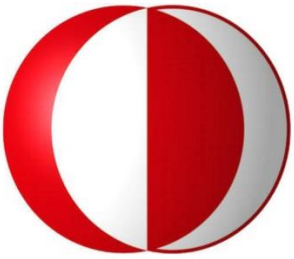

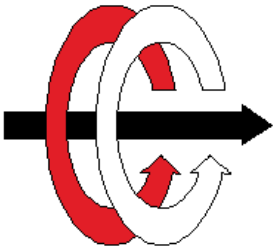

MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

EE402- DISCRETE TIME CONTROL SYSTEMS

RECITATION 4 REPORT

P, PD, PI, PID CONTROLLERS

Group Members:

Sena TEMEL

Semih YAĞLI

Semih GÖREN

Table of Contents

Introduction:

This report is written to analyze the recitation that was presented on 02.04.2013. The recitation was presented by Sena TEMEL, Semih YAĞLI and Semih GÖREN. It was 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 were 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.

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.

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.

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.

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.

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 $\frac{1}{s^2+10s+20}$ and unit step input. The results will lead to tuning methods

Simulations and Results to Find the Constraints on Loop Tuning:

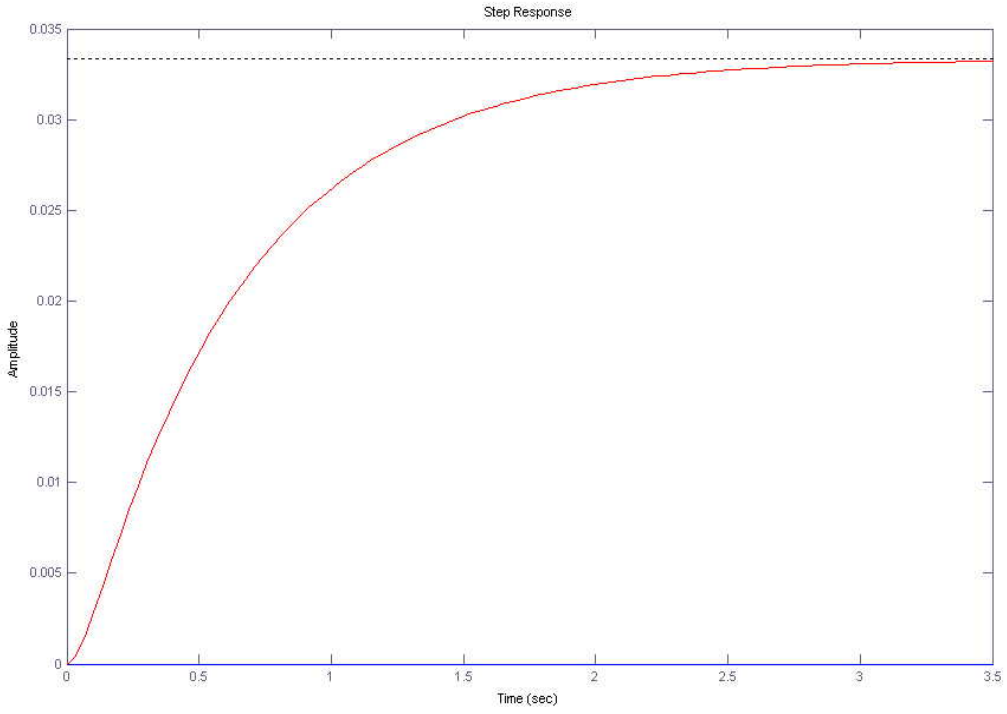

Figure 1: Step response without any controller

Results:

$$Steady\ State\ Error\ = e_{ss} = 0.965\ (too\ high)$$

$$Rise\ Time = t_r \cong 3\ seconds$$

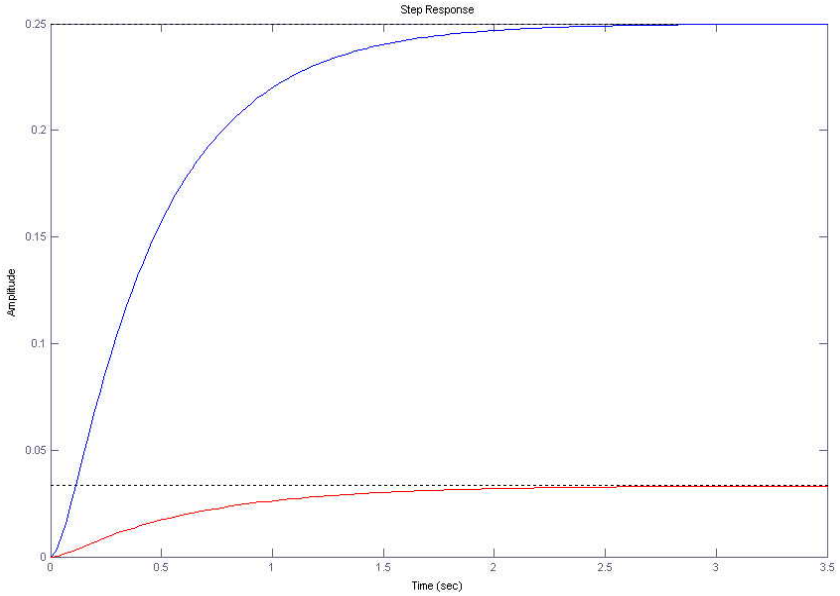

Figure 2: Step response with P controller, Kp = 10, Ki = 0, Kd = 0

Results:

$$Output\ response\ improved$$

$$Rise\ time\ decreased$$

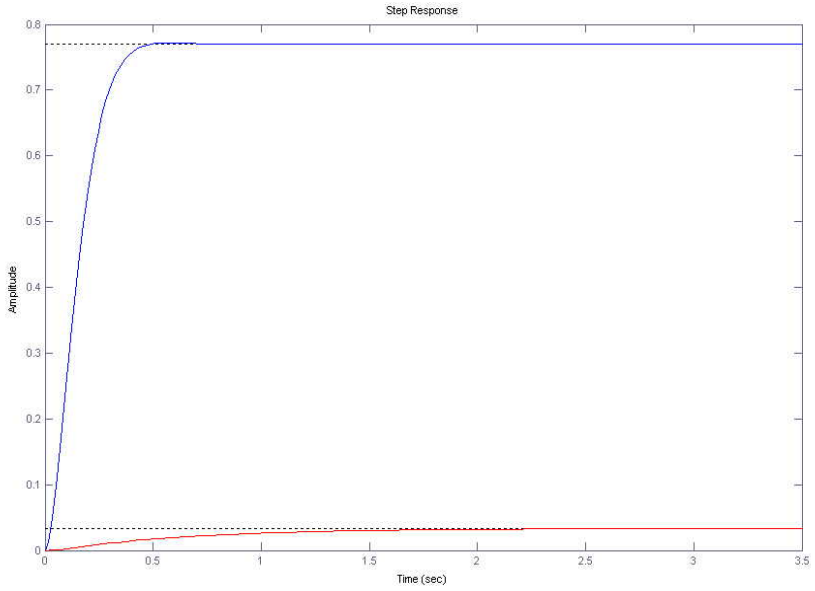

Figure 3: Step response with P controller, Kp = 100, Ki = 0, Kd = 0

Results:

$$Steady\ State\ Error\ = e_{ss} = 0.23\ (decreased)$$

$$Rise\ time\ decreased$$

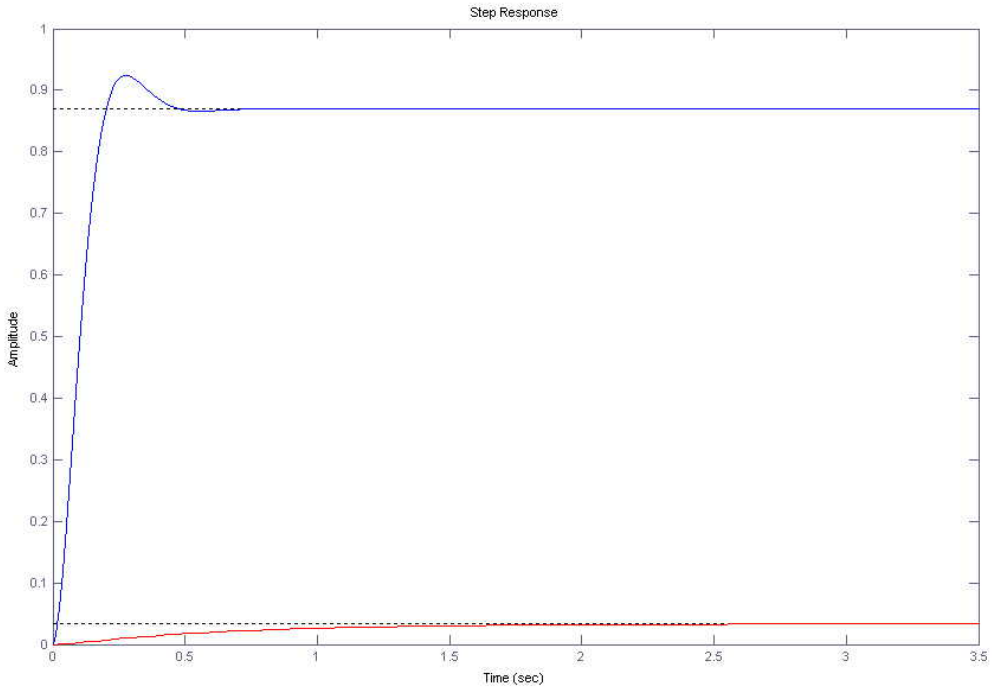

Figure 4: Step response with P controller, Kp = 200, Ki = 0, Kd = 0

Results:

$$Steady\ State\ Error\ = e_{ss} = 0.13\ (decreased)$$

$$Rise\ Time = t_r \cong 0.5\ seconds$$

$$Overshoot\ occurs\ at\ the\ output\ response$$

Conclusions:

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

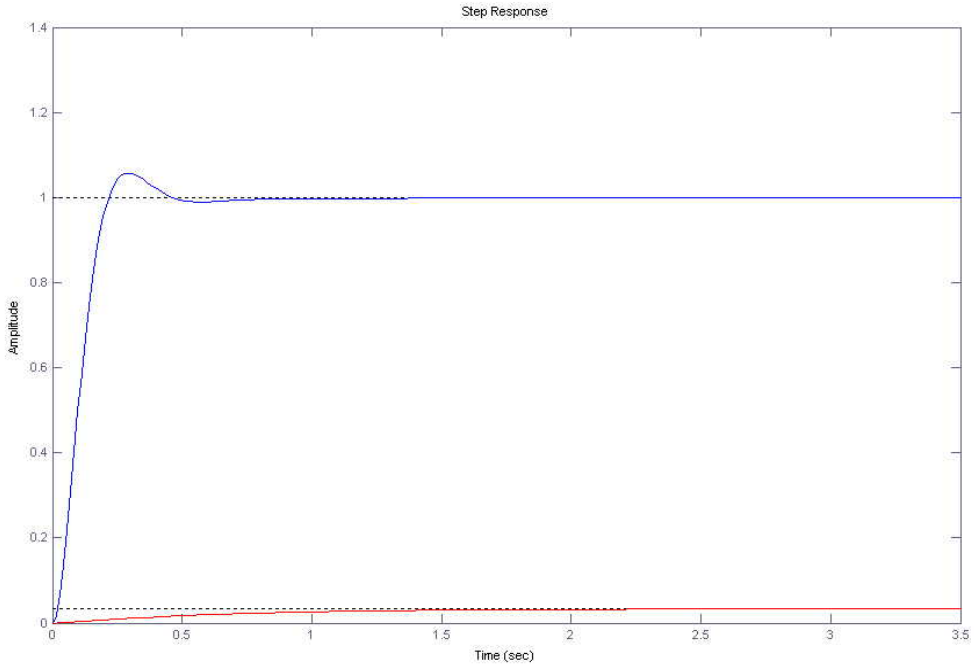

Figure 5: Step response with P-I controller, Kp = 200, Ki = 100, Kd = 0

Results:

$$Steady\ State\ Error\ = e_{ss} = 0\ (eliminated)$$

$$Rise\ Time = t_r \cong 0.3\ seconds$$

$$Settling\ Time = t_s \cong 0.7\ seconds$$

$$Overshoot\ remains$$

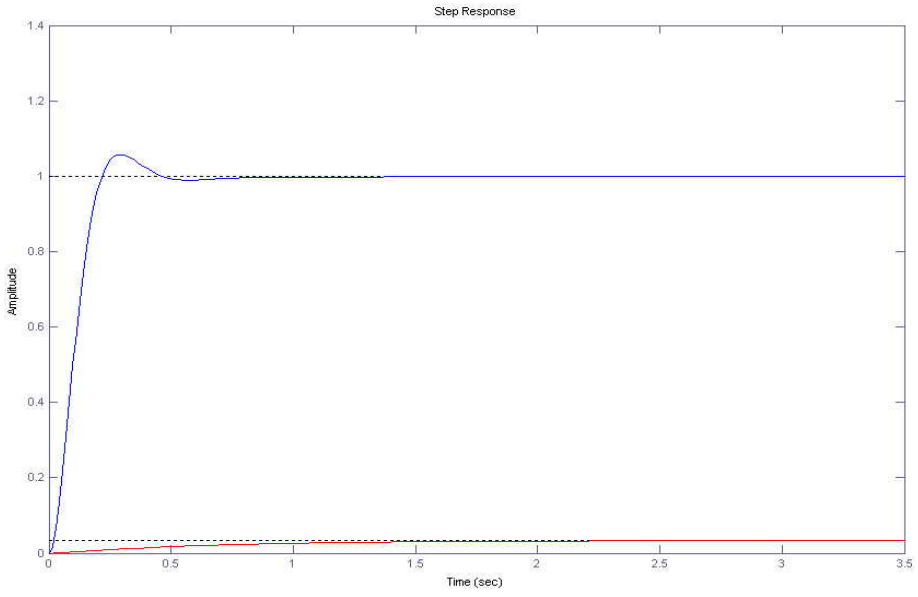

Figure 6: Step response with P-I controller, Kp = 200, Ki = 200, Kd = 0

Results:

$$Steady\ State\ Error\ = e_{ss} = 0$$

$$Rise\ Time = t_r \cong 0.3\ seconds\ (no\ significant\ change)$$

$$Settling\ Time = t_s \cong 0.7\ seconds\ (no\ significant\ change)$$

$$Overshoot\ remains$$

Conclusions:

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

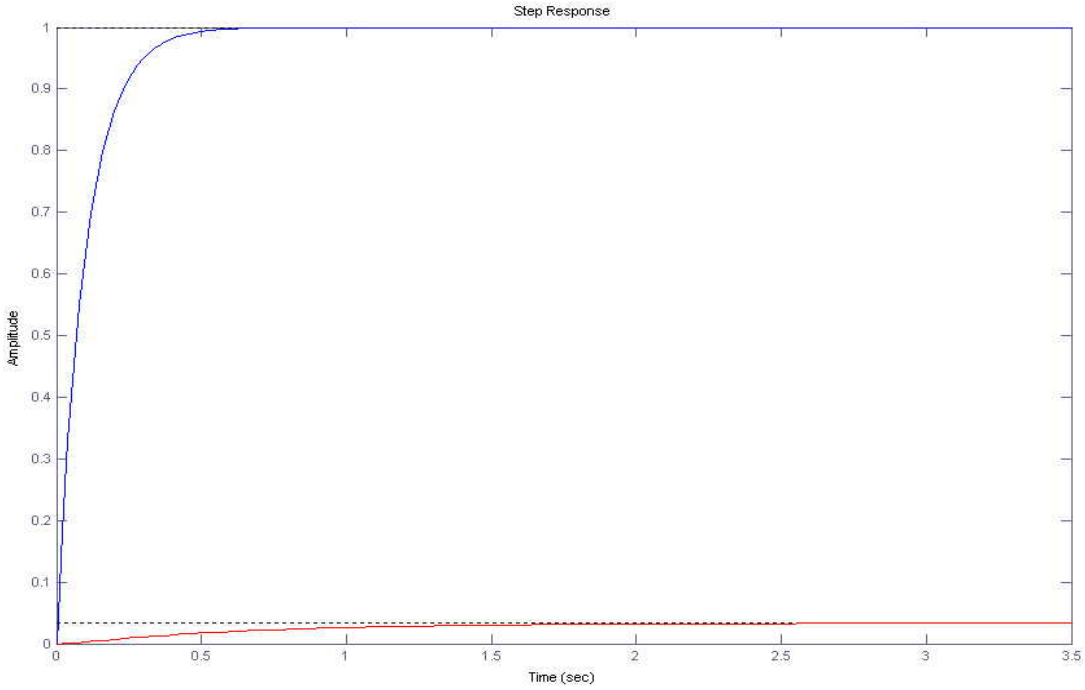

Figure 7: Step response with P-I-D controller, Kp = 200, Ki = 200 and Kd = 10

Results:

$$Steady\ State\ Error\ = e_{ss} = 0$$

$$Rise\ Time = t_r \cong 0.3\ seconds\ (no\ significant\ change)$$

$$Settling\ Time = t_s \cong 0.5\ seconds\ (no\ significant\ change)$$

$$Overshoot\ removed$$

Conclusions:

- Increasing K_d decreases the overshoot.
- Increasing K_d reduces the settling time.

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:

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

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

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.

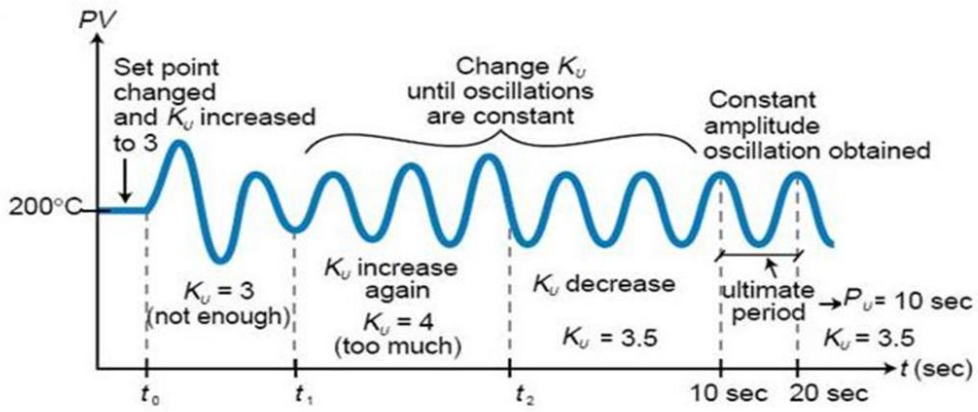

Figure 8: Ziegler-Nichols P-I-D controller tuning method

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	$0.50K_c$	0	0
PI	$0.45K_c$	$1.2K_p dT / P_c$	0
PID	$0.60K_c$	$2K_p dT / P_c$	$K_p P_c / (8dT)$

Figure 9: Ziegler-Nichols P-I-D controller tuning method, adjusting Kp, Ki and Kd

Advantages:

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

Disadvantages:

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

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 10.

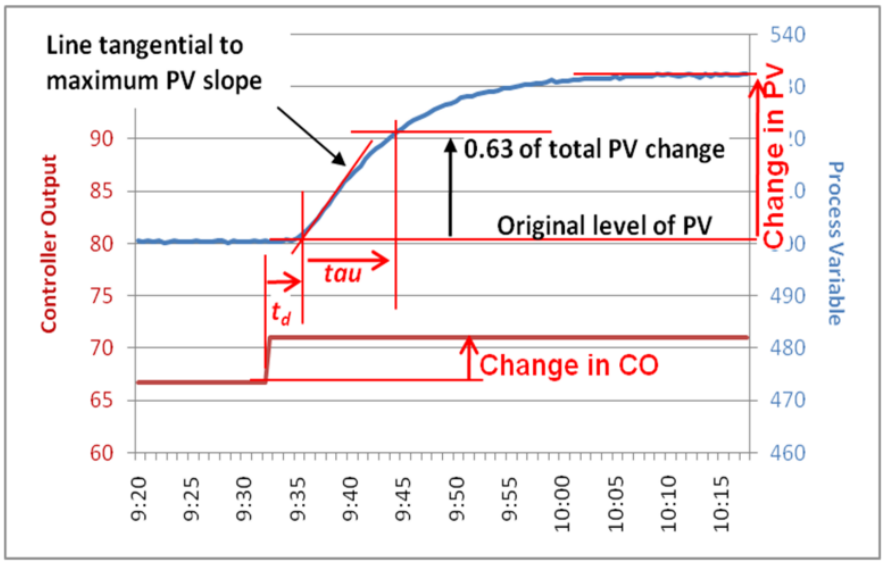

Figure 10: Cohen-Coon P-I-D Tuning Method

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:

$$gp = \frac{\Delta PV}{\Delta CO} \text{ (in \%)}$$

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 the dead time and the

PV 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 11 below.

	Controller Gain	Integral Time	Derivative Time
P Controller:	$K_C = \frac{1.03}{g_p}\left(\frac{\tau}{t_d} + 0.34\right)$		
PI Controller:	$K_C = \frac{0.9}{g_p}\left(\frac{\tau}{t_d} + 0.092\right)$	$T_I = 3.33 t_d \frac{\tau + 0.092 t_d}{\tau + 2.22 t_d}$	
PD Controller:	$K_C = \frac{1.24}{g_p}\left(\frac{\tau}{t_d} + 0.129\right)$		$T_D = 0.27 t_d \frac{\tau - 0.324 t_d}{\tau + 0.129 t_d}$
PID Controller: (Noninteracting)	$K_C = \frac{1.35}{g_p}\left(\frac{\tau}{t_d} + 0.185\right)$	$T_I = 2.5 t_d \frac{\tau + 0.185 t_d}{\tau + 0.611 t_d}$	$T_D = 0.37 t_d \frac{\tau}{\tau + 0.185 t_d}$

Figure 11: Cohen-Coon P-I-D Tuning Method, adjusting Kp, Ki and Kd

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 $\frac{1}{2}$ of the time constant, Cohen-Coon is tolerable until $\frac{3}{4}$ 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.

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.

- **Transient 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:

- Increasing gain decreases rise time (Advantage)
- Increasing gain increases percent overshoot and number of oscillations (Disadvantage)
- Increasing gain decreases steady state error (Advantage)
- Steady state is never zero if only-P type controller is used (Disadvantage)
- 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.

