Department of

Engineering and Design

University of Sussex

Case Study Number: 10

Candidate No: 181395

H6105: Control Engineering

Final Report

Date: 15/01/21

Word Count:

Table of Contents

1 Introduction

The purpose of this project was to design and evaluate different control methods for controlling the blade pitch angle of a wind turbine through the software MATLAB and Simulink. In industry, different methods are used depending on the system, for example they may want to control temperature or pressure. Therefore, a business may want to test multiple control methods to decide which is the most efficient or cost effective. These are important factors in respect to this project as the pitch blade angle of a wind turbine is very important. It is used to maintain a smooth power output so that there is a steady rate of power generated, rather than the power being generated proportionately to the wind which is very inconsistent and unreliable. The controllers that were designed in this project behaved differently to inputs as well as any disturbances therefore, the one that could be chosen would depend on the characteristics a client would desire for their system. These characteristics are what we have discussed and evaluated in this project.

The project objectives were as follows:

- To understand the behaviour of the system and simulate the response of the plant with no controller or disturbances.
- To design P, PI and PID controllers and analyse their characteristics.
- To design digital PI and PID controllers by converting the previous controllers from the continuous to discrete domain (analogue to digital).
- To observe and analyse how the digital PI and PID controllers respond to different time samples.
- To design lead and lag compensators and analyse how they affect the system's stability margins.
- To design a State-Space using state feedback control by assigning poles for the closed-loop system.
- To design an observer feedback system by assigning poles for the closed-loop system that contains the L matrix.
- To test how each type of control method performs when disturbances are put into the system.
- To discuss and evaluate the benefits of each system.

2 System Behaviour

2.1 Obtaining the Transfer Function

The first step was to obtain the transfer function for the plant. This is shown in figure 1 as "Turbine, motor and gears". This figure shows a block diagram representing the system I built, with a different controller each time. This is a feedback-controlled system where the controller adjusts the output based on the error between the desired and actual rotor speed, $\omega_d(s)$ and $\omega(s)$. $T_d(s)$ is a disturbance to the system which was tested later on.

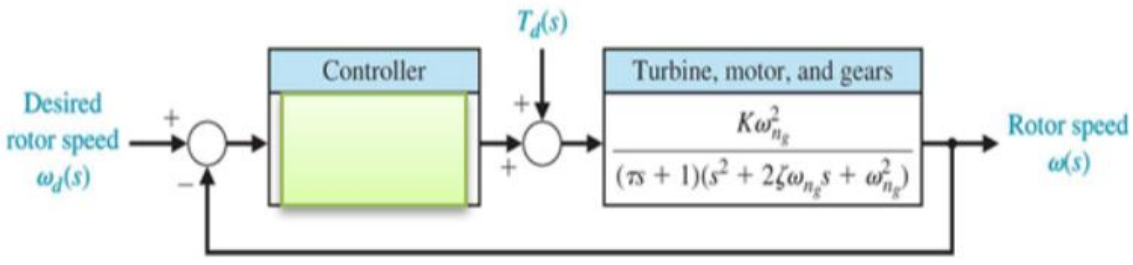


Figure 1: Block Diagram for Pitch Blade Angle

The transfer functions used in the plant were different for each student due to each being given different parameters. As seen in figure 1, the transfer function depends on the values of the variables within. The parameters I was given were as follows:

$$K = 2.3 \qquad \omega_{ng} = 1.9 \qquad \tau = 0.7 \qquad \xi\omega_{ng} = 1.2$$

The first step was to put these parameters into the transfer function given for the plant.

$$\frac{K\omega_{ng}^2}{(\tau s + 1)(s^2 + 2(\xi\omega_{ng})s + \omega_{ng}^2)} = \frac{(2.3)(1.9^2)}{\big((0.7)s + 1\big)(s^2 + 2(1.2)s + 1.9^2)}$$

$$= \frac{8.303}{(0.7s + 1)(s^2 + 2.4s + 3.61)} = \frac{8.303}{(0.7s + 1)(s^2 + 2.4s + 3.61)}$$

My transfer function is:

$$Turbine, motor\ and\ gears = \frac{8.303}{(0.7s^3 + 2.68s^2 + 4.927s + 3.61)}$$

2.2 Open Loop System

Once we found our transfer function to be used for our plant (Turbine, motor and gears), we could build an open loop system through Simulink, to observe how the plant responds to different inputs. This open loop system was like that shown in figure 1 but without the controller or any disturbances.

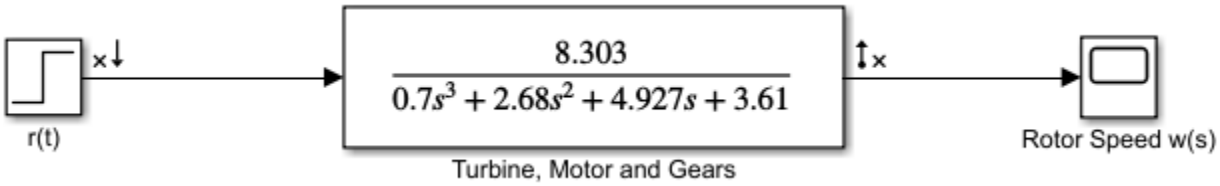


Figure 2: Simulink Block Diagram for Open Loop System

Figure 2 shows a unit step function r(t) = 1, being used as the input to the system and a scope to measure the output (response) of the system. Running the simulation allows us to observe the output graph recorded on the scope, the response of the transfer function that I just calculated. A linear analysis tool allowed me to view step and impulse responses as well as the pole-zero map to view the characteristics of the system in more detail.

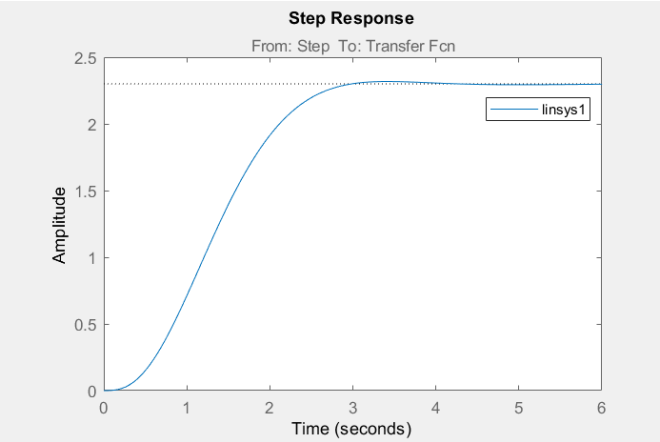


Figure 3: Step Response Using Linear Analysis Tool

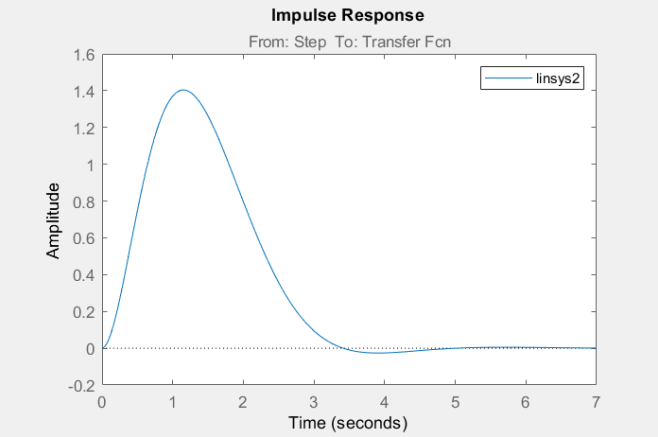


Figure 4: Impulse Response of Open-Loop System

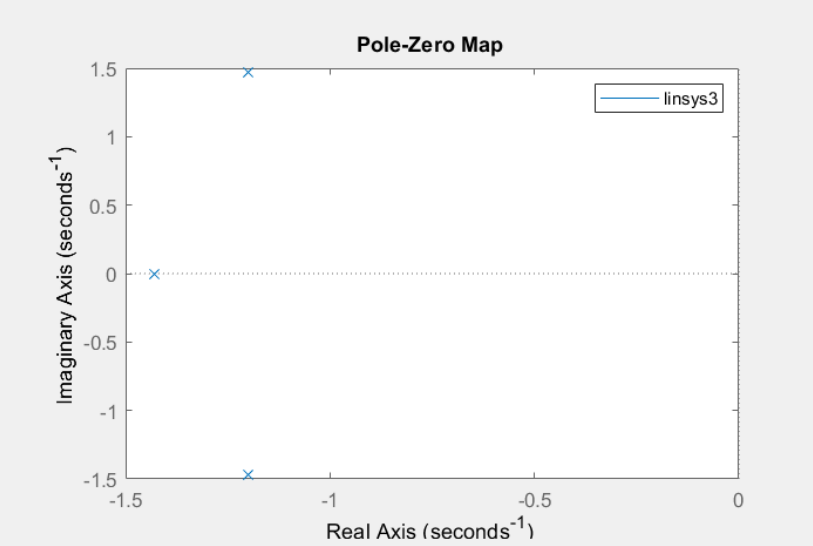


Figure 5: Pole-Zero Map of Open-Loop System

The pole-zero map is a representation of the 's-plane' and is shown in figure 6 and we can see that there are 2 poles at (-1.2+1.47i) and (-1.2-1.47i) and a zero at (-1.43). This shows that the system is stable as all poles and zeros are towards the left of the real axis. If any were on the right, the system would be unstable.

The linear analysis tool also allows us to observe other pieces of data through the MATLAB command stepinfo(linsys). These are things like the rising or settling time of the system as well as the steady state value.

t_r = 1.632s $t_s(2\%)$ = 2.698s y_{ss} = 2.3 e_{ss} = 1.3 OP% = 0.77%

2.3 Closed-Loop System

Similar to the open-loop control system previously built, I then built a closed-loop control system through the use of Simulink. This was the same system as that in figure 1, except there was no controller or disturbance included.

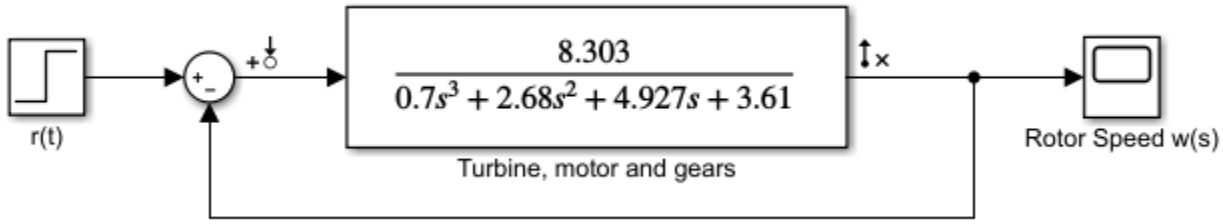


Figure 6: Block Diagram of Closed-Loop System

Again, through linear analysis we were able to observe different responses of the system from different inputs.

