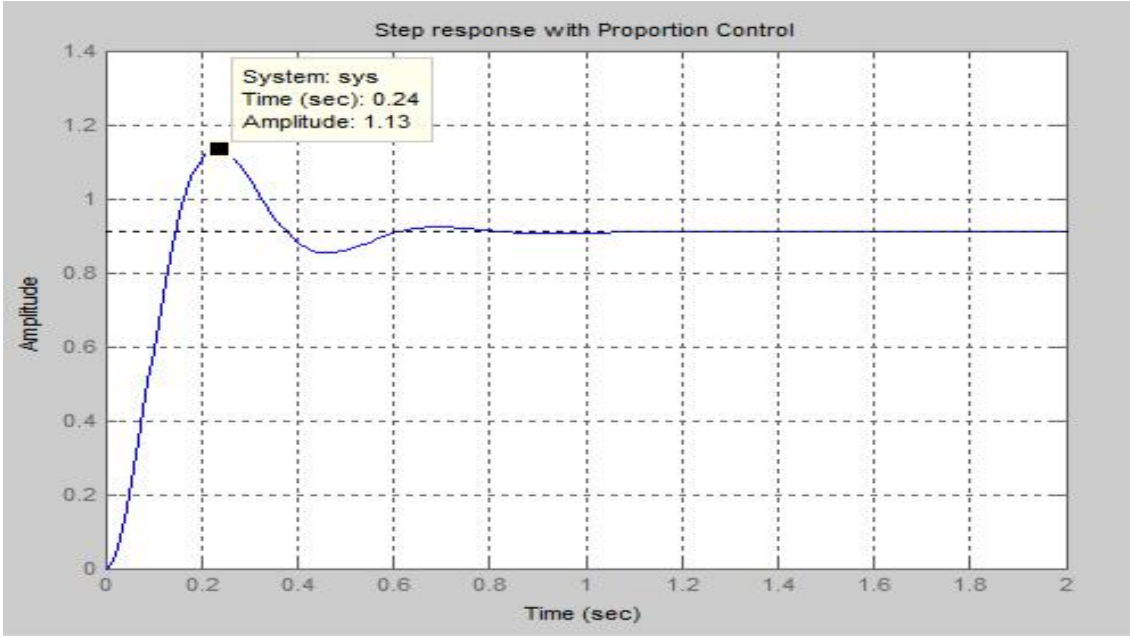

Figure 3.6.2:The MATLAB Result, Kp=100

The overshoot of the system with Kp = 100 is 25% which does not satisfy our design requirements. The settling time of the system is about 0.37 seconds. This satisfies our system requirement. The steady state error of the system is 0.1.

After adding derivative and integral controllers to the system; the block diagram of the system is the following;

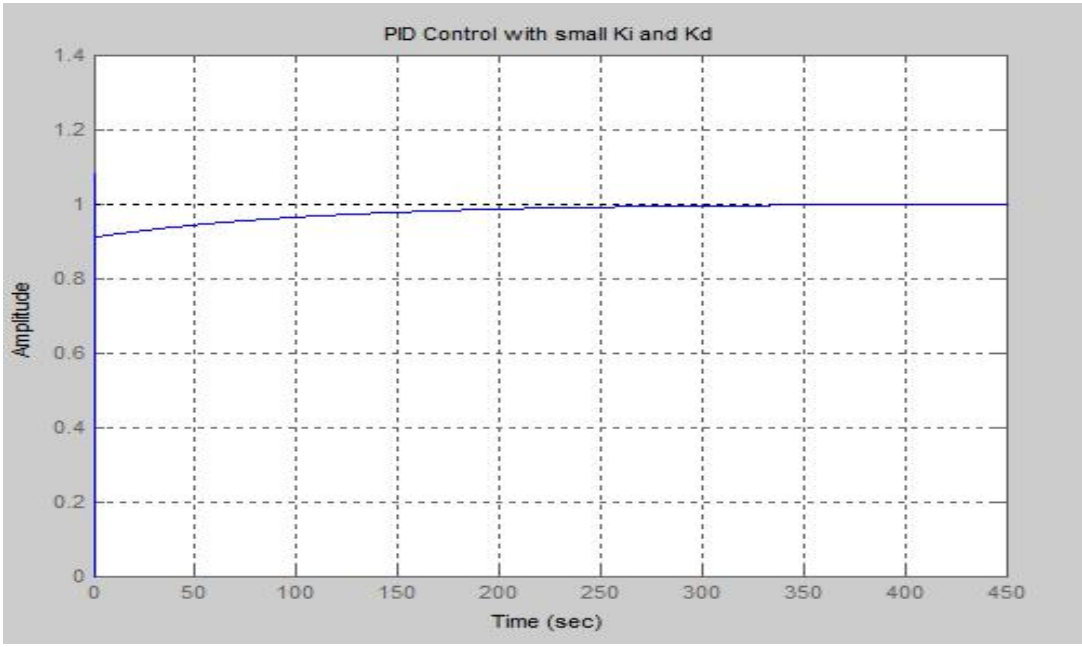

Figure 3.6.3:The MATLAB Result for Ki=1, Kd=1, Kp=100

The settling time of the new system is 400 seconds which is far away from satisfying our design requirement. There is also a pulse in t=0 which causes instability to our system. To obtain a better response, we have increased the value of Ki to 200;

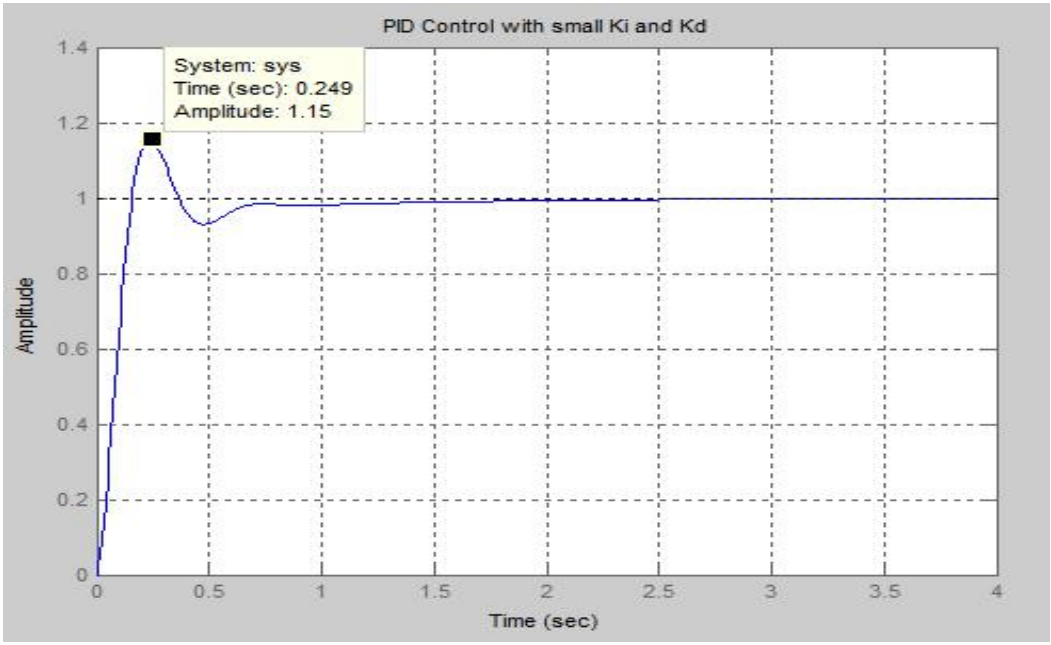

Figure 3.6.4:The MATLAB Result for Ki=200, Kd=1, Kp=100

As we have increased the value of Ki, the steady state value of the system becomes 0. Actually the aim in using the integral control is to make the steady state error zero. For the overshoot of the system does not satisfy thedesign requirement. For that reason let increase the value of Kd;

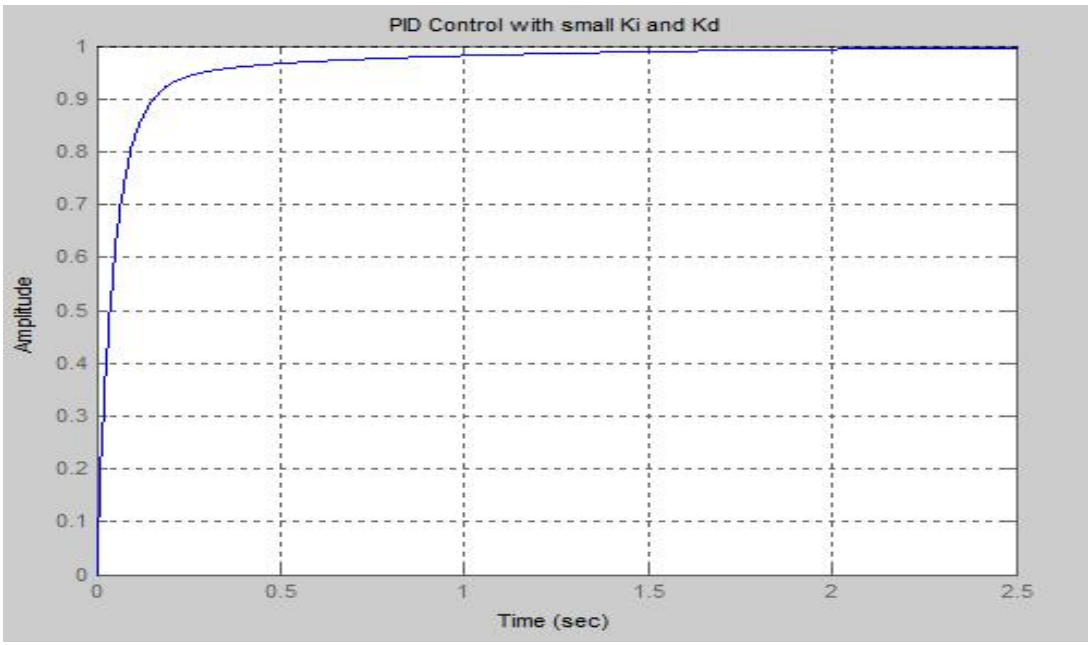

Figure 3.6.5: The MATLAB Result for Ki=200, Kd=10, Kp=100

As we have increased the value of Kp, the overshoot value of the system becomes 0 and with those parameters of P-I-D controller, we have obtained the system design requirements.

Note that these P-I-D parameters are found in continuous time system. So we have to check whether these parameters satisfy the system requirements in discrete time domain.

To be able to check it, first of all we have to obtain the DC motor transfer function in z domain. For the conversion from s to z, we have used ZOH method which is learnt in the class.