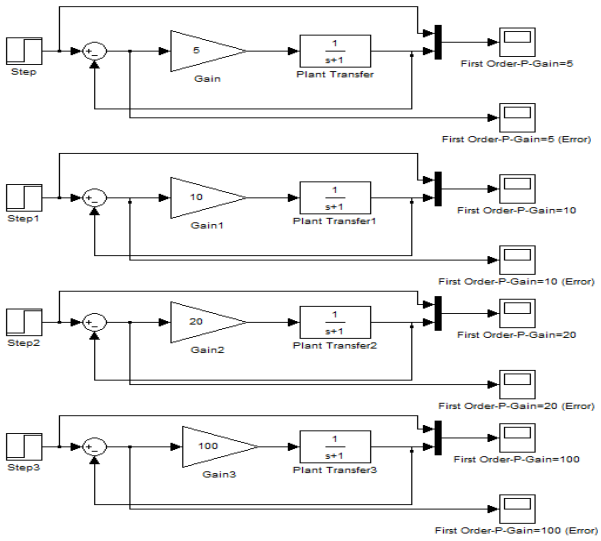

Figure 12: MATLAB-Simulink Diagram to show the effect of P control on first order plant

First order continuous plant transfer function:

$$G_p(s) = \frac{1}{s+1}$$

Input:

$$x(t) = 5u(t)$$

Controlled system outputs with for various K_p

For $K_p = 5$

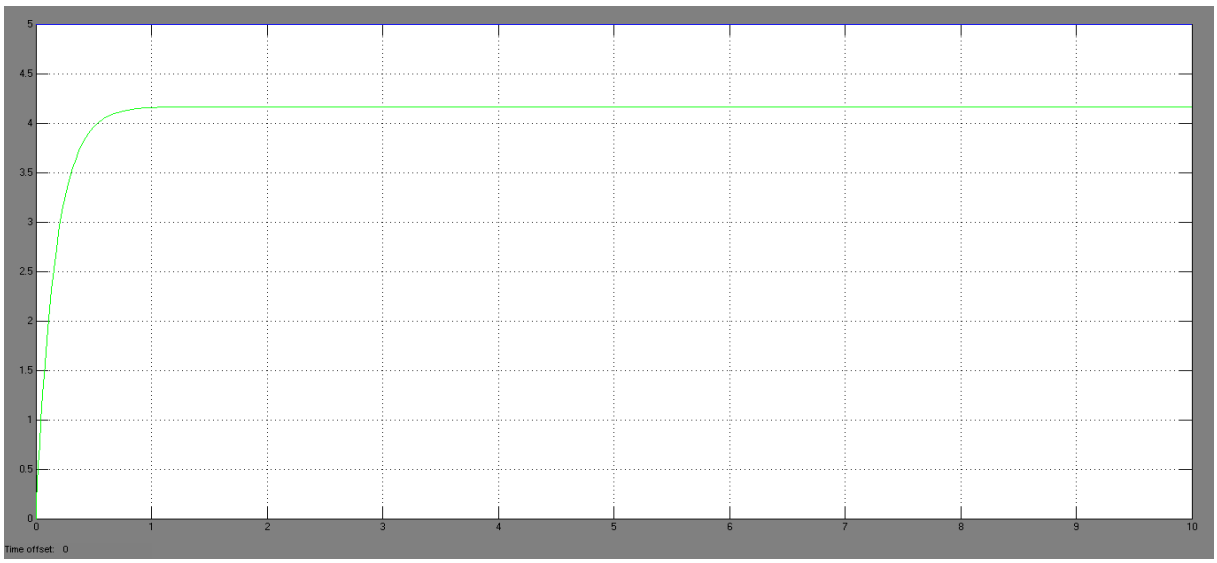

Figure 13: Output of the closed loop system with only P control, Kp = 5

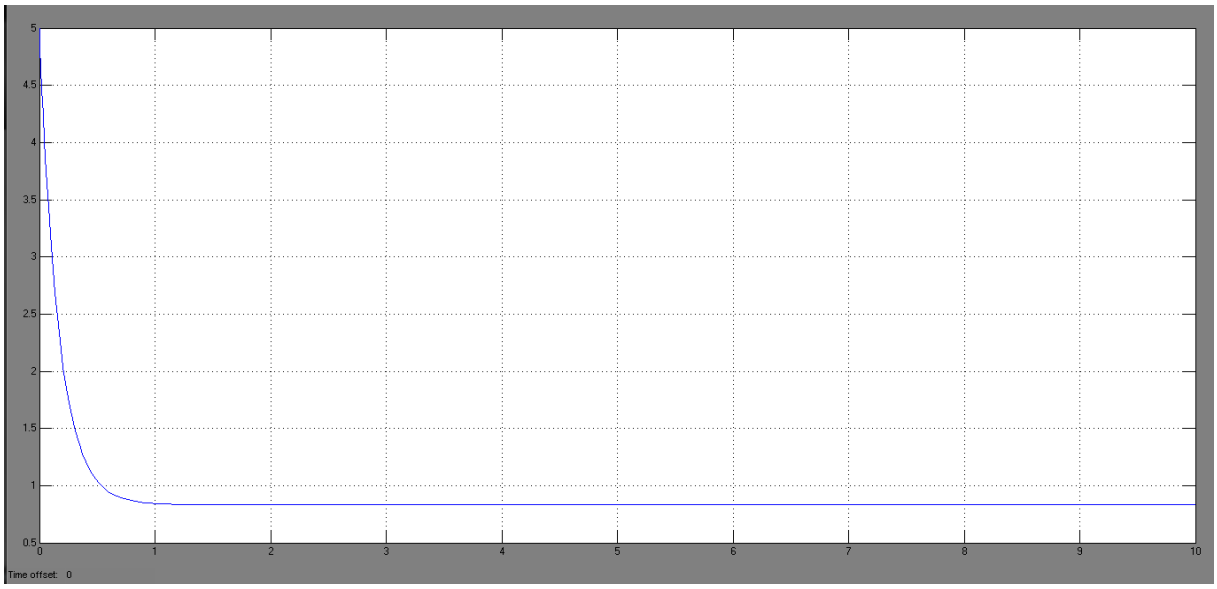

Figure 14: Error of the closed loop system with only P control, Kp=5

$$Steady\ State\ Error = e_{ss} = 0.84$$

$$Rise\ Time = \ t_r = 0.45\ secs$$

For $K_p = 10$

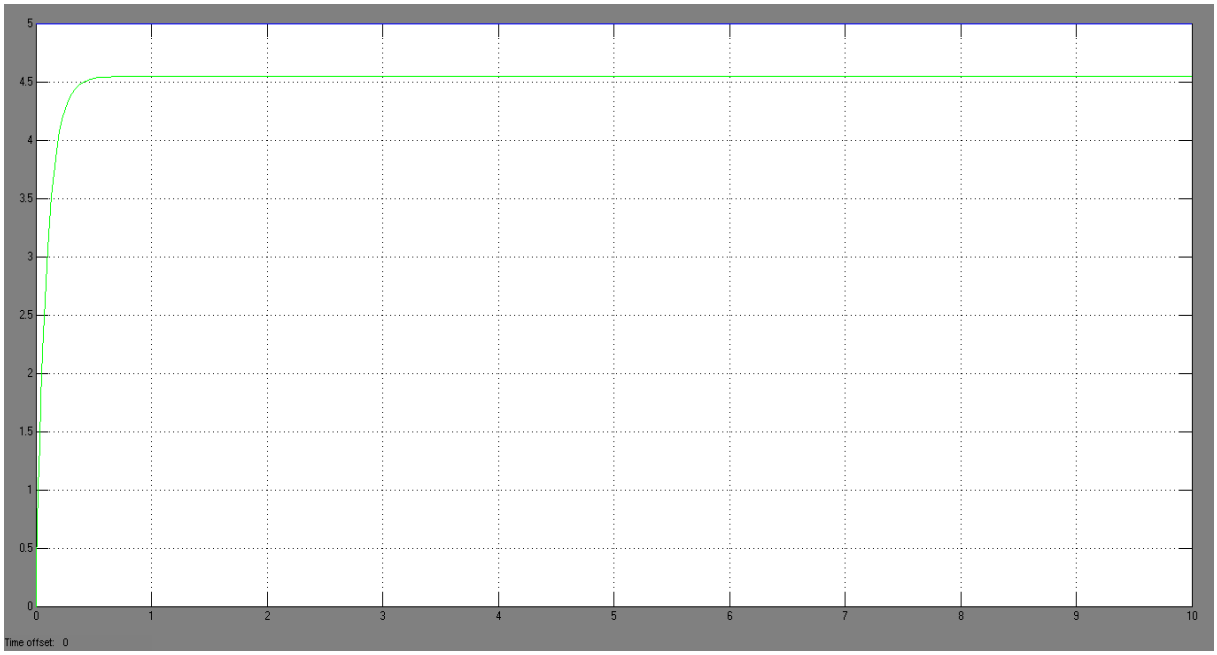

Figure 15: Output of the closed loop system with only P control, Kp=10

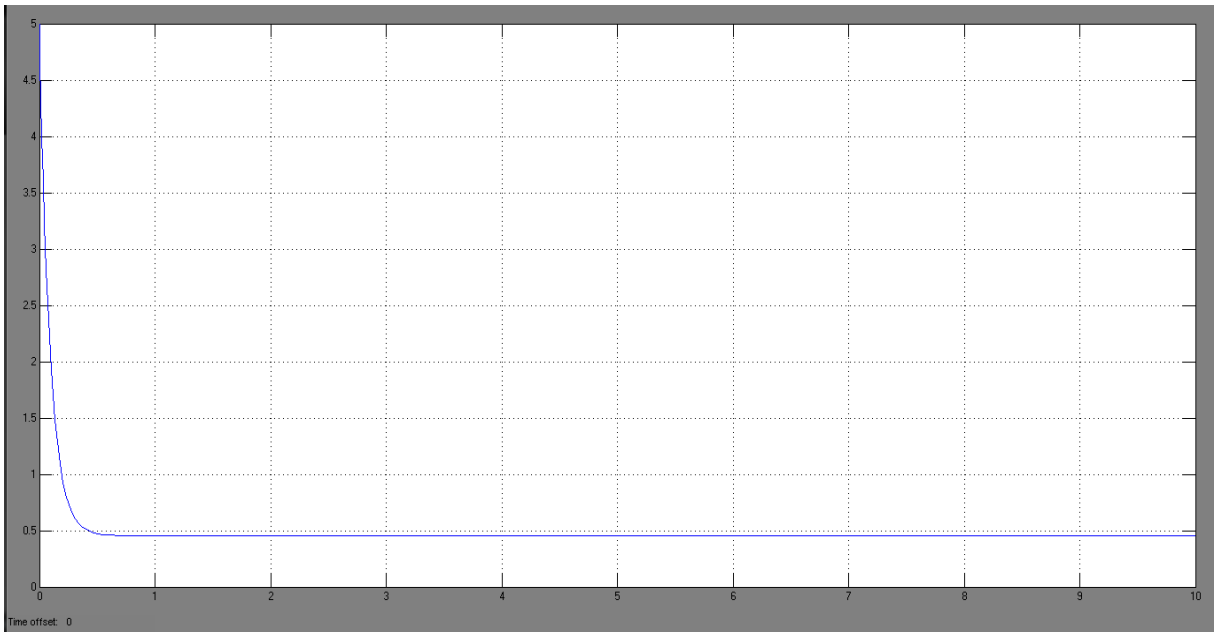

Figure 16: Error of the closed loop system with only P control, Kp=10

$$Steady\ State\ Error = e_{ss} = 0.45$$

$$Rise\ Time = \ t_r = 0.4\ secs$$

For $K_p = 20$

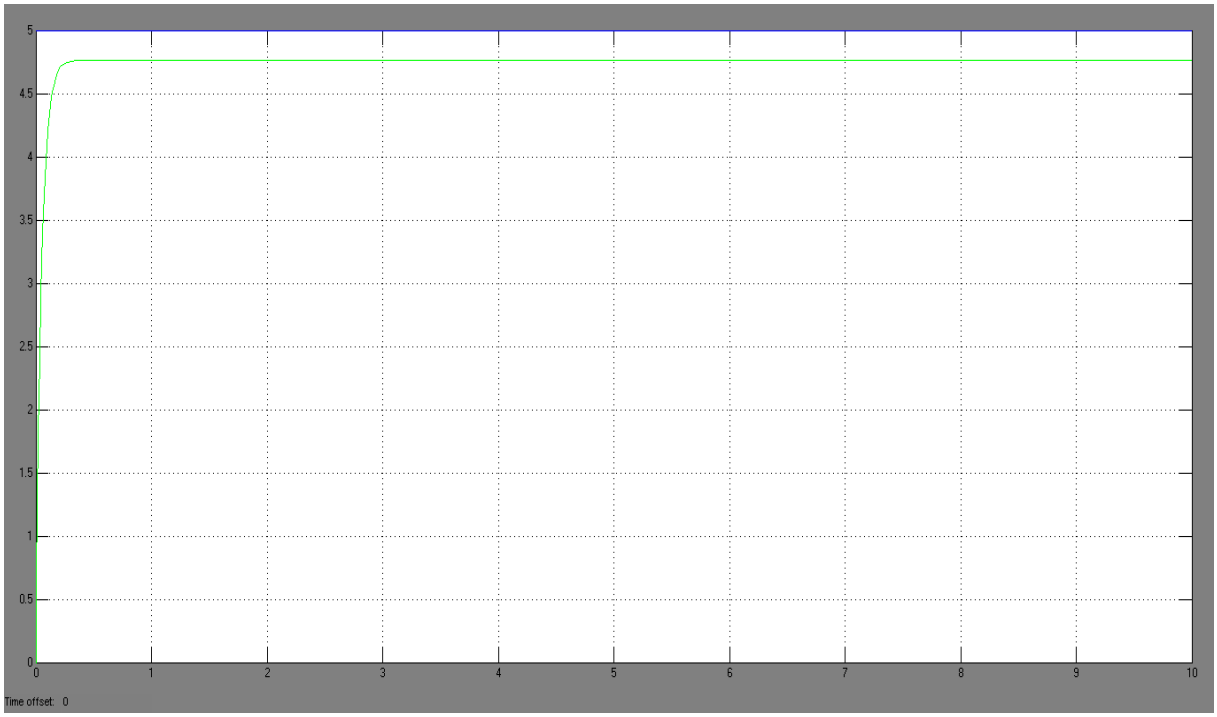

Figure 17: Output of the closed loop system with only P control, Kp=20

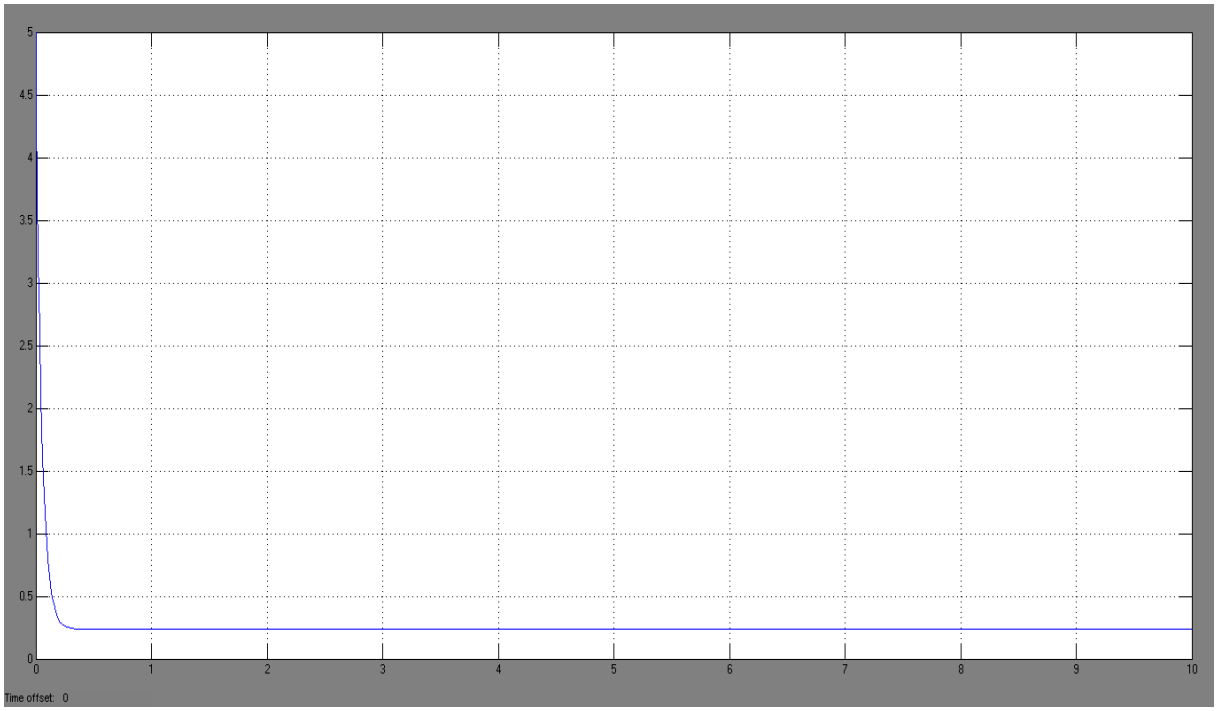

Figure 18: Error of the closed loop system with only P control, Kp=20

$$Steady\ State\ Error = e_{ss} = 0.24$$

$$Rise\ Time = \ t_r = 0.21\ secs$$

For $K_p = 100$

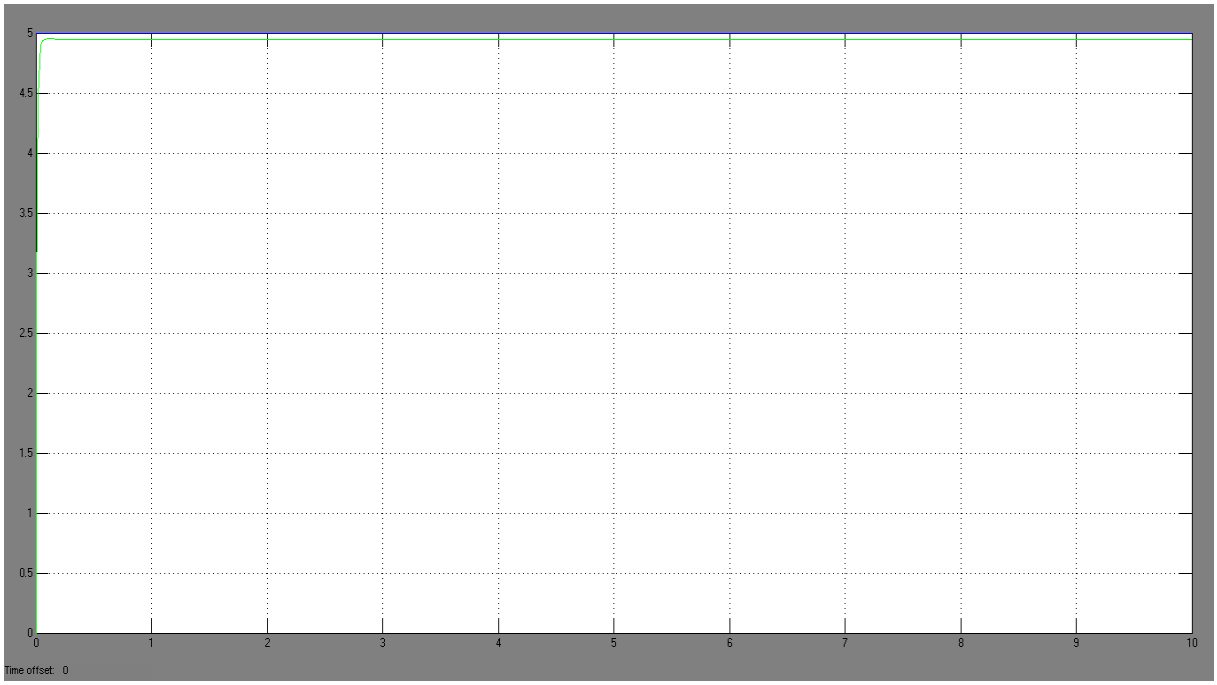

Figure 19: Output of the closed loop system with only P control, Kp=100

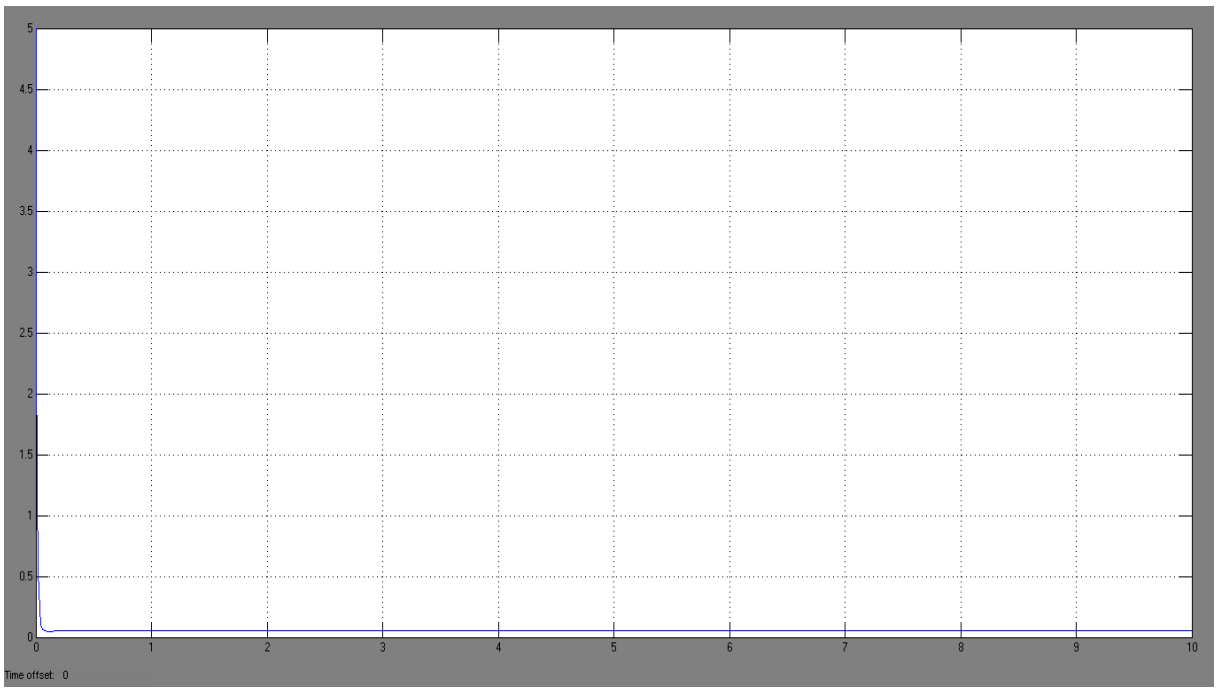

Figure 20: Error of the closed loop system with only P control, Kp=100

$$Steady\ State\ Error = e_{ss} = 0.05$$

$$Rise\ Time = \ t_r = 0.06\ secs$$

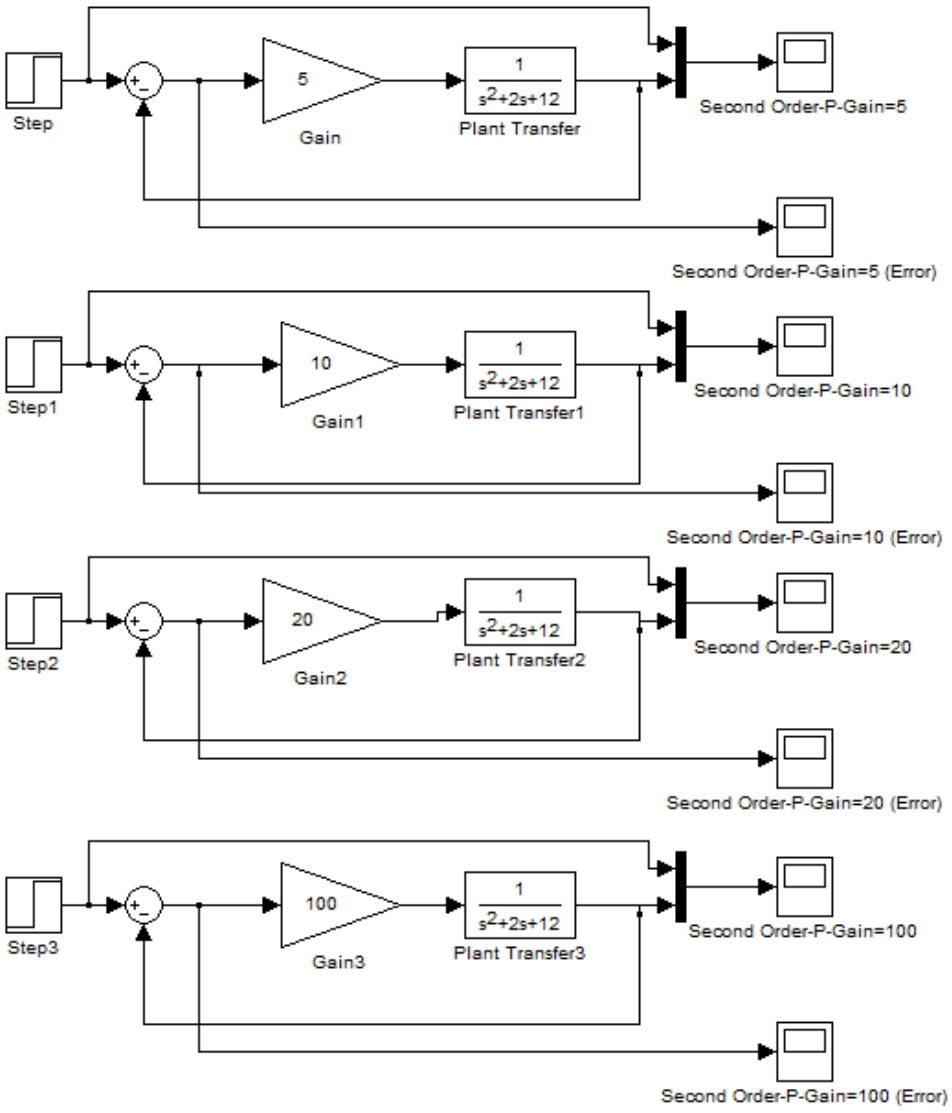

Figure 21: MATLAB-Simulink Diagram to show the effect of P control on second order plant