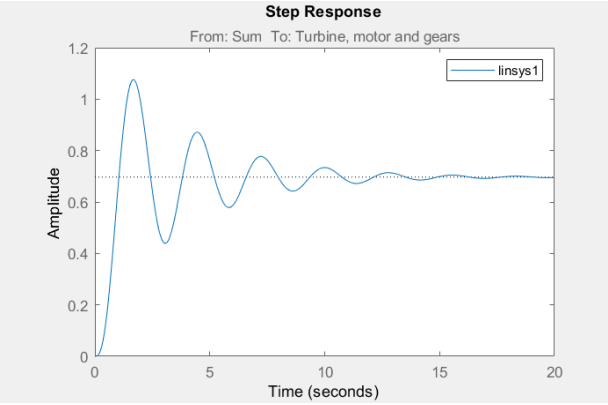


Figure 7: Step Response of Closed Loop System

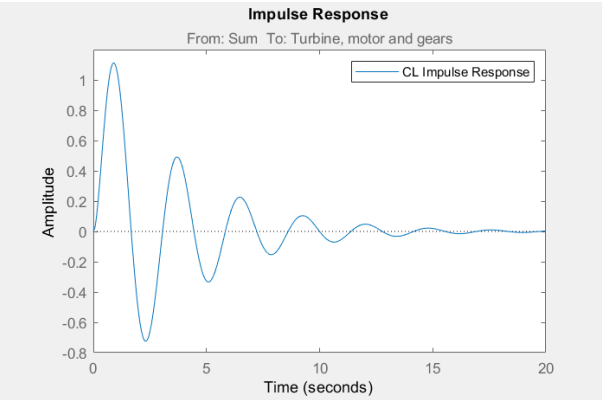


Figure 8: Impulse Response of Closed Loop System

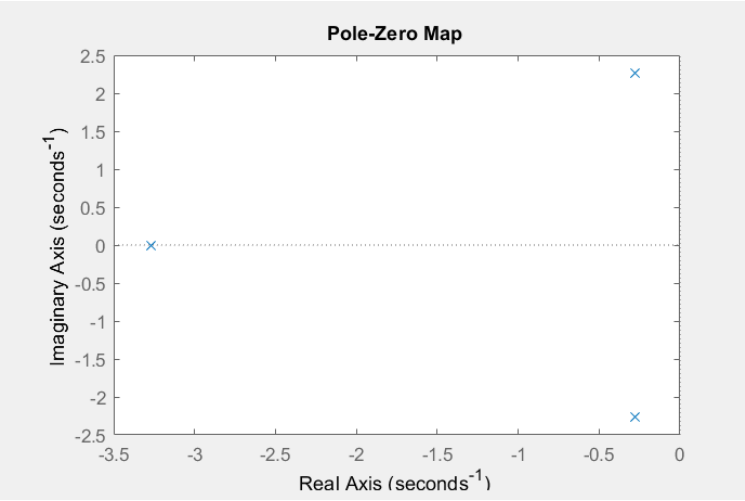


Figure 9: Pole-Zero Map of Closed Loop System

From figure 10, much like the open loop system, we can see that there are 3 poles at (-0.28+2.26i), (-3.27) and (-0.28-2.26i). This again shows that the system is stable as all poles are towards the left of the real axis.

Again, the linear analysis tool allows us to observe other pieces of data from the system.

$t_r = 0.6074s$ $t_s = 13.04s$ $y_{ss} = 0.697$ $e_{ss} = -0.303$ $OP\% = 54.36\%$

As you can observe from the open loop and closed loop system, the step and impulse responses are different depending on the system. The open loop step response has very little overshoot while the closed loop response has a very large overshoot. This could be an issue as it has gone far over the required output before settling which could be very dangerous depending on the situation. However, the rising time is much faster in the closed loop system which could be beneficial compared to the open loop system as it is more than twice as fast to react. We see the same pattern in the impulse responses with the open loop system being slower to react but faster to settle as there is little overshoot. The closed loop impulse response dives deep into negative values which could again be dangerous given the circumstance.

3 PID Controller Design

3.1 Finding Controller Parameters

In order to build a PID controller, there are 3 variables we must know the values of. These are the proportional gain K_p (P), the Integral time T_i (I) and the derivative time T_d (D). These are according to Ziegler-Nichols rules for tuning PID controllers. I calculated these values by applying these rules and measuring the response from our closed loop system with a gain function K in place of the controller. This allowed me to calculate the critical gain K_{cr} and the critical period P_{cr}.

The value of K_{cr} is the value of our gain function K when the output of the scope gives an oscillating waveform. When this happens, the poles of the transform function are on the real axis. I changed the value of K through MATLAB starting at K=1. I then increased the value of K until the scope was producing an oscillating waveform. Through trial and error, I produced the most accurate oscillating waveform I could through observation. A value of K = 1.83.

Through trial and error, K_{cr} was found. P_{cr} was then easy to find as it is just the period of the waveform (distance between two peaks). The critical period was found to be 2.369. Once the critical gain and period were found, we could calculate the parameters of the PID controllers.

There are two forms to PIDs. The first form can be calculated using the critical gain and period values that were just calculated. The second can be calculated from the first form.

$$Form\ 1\ = K_p(1 + \frac{1}{T_i s} + T_d s)\ =\ K_p + \frac{K_p}{T_i s} + K_p T_d s$$

$$Form\ 2\ = P + I\frac{1}{s} + Ds\ =\ P + I\frac{1}{s} + D\frac{1}{\frac{1}{N} + \frac{1}{s}}$$

In order to calculate the parameters used for form 1 of the PID, a conversion table was used. This table is part of the Ziegler-Nichols tuning method and it tells us about the relationship between the critical values and the components of the PID controllers. To find the form 1 values, the calculation for each parameter was performed as shown in figure 13.

Type of controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Figure 10: Ziegler-Nichols PID Tuning Table

To make the calculations easier to observe, I reproduced the Ziegler-Nichols table but replaced the formulas with values I calculated by using that table. The K_p values are different for each type of controller as different P, I and D values are used. For example, the P controller uses just the P parameter whereas the PI controller uses both P and I parameters. However, the P parameters for each controller are different as they are calculated with a different formula based on the table.

Type of controller	K_p	T_i	T_d
P	0.915	∞	0
PI	0.8235	1.9742	0
PID	1.0980	1.1845	0.2961

Figure 11: PID Form 1 Conversion Table

I then made another table, this time showing the values to be used for form 2. The values placed in this table were my final PID values and could be placed in the PID function used in Simulink. I obtained these values from the following formulas that convert form 1 parameters into form 2 parameters.

Type of controller	K_p	T_i	T_d
P	0.915	∞	0
PI	0.8235	0.4171	0
PID	1.0980	0.9270	0.3251

Figure 12: PID Form 2 Conversion Table

$$P = K_p \qquad I = \frac{K_p}{T_i} \qquad D = K_p T_d$$

3.2 Testing P, PI and PID Controller Performance

Once I found my PID parameters, I could simulate the closed loop system with a controller to observe how 3 different controllers could affect the response of the system. The block diagram in figure 16 was reproduced for P and PID controllers. The desired output is r(t) so the controller must make the system reach that value. Before testing, we expected that the P controller would have a large steady state error and has a low amount of damping, the PI controller would have a similar damping pattern but would not have the steady state error like the P controller, the PID should have more damping so less variance as well as the steady state error being removed.

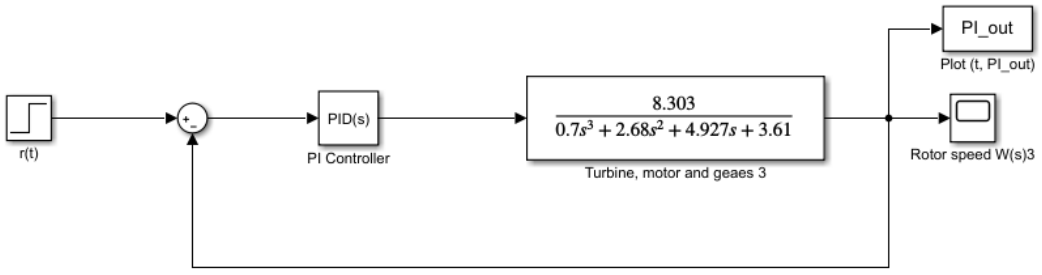


Figure 13: Block Diagram for System with PI Controller

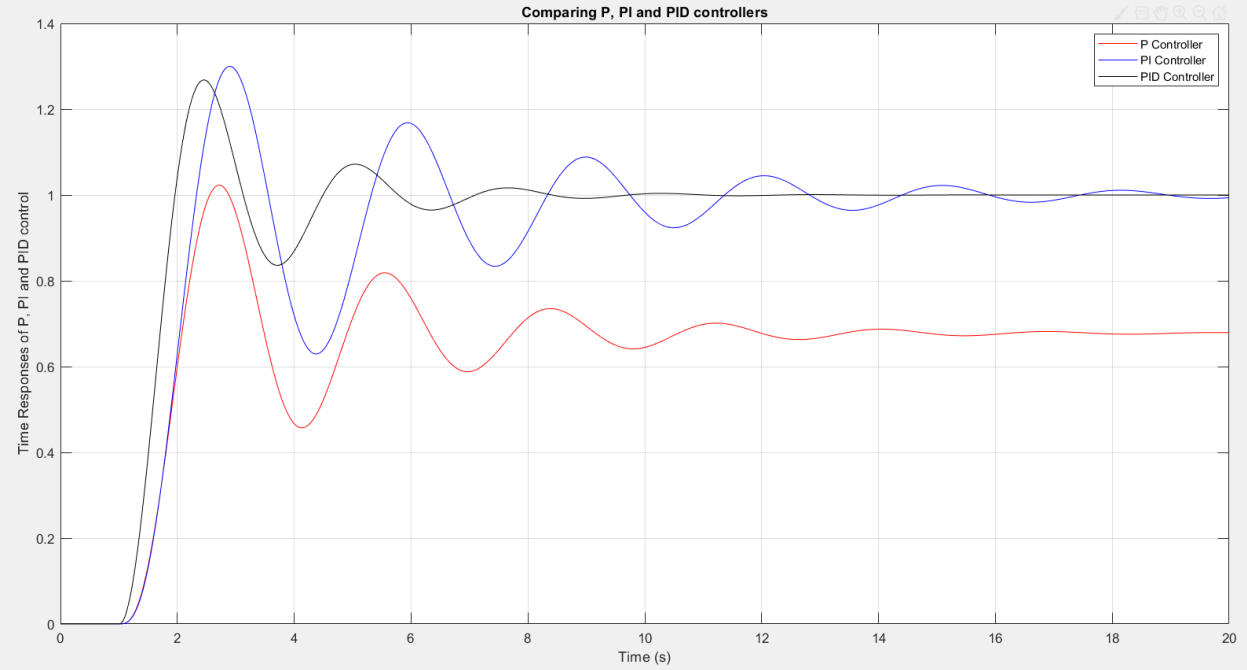


Figure 14: Graph Showing Outputs of P, PI and PID controllers

Here we can see that the hypothesis was correct, the controllers responded as predicted. The P controller had a large steady state error and little damping while the PI controller removed the steady state error but still had little damping and the PID controller increased the damping, so it reached its settling time faster. This is because the controllers have different components and the proportional, integral and derivative components make the system behave differently, the PID has all three therefore is the most accurate, while the P controller has just the proportional component and is less accurate. When assessing the performance of all three, the PID controller reacted the fastest, which could be beneficial as previously discussed. While there was an overshoot for the PID controller, it was the smallest out of the three and while not being safe in some situations, it should be safer than both P and PI controllers. The P controller does not reach the required output, therefore is not viable in most situations.

4 Digital Controller Design

4.1 Converting Analogue to Digital

The next task is to create a digital PID controller, rather than analogue PID controllers previously designed. In order to convert the controller from continuous to discrete form, it had to be transformed from the s-domain to the z-domain. This could be done by using the c2d (continuous to discrete) command on MATLAB to convert the $G_c(s)$ value into the $G_c(z)$ value. Part of the formula that is needed to make this conversion is the sampling time T. This is the rate at which a piece of data is recorded and plotted on the graph. For the first part, I used T = 0.05.

$$Digital\ PID\ Controller\ =\ G_c(z)\ =\ \frac{33.61z^2\ -\ 66.09z+\ 32.52}{z^2\ -1.007z\ +\ 0.006738}$$

$$Digital\ PI\ Controller\ =\ G_c(z)\ =\ \frac{0.8235z^2\ -\ 0.8082z+\ 0.005408}{z^2\ -1.007z\ +\ 0.006738}$$

4.2 Digital Controller Performance

Once I found my transfer functions for the z domain, I could enter these to the existing block diagrams by using 'zero order holds'. These blocks are used whenever there is a change from continuous to discrete or discrete to continuous.