3.7 s*-domain to z-domain with ZOH (only plant-DC motor)

$$T(s) = \frac{2}{(s+9.997)\,(s+2.003)}$$

$$T(z) = \frac{0.0020586\,(z+0.8189)}{(z-0.9047)\,(z-0.6066)}$$

Sampling time: 0.05

Note that sampling time of the system is defined according to dominant pole approximation which is clearly explained before.Now let investigate the step response of the plant with zero order hold;

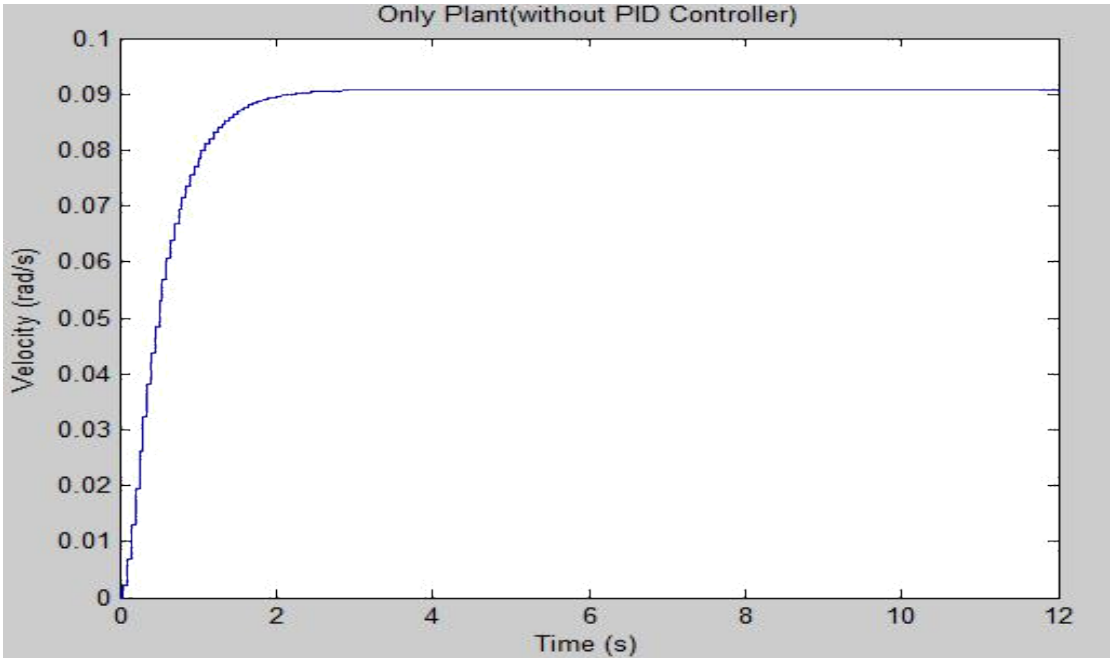

Figure 3.7.1-The Step Motor Response of the DC motor without PID controller

The steady state error of the system is increased to 0.9 which was 0.1 in continuous time. From that graph we can make the assumption that our system is required modification.Let investigate the step response of the compensated system with P-I-D;

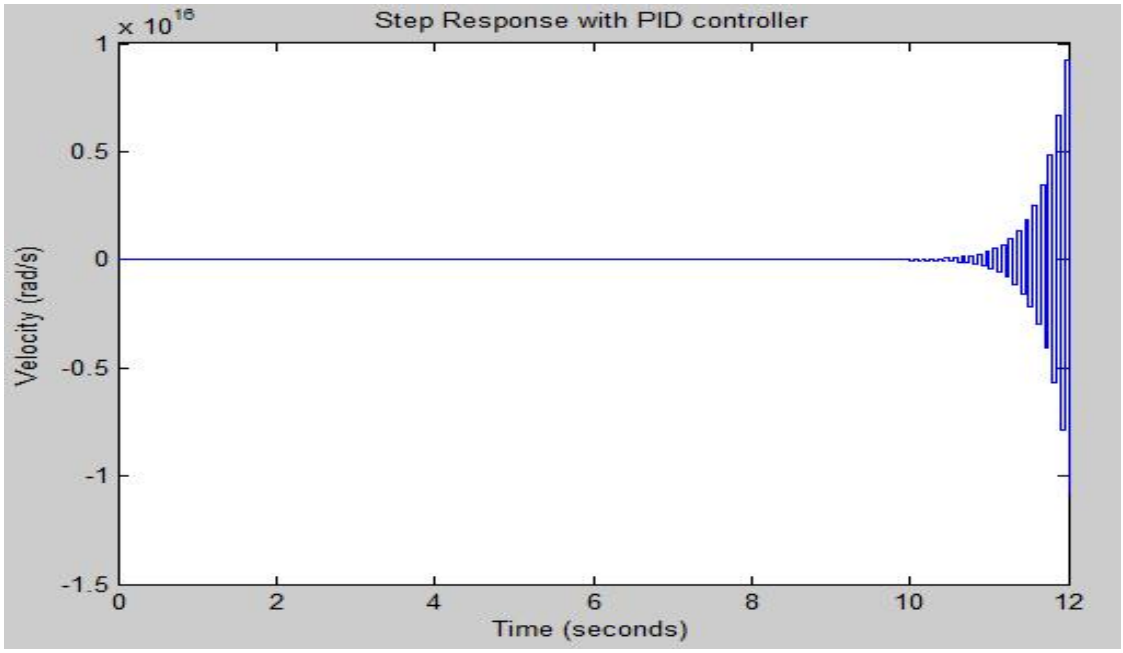

Figure 3.7.2-The Step Response of Plant with PID Controller

Our system's step response is unstable. To find the reason of instability, we have to check the root locus of the compensated system.The root locus of the system is the following;

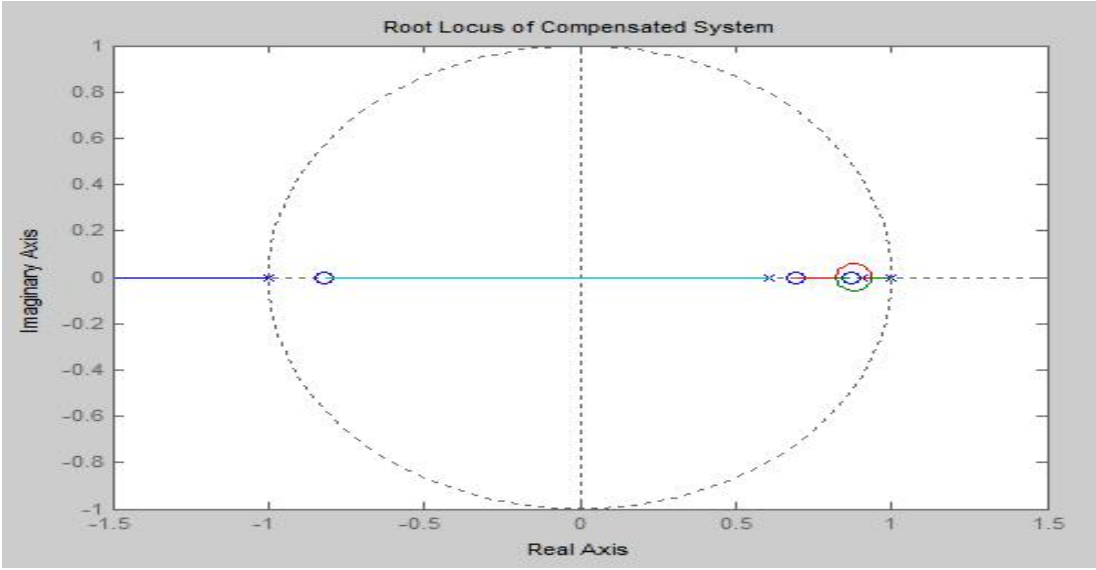

Figure 3.7.3-The Root Locus of the Compensated System

Note that the pole at -1 goes to infinity as the gain(K) of the system is increased. It is the reason of instability. To be able to make the system stable, let make a pole at -0.82.After adding a pole at -0.82 the root locus of the system is the following;After Adding a Pole at -0.82:

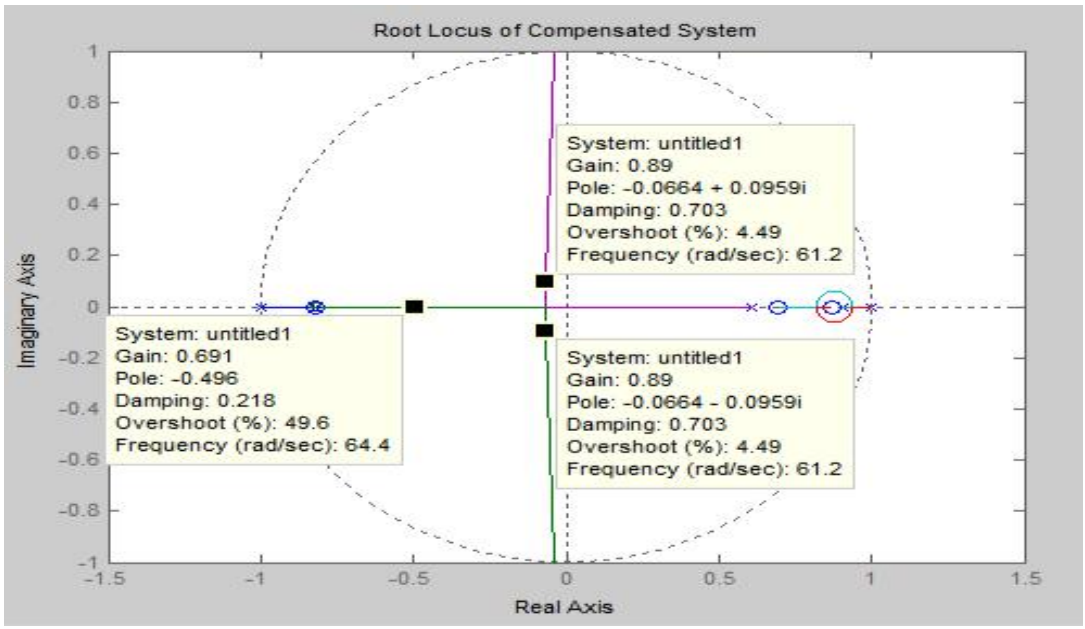

Figure 3.7.4-The Root Locus of the Compensated System

Note that for the values of the poles in the unit circle, we expect to obtain stable compensated system. For that purpose, to show the gain and other specifications of the system, we have taken 3 point. Note that any point in the unit circle can be chosen to obtain stable systems.at that point we have chosen gain=0.89;