Second order continuous plant transfer function:

$$G_p(s) = \frac{1}{s^2+2s+12}$$

Input:

$$x(t) = 5u(t)$$

Controlled system outputs with for various K_p

For $K_p = 5$

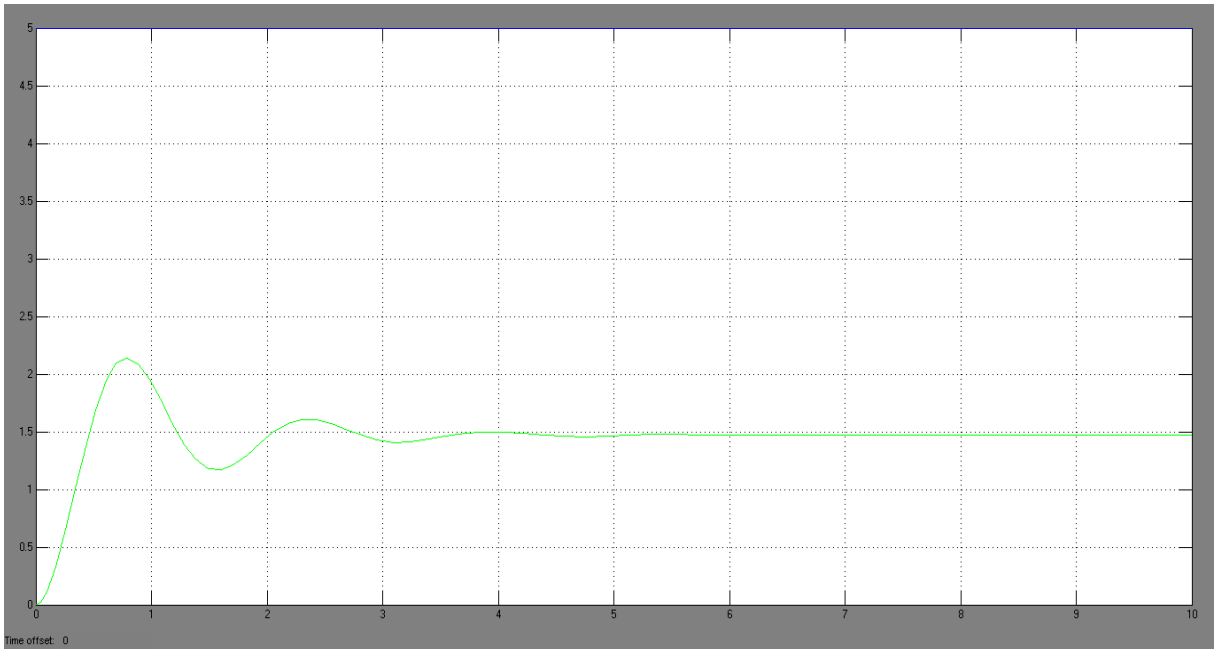

Figure 22: Output of the closed loop system with only P control, Kp=5

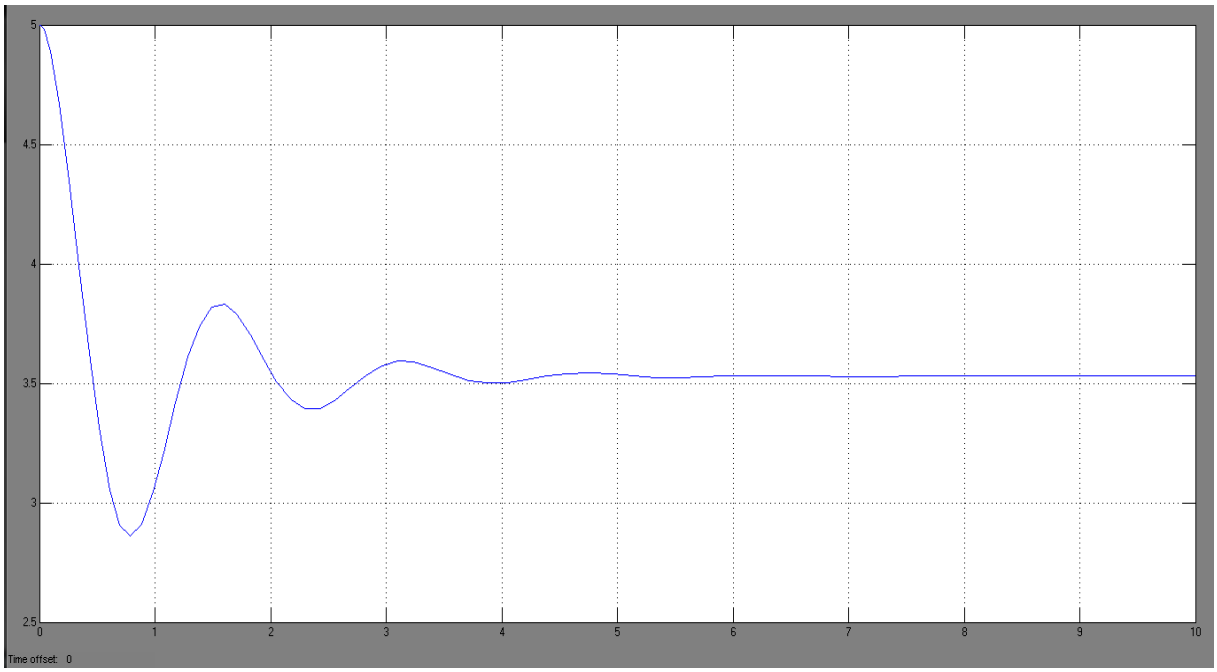

Figure 23: Error of the closed loop system with only P control, Kp=5

$$Steady\ State\ Error = e_{ss} = 3.5$$

$$Rise\ Time = \ t_r = 0.46\ secs$$

$$Percent\ Overshoot = 43.3\ \%$$

For $K_p = 10$

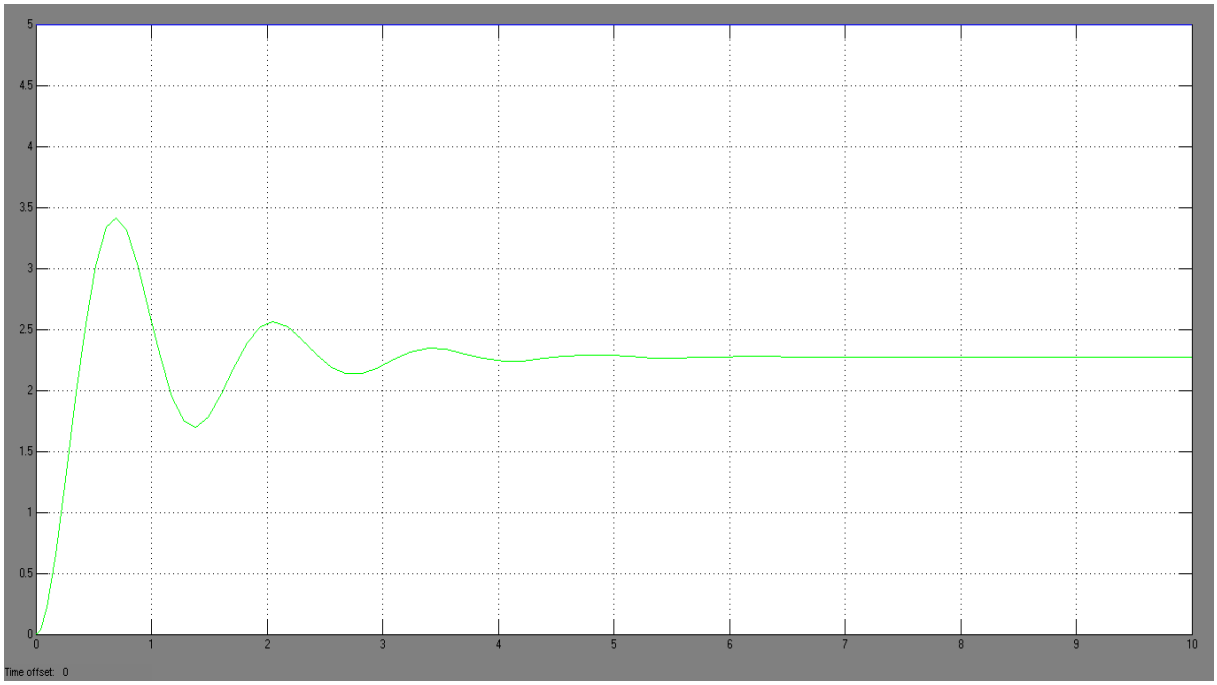

Figure 24: Output of the closed loop system with only P control, Kp=10

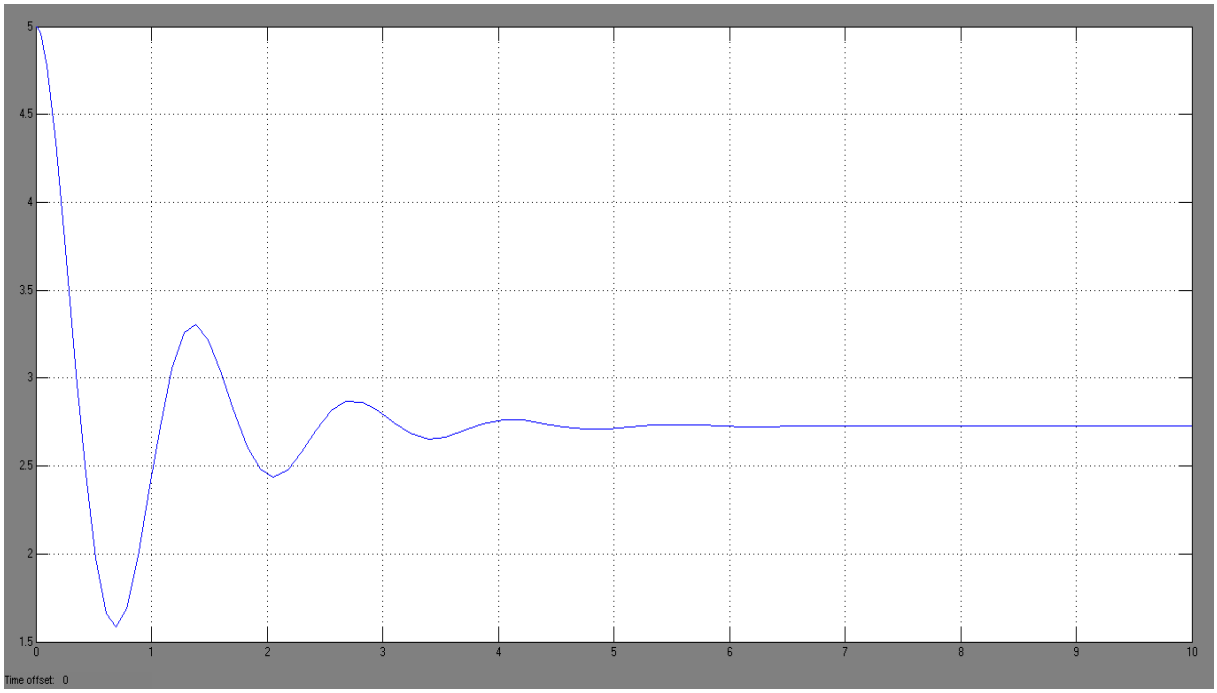

Figure 25: Error of the closed loop system with only P control, Kp=10

$$Steady\ State\ Error = e_{ss} = 2.7$$

$$Rise\ Time = \ t_r = 0.4\ secs$$

$$Percent\ Overshoot = 47.83\ \%$$

For $K_p = 20$

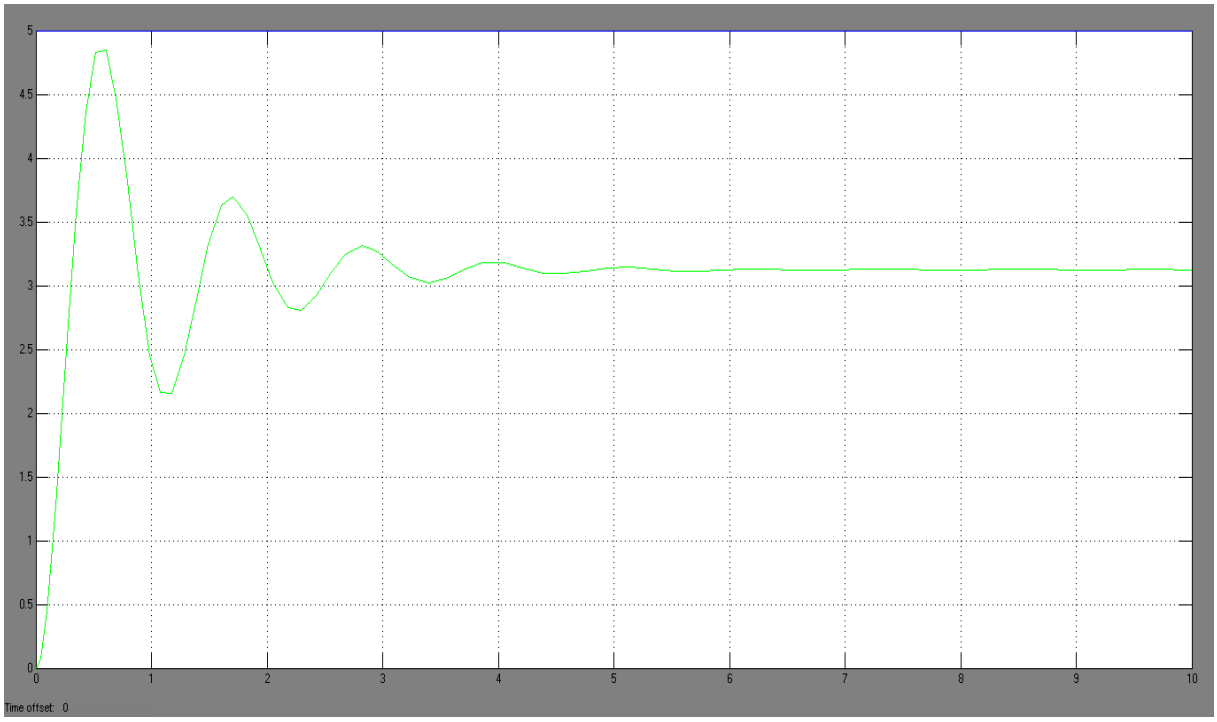

Figure 26: Output of the closed loop system with only P control, Kp=20

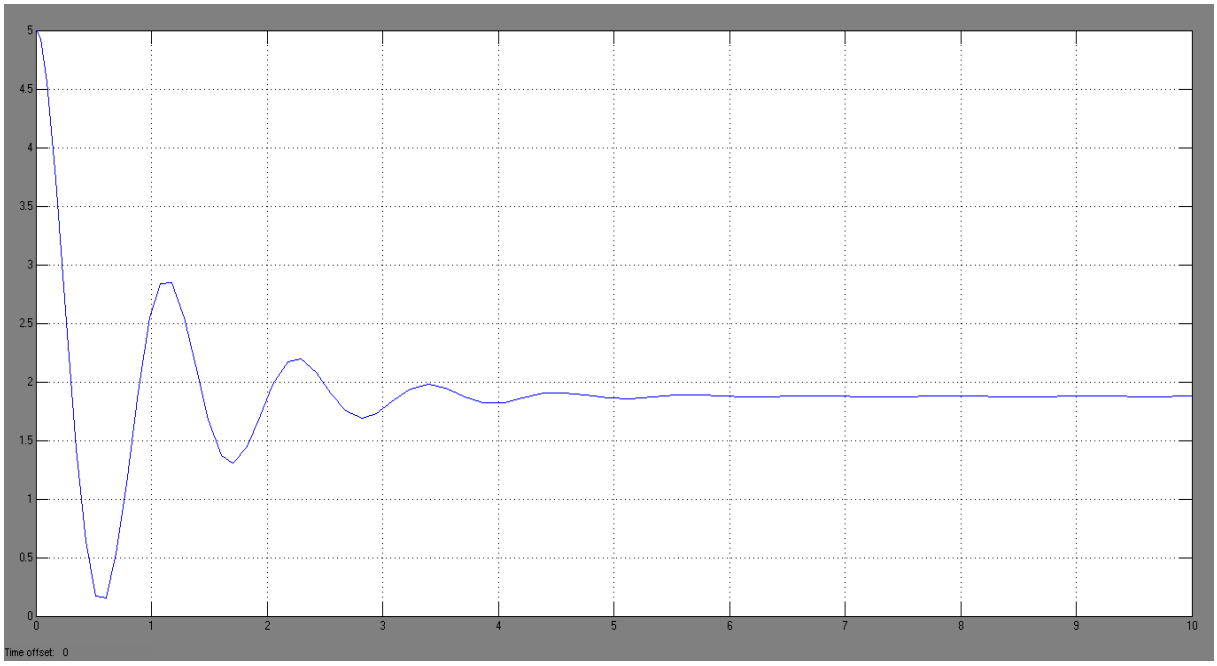

Figure 27: Error of the closed loop system with only P control, Kp=20

$$Steady\ State\ Error = e_{ss} = 1.9$$

$$Rise\ Time = \ t_r = 0.31\ secs$$

$$Percent\ Overshoot = 54.84\ \%$$

For $K_p = 100$

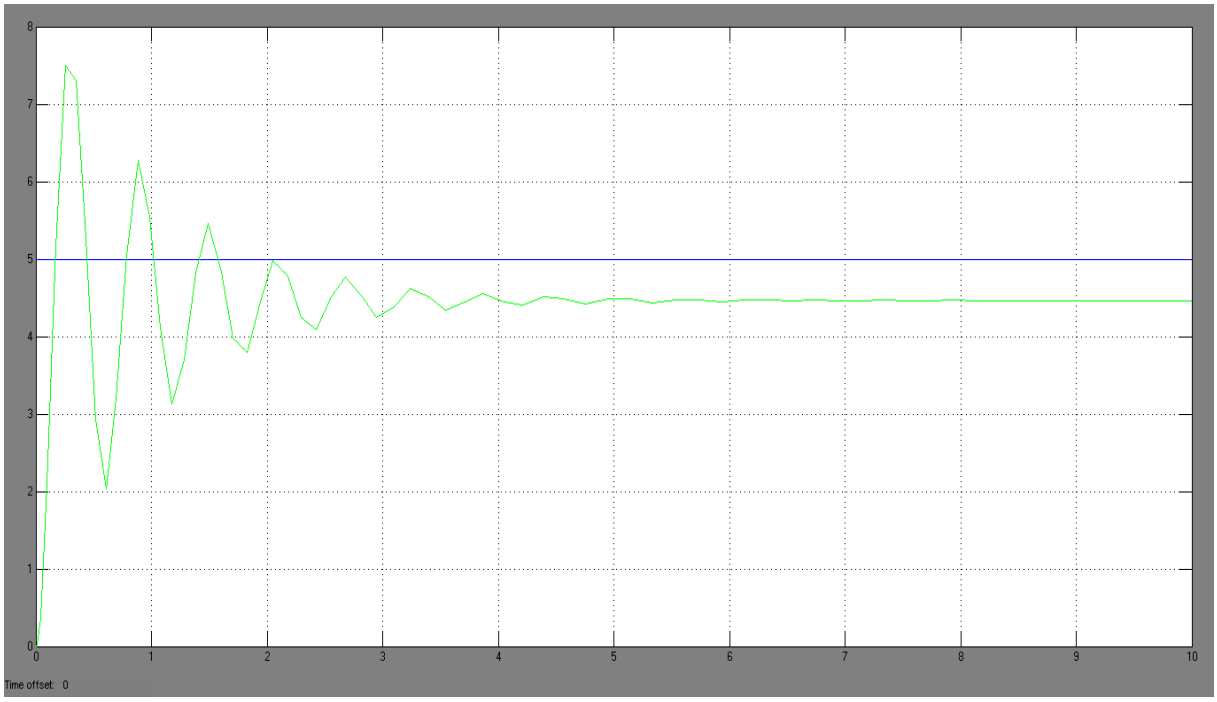

Figure 28: Output of the closed loop system with only P control, Kp=100

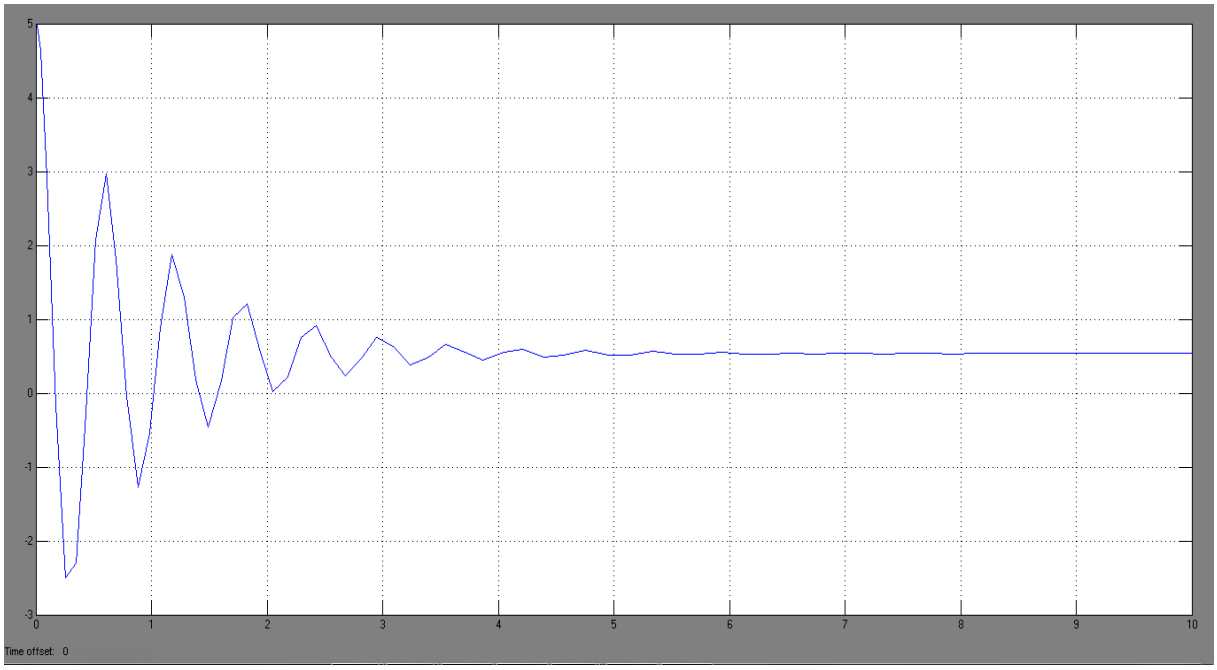

Figure 29: Error of the closed loop system with only P control, Kp=100

$$Steady\ State\ Error = e_{ss} = 0.5$$

$$Rise\ Time = \ t_r = 0.16\ secs$$

$$Percent\ Overshoot = 66.67\ \%$$

- **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.

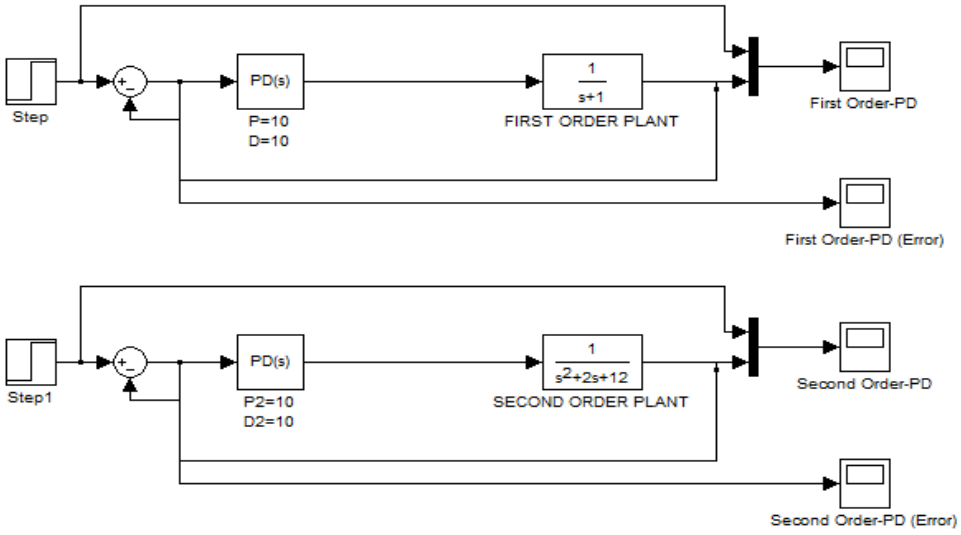

Figure 30: MATLAB-Simulink Diagram to show the effect of P-D control on first and second order plants

First order continuous plant transfer function:

$$G_p(s) = \frac{1}{s+1}$$

Second order continuous plant transfer function:

$$G_p(s) = \frac{1}{s^2+2s+12}$$

Input:

$$x(t) = 5u(t)$$

Controlled first order system outputs:

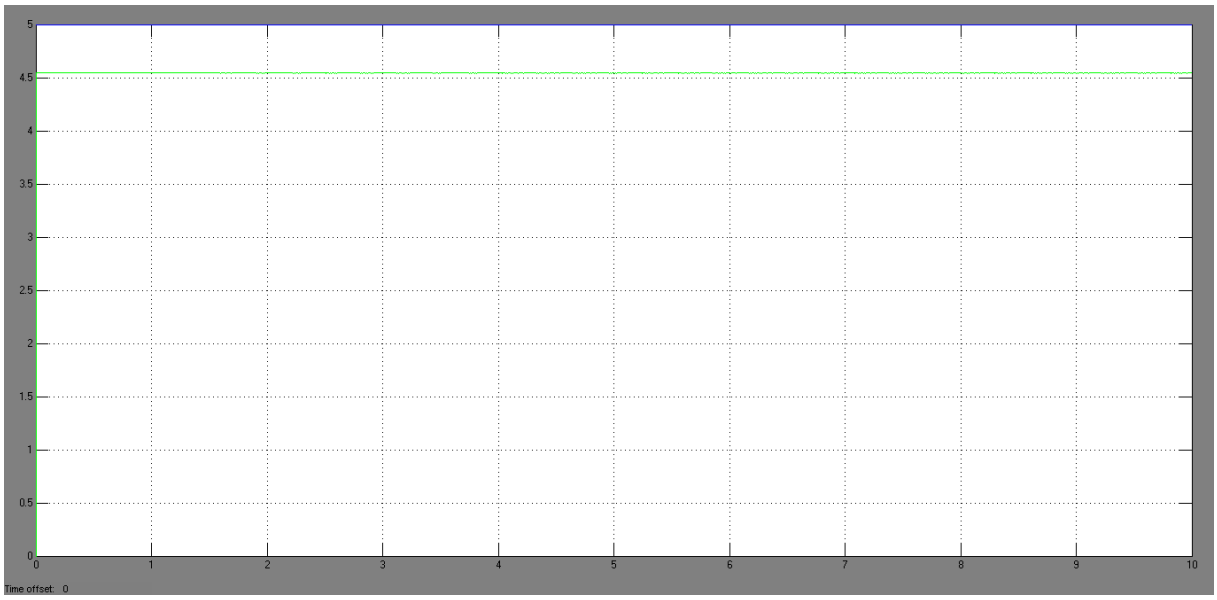

Figure 31: Output of the closed loop system (first order) with P-D control, Kp=10, Kd=10