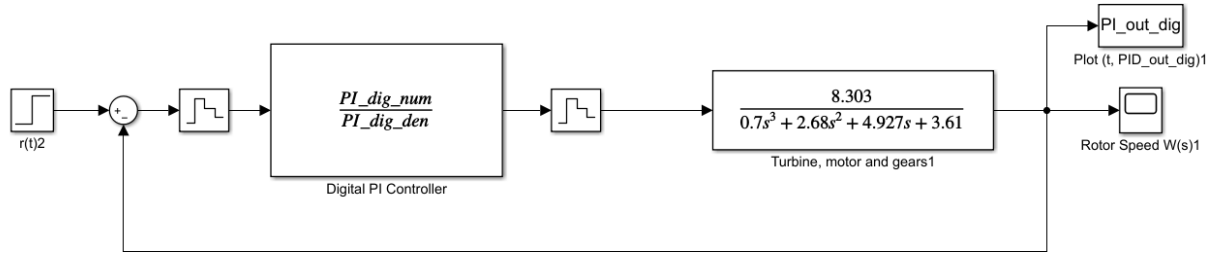


Figure 15: Block Diagram to Test Digital PI Controller Step Response

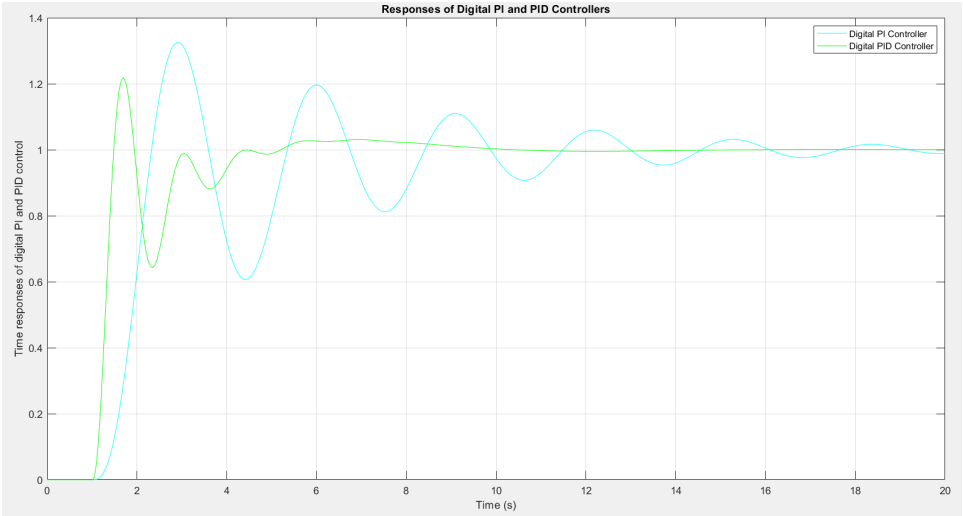


Figure 16: Graph Showing Digital PI and PID Step Responses

These two digital controllers reacted very similarly to the analogue PI and PID controllers with the PI controller having the same steady state error as the PID, but with little damping, hence it takes longer to reach its steady state value. The PID controller reacts faster with a smaller overshoot, therefore it should be the better option in most scenarios.

4.3 Effects of Sampling Time

The sampling time T that was previously mentioned is extremely important to a digital PID controller as it is the time difference between two consecutive samples in a graph, meaning the smaller the value of T, the more frequent a sample is taken, hence the more accurate it is. To observe this, I used a calculation that finds a range of values to be used for the sampling time: $T = t_r / n$ where $10 < n < 20$. These values were plotted on a graph as seen in figure 22.

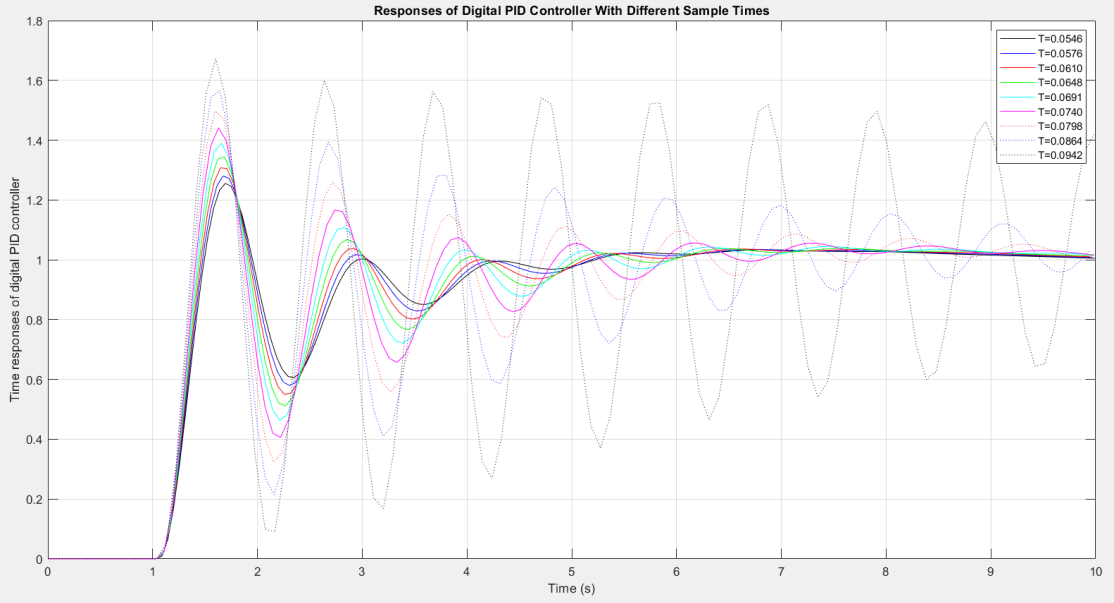


Figure 17: Effects of Sampling Times on PID controller

Figure 20 shows the importance of getting the sampling time correct as it greatly impacts the output generated. You can see that the smaller the value of T, the smaller the overshoot is and the faster the system reaches a steady state. This is because the smaller the value of T, the more accurate the graph is as it takes readings more often. A large value of T takes samples further apart, therefore only tells part of the story. The sample times at this range have a large impact on the digital PID controller but not as much impact on the PI controller as there is very little variance between these sample times on the digital PI controller system.

5 Lead and Lag Compensator Design

5.1 Bode Plots and Stability Margins

In order to design lead or lag compensators, the frequency response of the system should be analysed by using Bode plots. Bode plots consist of a magnitude plot, expressed in decibels (dB) or a phase plot, expressed in degrees, along with a log frequency scale. The Bode plot of the open loop system with no controller can be seen below in figure 18.

The frequency response can show a lot of information about the characteristics of a system, in particular the system's stability margins. These stability margins include the gain margin and phase margin and they

are defined as the amount of change in open-loop gain/phase needed to make a closed-loop system unstable. [1]

As seen below in figure 18, the gain margin of the system without a controller or compensator is at 5.28 dB, while the phase margin is at 30.8 deg. These margins are not high enough for an ideal system and the system is therefore unstable. Usually, engineers would aim for a 6 – 8 dB gain margin and a 45 – 65 deg phase margin. This is what we use a compensator for. The lead and lag compensators improve the stability margins in the system in order to change factors like response time, steady state error or overshoot.

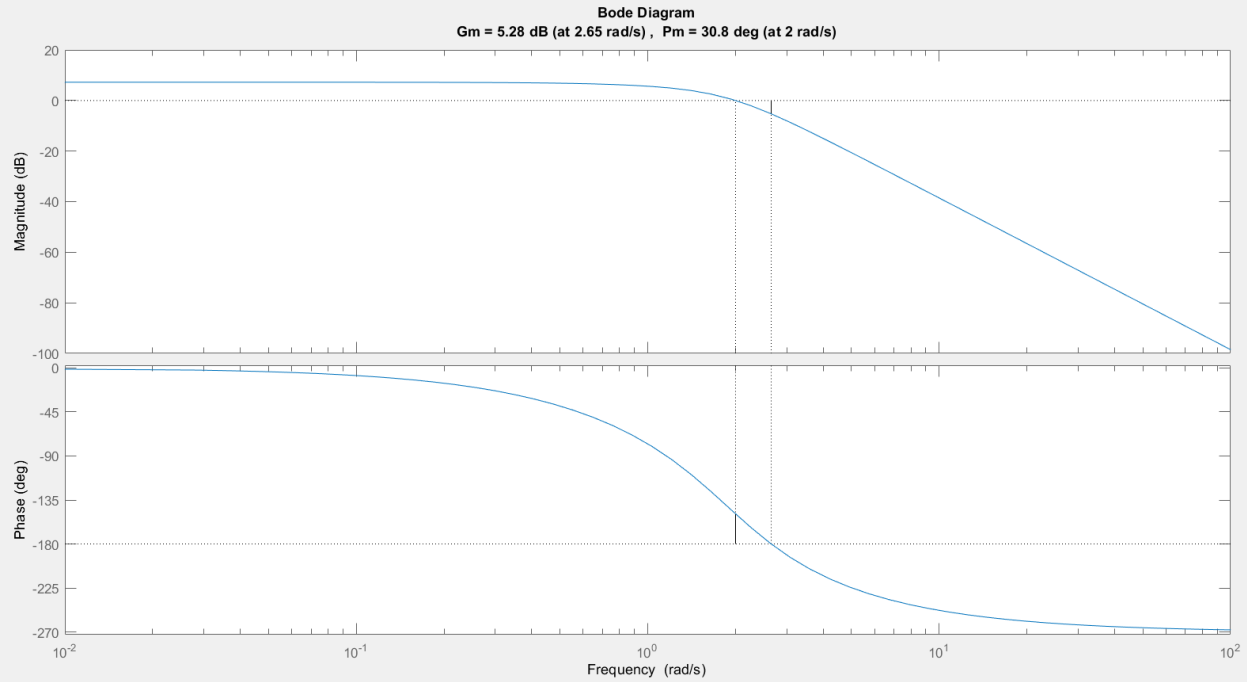


Figure 18: Bode plot showing frequency response and stability margins of the open loop system

5.2 Lead Compensator Design

The purpose of the lead compensator is to improve the transient response, hence make the response faster while also stabilizing the system. However, this is usually at a cost with a sacrifice in steady state error.

The design of a lead compensator involves finding the correct parameters for the compensator to shift the stability margins to the desired level. These parameters are seen in the equation below and changing these parameters changes the stability of the system. Firstly, the gain (K_c), should be found. Figure 22 shows how different values of K_c effect the time response of the system.

$$G_c(s) = K_c \frac{\alpha(Ts+1)}{\alpha Ts+1}$$

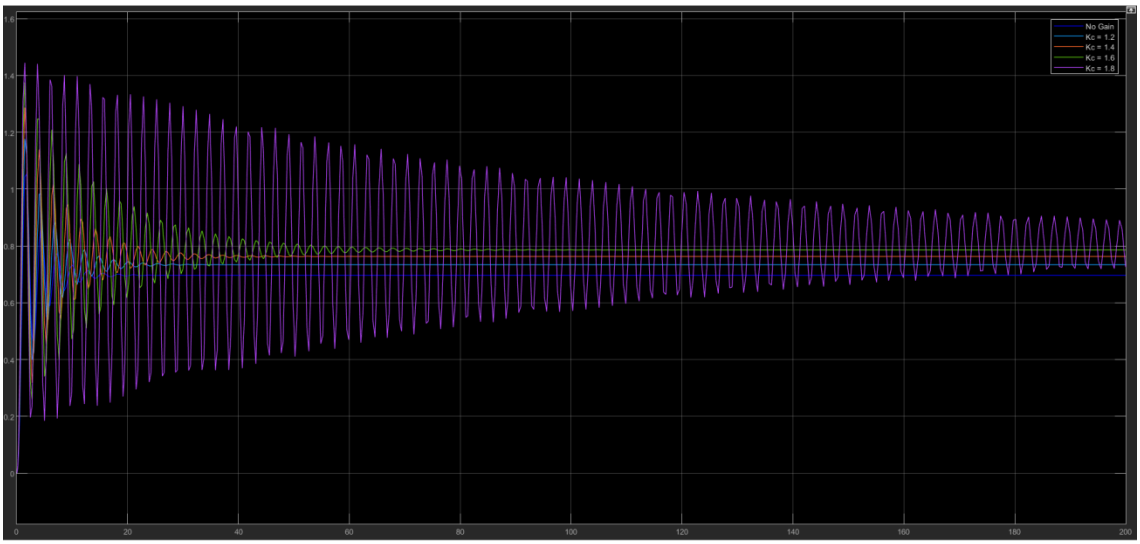


Figure 19: Graph showing time responses for systems with different K_c values.