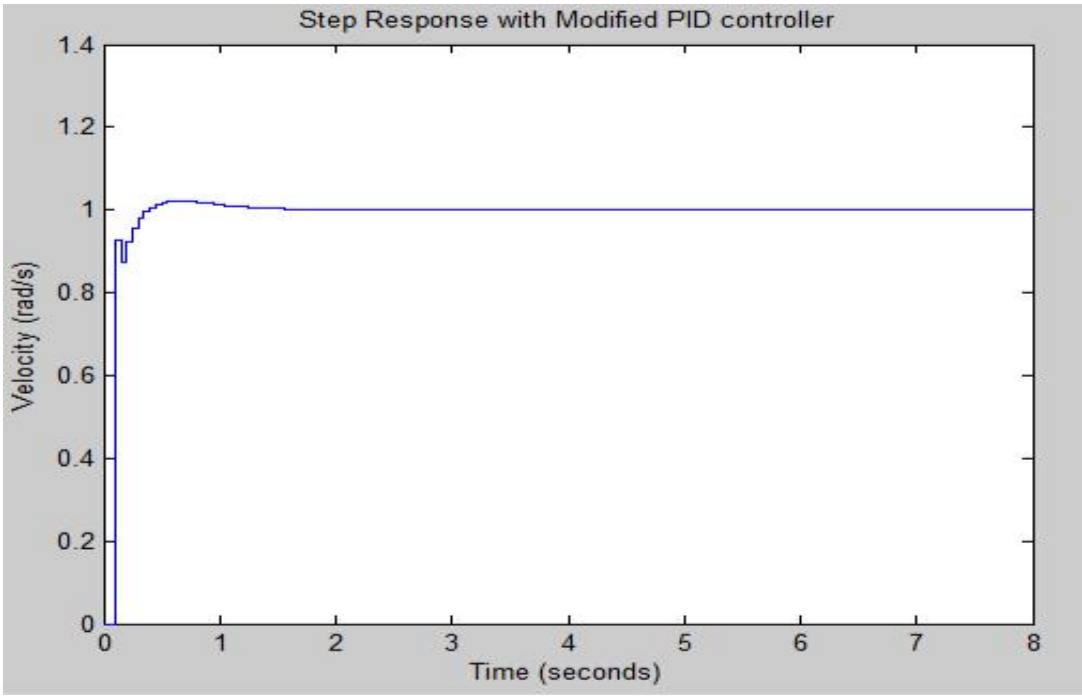

Figure 3.7.5-The Step Response of the System with modified P-I-D Controller

As it is seen from Figure 64, the system design requirements are also satisfied in discrete time model. In real life, the addition of pole can be done by adding a capacitor at the end of the PID controller.

Chapter 4

P-I-D Controller Design for Controlling DC Motor Position in the Project

4.1 Why we need to positioning the DC Motor?

In this project, the position control of the vehicle can be done with P-I-D controller. Note that the distance measurement of the vehicle to the gate is done with ultrasonic sensor. The speed of the vehicle is relatively low (DC motors are at 50 rpm) when the vehicle is at a distance greater than 15 cm. for successful passing operation, we need a faster vehicle. For that purpose our vehicle speed is relatively high (DC motors are at 200 rpm). While adjusting the speed of DC motors with P-I-D controllers, we also have to make car moving in correct position. For that purpose we have done a line follower vehicle with P-I-D controller. Note that the P-I-D controller is done with done segments which can be seen in Appendix. In this section we will try to explain how the parameters of the P-I-D are obtained.

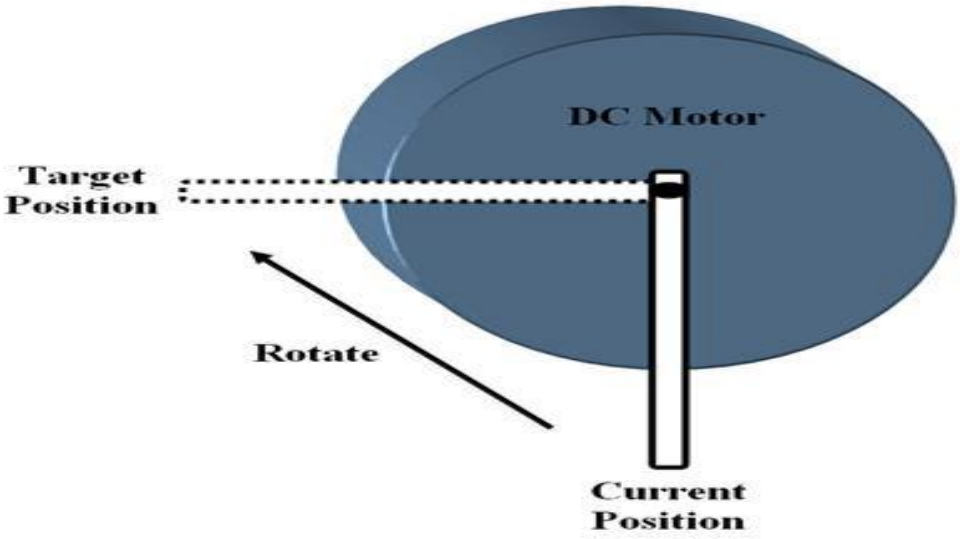

Figure 4.1.1-DC Motor Position Control

Before deciding the parameters of the P-I-D, let first derive the transfer function of the DC motor.Note that the only difference from the DC motor speed control is that we have a term 1/s which comes from the derivative of the position.Note that we have investigated the transfer function of the encoder and position sensors. But for some cases these parameters are directly connected to the DC motor. So we have used directly the transfer function of the DC motor while deciding our P-I-D controller parameters.

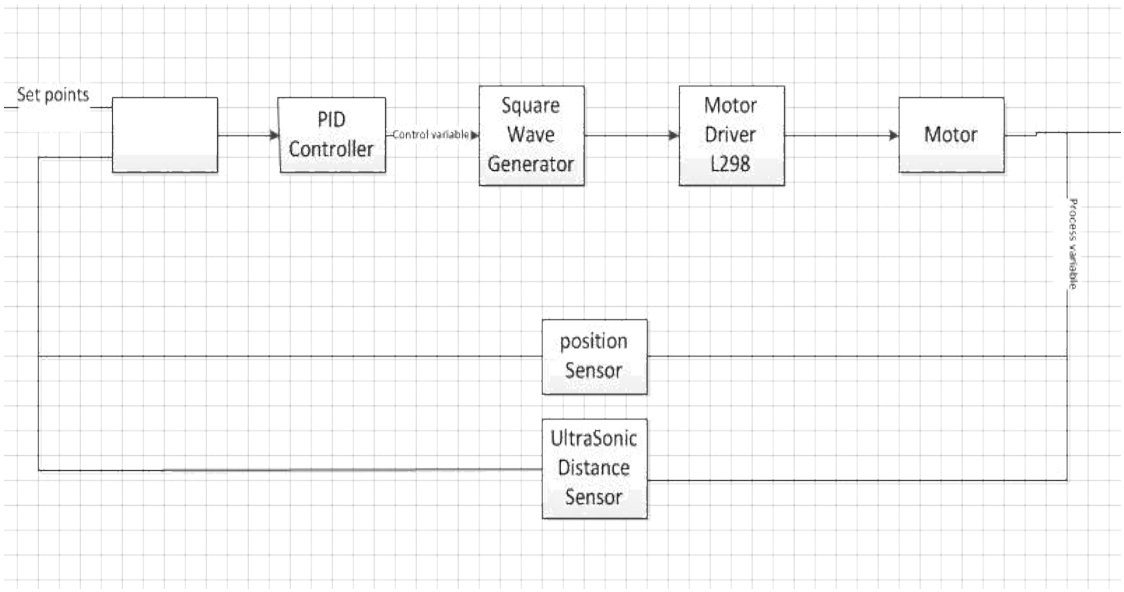

Figure 4.1.2-The Block Diagram of the DC Motor Position Control Loop

The ultrasonic sensor is also directly connected to the Arduino Microcontroller which also includes our P-I-D controller and square wave generator. For that reason we have not calculated the transfer function of the ultrasonic distance sensor.

for the position sensors, we have used 4 CNY70 sensors. These sensors are used with Schmidt triggers. Note that in Figure 66, position sensors include both CNY70 sensors and Schmidt triggers. Schmidt triggers are used to obtain either 0 or 5 Volts according to the output of the CNY70 sensors. CNY70 sensor output is HIGH (5V) if the sensor sees black line and it is LOW (0V) if it sees white area.

We have found the average and sum of the output of these sensors to be able to find the actual position and the error. According to the error, the given power to the motors is changing so we can stay on the black line. By using PID we can control the vehicle on the road at higher speed. It is a great advantage of using PID especially while passing under the gates.

4.2 The Design Requirements of the System

We can use the design parameters of the system which are described in controlling the DC motor speed.

- Settling time should be less than 2 seconds;
- Overshoot of the system should be less than 5%;
- Steady state error should be less than 1%

Let's start finding the P-I-D parameters from the proportional control constant.If we choose the proportional constant too low, we have large settling time.ForKp=1:

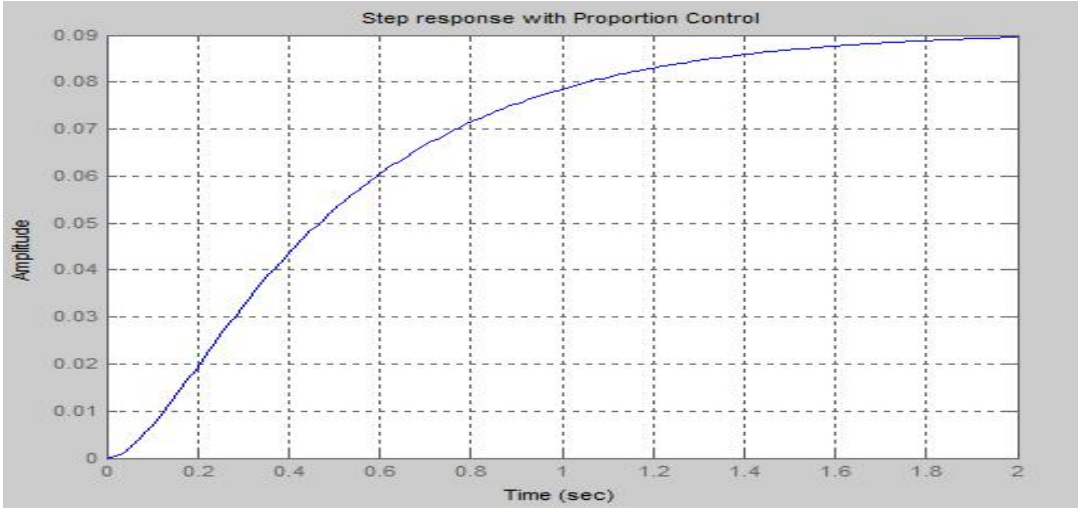

Figure 4.2.1-The Step Response of the DC Motor for Kp=1

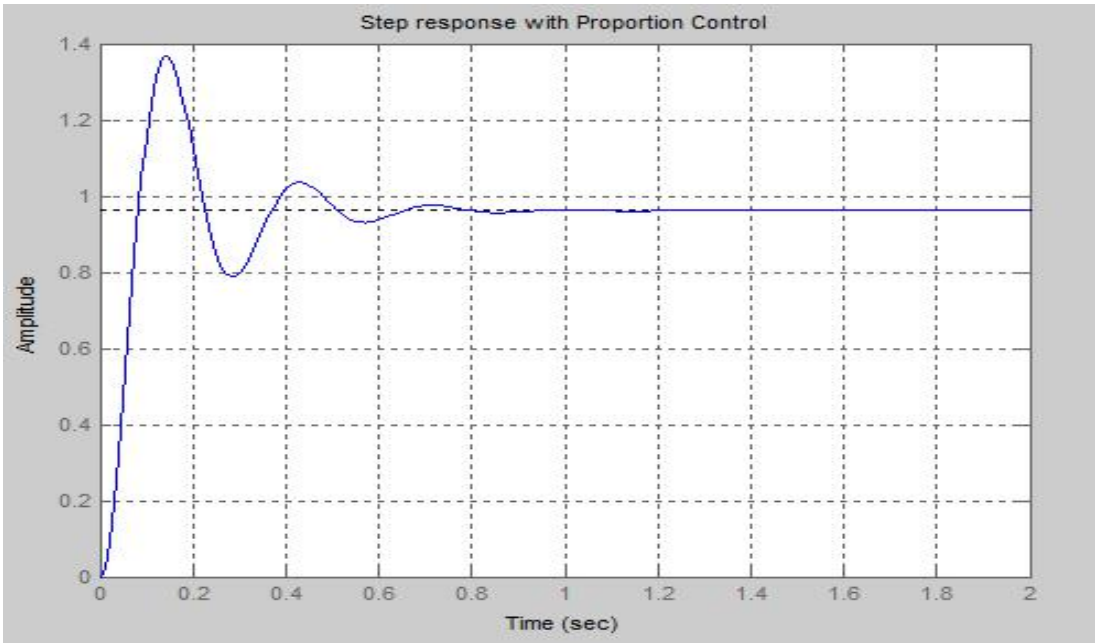

Figure 4.2.2- The Step Response of the DC Motor for Kp=250

The step response for Kp=250 seems to satisfy the system requirement. We have overshoot and steady state error but they can be improved with the addition of Ki and Kd ,ForKp=3000:

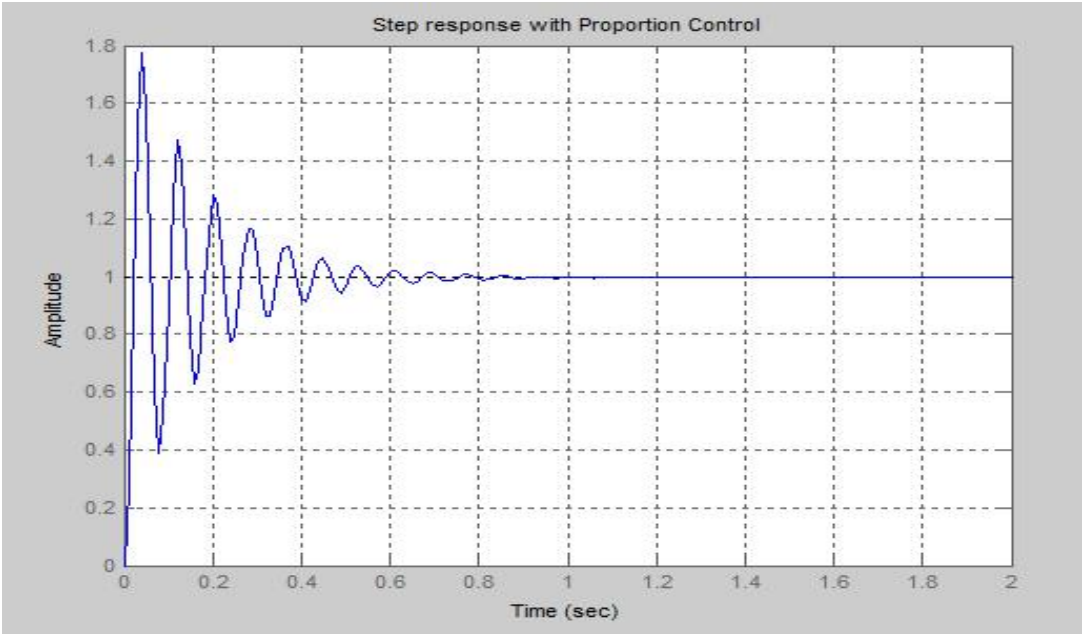

Figure 4.2.3- The Step Response of the DC Motor for Kp=3000

We have instability of the system for Kp=3000. The logical choice for Kp would be 100.After the addition of integral and derivative controller, the step response of the system would be as follows;

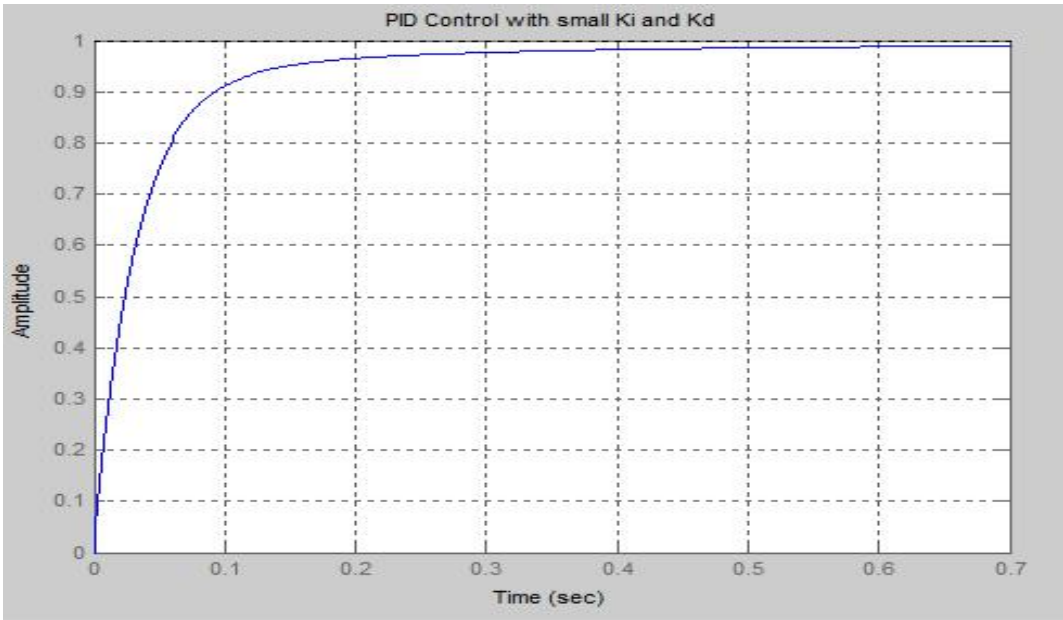

Figure 4.2.4-- The Step Response of the DC Motor for Kp=100, Ki=200, Kp=10

The requirements are satisfied for continuous time system. As it is done in controlling DC motor speed control, we have to pass from s*-domain to z-domain.The transfer function of the DC motor in z-domain is the following

4.3 The Transfer Function of the DC Motor with Zero Order Hold:

$$T(z) = \frac{0.0010389\,(z+0.9831)\,(z+9.256e\text{-}007)}{z\,(z-1)\,(z-0.9425)}$$

$$T(z) = \frac{0.0010389\,(z+0.9831)}{(z-1)\,(z-0.9425)}$$

Now let find the step response of the DC motor in z domain;

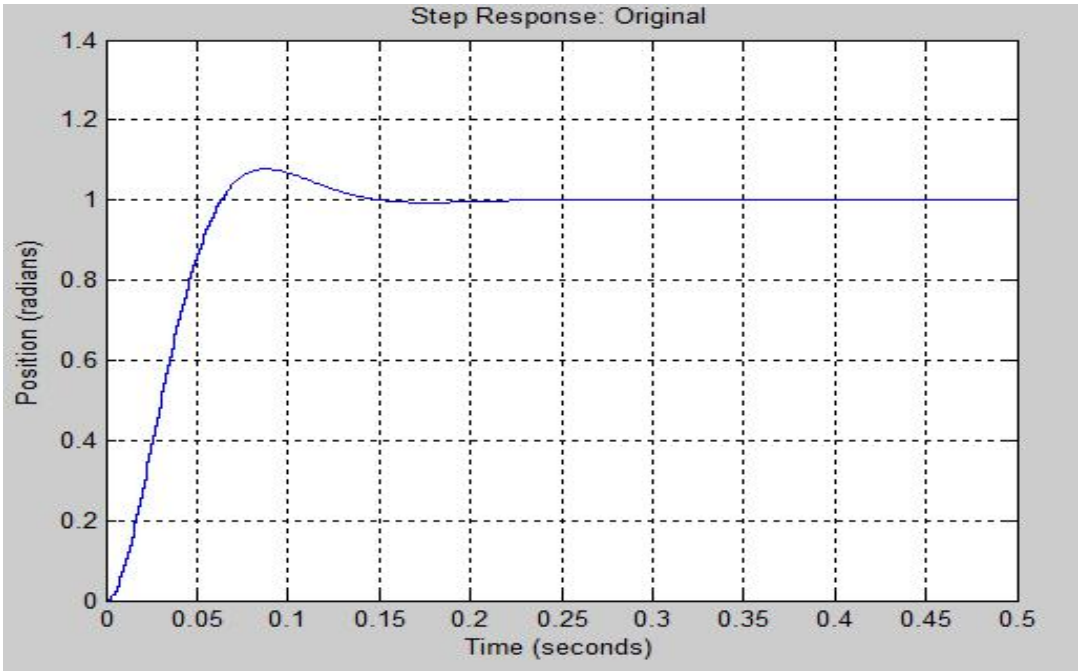

Figure 4.3.1-The Step Response of the DC Motor with ZOH

The step response of the system does not satisfy the design requirements. To find the reason of instability let us draw the root locus of the compensated system.The Root Locus of the Compensated System:

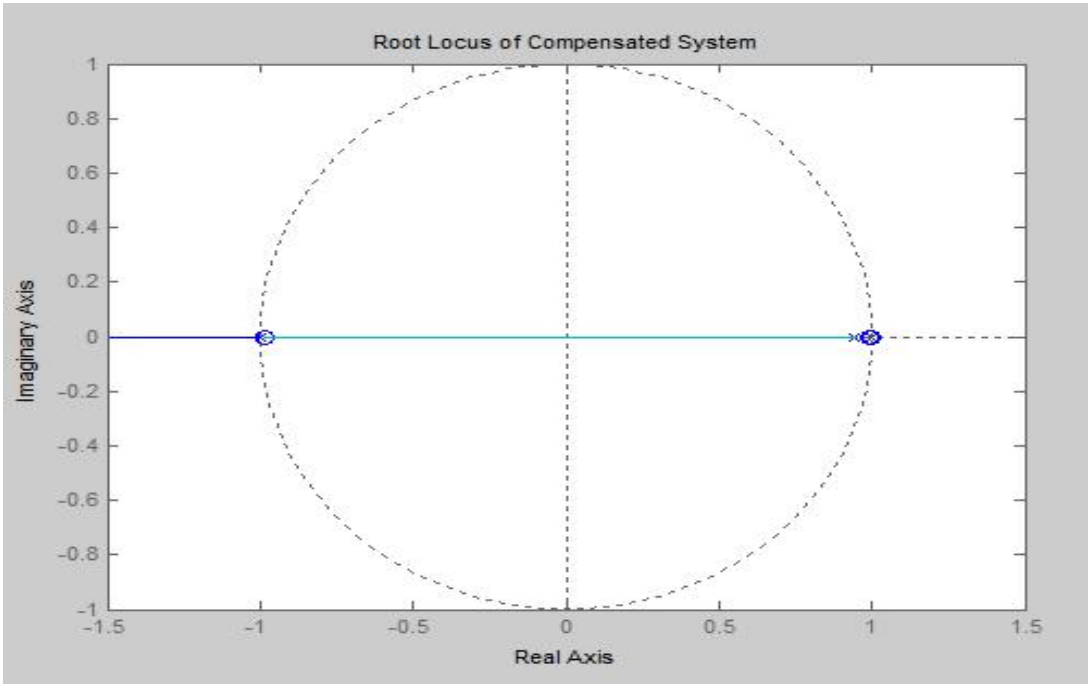

Figure 73-The Root locus of the Compensated System

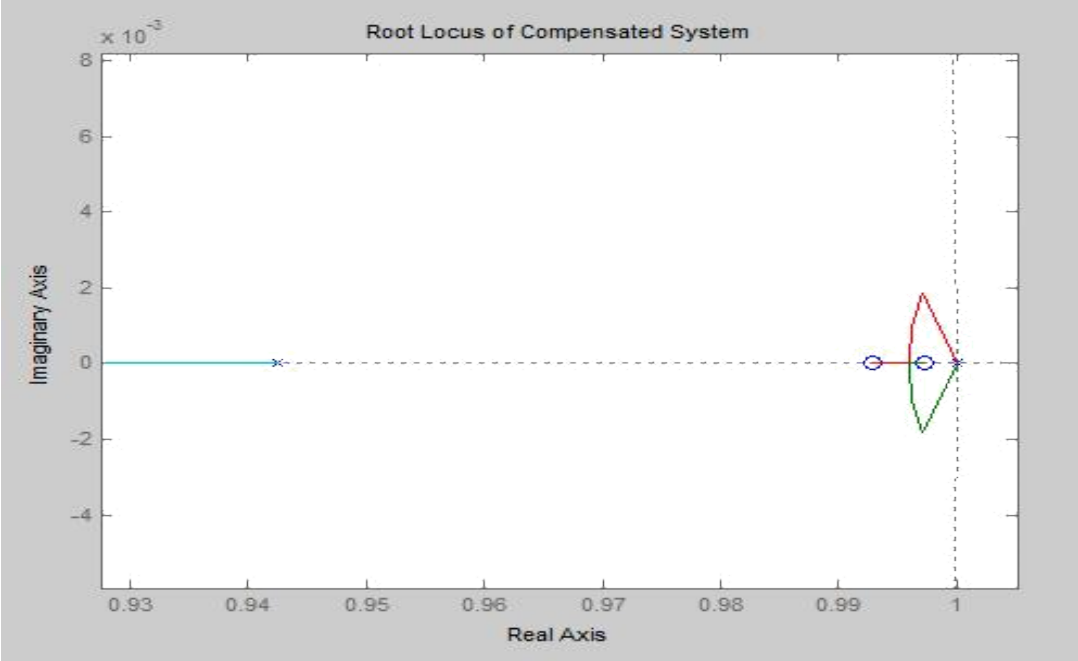

Figure 4.3.3-The RHS of the Root Locus

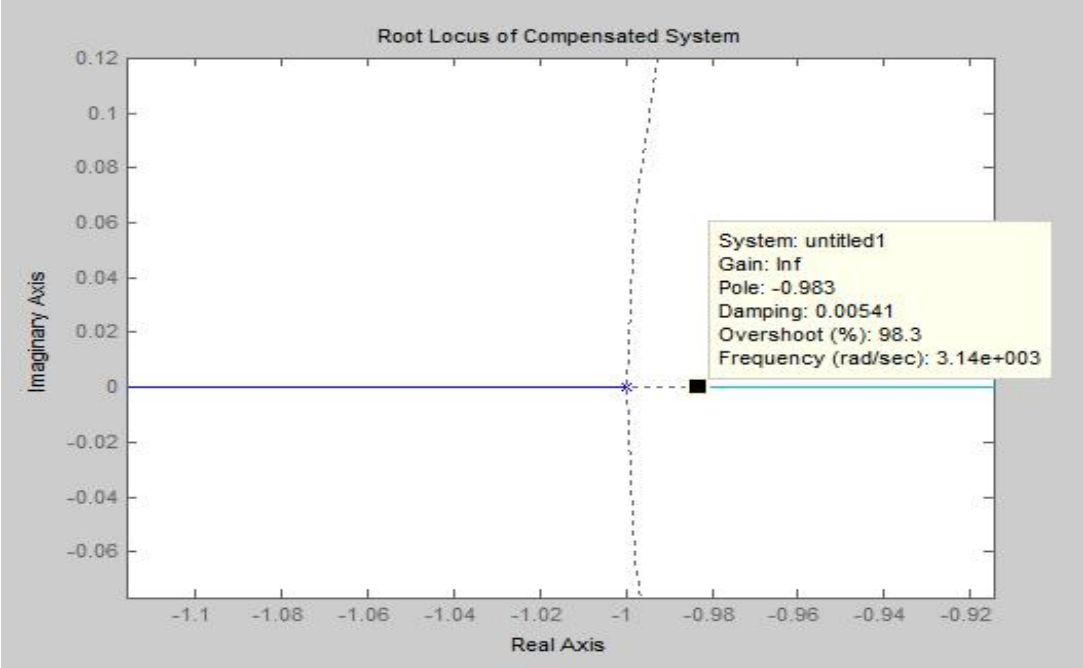

Figure 4.3.4- The LHS of the Root Locus

The pole at -1 goes to infinity as the gain of the system (K) increases. It causes the instability of the system. To make the system stable, let us add a pole at -0.983.

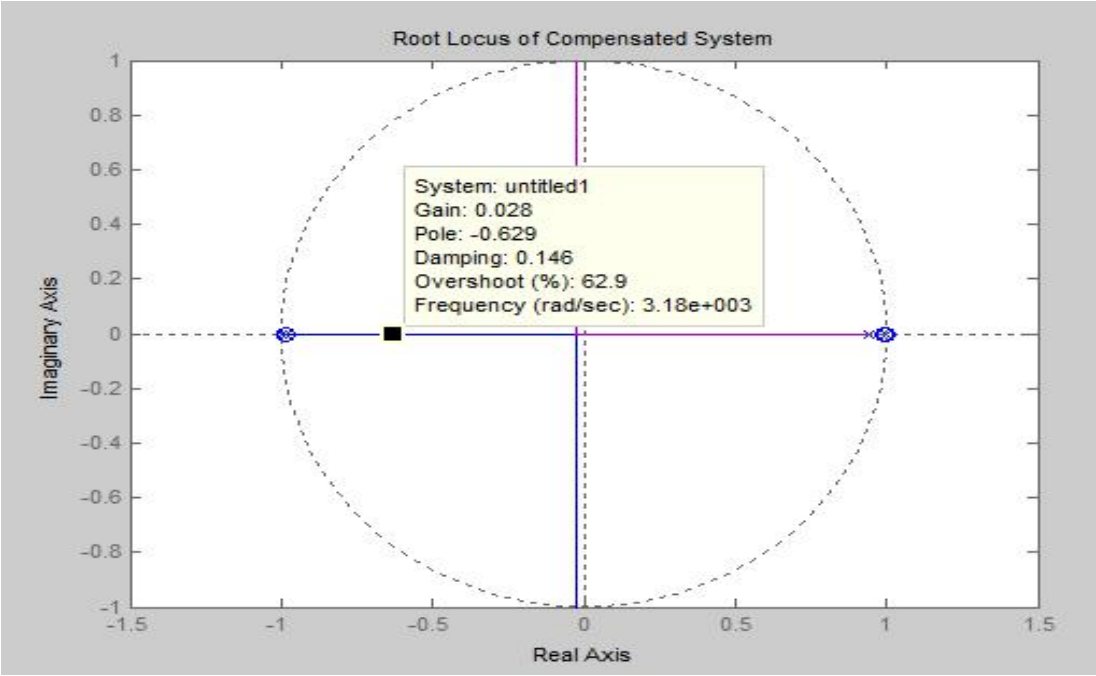

Figure 4.3.5-After the Addition of the Pole at -0.983

The system is stable if the gain value of the system is chosen in unit circle. Let us choose the gain value as 0.0028.

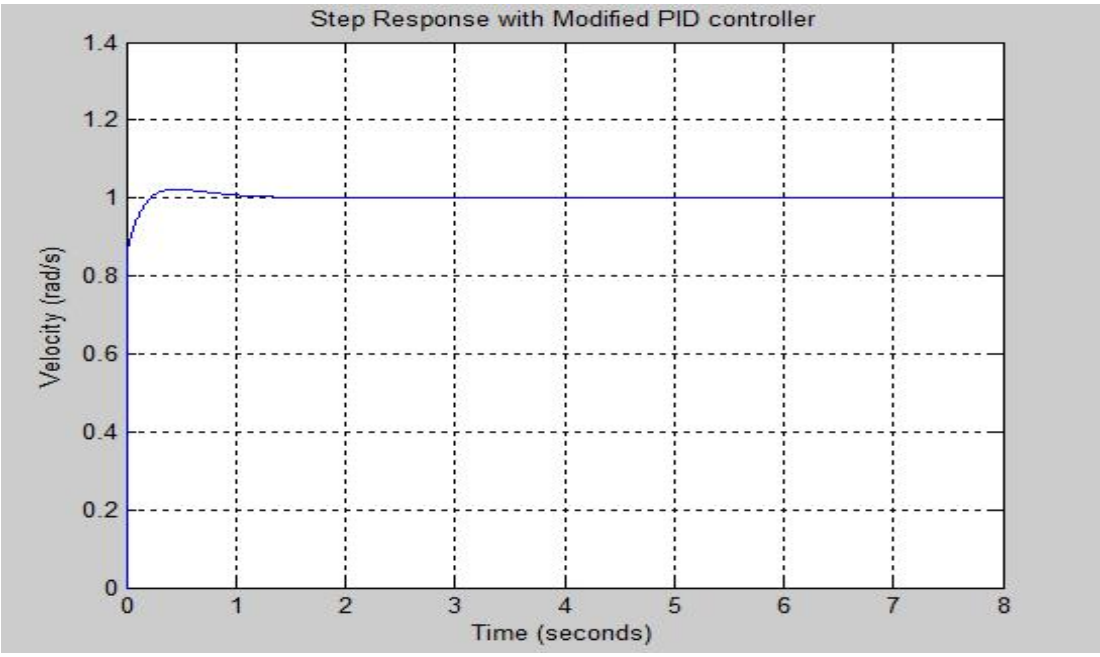

Figure 4.3.6-The Step Response of the System for gain=0.0028

Up to that point we have explained the individual control of speed and position of the DC motor. These two systems should be combined for the gate-vehicle project.