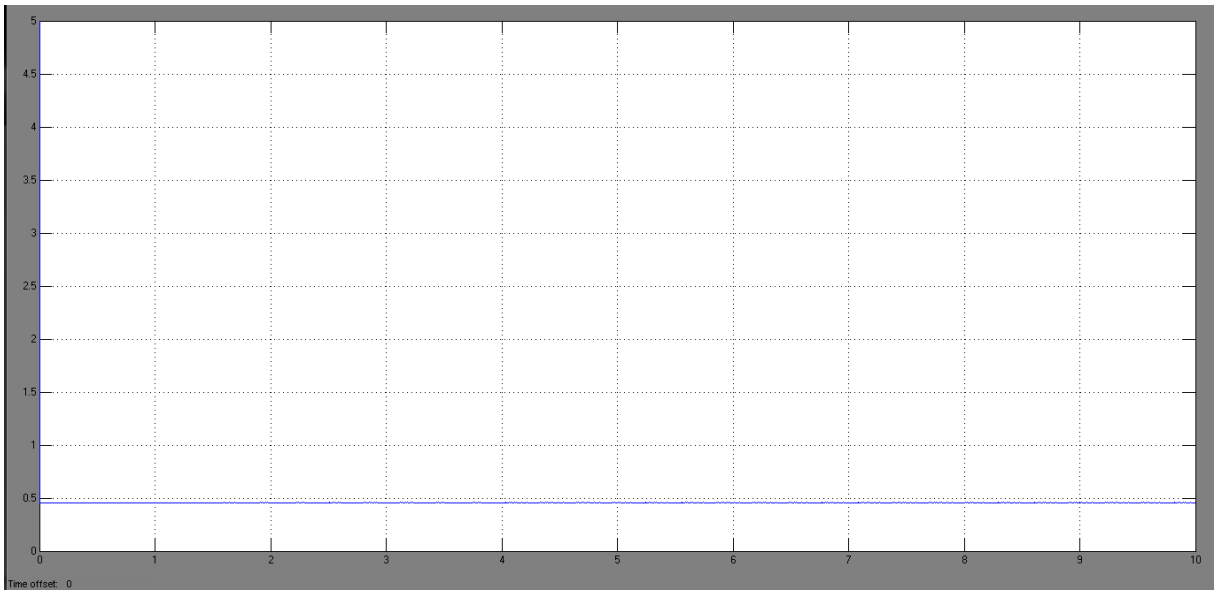

Figure 32: Error of the closed loop system (first order) with P-D control, Kp=10, Kd=10

$$Steady\ State\ Error = e_{ss} = 0.45$$

$$Rise\ Time = \ t_r = 0.005\ secs$$

Controlled second order system outputs:

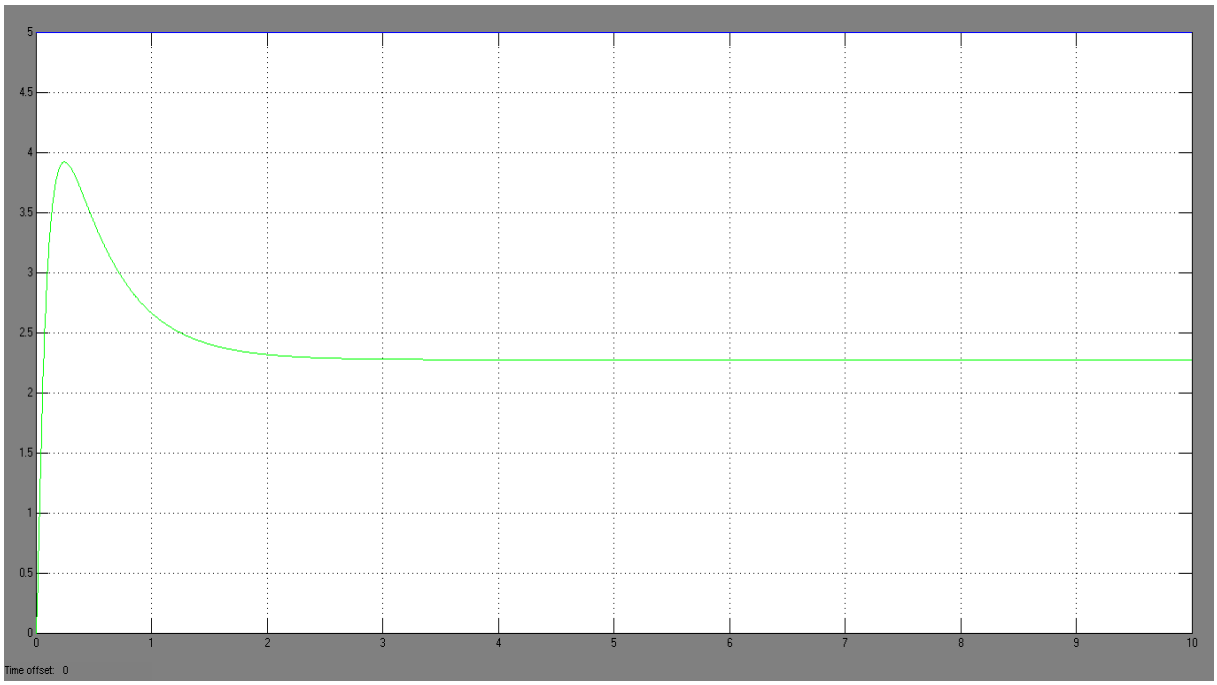

Figure 33: Output of the closed loop system (second order) with P-D control, Kp=10, Kd=10

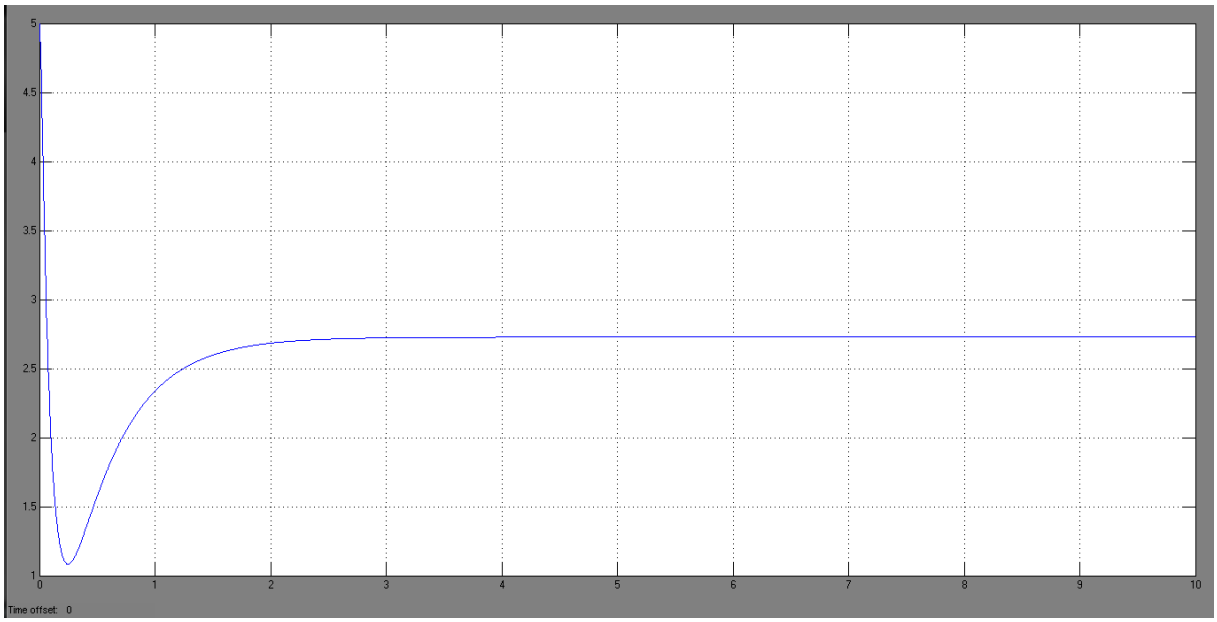

Figure 34: Error of the closed loop system (second order) with P-D control, Kp=10, Kd=10

$$Steady\ State\ Error = e_{ss} = 2.7$$

$$Rise\ Time = \ t_r = 0.07\ secs$$

$$Percent\ Overshoot = 69.5\ \%$$

- **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.

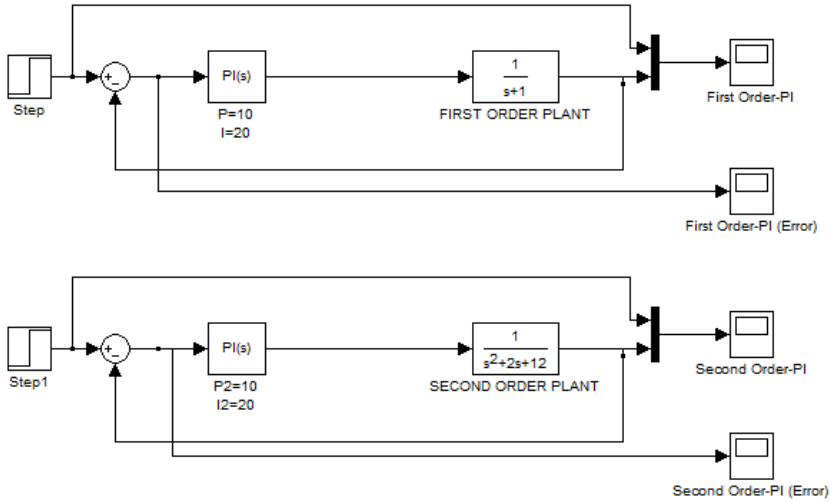

Figure 35: MATLAB-Simulink Diagram to show the effect of P-I control on first and second order plants

First order continuous plant transfer function:

$$G_p(s) = \frac{1}{s+1}$$

Second order continuous plant transfer function:

$$G_p(s) = \frac{1}{s^2+2s+12}$$

Input:

$$x(t) = 5u(t)$$

Controlled first order system outputs:

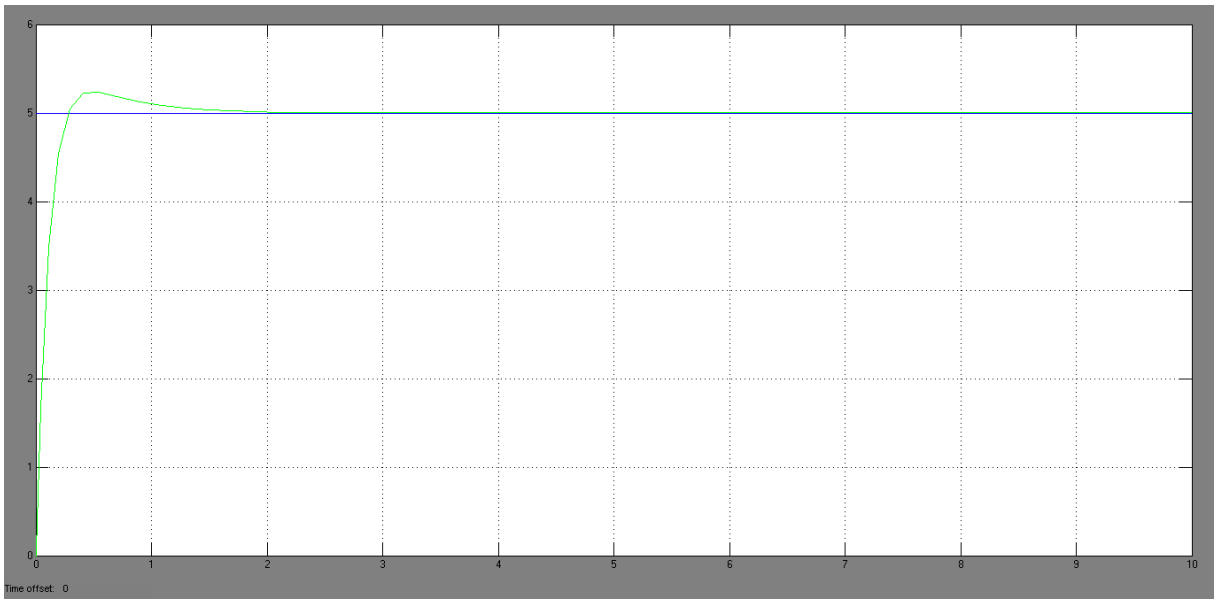

Figure 36: Output of the closed loop system (first order) with P-I control, Kp=10, Ki=20

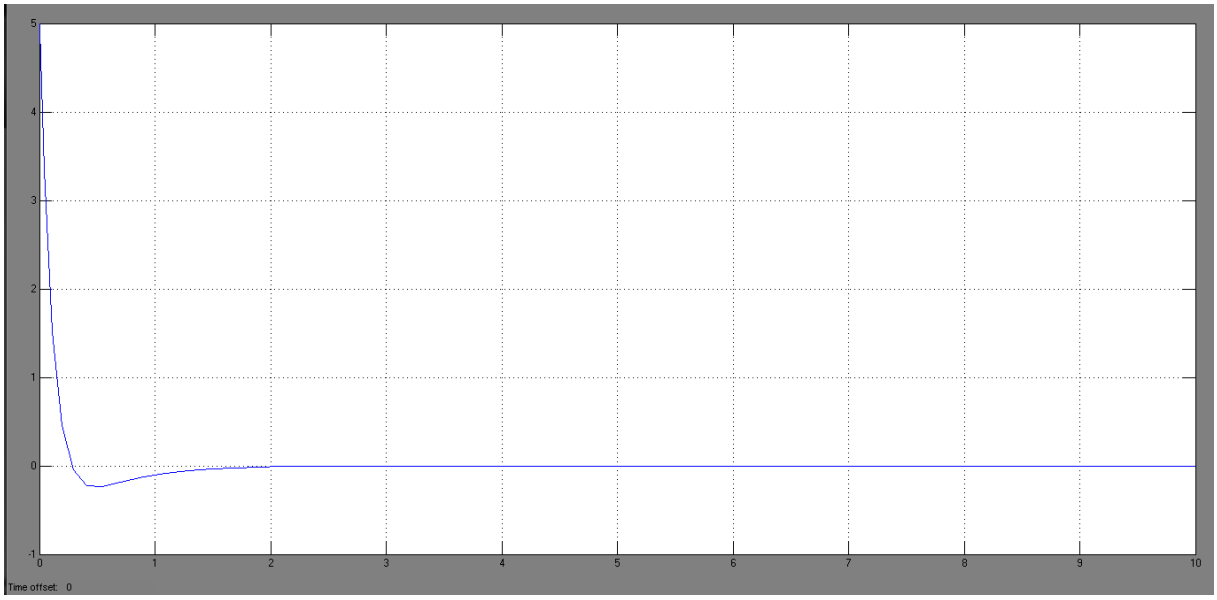

Figure 37: Error of the closed loop system (first order) with P-I control, Kp=10, Ki=20

$$Steady\ State\ Error = e_{ss} = 0$$

$$Rise\ Time = \ t_r = 0.28\ secs$$

$$2\%\ Settling\ time = \ t_s = 1.31\ secs$$

$$Percent\ Overshoot = 5\ \%$$

Controlled second order system outputs:

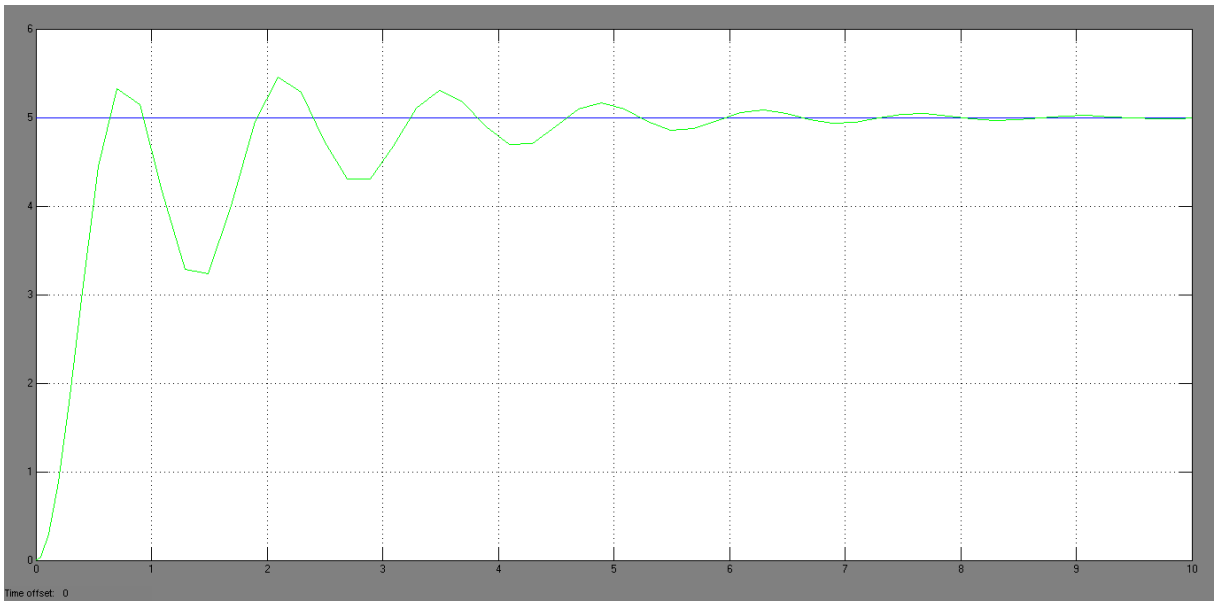

Figure 38: Output of the closed loop system (second order) with P-I control, Kp=10, Ki=20

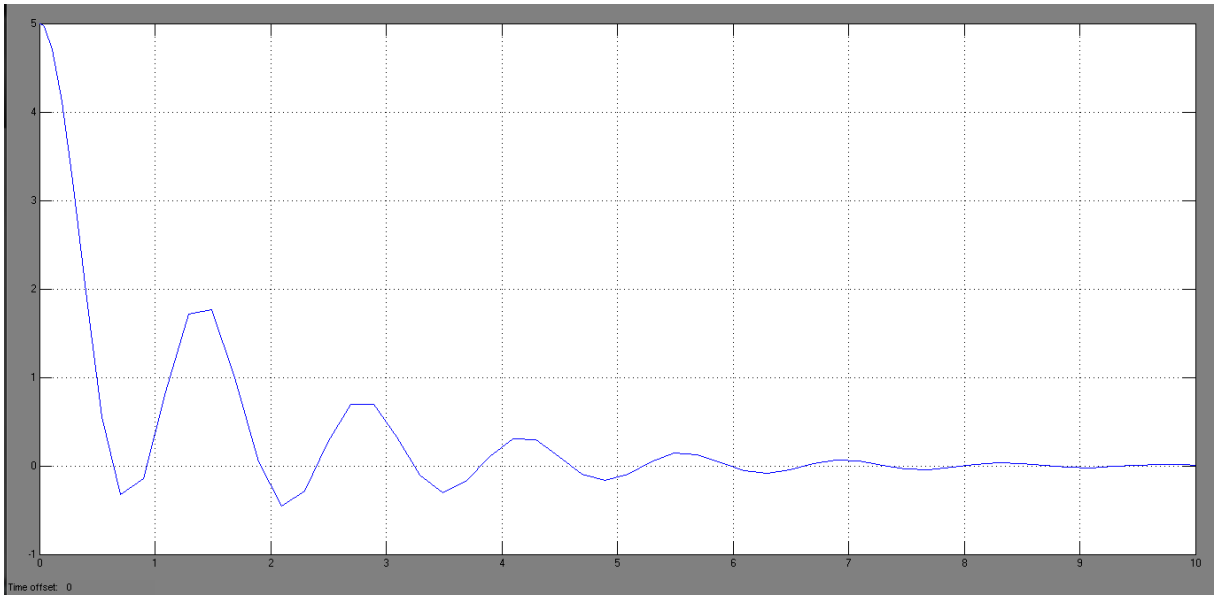

Figure 39: Error of the closed loop system (second order) with P-I control, Kp=10, Ki=20

$$Steady\ State\ Error = e_{ss} = 0$$

$$Rise\ Time = \ t_r = 0.65\ secs$$

$$2\%\ Settling\ time = \ t_s = 7.11\ secs$$

$$Percent\ Overshoot = 6\ \%$$

- **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.

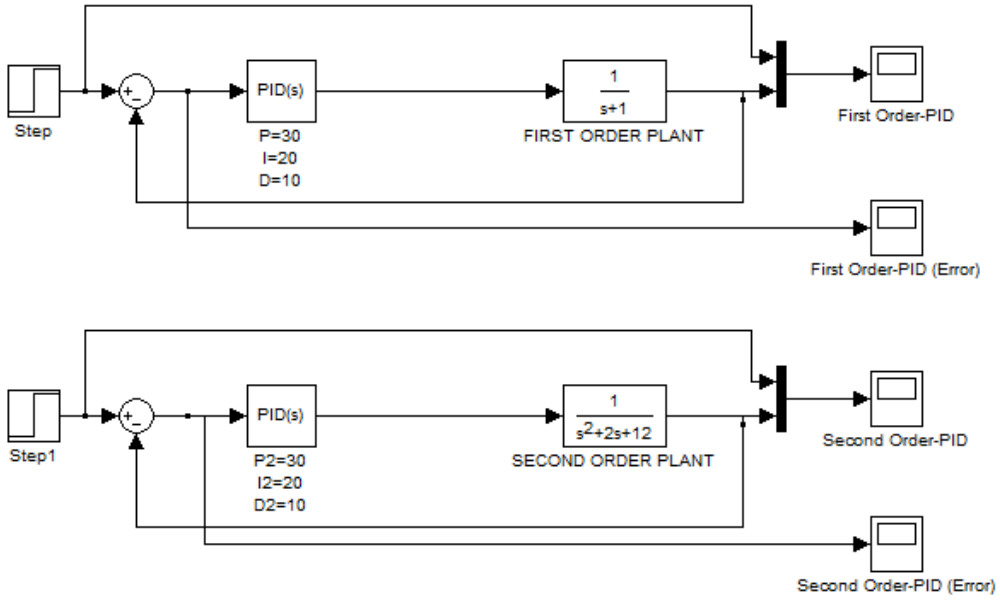

Figure 40: MATLAB-Simulink Diagram to show the effect of P-I-D control on first and second order plants

First order continuous plant transfer function:

$$G_p(s) = \frac{1}{s+1}$$

Second order continuous plant transfer function:

$$G_p(s) = \frac{1}{s^2+2s+12}$$

Input:

$$x(t) = 5u(t)$$

Controlled first order system outputs:

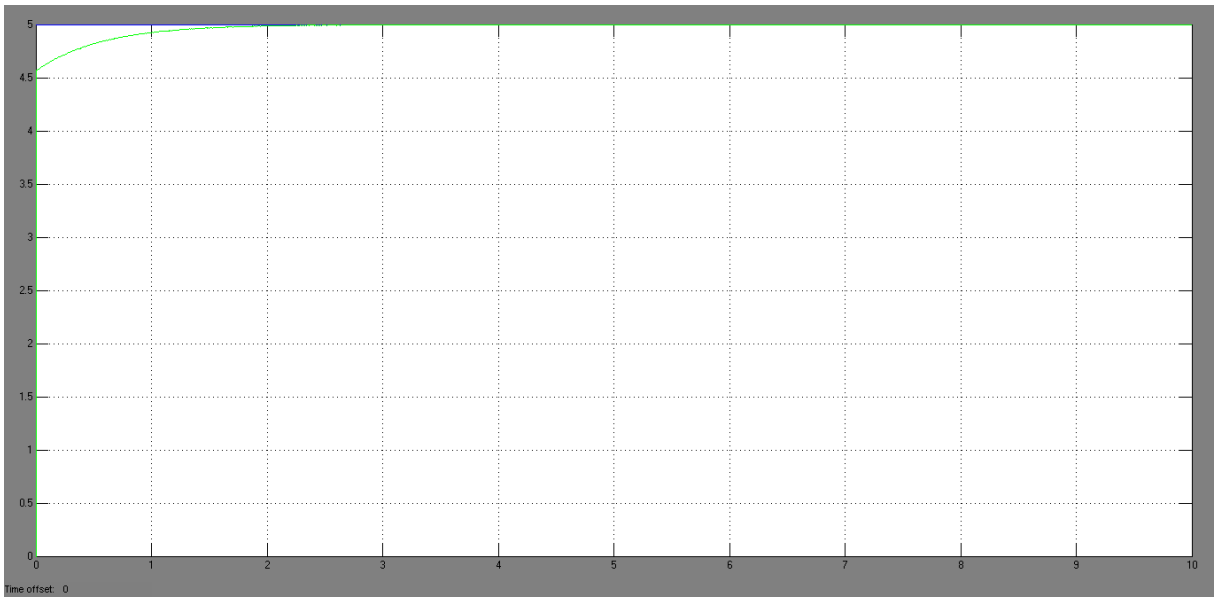

Figure 41: Output of the closed loop system (first order) with P-I-D control, Kp=30, Ki=20, Kd=10