As observed from figure 19, K_c has a great impact on the stability of the system. For the compensator to work best, a high K_c value is required, but not one so high that it makes the system unstable. For this system, 1.5 is an ideal value for Kc as it improves the steady state error by decreasing the gap between the input value and the value when the system reaches a steady state. It does this while not letting the settling time be too large with the system settling within 60 seconds, compared to around 20 without the gain added. If Kc > 1.5, the settling time is over 100 seconds which is far too large for this type of system. This can be seen more clearly in figure 20 below.

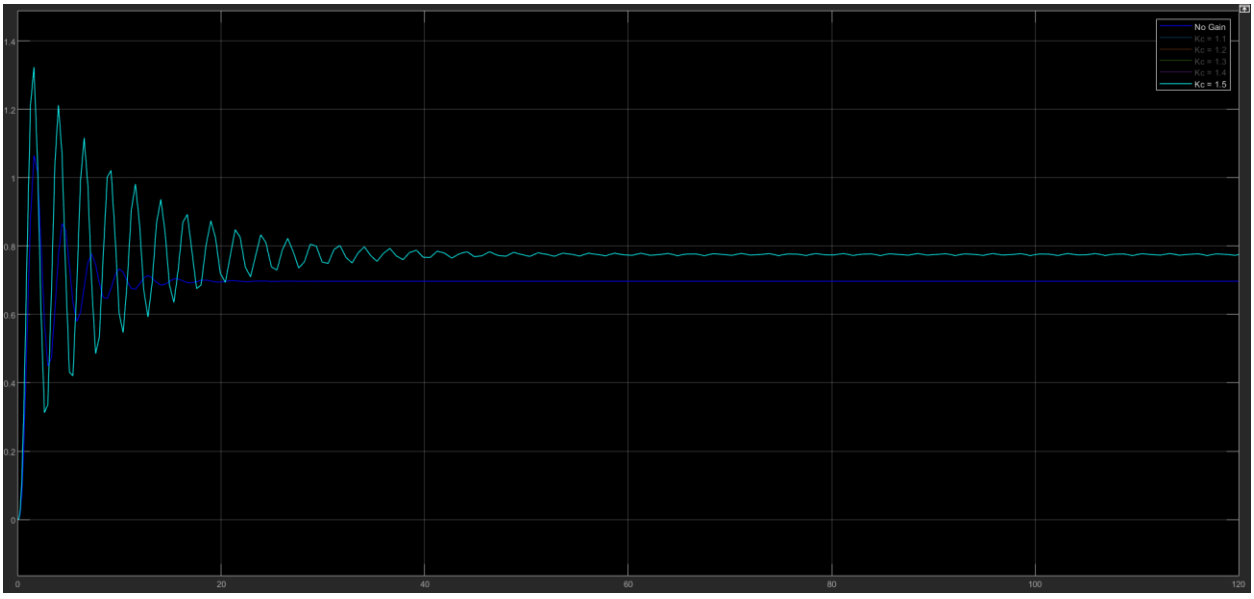


Figure 20: Time response of system without gain and with gain of K_c = 1.5

Now that the gain parameter K_c has been found, the others can be found, α and T. We know that 0.05 < α < 1, but we can find a more specific set of values from the equation:

$$sin\varphi_m = \frac{1 - \alpha}{1 + \alpha}$$

In order to get the phase margin in the correct stability region α should be between 0.05 and 0.17.

In these ranges there are four sets of values for the parameters to ensure the system is within the stability margin. These are: (T=14.5, α =0.05), (T=9, α =0.08), (T=7, α =0.1), (T=4, α =0.16).

With these values, we have designed lead compensators that meet the required stability margins of 8dB as the desired gain margin and a minimum of 45 deg for the minimum phase margin. Multiple combinations of a and T give the required stability margins and while their Bode plots are similar, the time response graphs are very different as can be seen in figure 22.

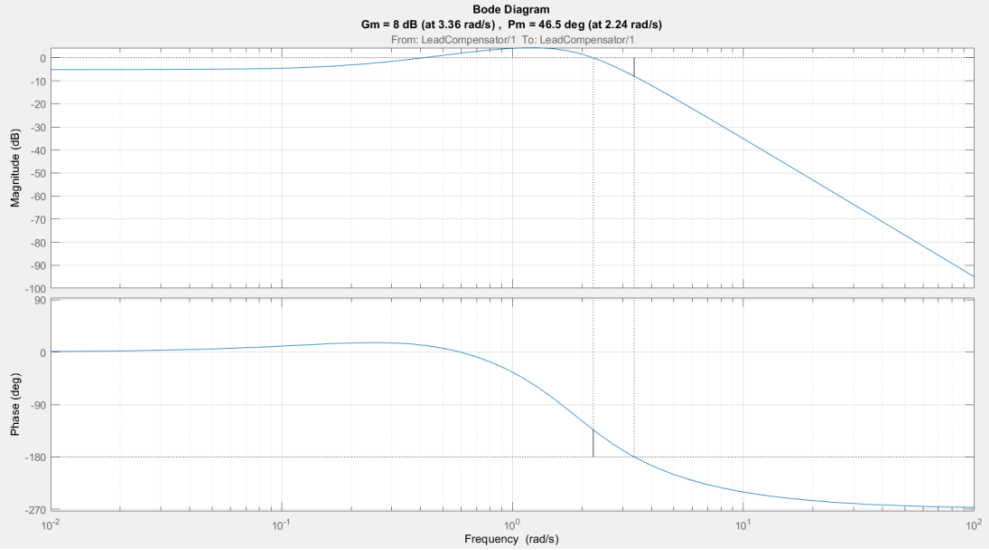


Figure 21: Bode plot of system with lead compensator (T=4, α =0.16)

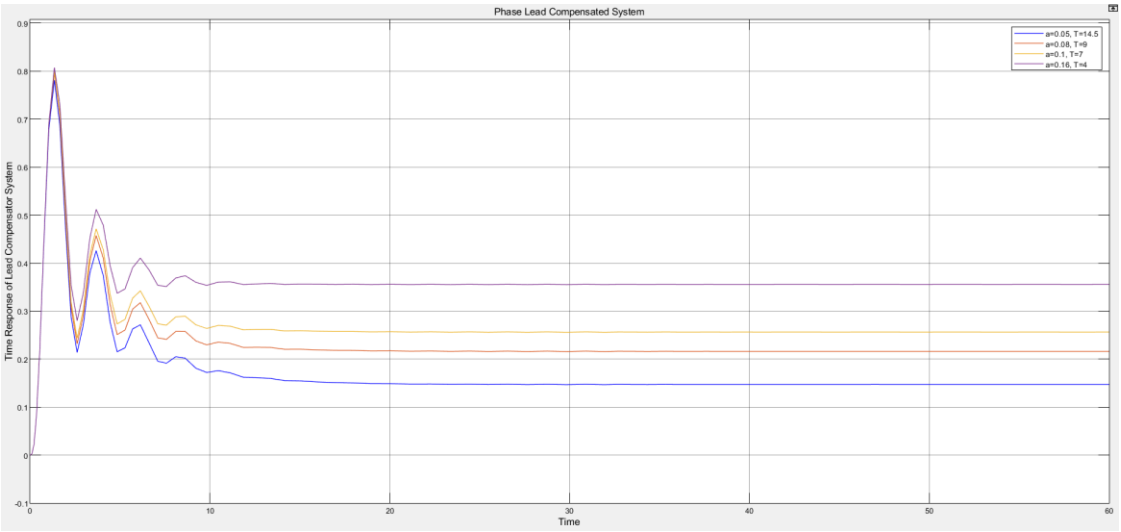


Figure 22: Time response graph showing compensated systems with different α and T values

It can be observed that different values for the compensator parameters can have a large effect on the time response as the higher the value of α and lower the value of T, the more accurate the steady state error is. Therefore, the compensator design with the best steady state error should be chosen (T=4, α =0.16). The bode plot and the time response of the compensated and uncompensated system can be seen in figures 23 and 25 respectively.

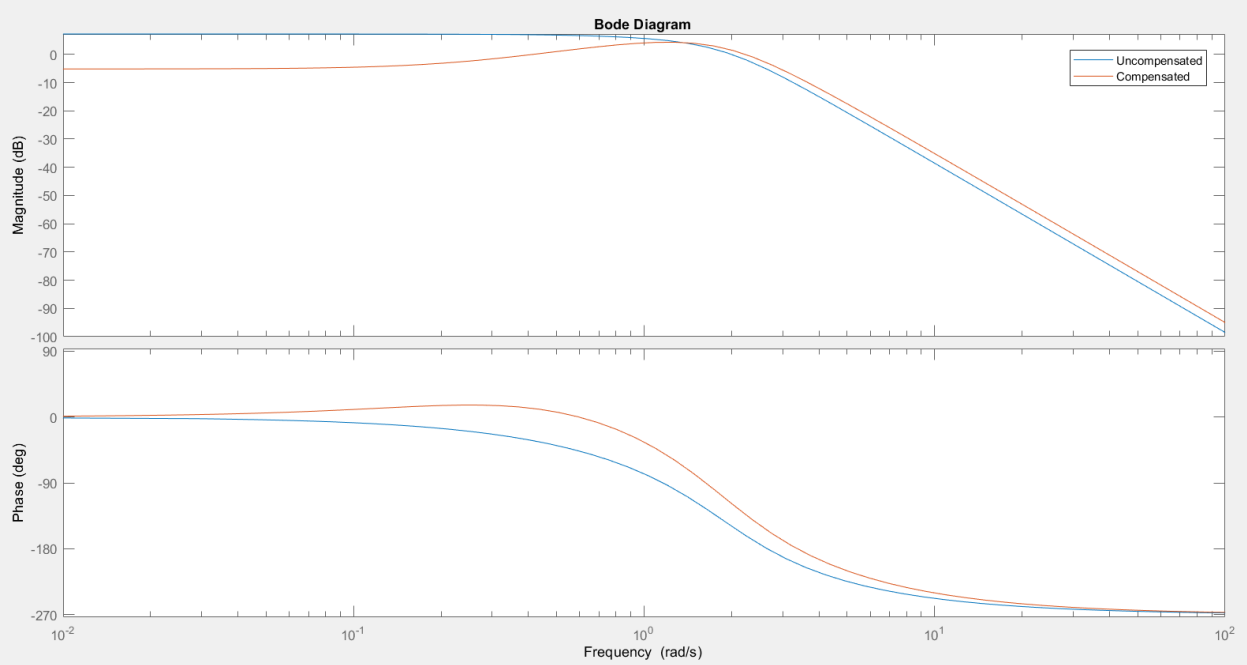


Figure 23: Bode plot of compensated and uncompensated system

To test the time response of the compensated system, a block diagram was made including the gain, compensator and plant.