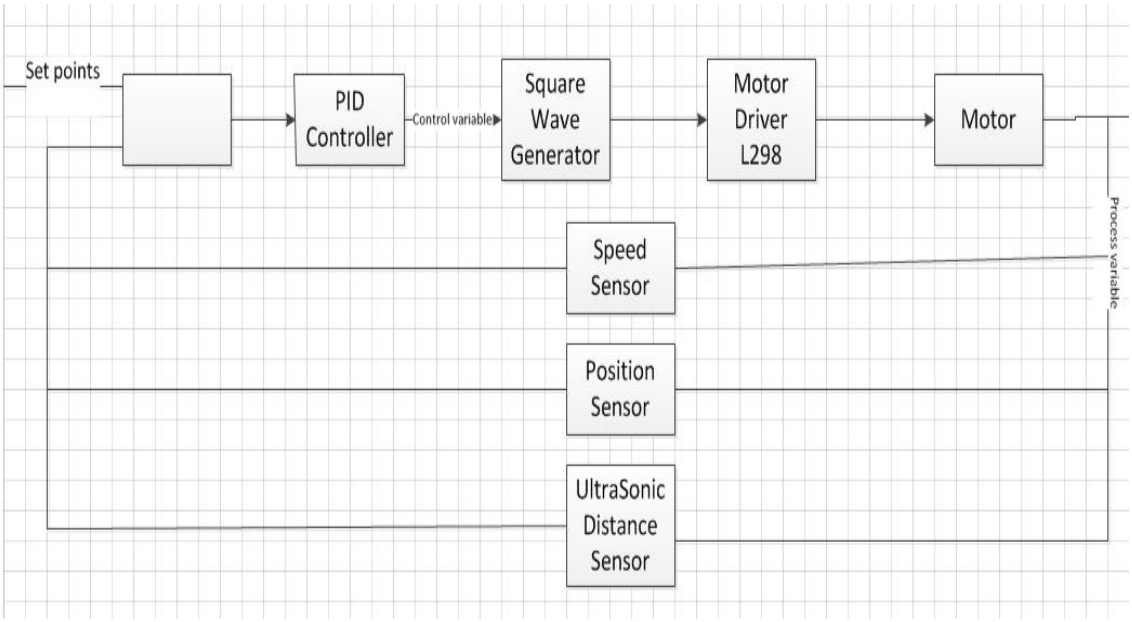

Figure 4.3.7-The Block Diagram of Both Speed and Position Control of DC Motor

CHAPTER 5

BACKGROND OF PROJECT

5.1 GENERAL

Permanent magnet direct current motor (PMDC) have been widely use in high-performance electrical drives and servo system. There are many difference DC motor types in the market and all with it good and bad attributes. Such bad attribute is the lag of efficiency. In order to overcome this problem a controller is introduce to the system.

There are also many types of controller used in the industry, such controller is PID controller. PID controller or proportional–integral–derivative controller is a generic control loop feedback mechanism widely used in industrial control systems. A PID controller attempts to correct the error between a measured process variable and a desired set point by calculating and then outputting a corrective action that can adjust the process accordingly. So by integrating the PID controller to the DC motor were able to correct the error made by the DC motor and control the speed or the position of the motor to the desired point or speed.

5.2 Problem Statement

The problem encounter when dealing with DC motor is the lag of efficiency and losses. In order to eliminate this problem, controller is introduce to the system. There's few type of controller but in this project, PID controller is chosen as the controller for the DC motor. This is because PID controller helps get the output, where we want it in a short time, with minimal overshoot and little error.

5.3 Permanent Magnet Direct Current Motor

A DC motor is designed to run on DC electric power [3]. An example is Michael Faraday's homopolar motor, and the ball bearing motor. There are two types of DC motor which are brush and brushless types, in order to create an oscillating AC current from the DC source and internal and external commutation is use respectively. So they are not purely DC machines in a strict sense .

A brushless DC motor (BLDC) is a synchronous electric motor which is powered by direct-current electricity (DC) and which has an electronically controlled commutation system, instead of a mechanical commutation system based on brushes [4]. In such motors, current and torque, voltage and rpm are linearly related [4]. BLDC has its own advantages such as higher efficiency and reliability, reduced noise, longer lifetime, elimination of ionizing sparks from the commutator, and overall reduction of electromagnetic interference (EMI). With no windings on the rotor, they are not subjected to centrifugal forces, and because the electromagnets are located around the perimeter, the electromagnets can be cooled by conduction to the motor casing, requiring no airflow inside the motor for cooling [4]. The disadvantageis higher cost, because of two issues. First, it requires complex electronic speed controller to run.

5.4 Control Theory

Control theory is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems [7]. The desired output of a system is called the *reference* [7]. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system [7].

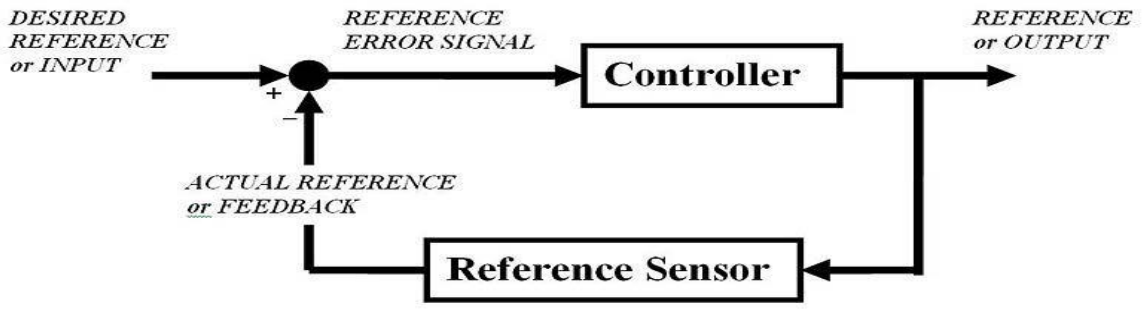

Figure 5.4 Concept of the Feedback Loop to Control the Dynamic Behavior of the reference

If we consider an automobile cruise control, it is design to maintain the speed of the vehicle at a constant speed set by the driver. In this case the system is the vehicle. The vehicle speed is the output and the control is the vehicle throttle which influences the engine torque output. One way to implement cruise control is by locking the throttle at the desired speed but when encounter a hill the vehicle will slow down going up and accelerate going down. In fact, any parameter different than what was assumed at design time will translate into a proportional error in the output velocity, including exact mass of thevehicle, wind resistance, and tire pressure [7]. This type of controller is called an open-loop controller because there is no direct connection between the output of the system (the engine torque) and the actual conditions encountered; that is to say, the system does not and cannot compensate for unexpected forces [7].

For a closed-loop control system, a sensor will monitor the vehicle speed and feedback the data to its computer and continuously adjusting its control input or the throttle as needed to ensure the control error to a minimum therefore maintaining the desired speed of the vehicle. Feedback on how the system is actually performing allows the controller (vehicle's on board computer) to dynamically compensate for disturbances to the system, such as changes in slope of the ground or wind speed [7]. An ideal feedback control system cancels out all errors, effectively mitigating the effects of any forces that may or may not arise during operation and producing a response in the system that perfectly matches the user's wishes [7].

5.5 Closed-Loop Transfer Function

The output of the system *y(t)* is fed back through a sensor measurement *F* to the reference value *r(t)*. The controller *C* then takes the error *e* (difference) between the reference and the output to change the inputs *u* to the system under control *P*. This is shown in the figure. This kind of controller is a closed-loop controller or feedback controller. This is called a single-input-single-output (*SISO*) control system; *MIMO* (i.e. Multi-Input-Multi-Output) systems, with more than one input/output, are common. In such cases variables are represented through vectors instead of simple scalar values. For some distributed parameter systems the vectors may be infinite-dimensional (typically functions).

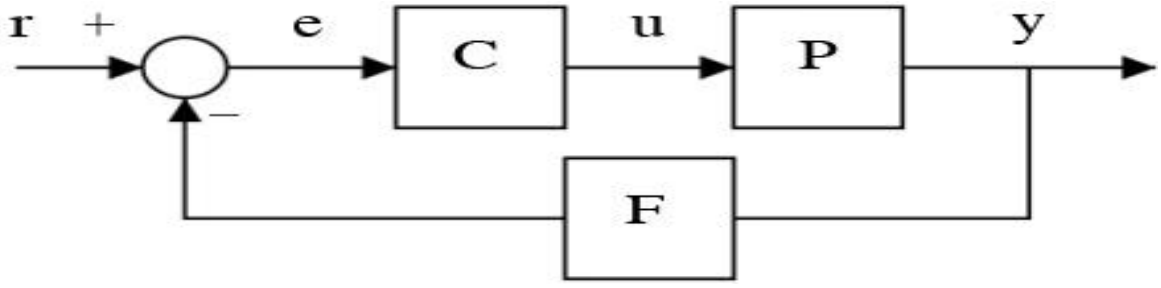

Figure 5.5 Closed-loop controller or feedback controller

If we assume the controller *C*, the plant *P*, and the sensor *F* are linear and time-invariant (i.e.: elements of their transfer function *C(s)*, *P(s)*, and *F(s)* do not depend on time), the systems above can be analyzed using the Laplace transform on the variables. This gives the following relations:

$$Y(s) = P(s)U(s)$$
$$U(s) = C(s)E(s)$$
$$E(s) = R(s) - F(s)Y(s)$$

Solving for $Y(s)$ in terms of $R(s)$ gives:

$$Y(s) = \left(\frac{P(s)C(s)}{1+F(s)P(s)C(s)}\right) R(s) = H(s)R(s)$$

The expression

$$H(s) = \left(\frac{P(s)C(s)}{1+F(s)P(s)C(s)}\right)$$

(open-loop) gain from *r* to *y*, and the denominator is one plus the gain in going around the is referred to as the closed-loop transfer function of the system. The numerator is the forward feedback loop, the so called loop gain.

CHAPTER 6

P I D CONTROLLER

6.1 GENERAL

PID Control (proportional-integral-derivative) is by far the widest type of automatic control used in industry. Even though it has a relatively simple algorithm/structure, there are many subtle variations in how it is applied in industry [5]. A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism widely used in industrial control systems [1]. A PID controller will correct the error between the output and the desired input or set point by calculating and give an output of correction that will adjust the process accordingly. A PID controller has the general form

$$u(t)= Kp\ e(t) + Ki \int_0^t e(T)\ dT + Kd \frac{de}{dt}$$

Where K_p is proportional gain, K_i is the integral gain, and K_d is the derivative gain.

The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values [1]. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing [1]. 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, the power supply of a heating element or DC motor speed and position.

6.2 Pulse Width Modulation

Pulse-width modulation (PWM) of a signal or power source involves the modulation of its duty cycle, to either convey information over a communications channel or control the amount of power sent to a load.