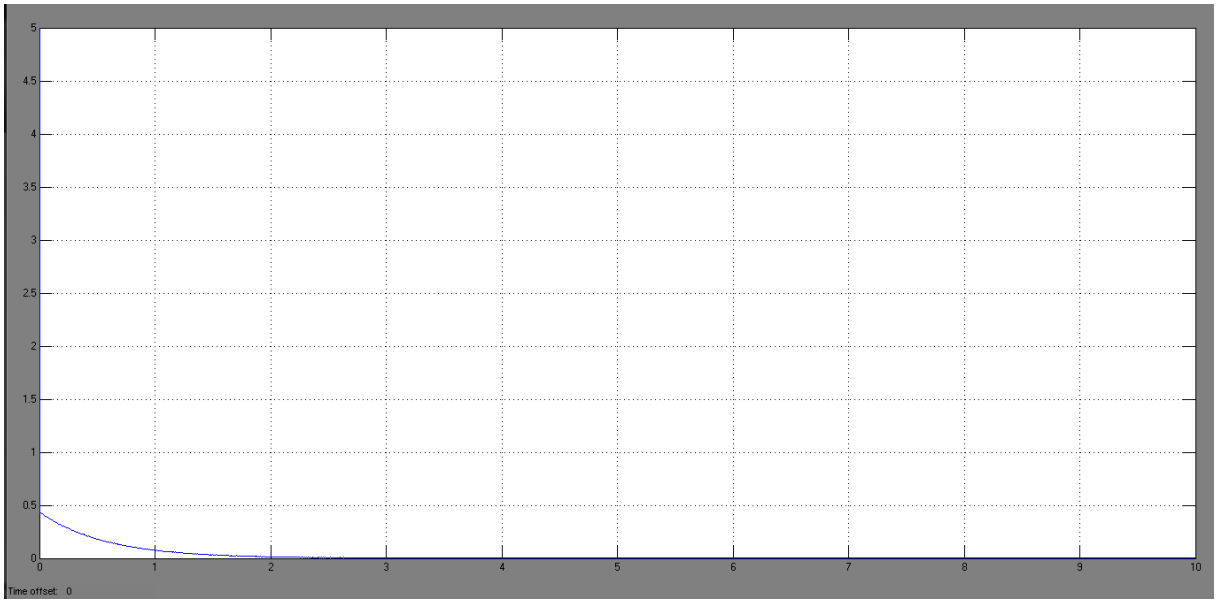

Figure 42: Error of the closed loop system (first order) with P-I-D control, Kp=30, Ki=20, Kd=10

$$Steady\ State\ Error = e_{ss} = 0$$

$$Rise\ Time = \ t_r = 2.4\ secs$$

Controlled second order system outputs:

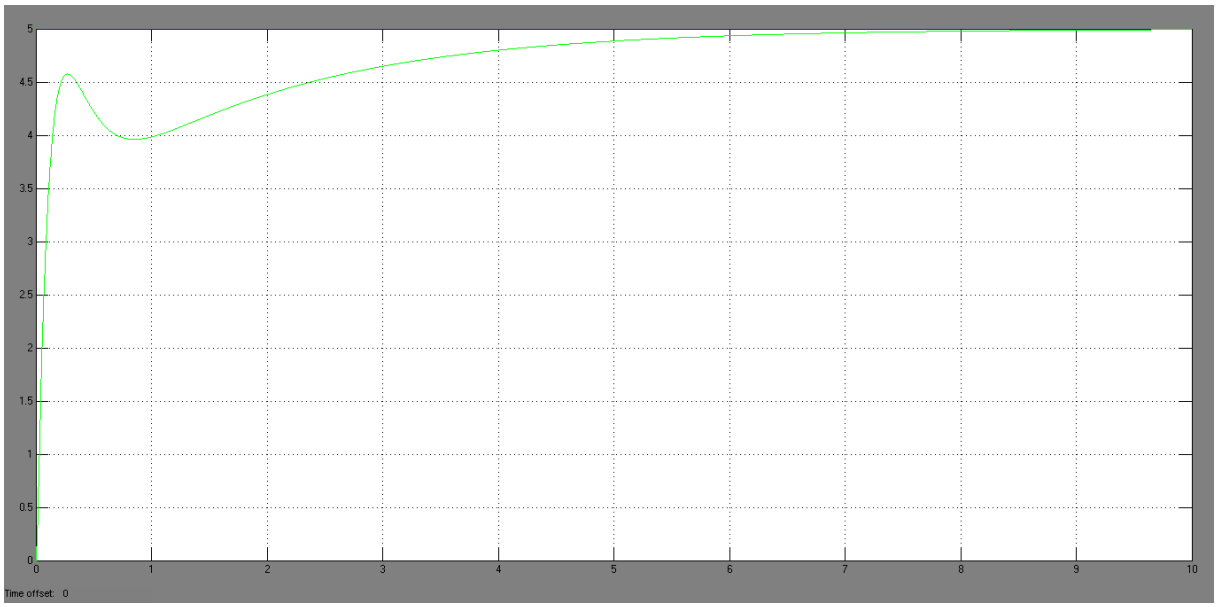

Figure 43: Output of the closed loop system (second order) with P-I-D control, Kp=30, Ki=20, Kd=10

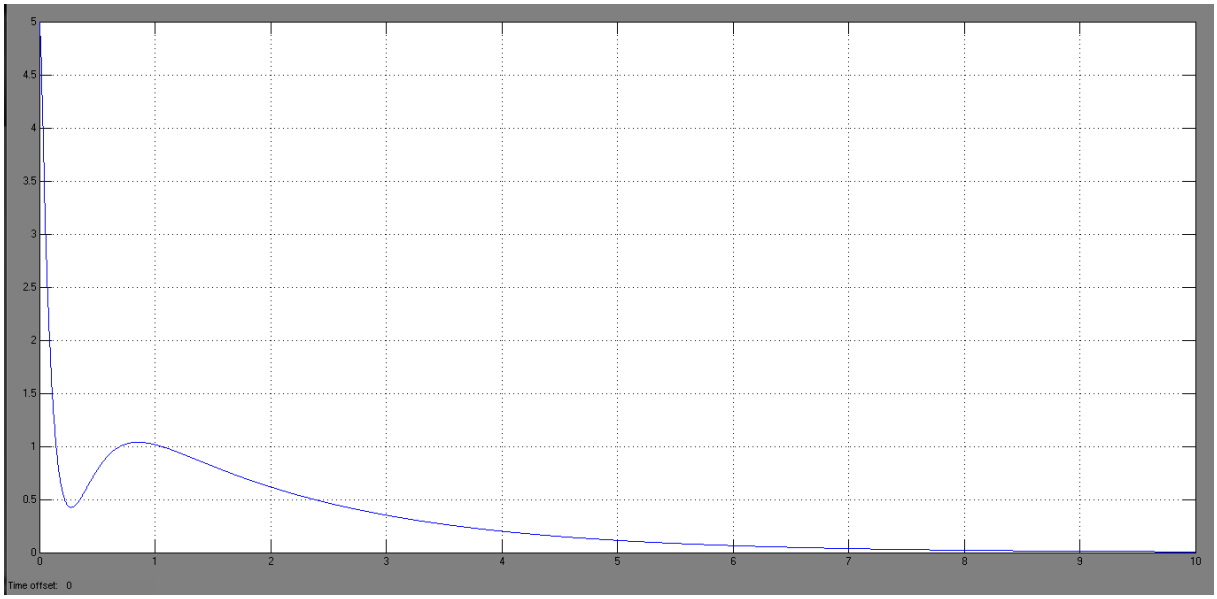

Figure 44: Error of the closed loop system (second order) with P-I-D control, Kp=30, Ki=20, Kd=10

$$Steady\ State\ Error = e_{ss} = 0$$

$$Rise\ Time = \ t_r = 0.2\ secs$$

$$2\%\ Settling\ time = \ t_s = 5.24\ secs$$

$$Percent\ Overshoot = 0\ \%$$

The results above show that P-I-D controller for second order plant requires tuning. Using the automatic tuning option of MATLAB-Simulink one can get the following results:

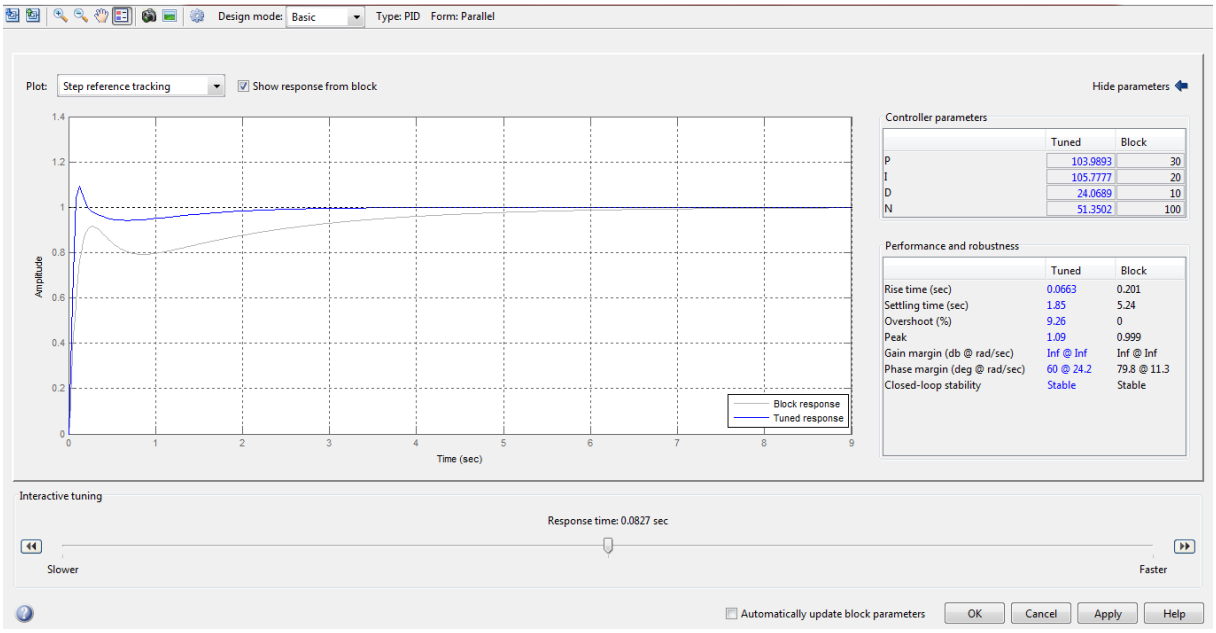

Figure 45: Output of the closed loop system (second order) with P-I-D control after tuning, Kp=104, Ki=106, Kd=24

$$Steady\ State\ Error = e_{ss} = 0$$

$$Rise\ Time = \ t_r = 0.0663\ secs$$

$$2\%\ Settling\ time = \ t_s = 1.85\ secs$$

$$Percent\ Overshoot = 9.26\ \%$$

$$Phase\ Margin = 60\ degree$$

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.

The continuous time plant transfer function of the DC motor is as follows

$$G_p = \frac{2}{(s + 9.997) \times (s + 2.003)}$$

In this transfer function, pole at s=2.003 is the dominant pole as it is closer to the jw − axis. Rise time for the output will approximately be

$$t_r = \frac{4}{|Dominant\ pole\ location|} = \frac{4}{2} = 2\ secs$$

Hence, the sampling period should be much less than 2 seconds.

The other critical step is the determination of s*-domain to z-domain transformation method. Various methods in MATLAB are indicated in the MATLAB-code as comments. MATLAB-code can be found in Appendix.

Below the effect of the sampling time is illustrated.

Sampling period, $T_s = 0.05\ secs$

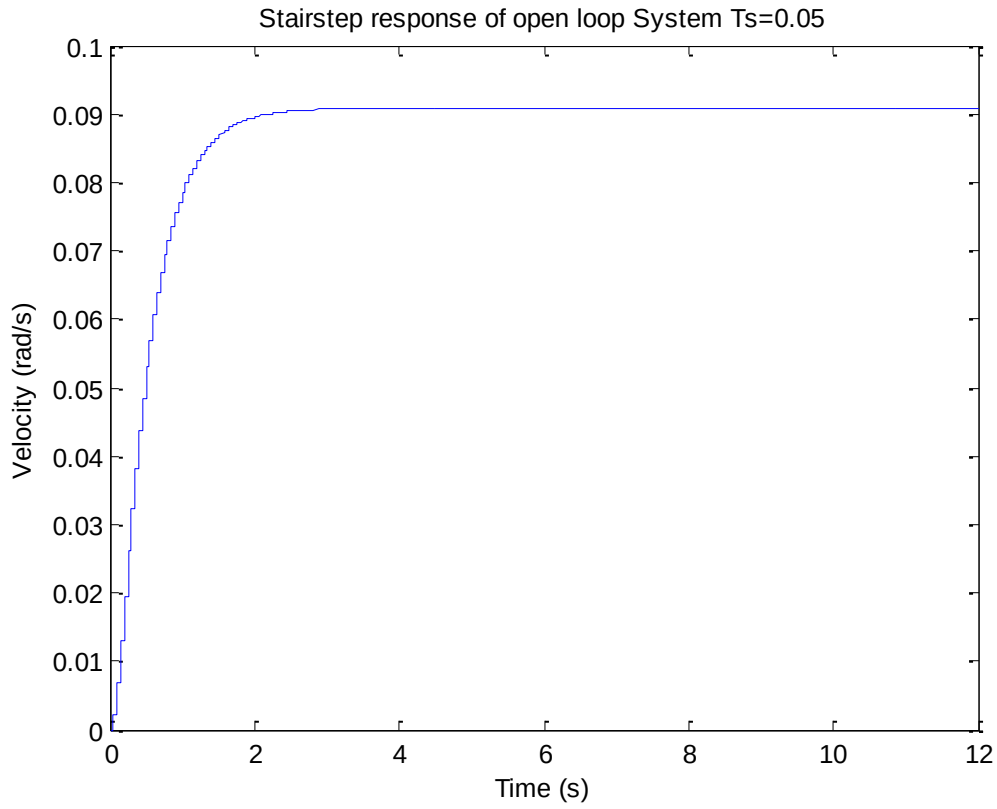

Figure 46: Stairstep response of open loop system, sampling period Ts=0.05 seconds

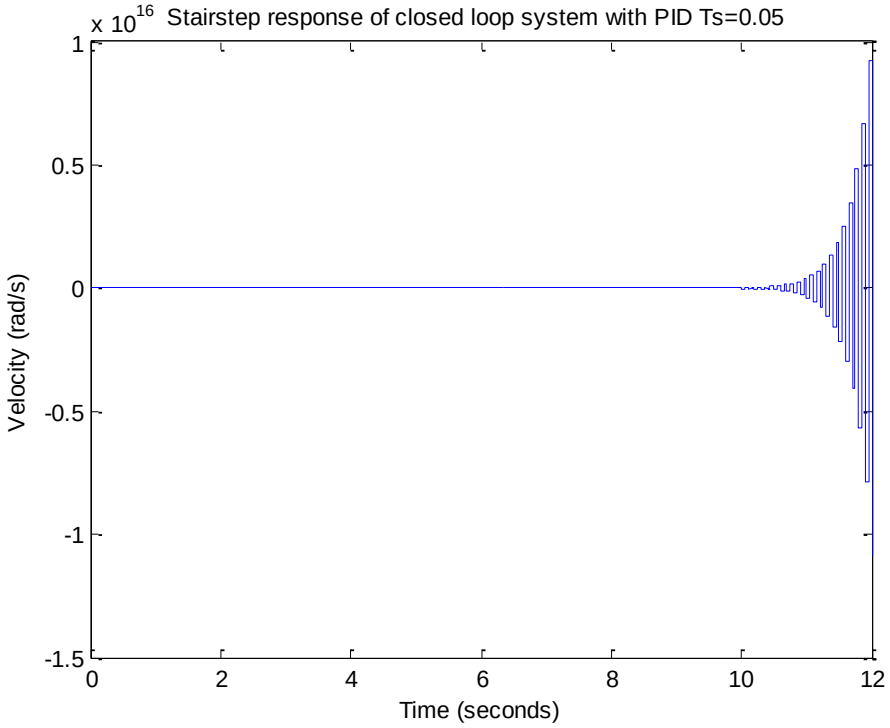

Figure 47: Stairstep response of closed loop system with PID, sampling period Ts=0.05 seconds

System is unstable, this is due to the conversion of s*-domain to z-domain. Pole placement is required. Figure 48 indicates the stairstep response of the closed loop system after proper pole placement.

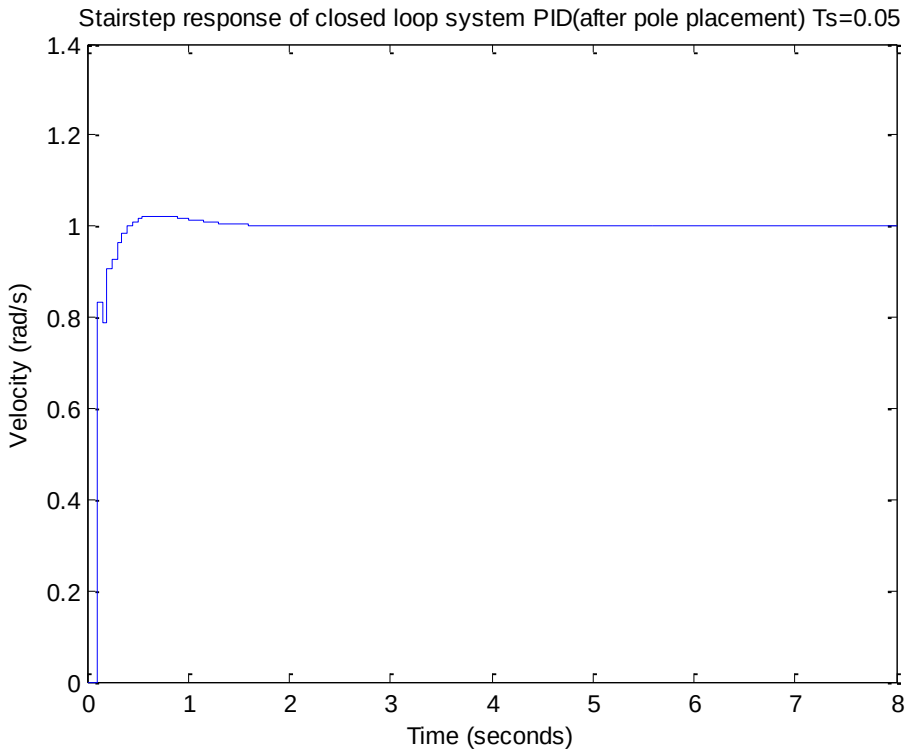

Figure 48: Stairstep response of closed loop system with PID after pole placement, sampling period Ts=0.05 seconds

Sampling period, $T_s = 1\ secs$

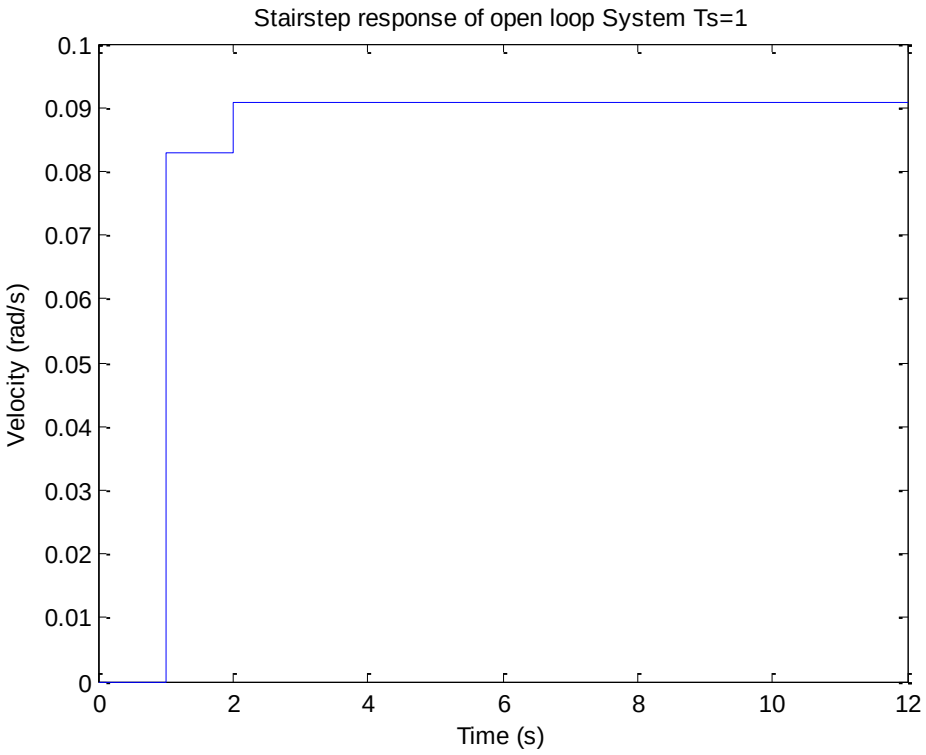

Figure 49: Stairstep response of open loop system, sampling period Ts=1 second

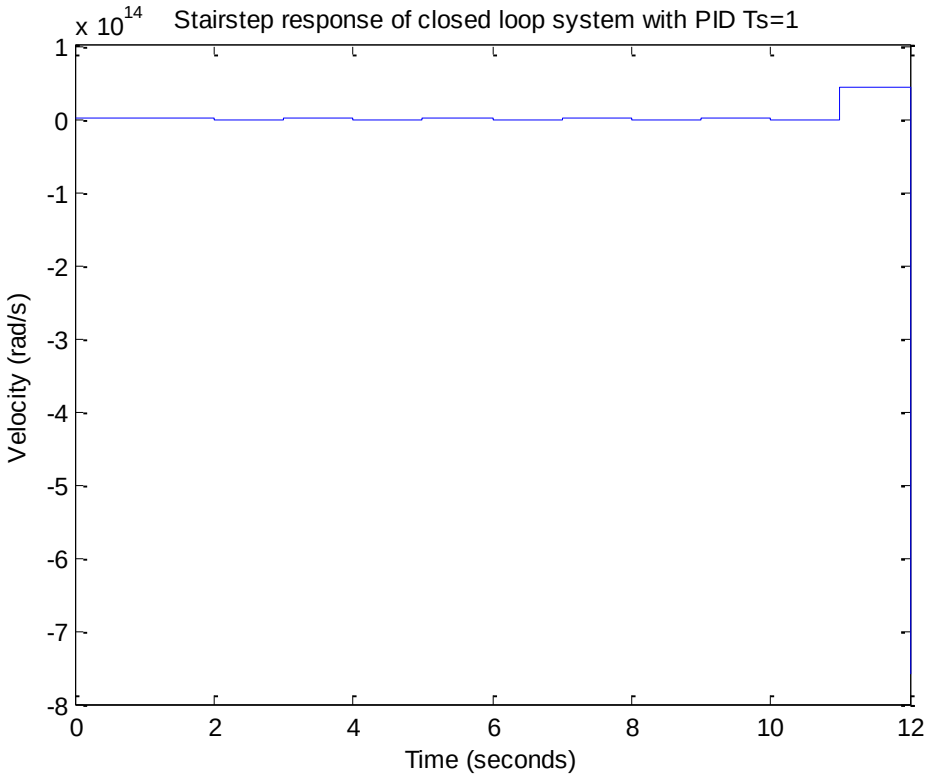

Figure 50: Stairstep response of closed loop system with PID, sampling period Ts=1seconds

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, $T_s = 4\ secs$

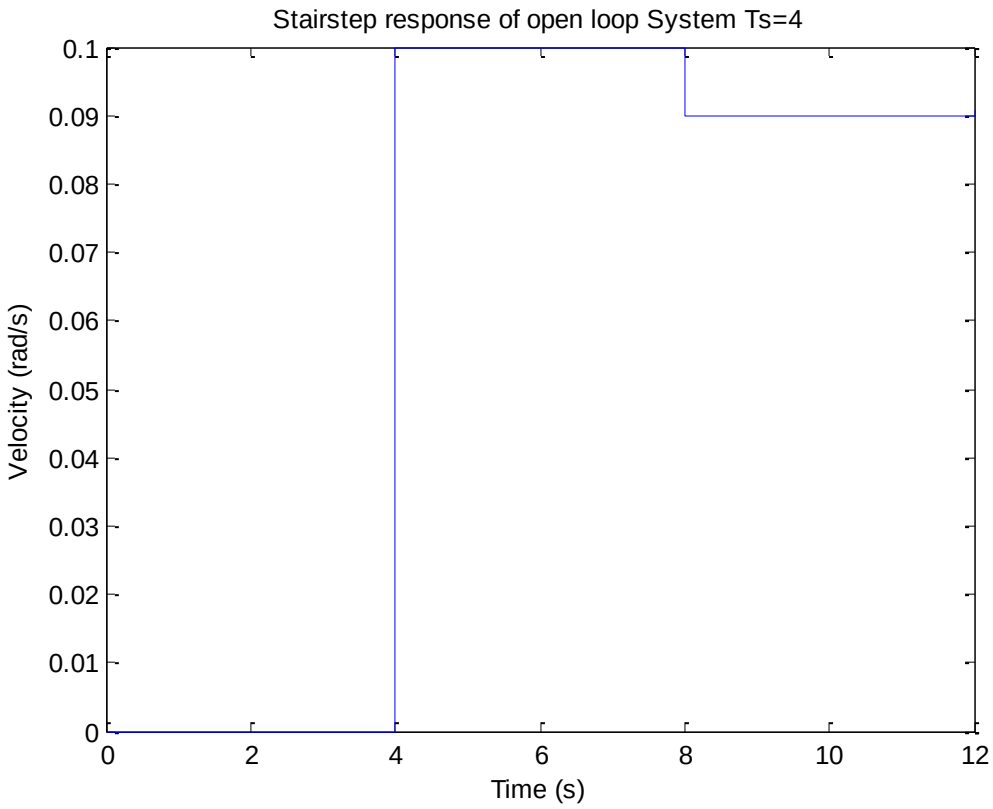

Figure 51: Stairstep response of open loop system, sampling period Ts=4 seconds

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.