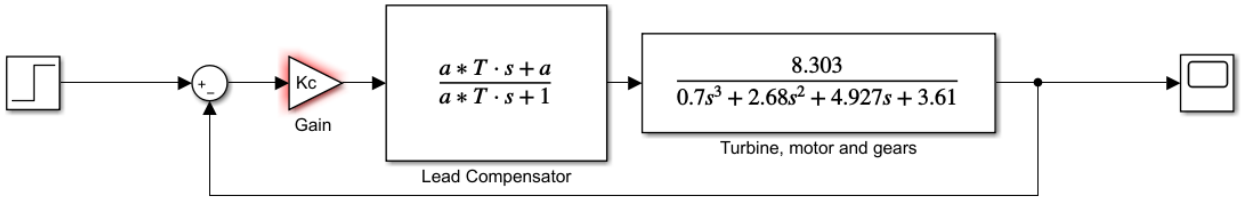


Figure 24: Block Diagram of Lead Compensated System

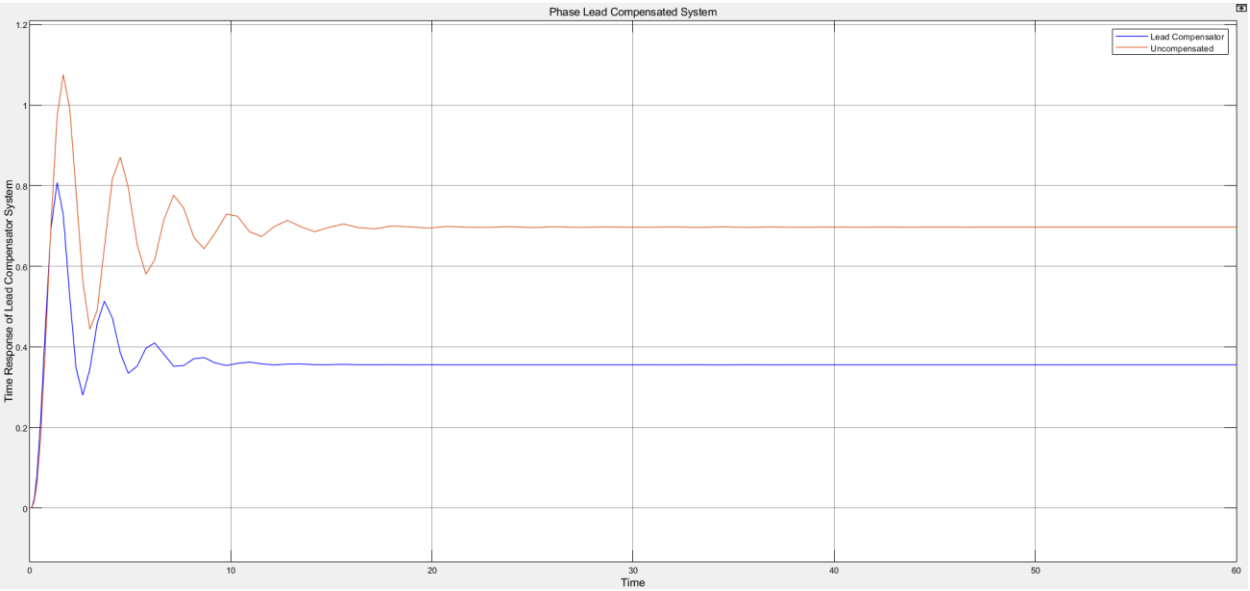


Figure 25: Time response of compensated and uncompensated systems

From figure 25, we can see that the lead compensator performed as expected with the improved settling time and faster response than that of the uncompensated system. However, the steady state error is very high with the final value at 0.37 rather than 1. This steady state error would be even higher if other parameter values were used as observed in figure 22.

5.3 Lag Compensator Design

Design of a lag compensator is similar to that of a lead compensator, except that rather than increasing gain magnitude, you reduce it. This is done by swapping the pole and zero so that the pole comes before the zero when you plot the system's roots in the s domain. This is opposed to the zero coming before the pole in the lead compensator design. This is where the lead and lag compensators get their names. The idea behind the lag compensator is to shift the phase magnitudes to the left which will in turn increase the gain margin. The Lag compensator should increase the stability margins of the system as well as improve the steady state error. The lag compensator may be slower than other compensators and controllers.

Rather than finding the gain (K_c) first, the gain should be found last in this design, the dynamic part of the compensator should be designed first using the equations:

$$G_{\text{lag}}(s) = K_c \frac{\text{b}(Ts+1)}{\text{b}Ts+1}$$

Where $1 < \text{b} < 12$

Due to the large value of b when compared to α, the frequency response of the lag compensator is much different to that of the lead compensator.

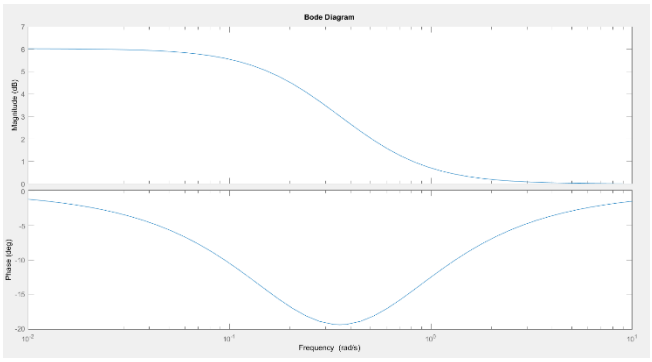


Figure 26: Bode plot of lag compensator

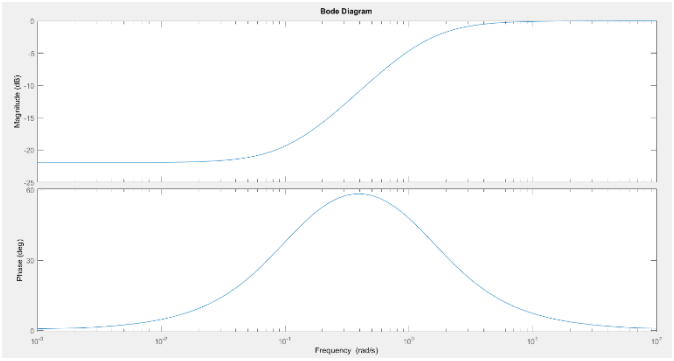


Figure 27: Bode plot of lead compensator

The final Bode plot that we get from the compensated system is the addition of the bode plot of the compensator and the bode plot of the uncompensated system. The two figures above show how different each compensator is designed.

By changing the parameters for the compensator, the relationship between those parameters and the frequency response can be observed. Figure 28 shows a constant T value but b increases from 2 to 11. It is clear why the value of b should be within those parameters as the gain and phase margins start to stagnate at b > 10, which can be seen from the bode plot in figure 28, and the graphs in figures 33 and 34. This is also the case for T values where gain and phase margin values start to stagnate when T > 20, as seen in figures 29, 35 and 36.

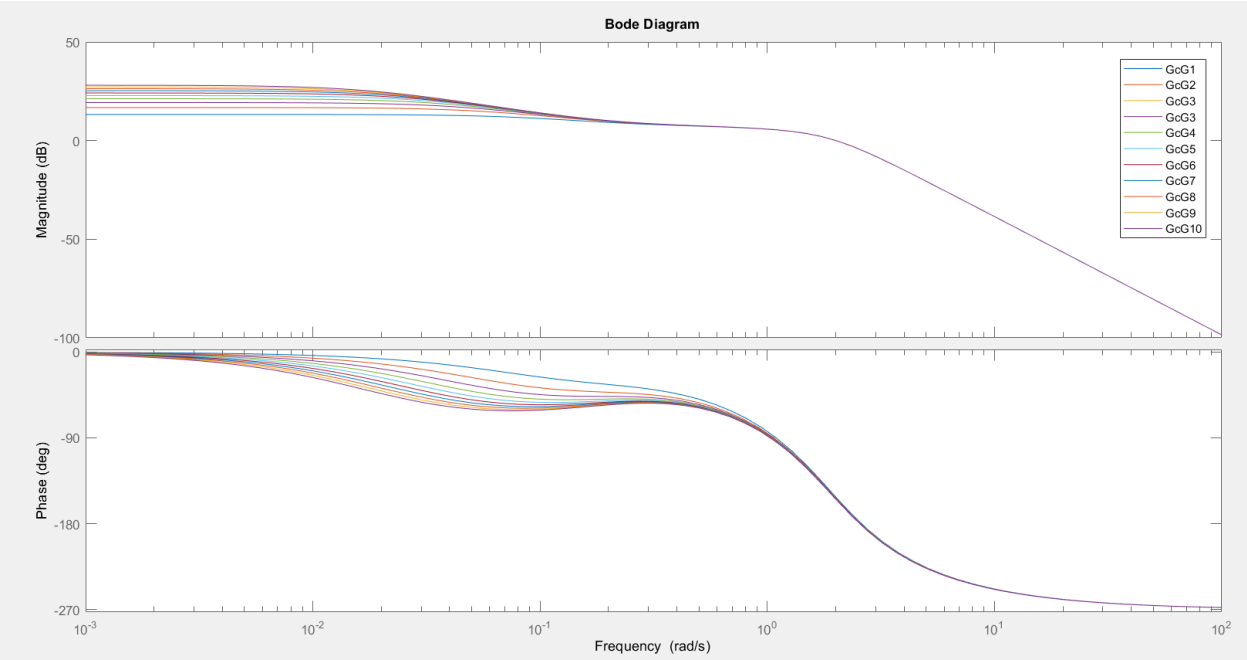


Figure 28: Bode plot showing compensators with T=5 but b moving from 2 (GcG1) to 11 (GcG10)

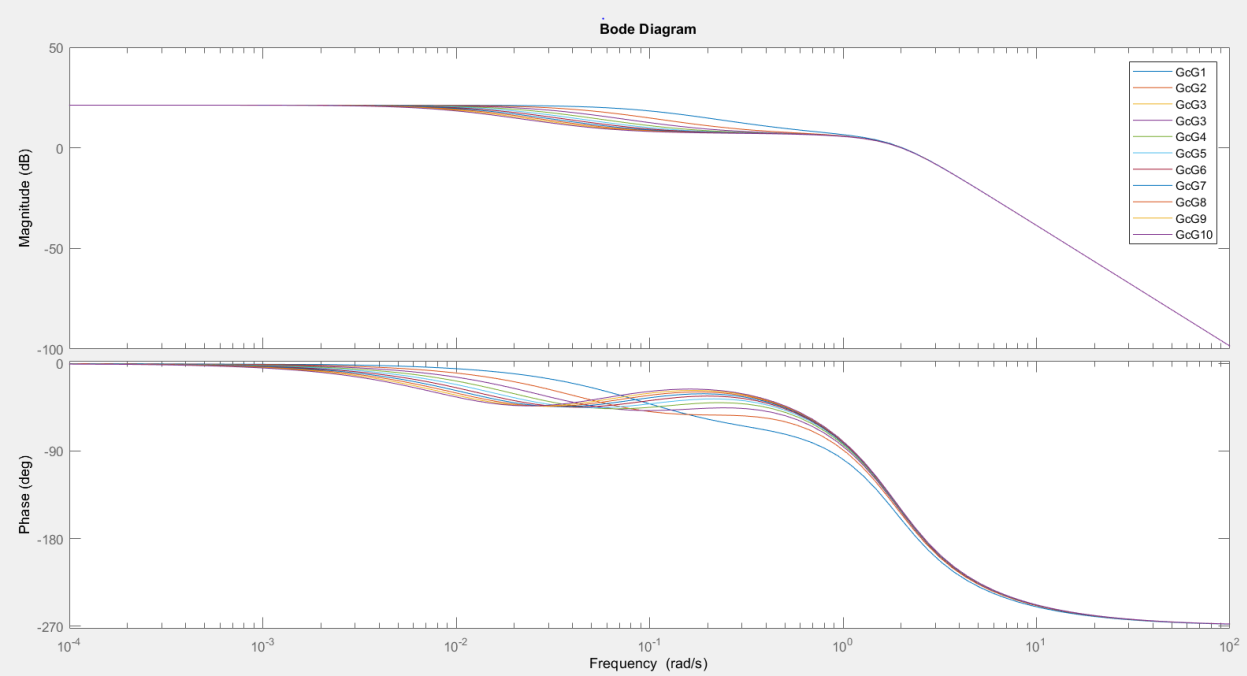


Figure 29: Bode plot showing compensators with b=5 but T moving from 2 (GcG1) to 20 (GcG10)