Pulse-width modulation uses a square wave whose pulse width is modulated resulting in the variation of the average value of the waveform. If we consider a square waveform $f(t)$ with a low value ymin, a high value ymax and a duty cycle D (see figure 6.2.1), the average value of the waveform is given by:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)\,dt$$

—

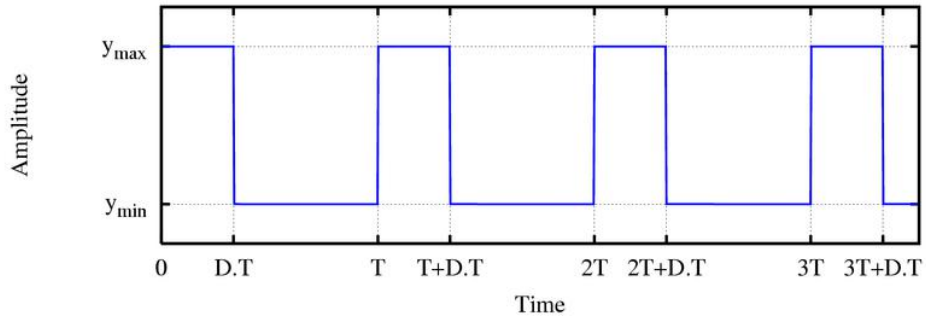

Figure 6.2: A Square Wave showing the definitions of ymin, ymax and D

As $f(t)$ is a square wave, its value is ymax for $0 < t < D.T$ and ymin for $D.T < t < T$. The above expression then becomes:

$$\bar{Y} = \frac{1}{T}\left(\int_0^{DT} y_{max}\,dt + \int_{DT}^{T} y_{min}\,dt\right)$$

$$= D.T.\,y_{max} + T(1-D)\,y_{min}$$

$$= D.\,y_{max} + (1-D)\,y_{min}$$

This latter expression can be fairly simplified in many cases where $y_{min} = 0$ as

$$y = D.\ y_{max}$$

From this, it is obvious that the average value of the signal ($\bar{y}$) is directly dependent on the duty cycle D.

The simplest way to generate a PWM signal is the intersective method, which requires only a sawtooth or a triangle waveform (easily generated using a simple oscillator) and a comparator. When the value of the reference signal (the green sine wave in figure 2.4) is more than the modulation waveform (blue), the PWM signal (magenta) is in the high state, otherwise it is in the low state.

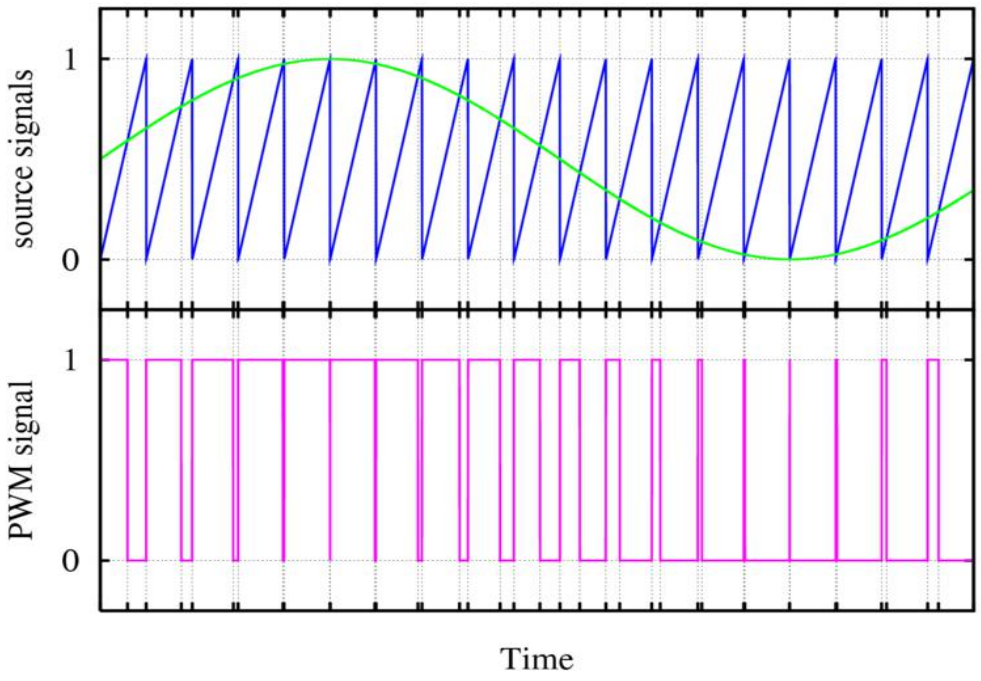

Figure 6.2.2: PWM Pulse Generate from Comparing Sinewave and Sawtooth

CHAPTER 7

MATLAB AND SIMULINK

7.1 GENERAL

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

6. Math and computation
7. Algorithm development
8. Data acquisition
9. Modeling, simulation, and prototyping
10. Data analysis, exploration, and visualization
11. Scientific and engineering graphics
12. Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized.

Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs.

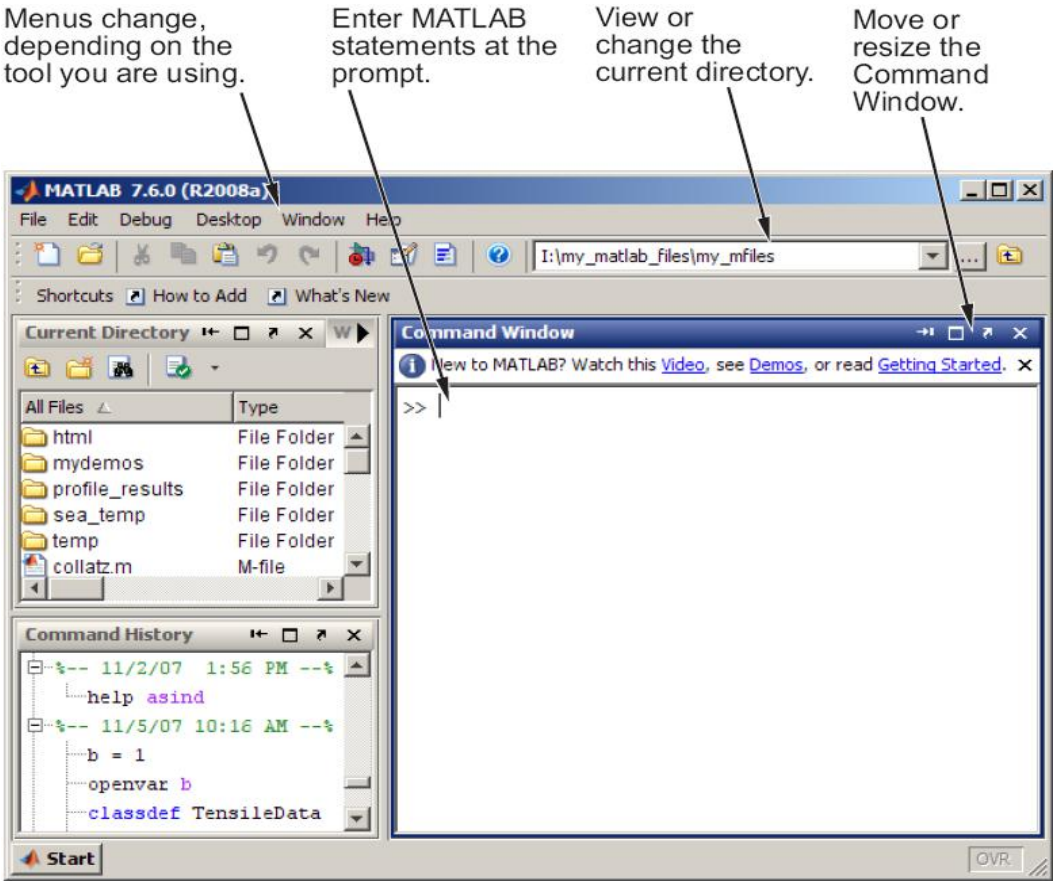

Figure 7.1.1: MATLAB default command windows

Simulink is software for modeling, simulating, and analyzing dynamic systems. Simulink enables you to pose a question about a system, model it, and see what happens with Simulink, you can easily build models from scratch, or modify existing models to meet your needs. Simulink supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate having different parts that are sampled or updated at different rates.

Thousands of scientists and engineers around the world use Simulink® to model and solve real problems in a variety of industries, including:

1. Aerospace and Defense
2. Automotive
3. Communications
4. Electronics and Signal Processing
5. Medical Instrumentation

Model analysis tools include linearization and trimming tools, whichcan be accessed from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes. Because MATLAB® and Simulink are integrated; you can simulate, analyze, and revise your models in either environment at any point.

Simulink is tightly integrated with MATLAB. It requires MATLAB to run, depending on MATLAB to define and evaluate model and block parameters. Simulink can also utilize many MATLAB features. For example, Simulink can use MATLAB to:

1. Define model inputs.
2. Store model outputs for analysis and visualization.
3. Perform functions within a model, through integrated calls to MATLAB operators and functions.

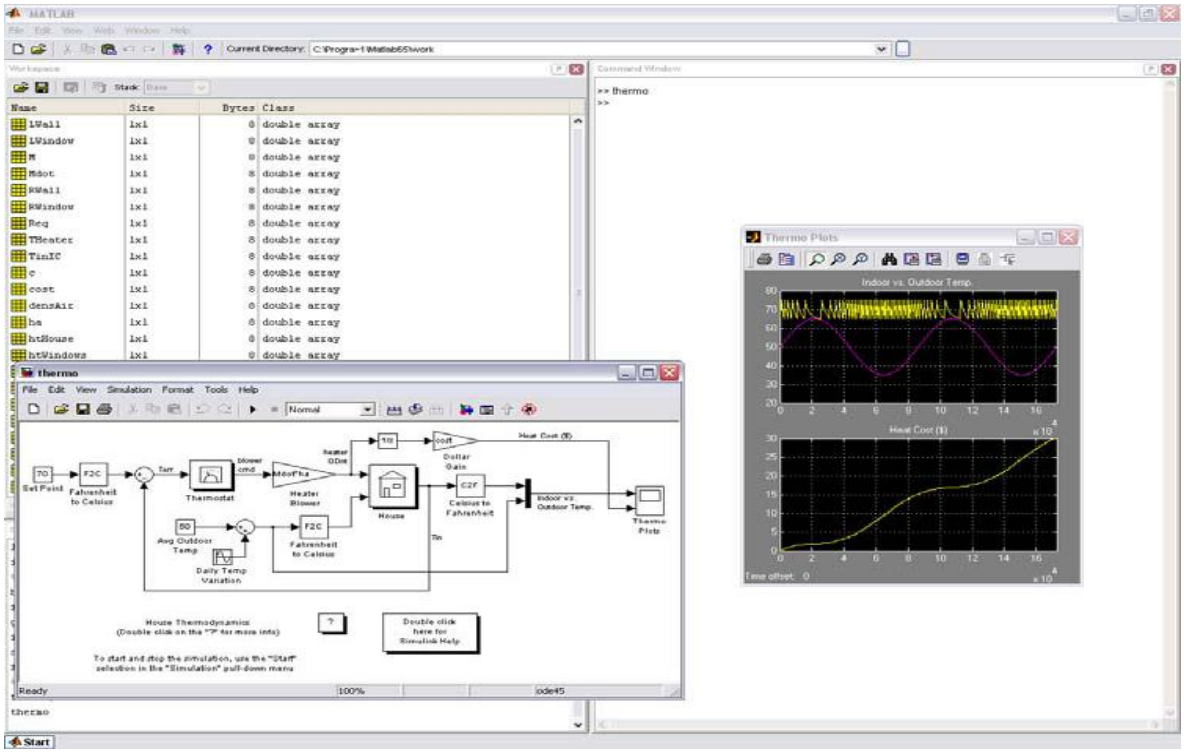

Figure 7.1.2: Simulink Running a Simulation of a Thermostat-Controlled Heating System