PID Controller Design for Controlling DC Motor Speed in the Project

"A Vehicle Which Can Pass a Series Of Periodically Operated Gates"

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.

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.

The Block Diagram of the DC Motor Speed Control Loop:

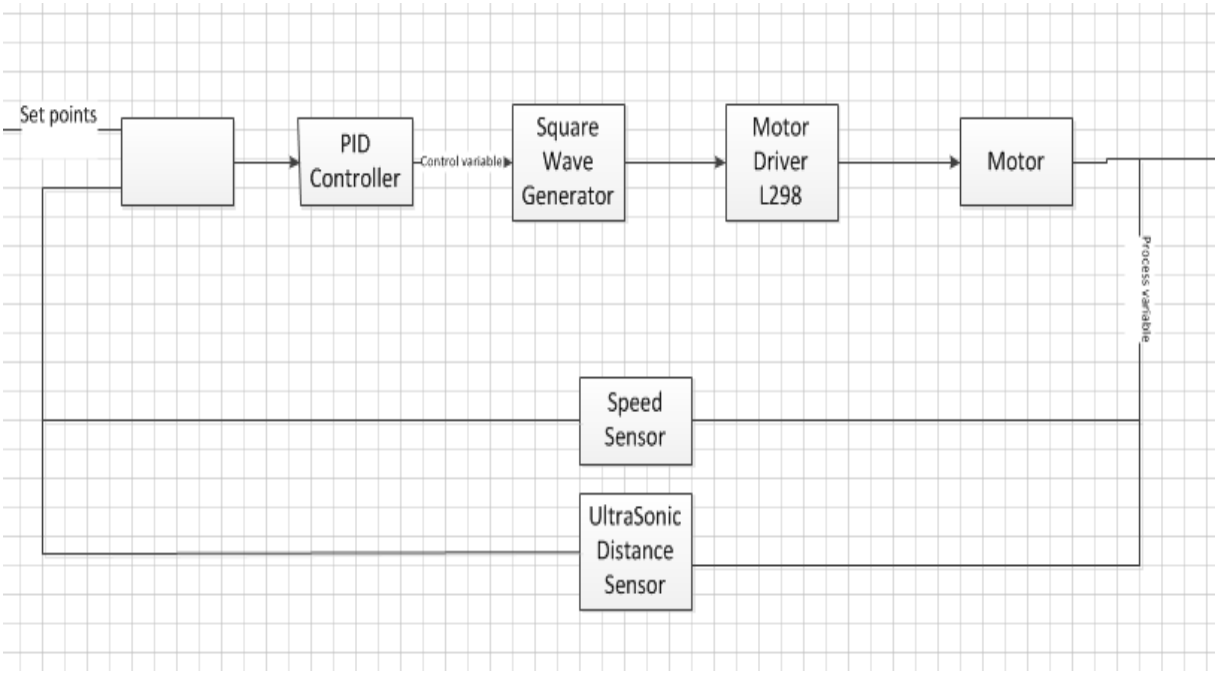

Figure 52: The Block Diagram of the DC Motor Speed Control Loop

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.

PID Parameters:

PID controller can be investigated under 3 main categories. Each controller has different properties in terms of controlling the whole system.

In proportional control, adjustments are based on the current difference between the actual and desired speed.

In ıntegral control, adjustments are based on recent errors.

In derivative control, adjustments are based on the rate of change of errors.

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;

- 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%

The Schematic of the DC Motor:

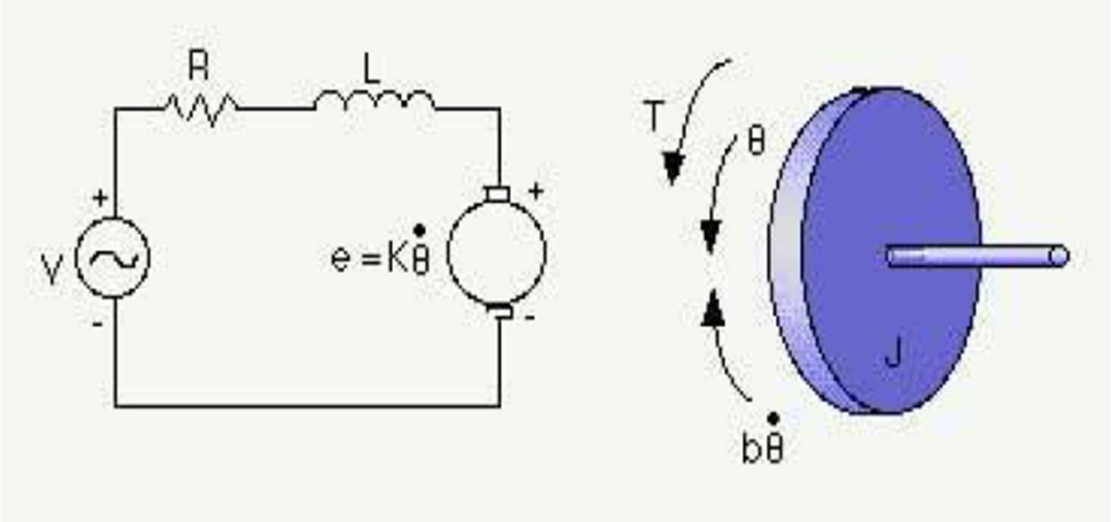

Figure 53-The Schematic of the DC Motor

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;

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

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 53.

- $s(Js + b)Q(s) = KI(s)$
- $(Ls + R)I(s) = V(s) - KQ(s)$

$$\frac{Q(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2}$$

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;

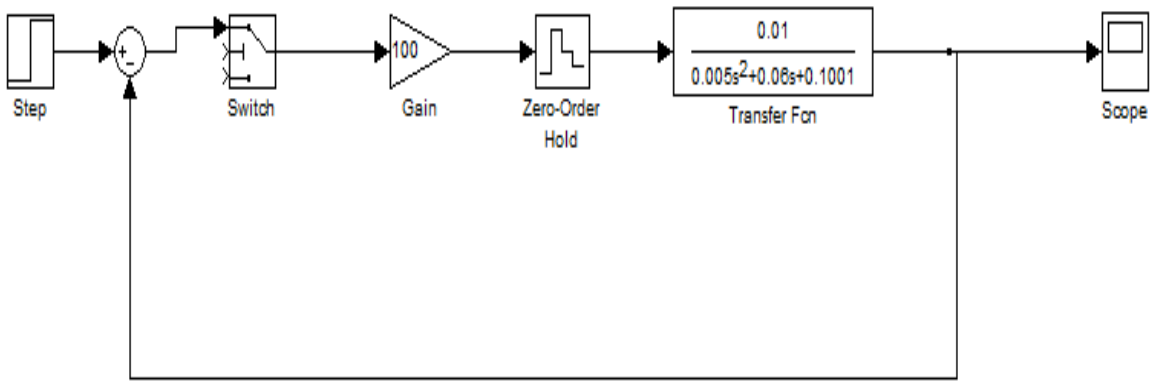

Figure 54-The Block Diagram of the System with Proportional Controller

The MATLAB Result for Kp=100:

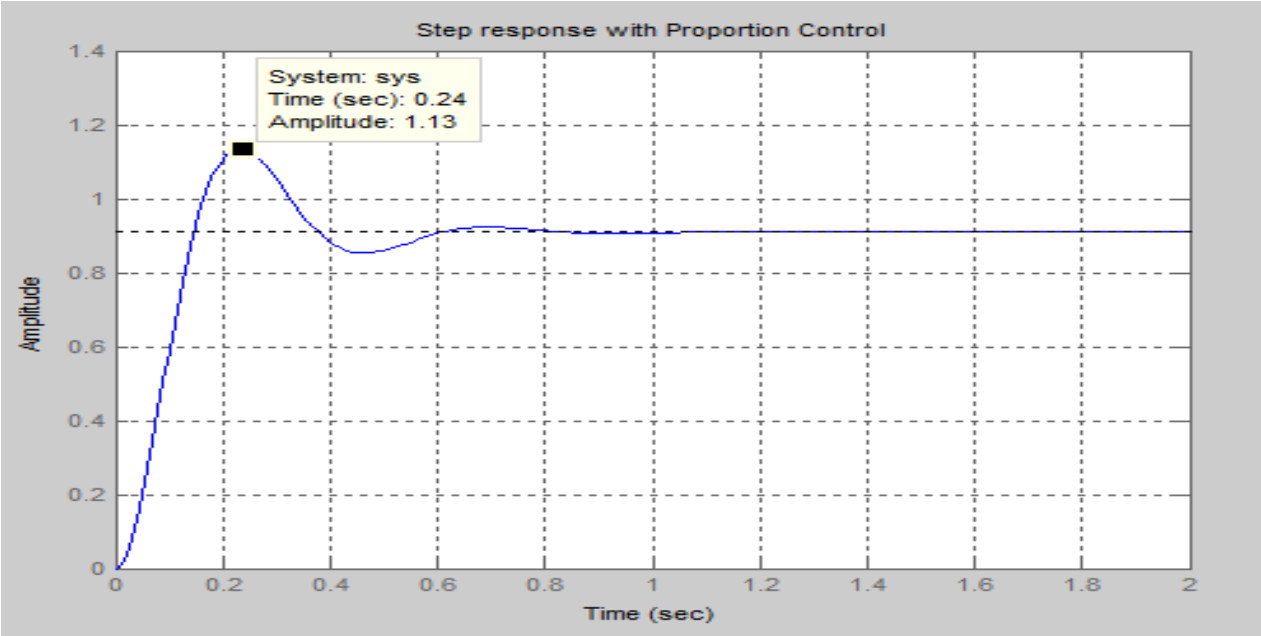

Figure 55-The MATLAB Result, Kp=100

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;

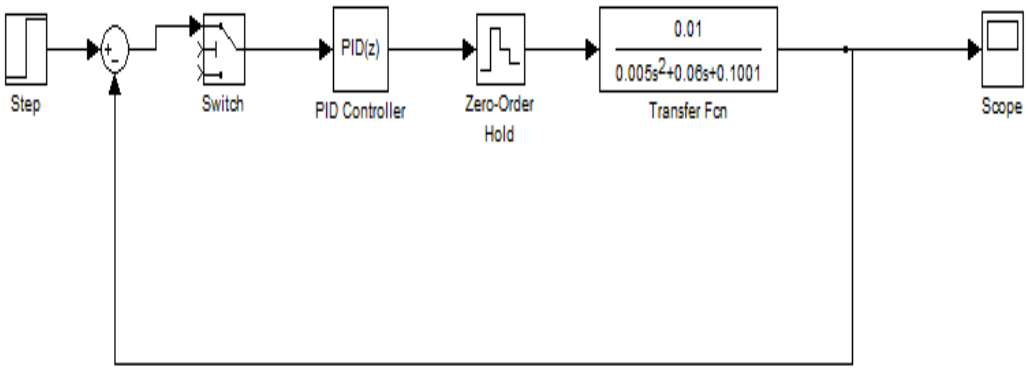

Figure 56-The Block Diagram of the Overall System after Adding İntegral and Derivative Controllers

Inıtially we have chosen both of our integral and derivative controllers' parameters as 1;

The MATLAB Result For Ki=1; Kd=1; Kp=100:

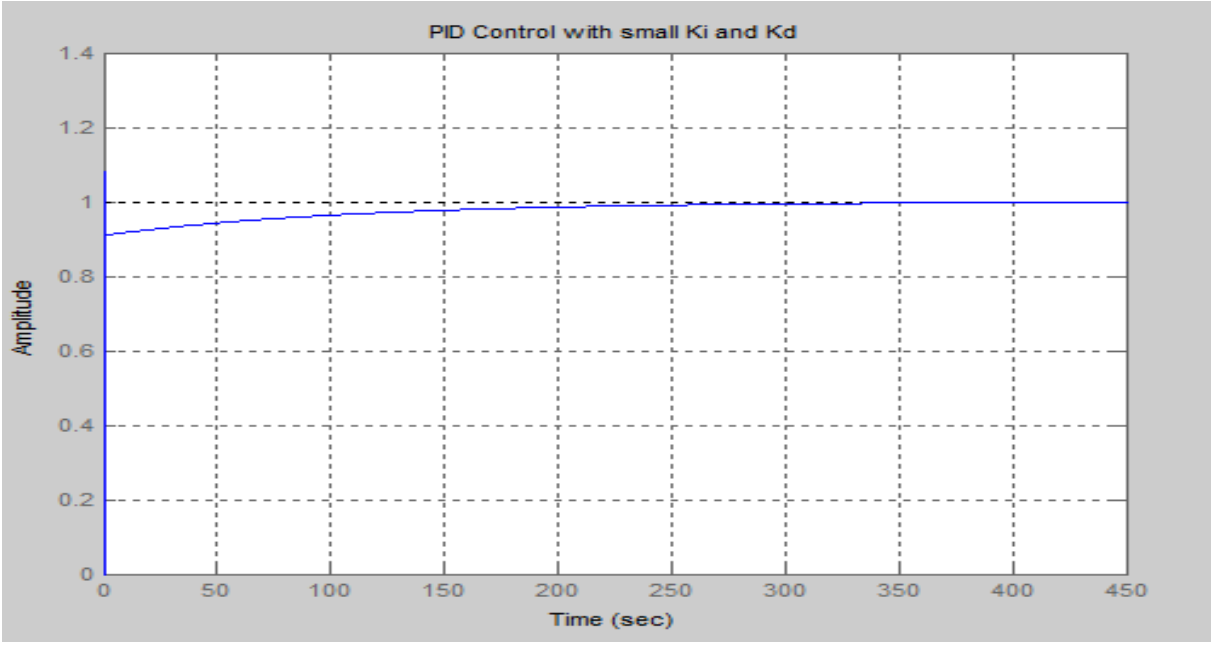

Figure 57-The MATLAB Result for Ki=1, Kd=1, Kp=100

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;

The MATLAB Result For Ki=200; Kd=1; Kp=100:

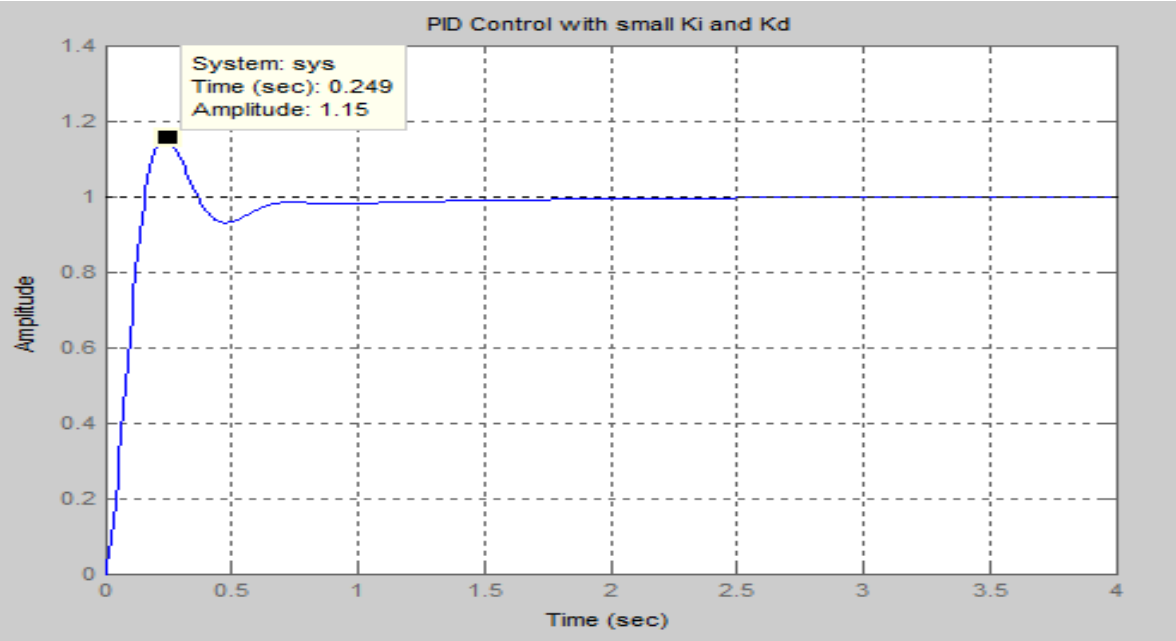

Figure 58- The MATLAB Result for Ki=200, Kd=1, Kp=100

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 $K_i = 200$, $M_p = 15\%$, $e_{ss} = 0$, $t_s = 1\ sec$. The overshoot of the system does not satisfy the design requirement. For that reason let increase the value of Kd;

The MATLAB Result For Ki=200; Kd=10; Kp=100:

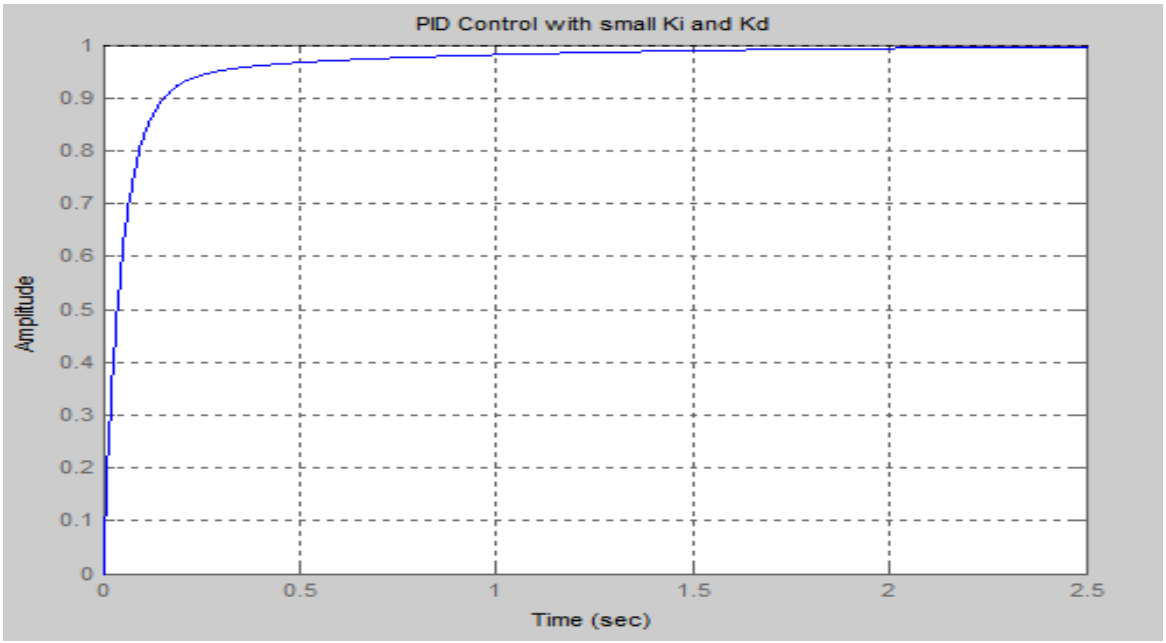

Figure 59- The MATLAB Result for Ki=200, Kd=10, Kp=100

As we have increased the value of Kp, the overshoot value of the system becomes 0, $e_{ss} = 0$ and $t_s = 2\,secs$. 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.

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;

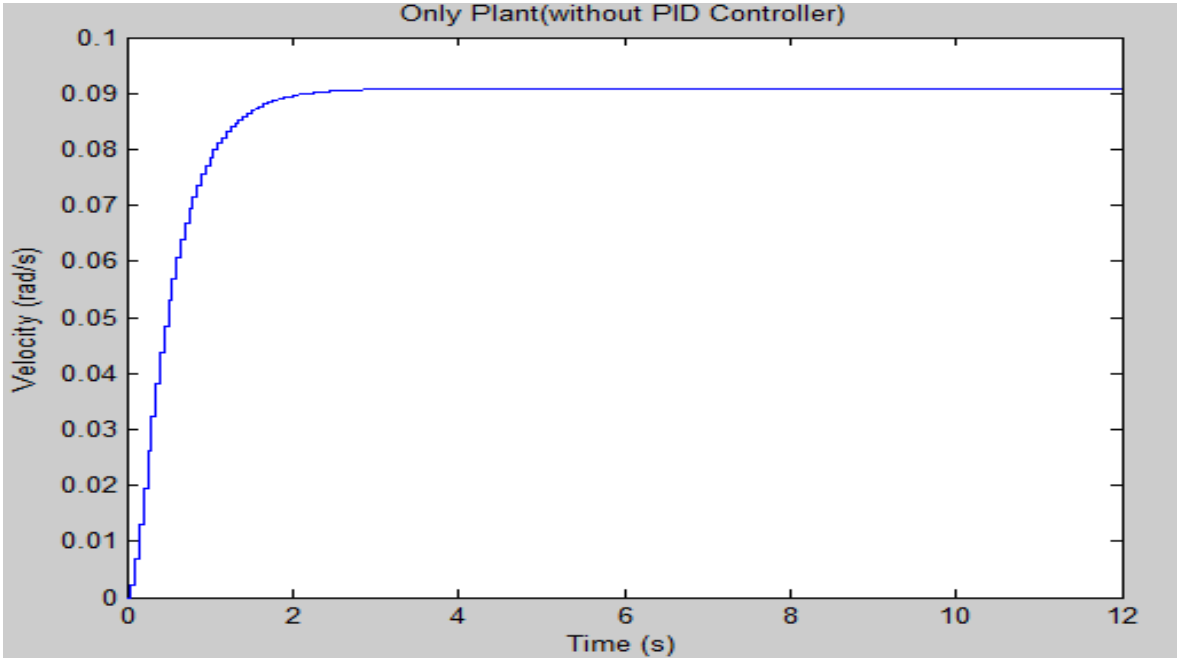

Figure 60-The Step Motor Response of the DC motor without PID controller

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;

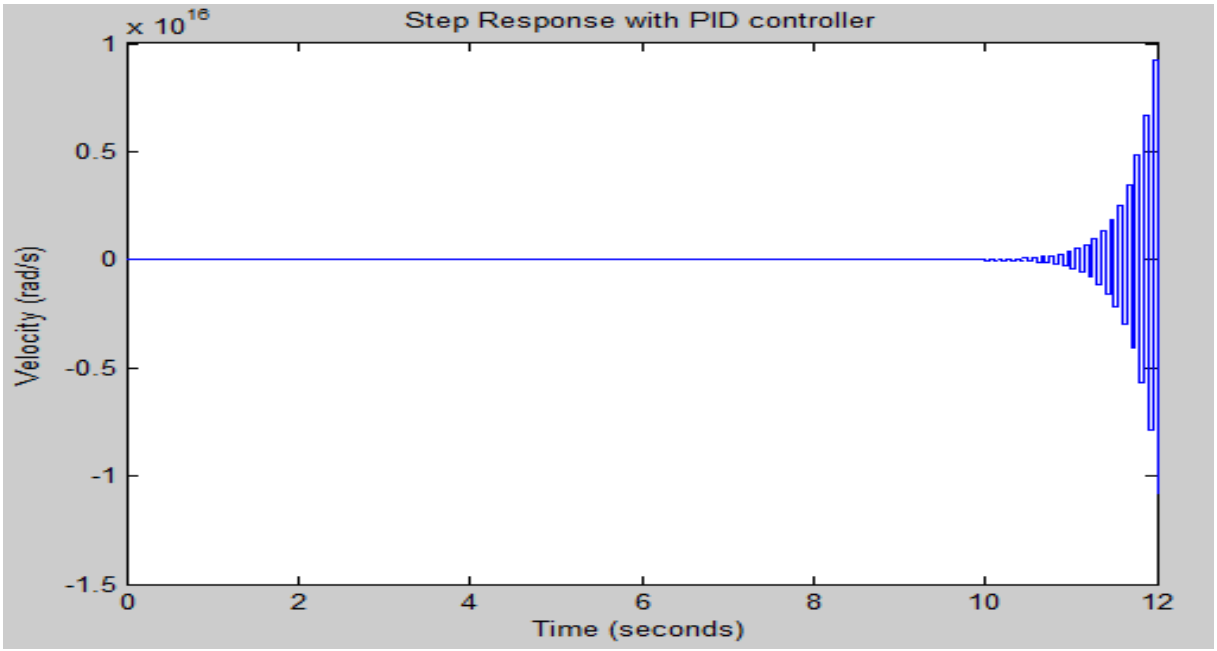

Figure 61-The Step Response of Plant with PID Controller

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;

Root Locus of the Compensated System:

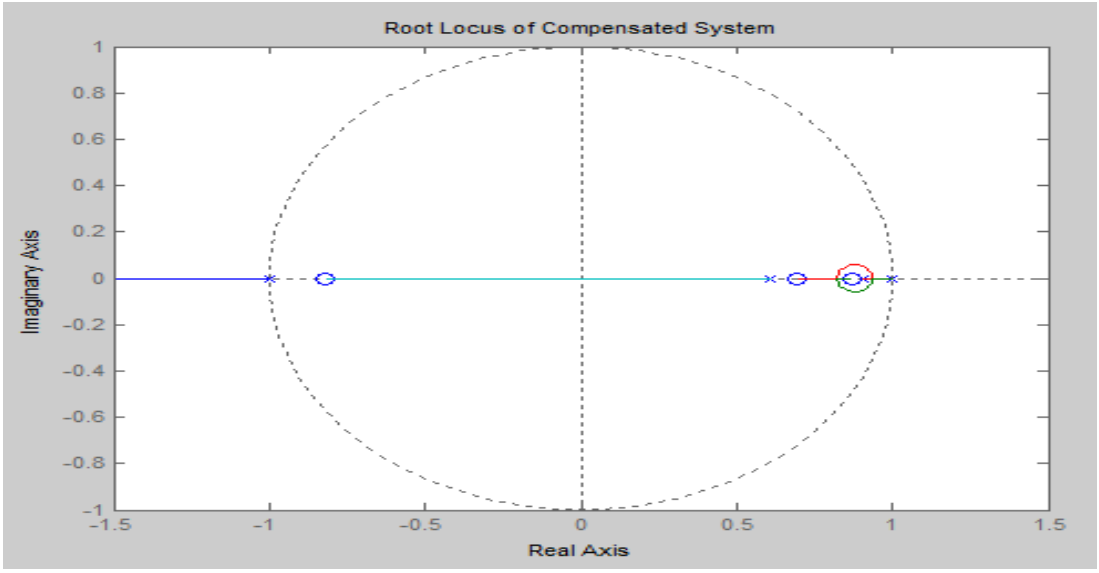

Figure 62-The Root Locus of the Compensated System

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:

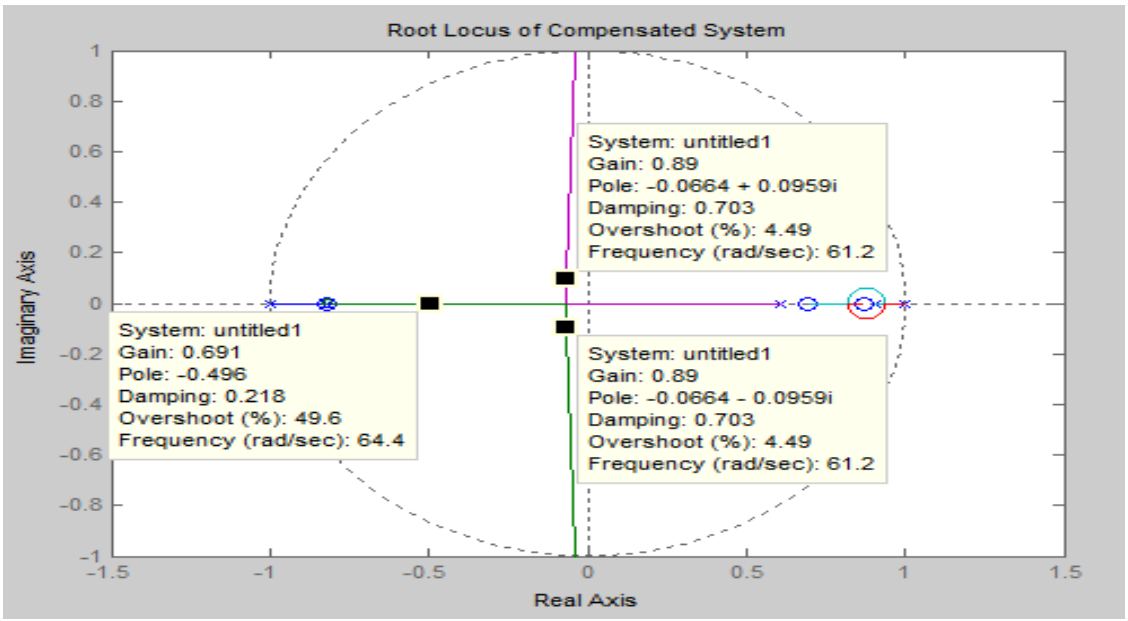

Figure 63-The Root Locus of the Compensated System

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;

The Step Response of the System with P-I-D Controller:

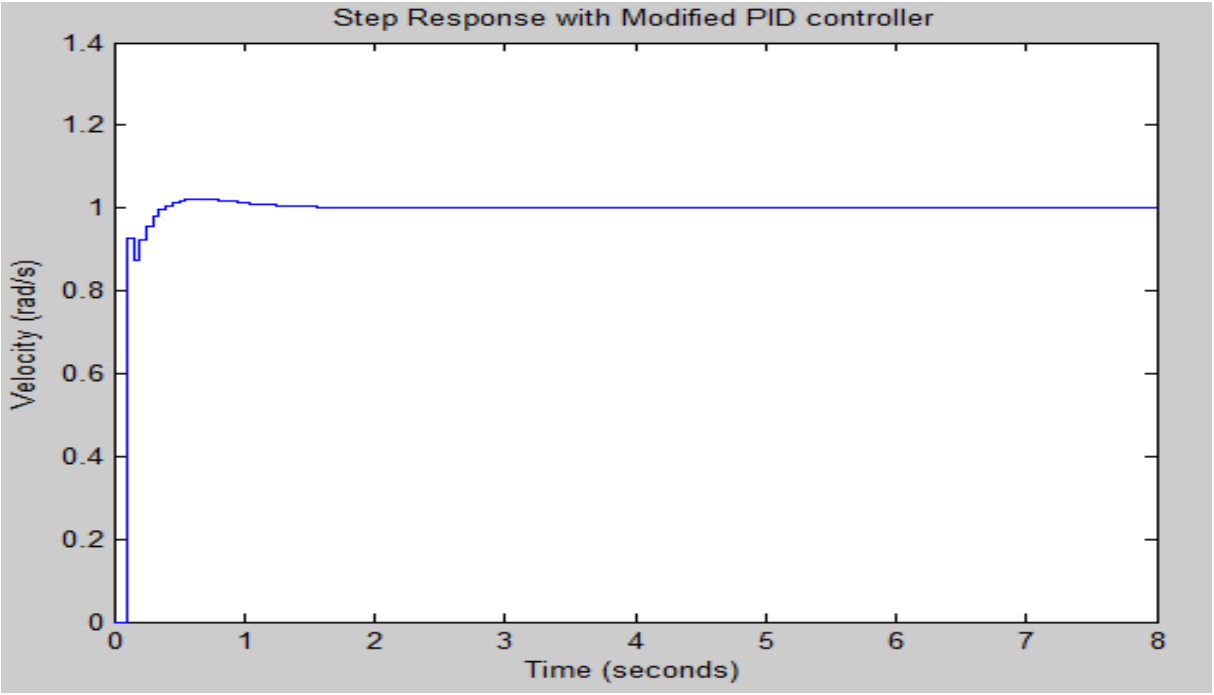

Figure 64-The Step Response of the System with modified P-I-D Controller

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.

P-I-D Controller Design for Controlling DC Motor Position in the Project "A Vehicle Which Can Pass a Series Of Periodically Operated Gates"

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.

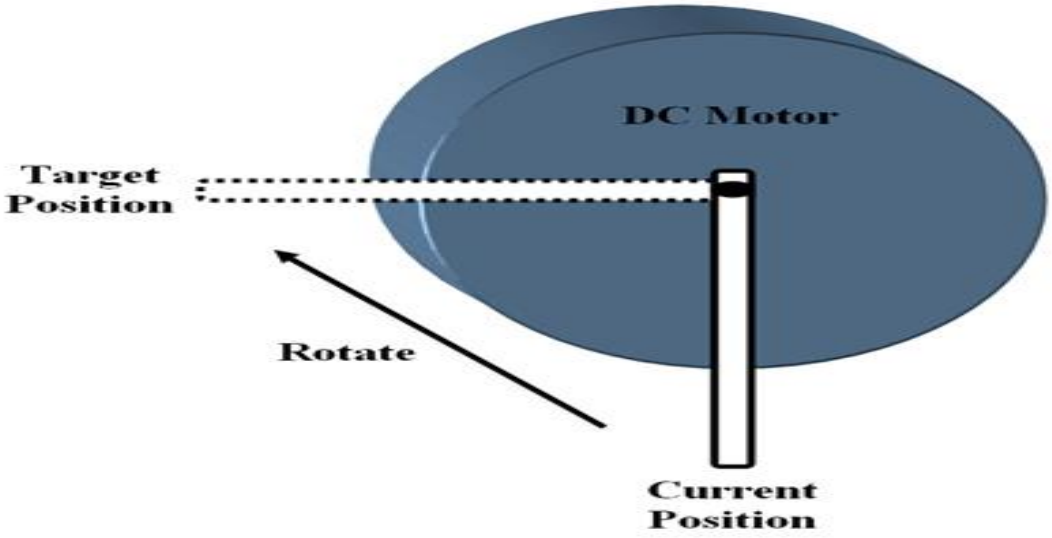

Figure 65-DC Motor Position Control

Before deciding the parameters of the P-I-D, let first derive the transfer function of the DC motor.

$$\frac{Q(s)}{V(s)} = \frac{K}{s * ((Js + b)(Ls + R) + K^2)}$$

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.

$$sQ(s) = speed\ of\ the\ motor$$

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.

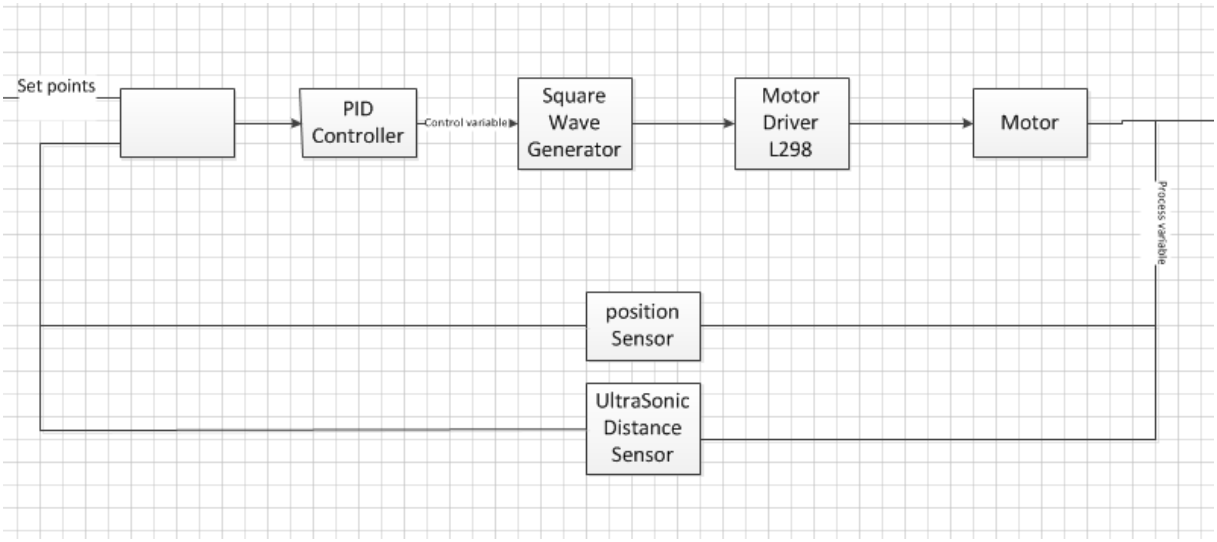

Figure 66-The Block Diagram of the DC Motor Position Control Loop

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.

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.

For Kp=1:

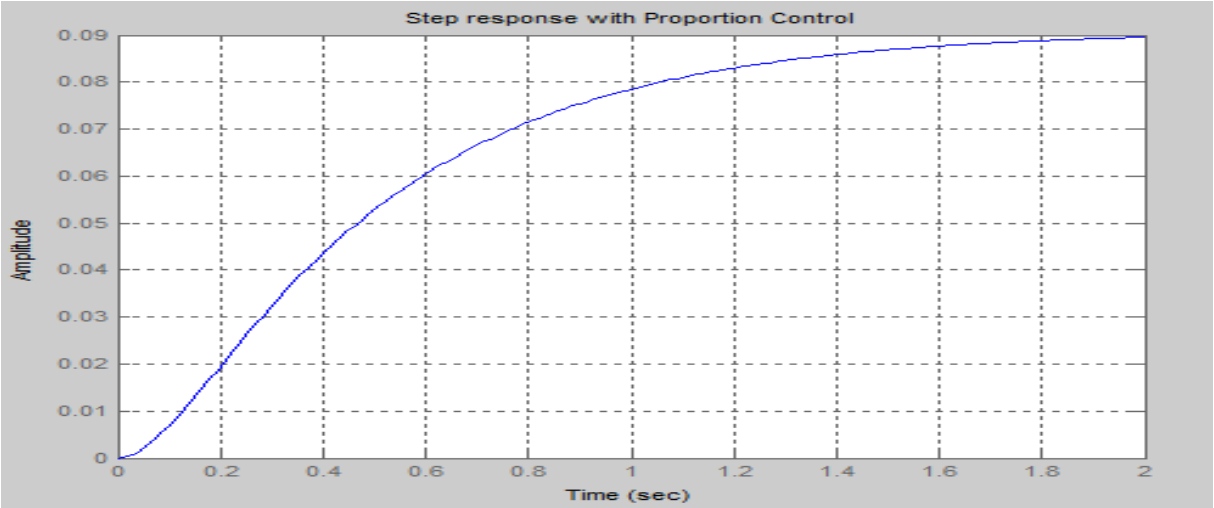

Figure 67-The Step Response of the DC Motor for Kp=1

As it is expected, we have obtained the settling time does not satisfy the design requirement.

For Kp=250:

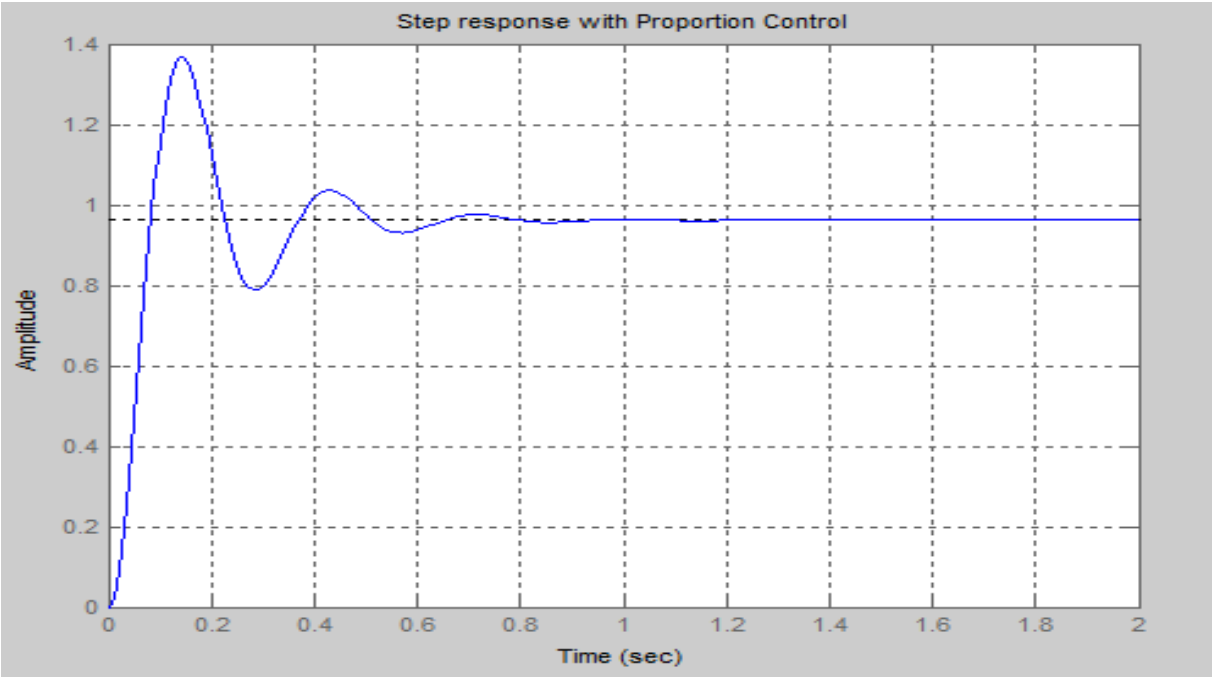

Figure 68- The Step Response of the DC Motor for Kp=250

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.

For Kp=3000:

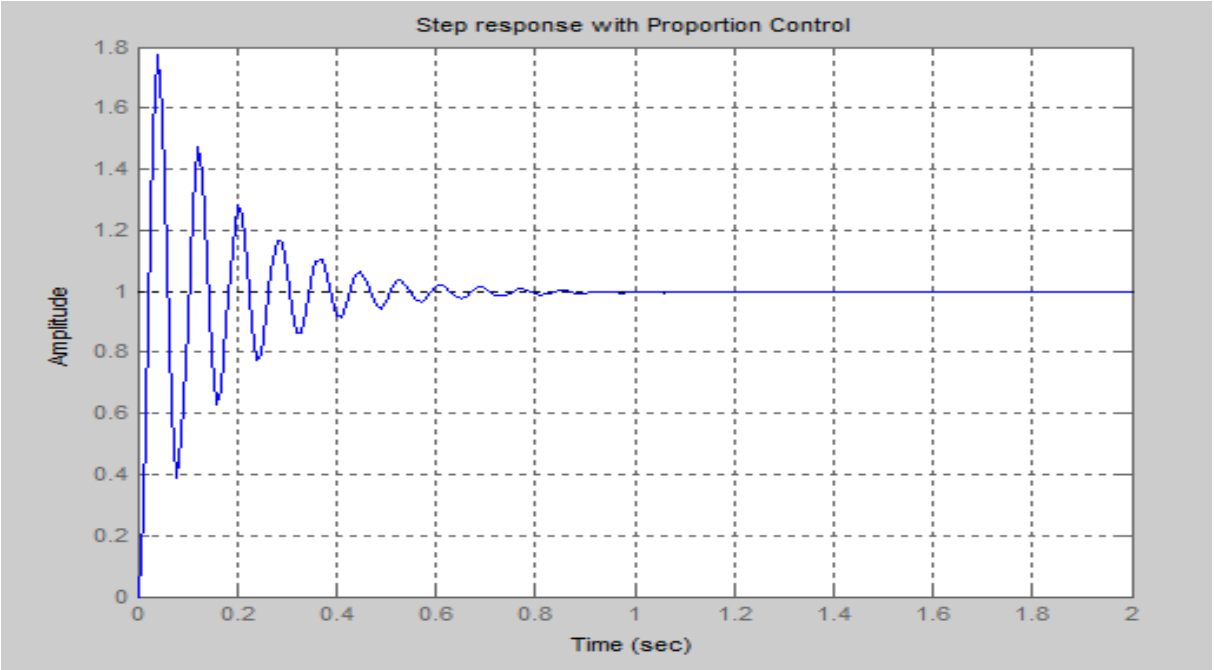

Figure 69- The Step Response of the DC Motor for Kp=3000

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;

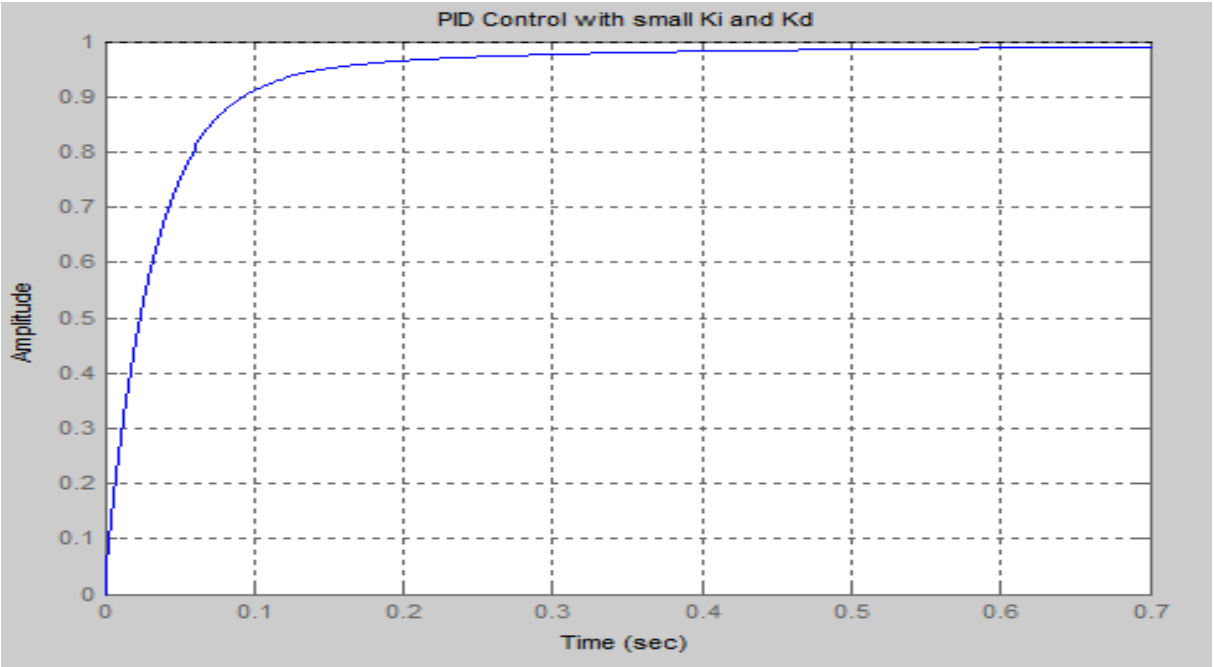

Figure 70-- The Step Response of the DC Motor for Kp=100, Ki=200, Kp=10

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

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;

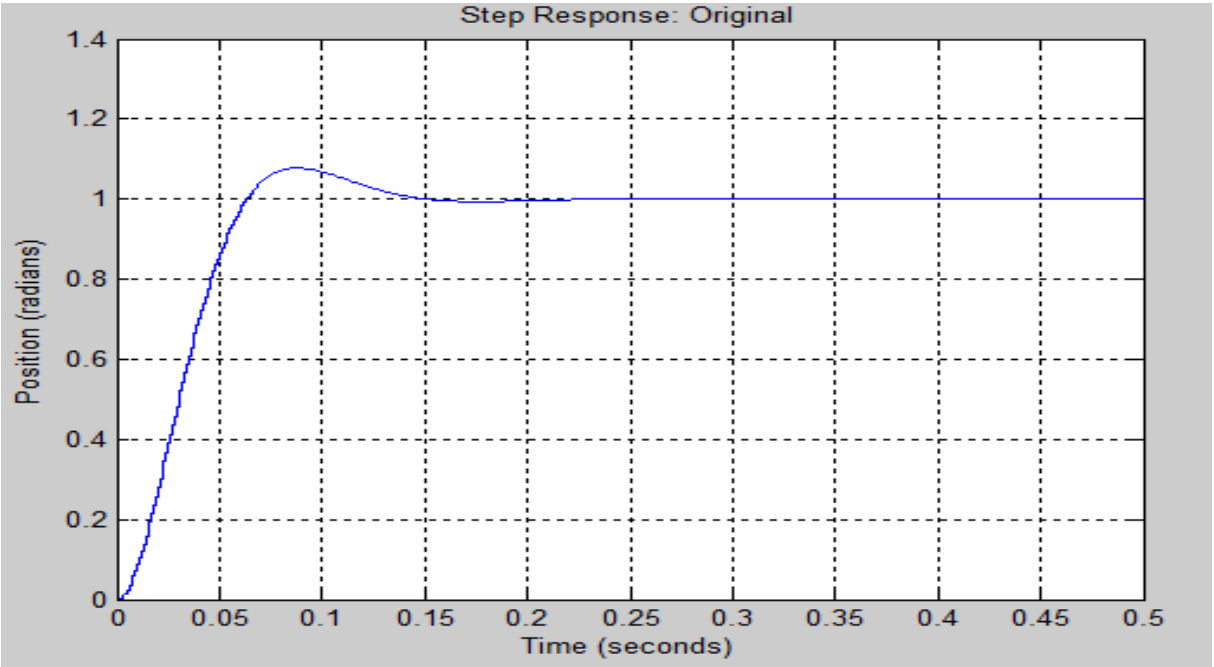

Figure 71-The Step Response of the DC Motor with ZOH

The step response of the compensated system is the following;