Value of b	Gain Margin (dB)	Phase Margin (deg)
2	4.79	27.7
3	4.63	26.7
4	4.55	26.2
5	4.5	25.9
6	4.47	25.7
7	4.45	25.6
8	4.43	25.5
9	4.41	25.4
10	4.4	25.3
11	4.39	25.3

Figure 31: Table showing stability margins against b

Value of T	Gain Margin (dB)	Phase Margin (deg)
2	3.23	17.9
4	4.3	24.6
6	4.63	26.7
8	4.8	27.8
10	4.9	28.4
12	4.96	28.8
14	5.01	29.1
16	5.04	29.3
18	5.07	29.5
20	5.09	29.6

Figure 32: Table showing stability margins against T

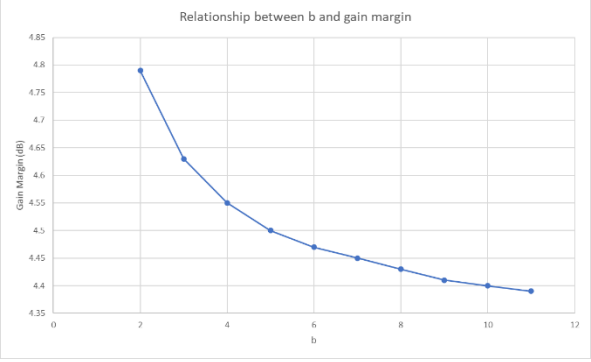


Figure 33: b vs gain margin

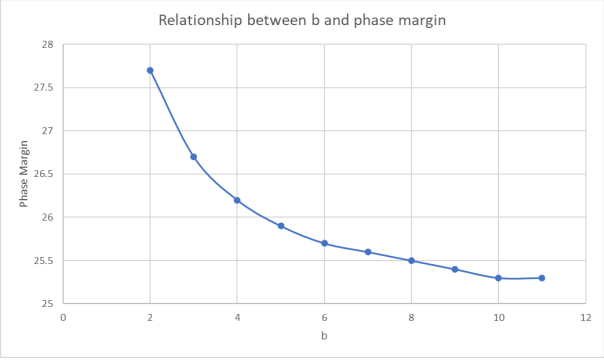


Figure 34: b vs phase margin

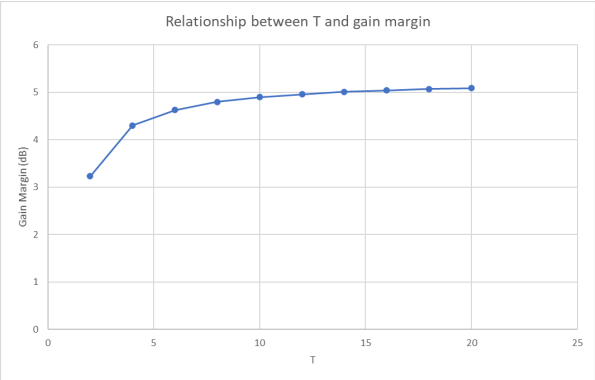


Figure 35: T vs gain margin

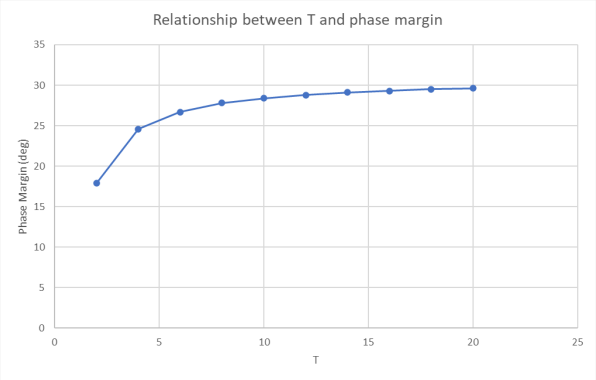


Figure 36: T vs phase margin

From figures 28 to 36, the stability margins are clearly only affected when $1 < b < 12$ and $1 < T < 20$. Since the gain and phase margins are too small however, a gain must be used to increase them. Using $K_c = 0.7$, $b = 5$, $T = 10$, the following Bode plot was obtained for a lag compensated system.

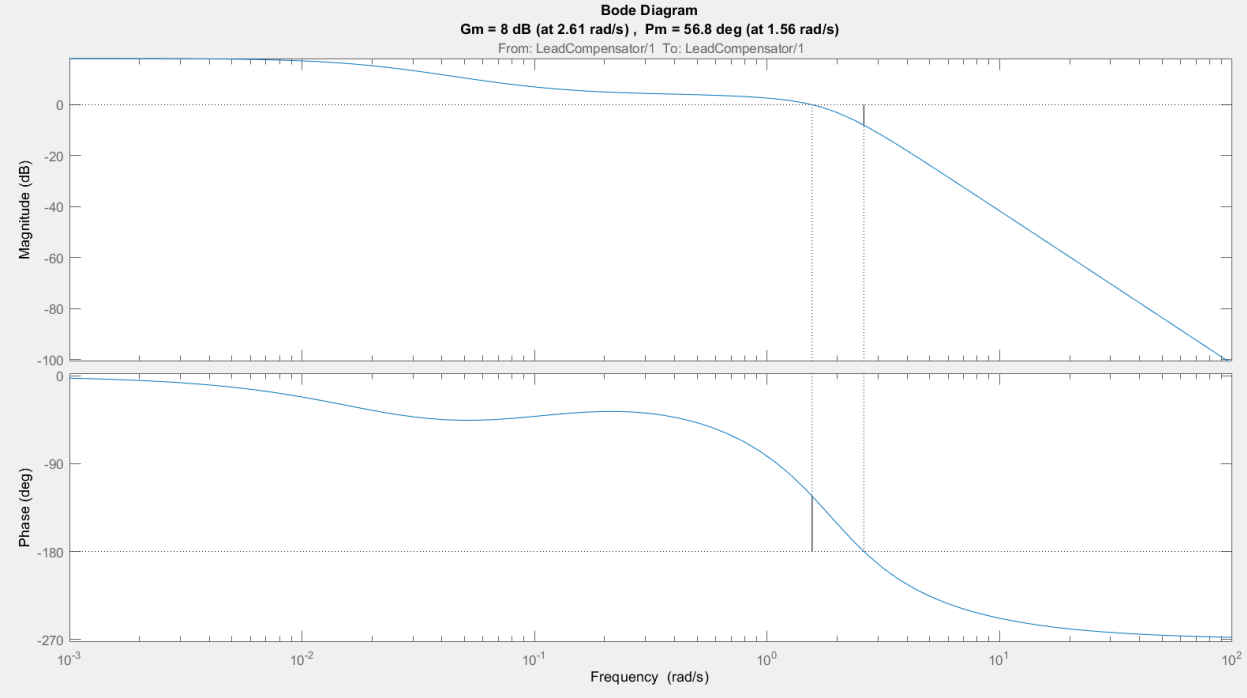


Figure 37: Bode plot showing stability margins of lag compensated system.

To test the time response of the compensated system, a block diagram was made including the gain, compensator, and plant.

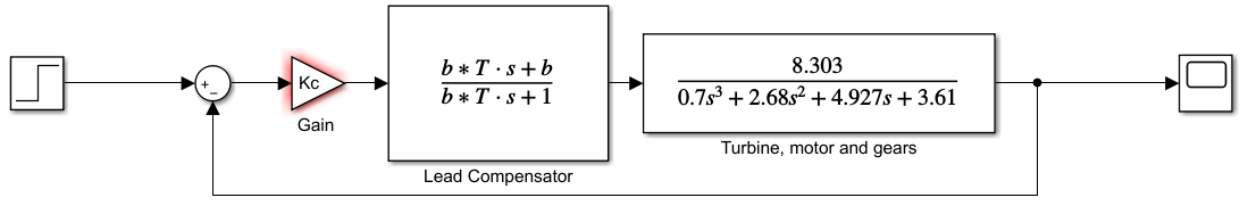


Figure 38: Block diagram for lag compensated system

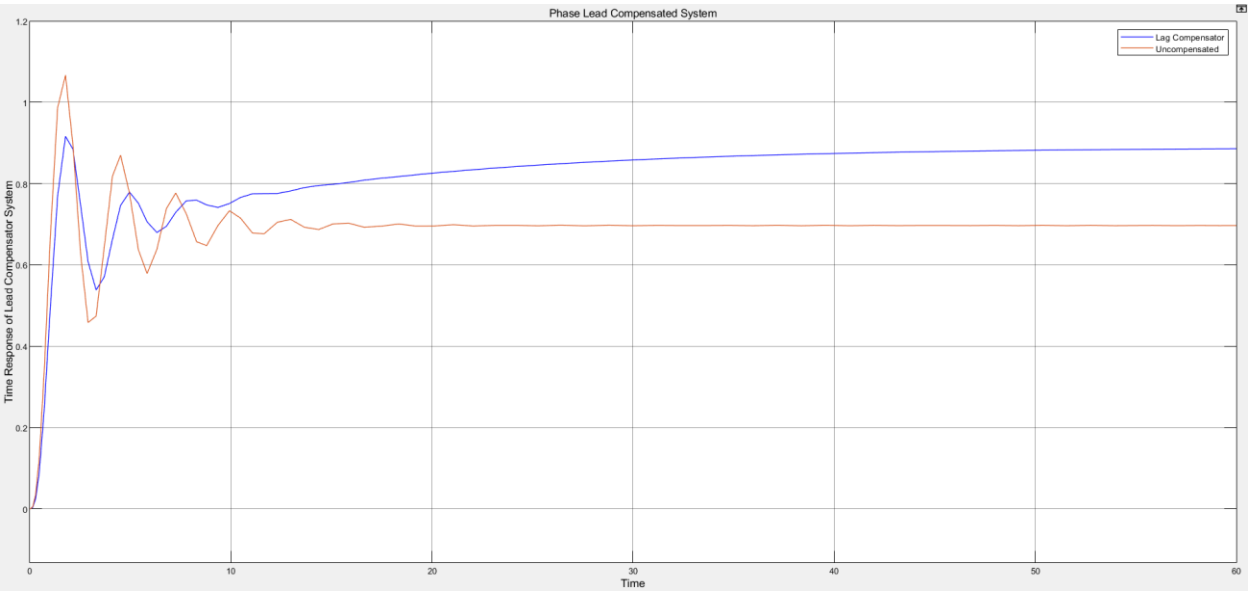


Figure 39: Time Response of lag compensated and uncompensated system

Figure 39 shows that the lag compensator also behaves as predicted with the steady state error improving, the final value is closer to 1 than that of the uncompensated system. The system is now more stable as there is less overshoot and the settling time is faster. Despite the lag compensator having a slower response than the lead compensator, the lag compensator still has a faster response than the uncompensated system.

6 State Space Design

6.1 State Variables and Pole Placements

State space models are models that use state variables to describe a system by one single set of differential equations rather than by one or more n^{th} order differential equation. These state variables are not measured during a design, rather they are constructed from the data used in the system. [2] This single set of differential equations is represented as a set of matrices in the form $\dot{x} = Ax + Bu$ and $y = Cx + Du$. Each parameter in these equations represents a matrix. Usually systems give the parameter of D to be zero as there are not many cases of that type of control system, which is also the case for this design.

State Space models are very useful as they make complex systems much easier to solve by designing a new representation for the same transfer function used in a system. This is a modern approach to control engineering and is used widely in industry today, for its fast response times and small levels of steady state error.

To start, the transfer function of the plant must be converted to the form described above. This can be done easily through MATLAB with the function “tf2ss”. This is the conversion from a transfer function to steady state form. A block diagram can be seen in figure 40 showing the new configuration for this system in its state space form, with state variables A, B and C represented as gains.