7.2 System Description

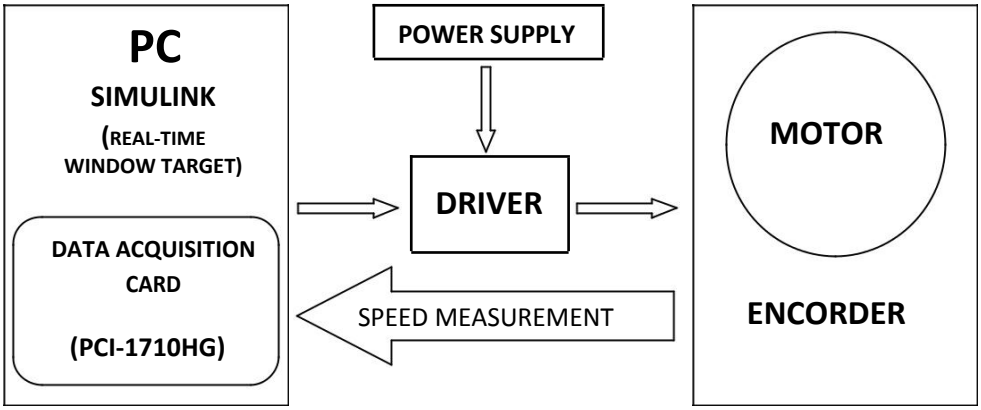

Figure 7.2: Block Diagram of the System

The system block diagram is as shown in Figure 3.1. It consist of 2 main block (PC and Motor) that are connected through a driver and supplied by a power supply. The control algorithm is builded in the Matlab/Simulink software and compiled with Real-Time Window Target. The Real-Time Window Target Toolbox include an analog input and analog output that provide connection between the data acquisition card (PCI-1710HG) and the simulink model. For example, the speed of the DC motor could be controlled by supplying certain voltage and frequency from signal generator block to the analog output in Simulink. From the analog input, the square received is displayed in a scope. The square wave pulse then is derived using the velocity equation to get the velocity of the DC motor speed. The speed acquired and the signal send can create a closed loop system with PID controller to control the speed of the DC motor. Figure 3.2 to Figure 3.9 shows the DC motor, driver, and other hardwareused in this project and the DC motor specification.

7.3 HARDWARE

Figure 7.3.1 to Figure 7.3.7 shows the DC motor, driver, and other hardware used in this project and the DC motor specification.

Figure 7.3.1:DC Motor

Figure 7.3.2 Personal Computer

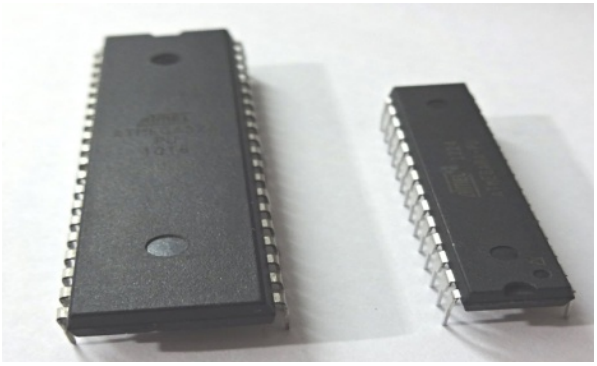

Figure 7.3.3:Micro Controller IC

Figure7.3.4 AT Mega 8 IC

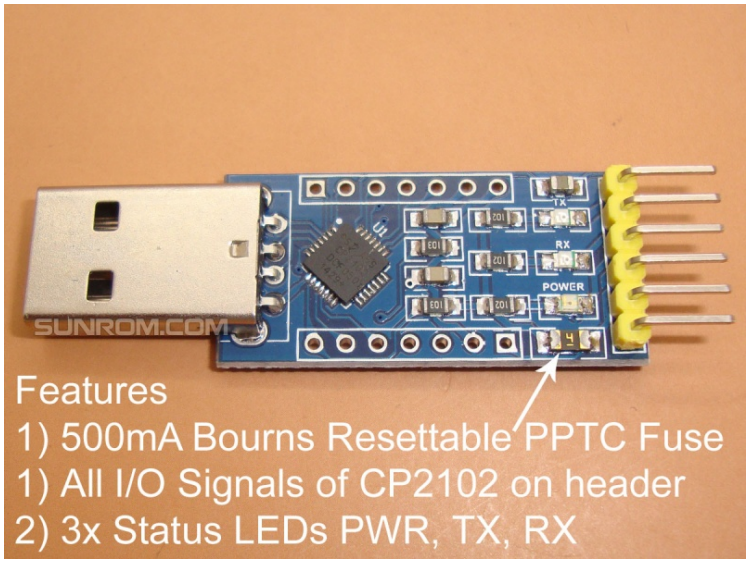

Figure 7.3.5 USB TTL

Figure 7.3.6-Jumper Wires Male-Female

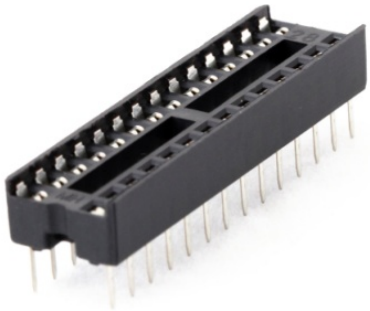

Figure7.3.7 IC Base 28 pin

CHAPTER 8

8.1 MATLAB-CODE

```
function varargout = Motor_control(varargin)
% MOTOR_CONTROL MATLAB code for Motor_control.fig
%      MOTOR_CONTROL, by itself, creates a new MOTOR_CONTROL or raises the existing
%      singleton*.
%
%      H = MOTOR_CONTROL returns the handle to a new MOTOR_CONTROL or the handle
to
%      the existing singleton*.
%
%      MOTOR_CONTROL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MOTOR_CONTROL.M with the given input arguments.
%
%      MOTOR_CONTROL('Property','Value',...) creates a new MOTOR_CONTROL or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Motor_control_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Motor_control_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Motor_control

% Last Modified by GUIDE v2.5 22-Apr-2017 22:21:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Motor_control_OpeningFcn, ...
                   'gui_OutputFcn',  @Motor_control_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
```

```
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Motor_control is made visible.
function Motor_control_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Motor_control (see VARARGIN)

% Choose default command line output for Motor_control
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Motor_control wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Motor_control_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
```

```
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');


s = serial(port);
fopen(s);
fprintf(s,'0');
fclose(s);
set(handles.edit1,'String','0%');


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');



s = serial(port);
fopen(s);
fprintf(s,'1');
fclose(s);
set(handles.edit1,'String','20%');


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');


s = serial(port);
fopen(s);
fprintf(s,'2');
fclose(s);
set(handles.edit1,'String','40%');


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');



s = serial(port);
fopen(s);
fprintf(s,'3');
fclose(s);
set(handles.edit1,'String','60%');



% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');



s = serial(port);
fopen(s);
fprintf(s,'4');
fclose(s);
set(handles.edit1,'String','80%');


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
port = get(handles.edit2,'String');
s = serial(port);
fopen(s);
fprintf(s,'5');
fclose(s);
set(handles.edit1,'String','100%');


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
global flag1
flag1 = get(hObject,'Value');

%save('flag1.mat','flag1');

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2

global  flag2
flag2 = get(hObject,'Value');

%save('flag2.mat','flag2');

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
global flag3
flag3 = get(hObject,'Value');
```

```
%save('flag3.mat','flag3');



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global flag1 flag2 flag3

J=0.01;     %kg*m^2
b=0.1;      %N*m/(rad/s)
K=0.01;     %V/(rad/s)
R=1;        %Ohm
L=0.5;      %H


A = [-b/J K/J; -K/L -R/L];
B = [0 -1/J; 1/L 0];
C = [1 0];
D = [0 0];

[sys] = ss(A,B,C,D);
```

```
[num,den] = ss2tf(A,B,C,D,1);
sys = tf([num],[den]);


figure,
bode(sys);

figure,
step(sys);

%load('flag1.mat');
%load('flag2.mat');
%load('flag3.mat');



flag1
flag2
flag3

if(flag1==1)
   C = pidtune(sys,'pi');
   Kp = C.Kp;

    Ki = C.Ki;
    sys = pid(Kp,Ki,0);

   axes(handles.axes1);
   bode(sys);

end

if(flag2==1)
   C = pidtune(sys,'pd');
   Kp = C.Kp;

   Kd = C.Kd;

   sys = pid(Kp,Kd,0);

   axes(handles.axes1);
   bode(sys);
```

```
end

if(flag3==1)
   C = pidtune(sys,'pid');
    Kp = C.Kp;

    Kd = C.Kd;

    Ki = C.Ki;


   sys = pid(Kp,Ki,Kd,0);

   axes(handles.axes1);
   bode(sys);

end


function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

Conclusions

P-I-D control and its variations are commonly used in the industry. They have so many applications. Control engineers usually prefer P-I controllers to control first order plants. On the other hand, P-I-D control is vastly used to control two or higher order plants. In almost all cases fast transient response and zero steady state error is desired for a closed loop system. Usually, these two specifications conflict with each other which makes the design harder. The reason why P-I-D is preferred is that it provides both of these features at the same time.

In this recitation, it was aimed to explain how one can successfully use P-I-D controllers in their prospective projects. We tried to focus on almost all aspects of P-I-D control. However, it is almost impossible to fit the explanation of P-I-D controllers within one hour. We suggest for the future Discrete Time Control System students to split the P-I-D controller subject into pieces and explain it more than one recitation hour. Being prospective control engineers, we feel lucky to give a presentation on the P-I-D subject. Finally, we encourage prospective control engineers to use P-I-D controllers wherever necessary, especially, when a great controller is required.

REFERENCES

1. http://atmega8.cc/en/uploads/Main/Atmega8-schematic
2. http://masters.donntu.edu.ua/2013/fkita/abakumov/library/article5
3. http://cache.freescale.com/files/sensors/doc/fact_sheet/PROXFAMFS
4. http://google.in/speedcontrol/matlabcode
5. http://wiki/proportional-integral-derivative/speedcontrol