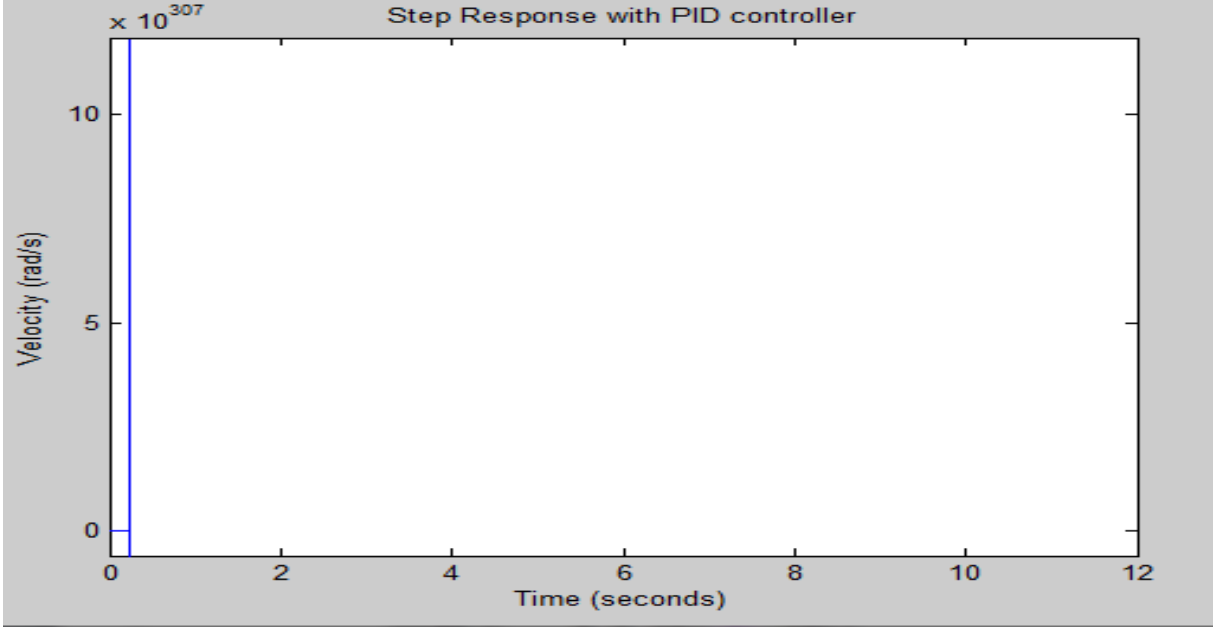

Figure 72-The Step Response of the Compensated System

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:

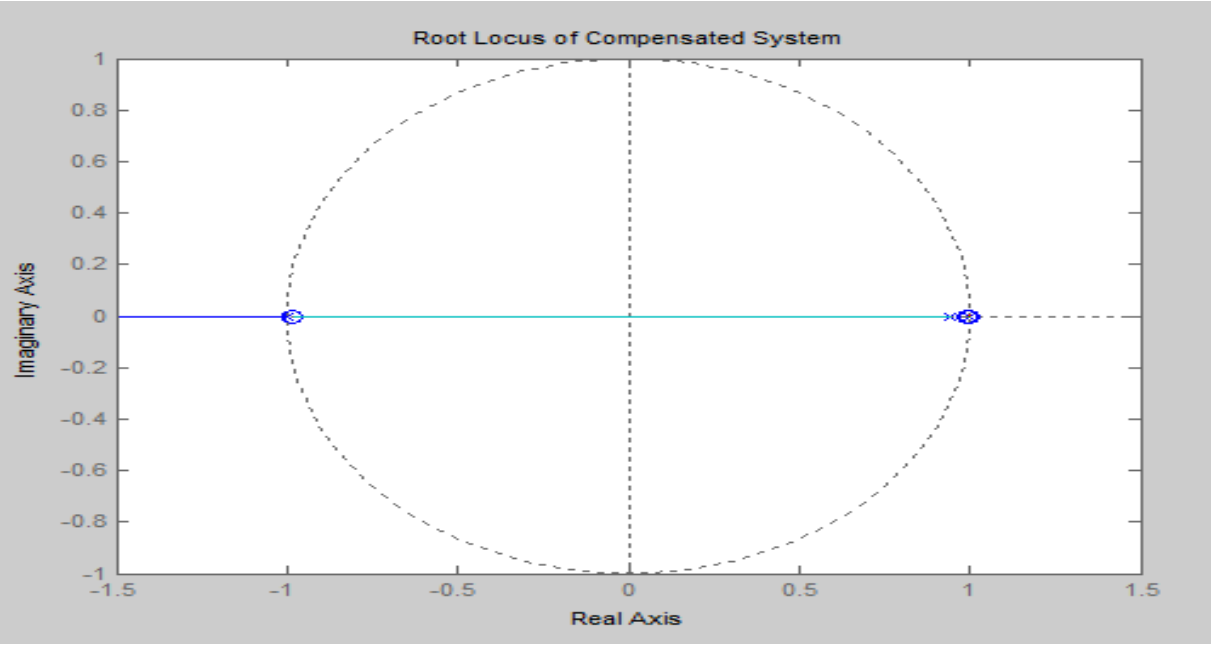

Figure 73-The Root locus of the Compensated System

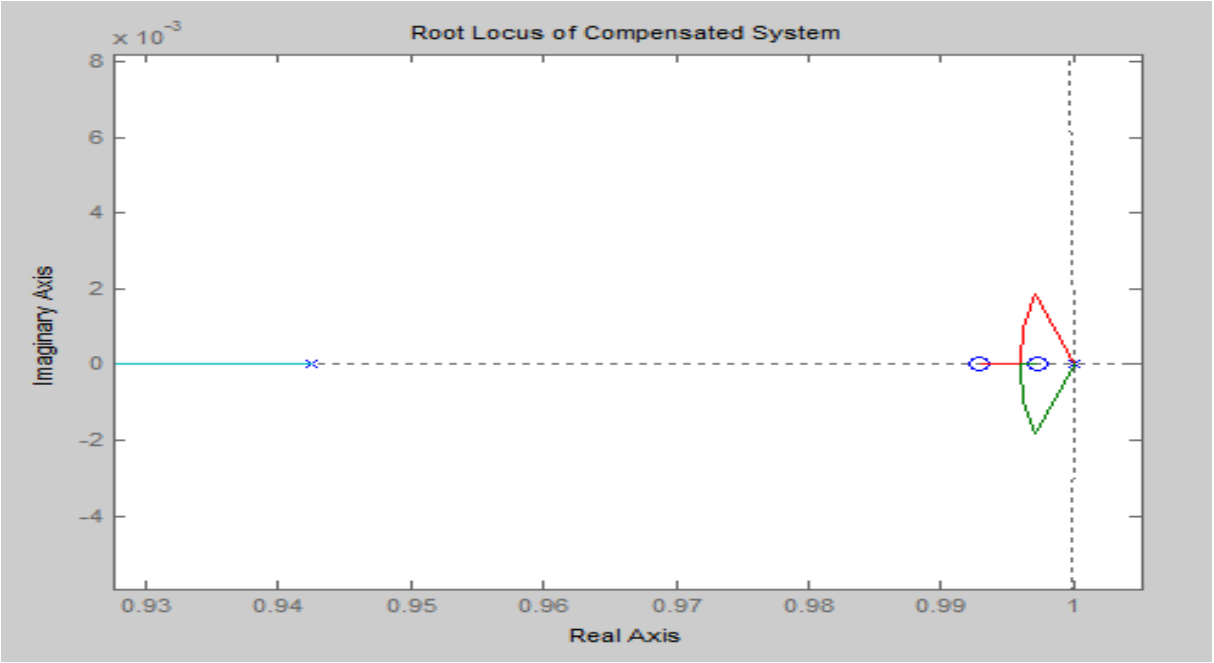

Figure 74-The RHS of the Root Locus

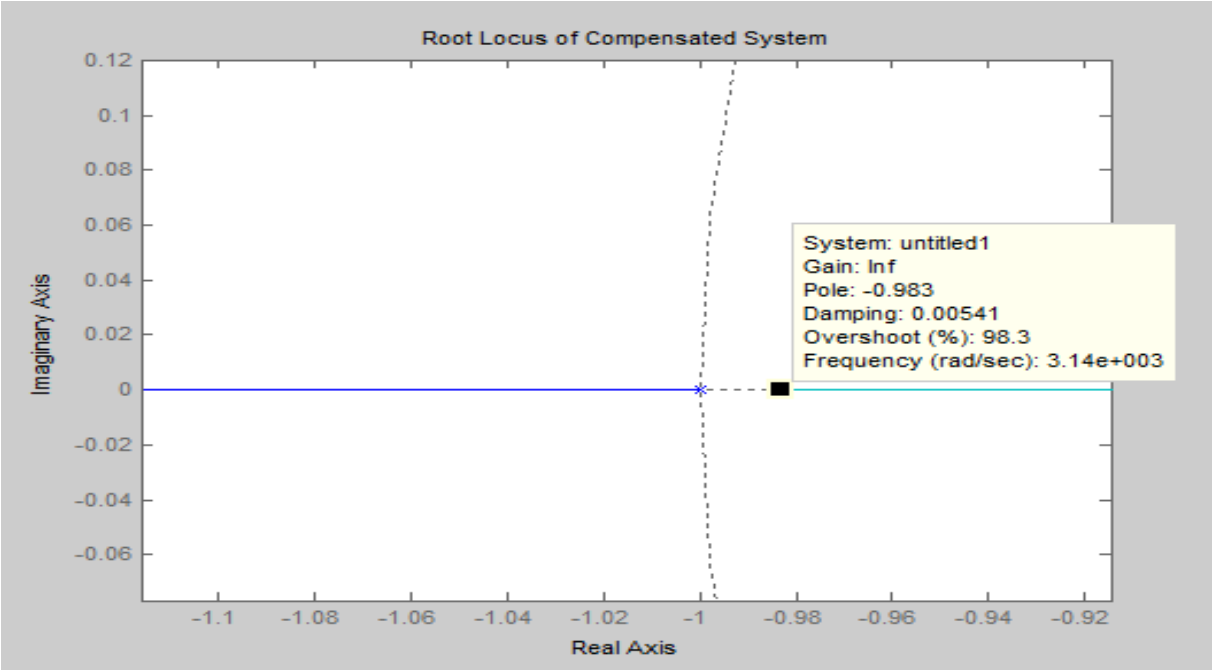

Figure 75- The LHS of the Root Locus

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.

After the addition of the pole at -0.983:

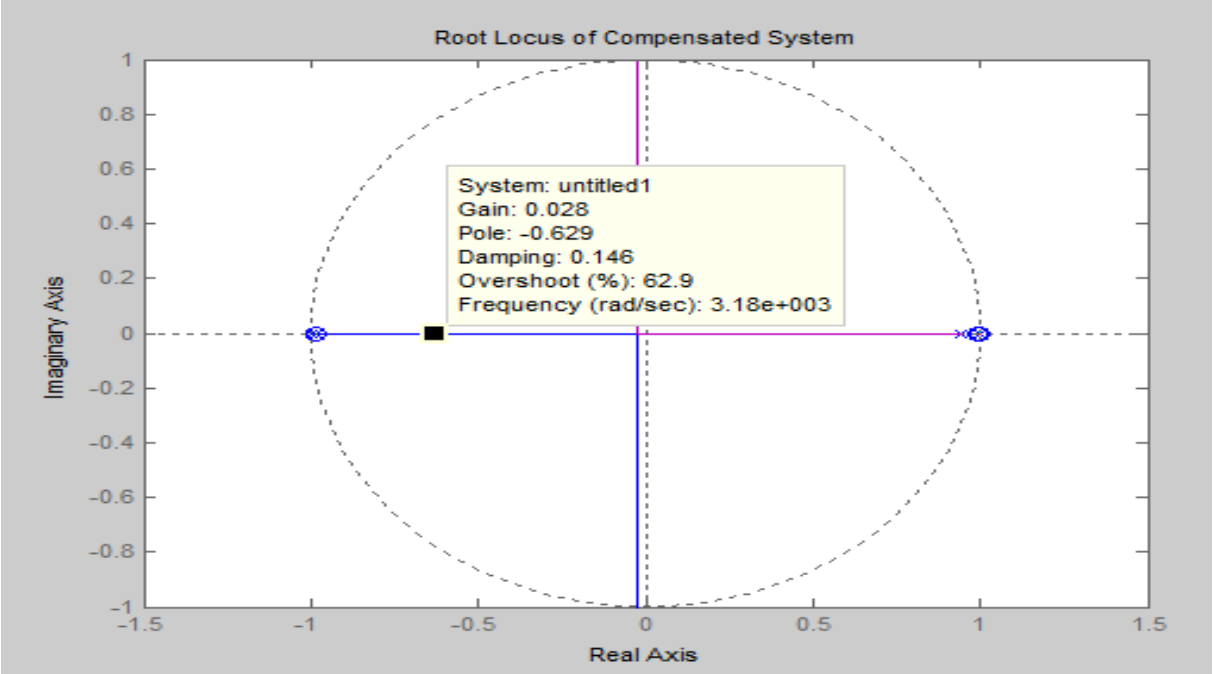

Figure 76-After the Addition of the Pole at -0.983

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.

The step response of the system for gain=0.0028:

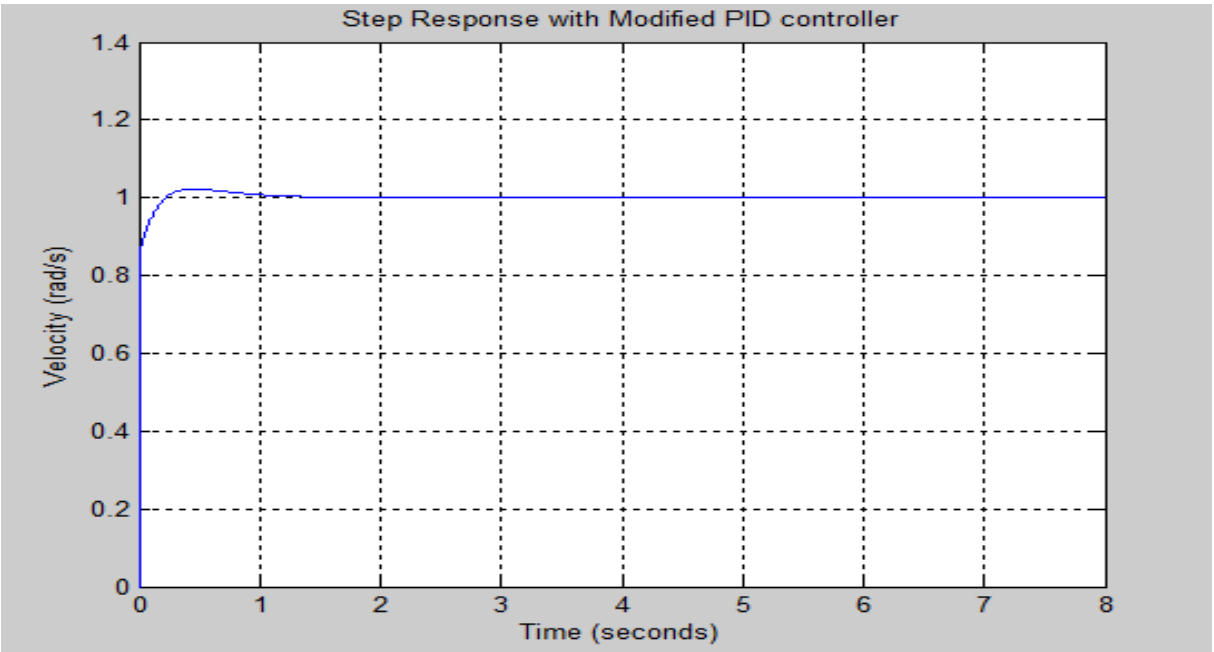

Figure 77-The Step Response of the System for gain=0.0028

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.

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

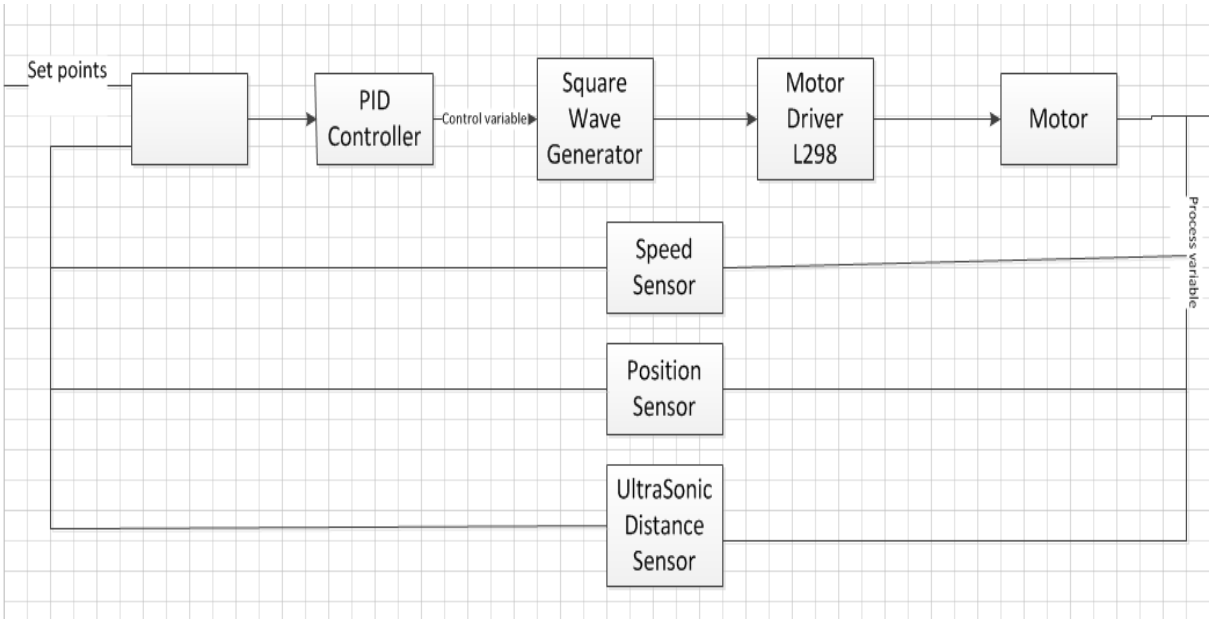

Figure 78-The Block Diagram of Both Speed and Position Control of DC Motor

The whole MATLAB and Arduino codes can be seen in Appendix.

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.

APPENDIX

MATLAB-code showing the effects of sampling time (with proper comments):

```
close all
%% Plant Transfer function
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
s = tf('s');
P_motor = K/((J*s+b)*(L*s+R)+K^2);
zpk(P_motor) % Zero/Pole/Gain
%% Discretized Transfer Function
Ts =0.05; % Sampling Period (Should be less than 2 secs for this plant)
% One can try various values for Ts changing the above value
dP_motor = c2d(P_motor,Ts, 'zoh'); % Continuous to discrete conversion
% One can also try ('matched'),('prewarp',0.5),('tustin'),('foh') methods
% Of course for each type different pole placement may be required to
% stabilize the closed loop system, the we one designed below is for 'zoh'
% method
zpk(dP_motor)
%% Stair step response Feedback system (closed loop) (with no PID)
sys_cl = feedback(dP_motor,1); % closed loop system transfer function
[y,t] = step(sys_cl,12); % Step response of closed loop system
figure, stairs(t,y); % Stair step response (Zero-Order Hold Step Response)
xlabel('Time (s)')
ylabel('Velocity (rad/s)')
title('Stairstep response of open loop System')
%% PID controller
% Note here that instead of conversion z=exp(s*Ts) we use 'tustin' method
% (s=2/Ts*[(z-1)/(z+1)]). Because exp(s*Ts) is transcendental and not
% suitable for computer programing, on the other hand, s=2/Ts*[(z-1)/(z+1)]
%  is just ratio of two polinomials, something that MATLAB can easily
%  handle
Kp = 100;
Ki = 200;
Kd = 10;
C = Kp + Ki/s + Kd*s;
dC = c2d(C,Ts,'tustin') % Tutsin is a method for bilinear transformation
% 'tutsin' converts continuous model to discrete time
%% Stair step response Feedback system with PID
sys_cl = feedback(dC*dP_motor,1);
[x2,t] = step(sys_cl,12);
figure, stairs(t,x2)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
title('Stairstep response of closed loop system with PID')
%% Pole addition to stabilize (for 'zoh' method)
% may or may not work for other methods ! Designer may/may not need to
% redesign if he/she changes the transformation method !!
z = tf('z',Ts);
dC = dC/(z+0.82);
%% Stair step response Feedback system with PID after pole placement
sys_cl = feedback(0.8*dC*dP_motor,1);
[x3,t] = step(sys_cl,8);
figure, stairs(t,x3)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
```

```
title('Stairstep response of closed loop system PID(after pole placement)')
%% root locus (if it is needed to see what is going on with the poles)
%{
figure, rlocus(dC*dP_motor)
axis([-1.5 1.5 -1 1])
title('Root Locus of Compensated System')
%}
```

MATLAB-code (for Speed Control)

```
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
s = tf('s');
P_motor = K/((J*s+b)*(L*s+R)+K^2);
zpk(P_motor)
Ts = 0.05;
dP_motor = c2d(P_motor,Ts,'zoh');
zpk(dP_motor)
sys_cl = feedback(dP_motor,1);
[y,t] = step(sys_cl,12);
stairs(t,y);
xlabel('Time (s)')
ylabel('Velocity (rad/s)')
title('Only Plant(without PID Controller)')
%%
%adding PID controller
Kp = 100;
Ki = 200;
Kd = 10;

C = Kp + Ki/s + Kd*s;
dC = c2d(C,Ts,'tustin')

sys_cl = feedback(dC*dP_motor,1);
[x2,t] = step(sys_cl,12);
stairs(t,x2)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
title('Step Response with PID controller')
%%
rlocus(dC*dP_motor)
axis([-1.5 1.5 -1 1])
title('Root Locus of Compensated System')
%%
z = tf('z',Ts);
dC = dC/(z+0.82);
rlocus(dC*dP_motor);
axis([-1.5 1.5 -1 1])
title('Root Locus of Compensated System');
%%
sys_cl = feedback(0.89*dC*dP_motor,1);
[x3,t] = step(sys_cl,8);
stairs(t,x3)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
title('Step Response with Modified PID controller')
```

MATLAB-code (for Position Control)

```
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
s = tf('s');
P_motor = K/(s*((J*s+b)*(L*s+R)+K^2));
zpk(P_motor)

Ts = 0.001;
dP_motor = c2d(P_motor, Ts, 'zoh');
zpk(dP_motor)

dP_motor = minreal(dP_motor,0.001);
zpk(dP_motor)

sys_cl = feedback(dP_motor,1);
[x1,t] = step(sys_cl,.5);
stairs(t,x1)
xlabel('Time (seconds)')
ylabel('Position (radians)')
title('Step Response: Original')
grid
%%
Kp = 100;
Ki = 200;
Kd = 10;

C = Kp + Ki/s + Kd*s;
dC = c2d(C,Ts,'tustin')

sys_cl = feedback(dC*dP_motor,1);
[x2,t] = step(sys_cl,12);
stairs(t,x2)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
title('Step Response with PID controller')
%%
rlocus(dC*dP_motor)
axis([-1.5 1.5 -1 1])
title('Root Locus of Compensated System')
%%
z = tf('z',Ts);
dC = dC/(z+0.982);
rlocus(dC*dP_motor);
axis([-1.5 1.5 -1 1])
title('Root Locus of Compensated System');
%%
sys_cl = feedback(0.028*dC*dP_motor,1);
[x3,t] = step(sys_cl,8);
stairs(t,x3)
xlabel('Time (seconds)')
ylabel('Velocity (rad/s)')
title('Step Response with Modified PID controller')
grid on
```

Arduino P-I-D controlled DC motor speed and position control

```
int integral=0;
int derivative=0;
int error=0;
int previous_error=0;
long sensors_average;
int sensors_sum;
int position;
long sensors[] = {0, 0, 0, 0};   // Array used to store 5 readings for 5sensors.
 void setup()
 {
  Serial.begin(9600);
 }
  void loop()
  {
    sensors_read();  //Reads sensor values and computes sensor sum and weighted average
    pid_calc();      //Calculates position[set point] and computes Kp,Ki and Kd
    calc_turn();     //Computes the error to be corrected
    motor_drive(right_speed, left_speed);  //Sends PWM signals to the motors
  }
  void sensors_read()
 {
 sensors_average = 0;
 sensors_sum = 0;

for (int i = 0; i < 4; i++)
 {
  sensors[i] = analogRead(i);
```

```
 sensors_average += sensors[i] * i * 1000;   //Calculating the weighted mean
 sensors_sum += int(sensors[i]);
}                                  //Calculating sum of sensor readings


 position = int(sensors_average / sensors_sum);
 Serial.print(sensors_average);
 Serial.print(' ');
 Serial.print(sensors_sum);
 Serial.print(' ');
 Serial.print(position);
 Serial.println();
 delay(2000);
}
void pid_calc()
{
 position = int(sensors_average / sensors_sum);
 error = position – set_point;       // Replace set_point by your set point-Aracı siyah çizginin ortasına koy serial porttan okudugun position datası
 integral = integral + error;
 derivative = error - previous_error;
 previous_error = error;
 error_value = int(error * Kp + integral * Ki + derivative * Kd);
}
 void calc_turn()
{  //Restricting the error value between +256.
 if (error_value< -256)
 {
        error_value = -256;
 }
```

```
  if (error_value> 256)
  {
        error_value = 256;
  }
// If error_value is less than zero calculate right turn speed values
  if (error_value< 0)
  {
       right_speed = max_speed + error_value;
       left_speed = max_speed;
  }
// If error_value is greater than zero calculate left turn values
  else
  {
      right_speed = max_speed; // max value should be defined
      left_speed = max_speed - error_value; //max value should be defined
  }
  }
 void motor_drive(int right_speed, int left_speed)
  {
// Drive motors according to the calculated values for a turn
  analogWrite(motor_right, right_speed); //motor_right the pin which the right motor is connected (Eksik var-pin numarası)
  analogWrite(motor_left, left_speed);    //motor_left the pin which the left motor is connected (Eksik var-pin numarası)
  delay(50);          // Optional
  }
```