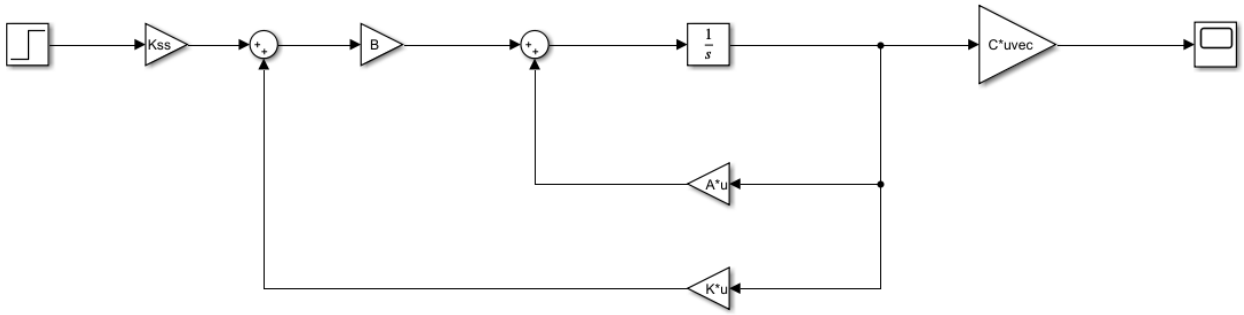


Figure 40: Block diagram for state space design

There is also a K value in this block diagram. This is a parameter to be designed in a way that it effects the placement of the poles of the system, effectively enabling the stability of the system, by moving the poles to the left. Where u is the input to B and r is the desired value, with Kx being the parameter we must design, we know that u = (r – Kx). Therefore, $\dot{x} = (A - Bk)x + Br$ where (A-Bk) replaces the previous parameter A. Here we see that only A, B and K effect the stability of the system, where we can change the value of K to change the roots of the system, hence the stability will change.

A new matrix of values can be used as coordinates for the routes of the system and the MATLAB function will place that matrix into the system along with A and B to change the characteristics by using the "place" function. These new poles should be to the right of the current poles for the uncompensated system as the further from the imaginary axis the routes of a system are, the more stable it should be. However, to observe the relationship between route placement and stability, time responses were plotted. Using different pole placements for different systems allowed for this observation as seen in figures 41 – 43.

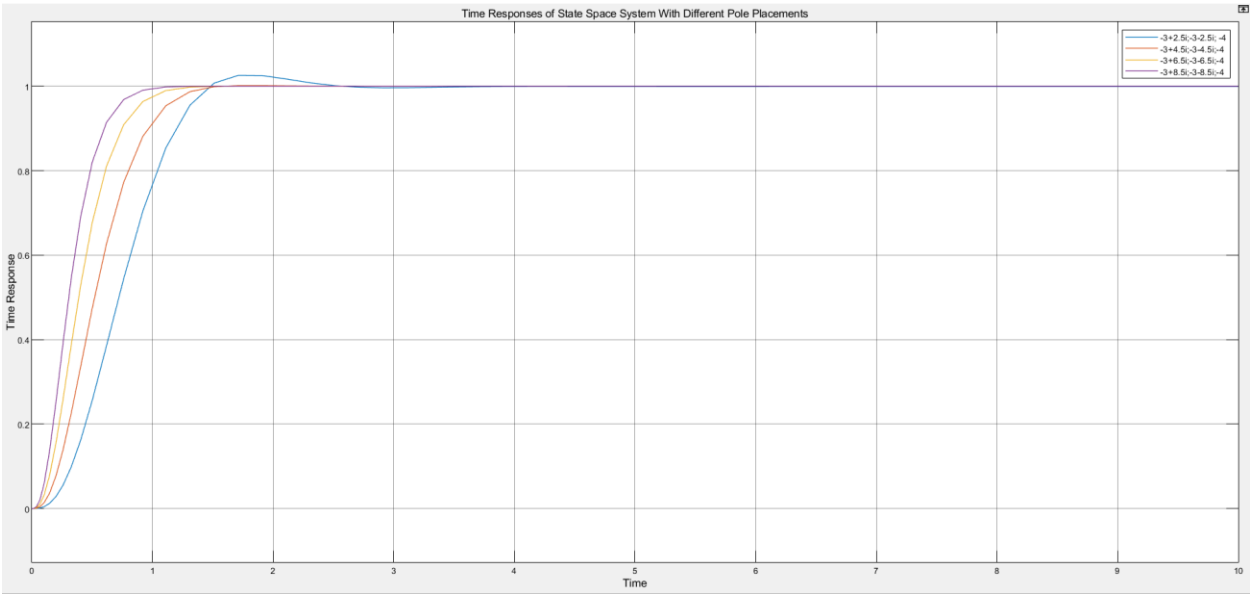


Figure 41: Time responses of state space designs with different poles placed along real axis

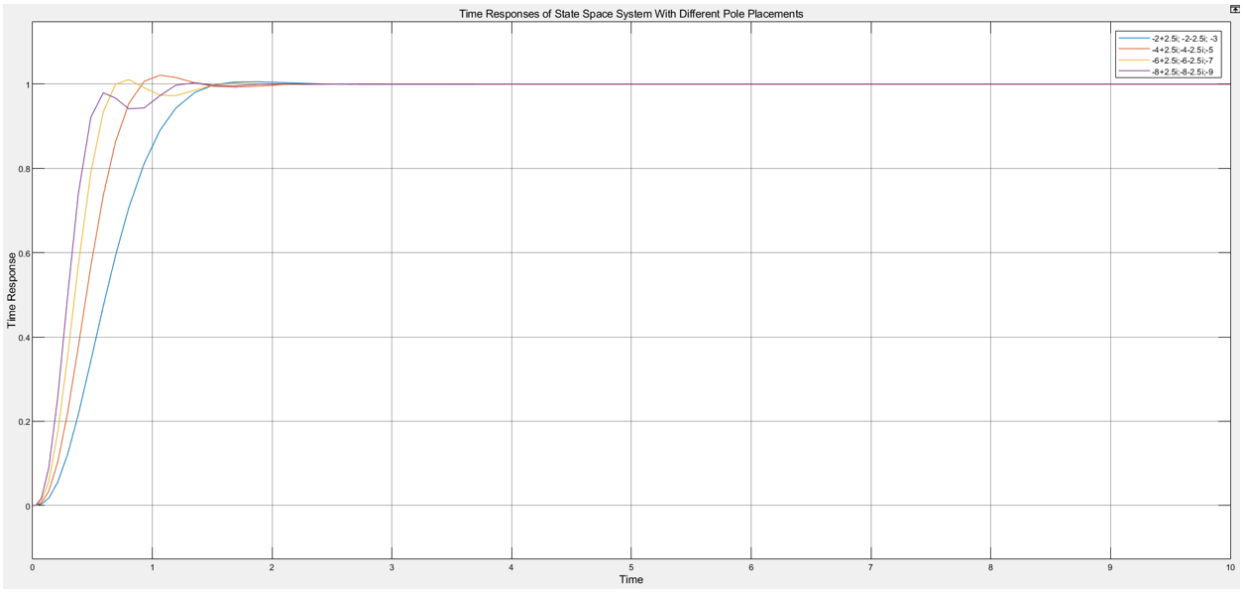


Figure 42: Time responses of state space designs with different poles placed along imaginary axis

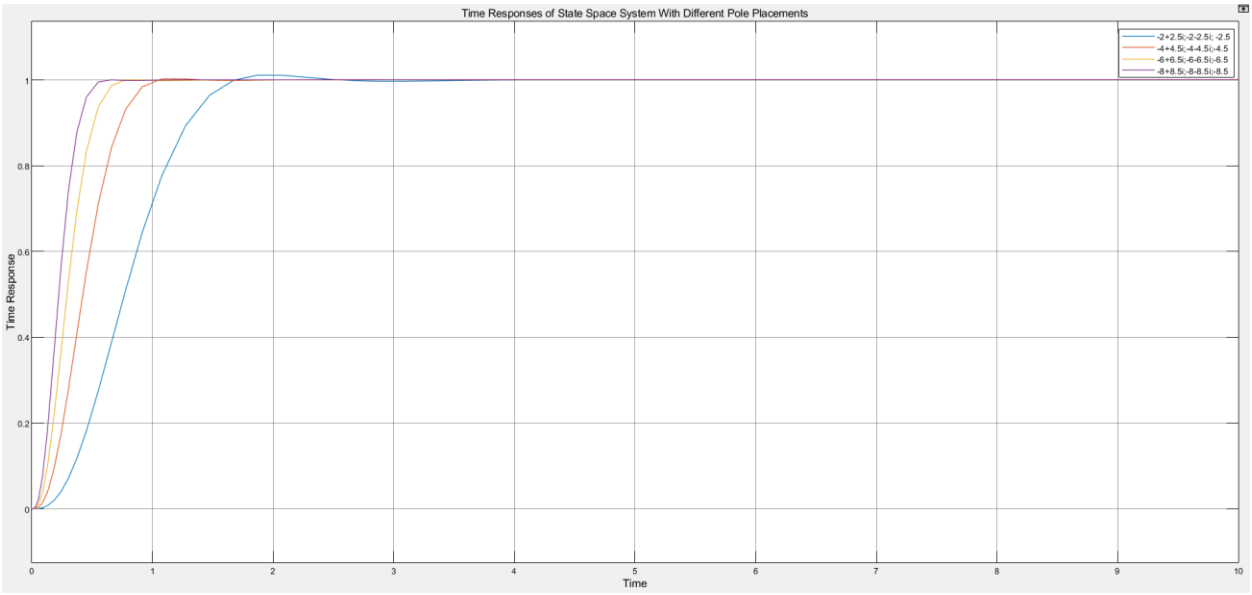


Figure 43: Time responses of state space designs with different poles placed diagonally

It can be seen that the further from both the real and imaginary axis the poles are placed, the faster the response time is. Furthermore, there is no overshoot after poles are placed to a certain point which improves accuracy of the system. However, some gain is needed for each system (K_{ss}) to make the steady state error zero. Some pole placements require more gain than others as it takes more gain to move the poles further from the axis. Therefore, it is important not to place poles too far away as some systems in industry do not have actuators of the required magnitude to provide that gain. Due to the fast response time, low amount of overshoot and low value of gain needed to reduce steady state error, the poles of this system will be placed at [-4+4.5i, -4-4.5i, -4.5+0i].

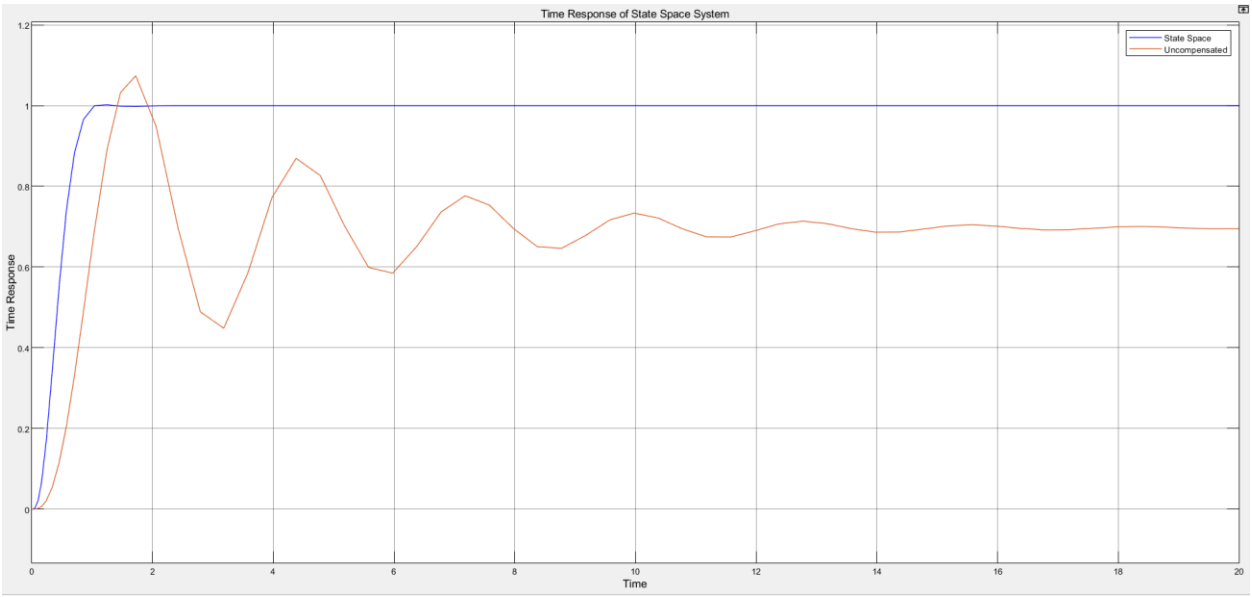


Figure 44: Comparison of state space design against uncompensated system

The time response of this state space system can be observed in figure 44. It can be seen that the system behaved as expected due to the fast response time and minimal steady state error, particularly when compared to the uncompensated time response. The state space design is very accurate and quick so is therefore useful to have in industry, however it requires constant measurement of each state to perform, therefore is very costly to measure this system. One solution to this is an observer feedback design.

6.2 Observer Design

Observer feedback design is the application of a software sensor to the state space system. The observer estimates the value for x(t) based on measurements from u(t) and y(t). This allows for the use of a state space design, without the issues that come with sensors like costs and noise. The system will look as seen in figure 45.

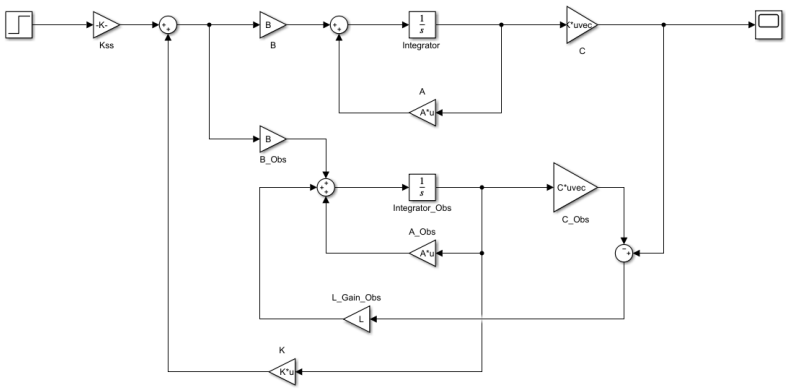


Figure 45: Block diagram showing state space and observer design

The observer uses the same parameters as the state space model, however the matrix L should be found instead of K. This is a matrix that allows for the change of pole placements, but for the observer, the poles should be placed further to the left than in the state space design. This is to give the observer an extremely quick response time to allow it to keep up and keep estimating values for the state space design. The good thing about this is that the pole placement for the observer, does not affect the gain of the system. Figure 46 shows how pole placements for the observer affect the time response of the system. It can be seen that the placement of the poles has little effect on the time response apart from making it slightly faster. Furthermore, in the block diagram, the integrator used for the observer, starts 0.2 seconds after the simulation starts. This is to allow it to estimate the state space model values.

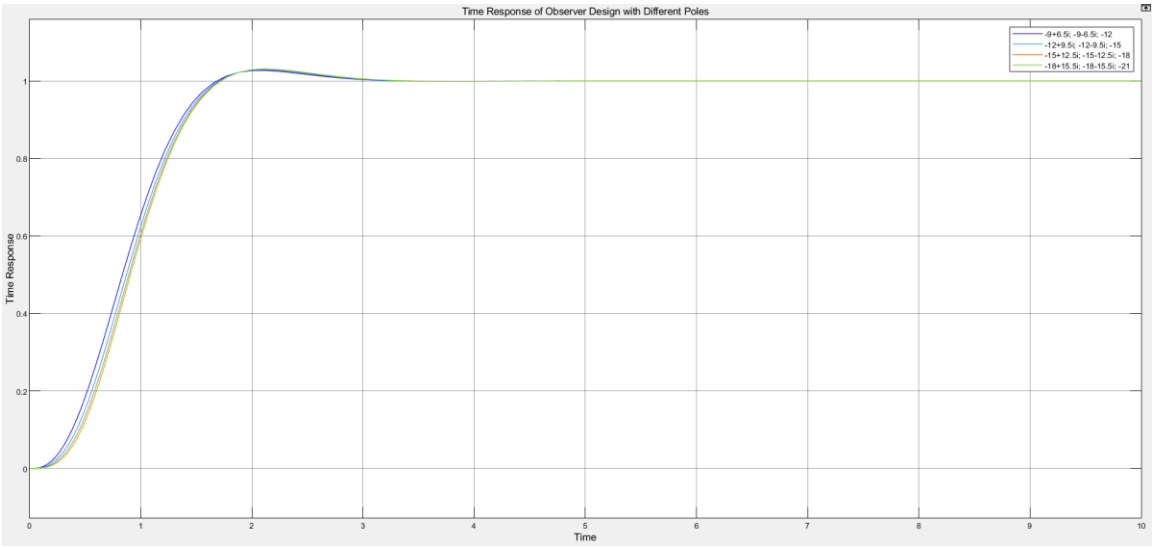


Figure 46: Time responses of observer designs with different poles placed diagonally

Since placing the poles further to the left does not largely affect the time response I will place the poles at [-9+6.5i, -9-6.5i, -12+0i]. This is far enough towards the left to give the observer the fast response time needed to match the state space model.

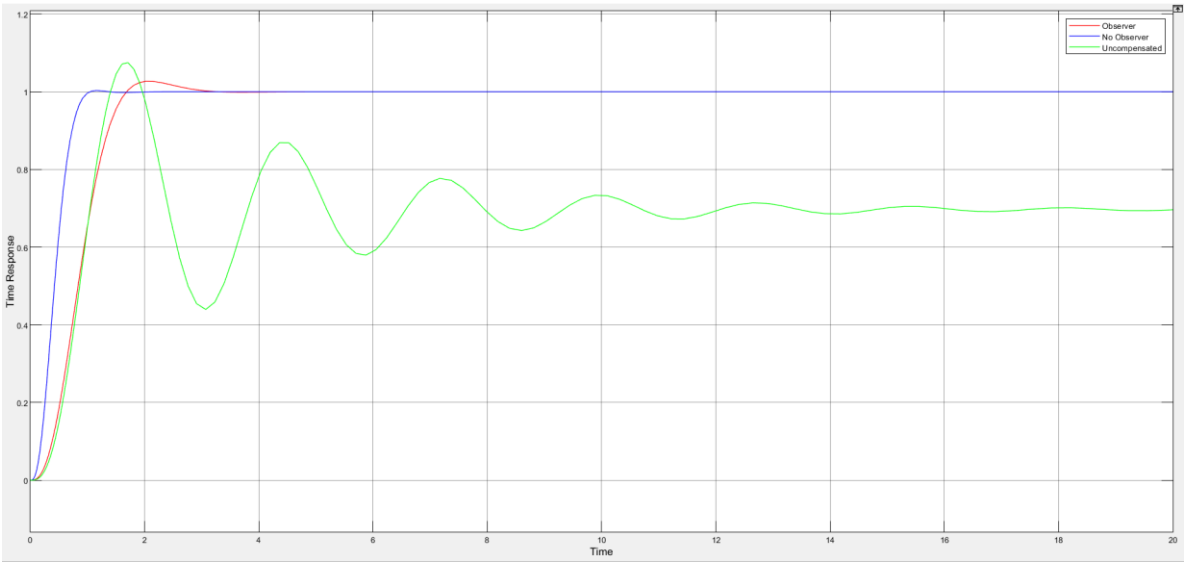


Figure 47: Time response of state space, observer and uncompensated system

Figure 47 shows the time response of the state space system with and without an observer as well as the uncompensated system. The fastest response is from the state space model with it also having little overshoot as previously discussed. However, this system can be costly so depending on the customer, an observer feedback state space system may be beneficial as it has a fast response time and little overshoot without the costs for sensors. The observer is clearly behind the state space model due to the measurements taking place 0.2 seconds after the simulation takes place, from the integrator seen in the block diagram.

7 Further Investigation

7.1 Effects of Noise on Controllers

Using the following block diagram, a system was constructed to investigate the effects of noise on the 4 best controllers or compensators from each type of design.

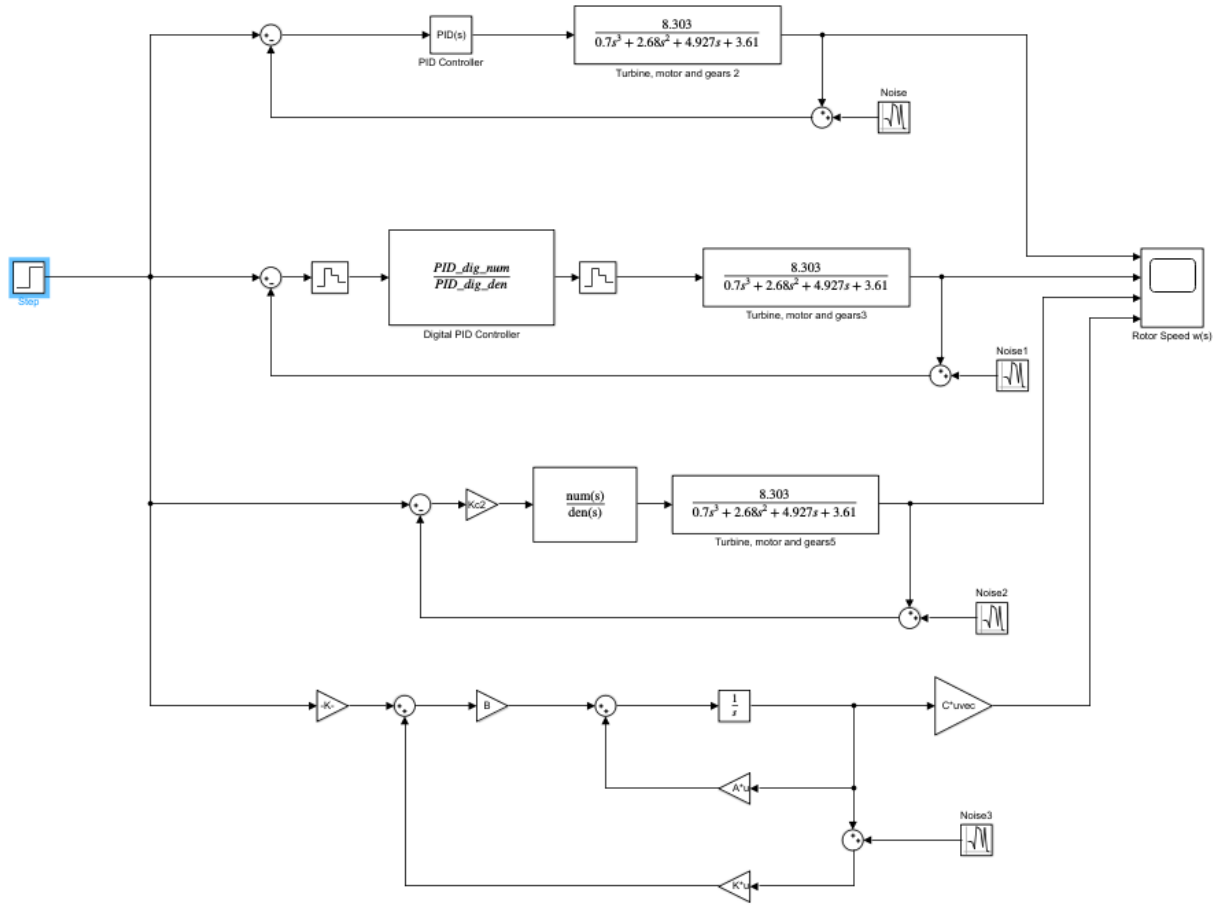


Figure 48: Block diagram testing noise on different controllers

Figure 49 shows the time response generated from this block diagram where we can see how different types of controllers reacted differently to noise put into the system.

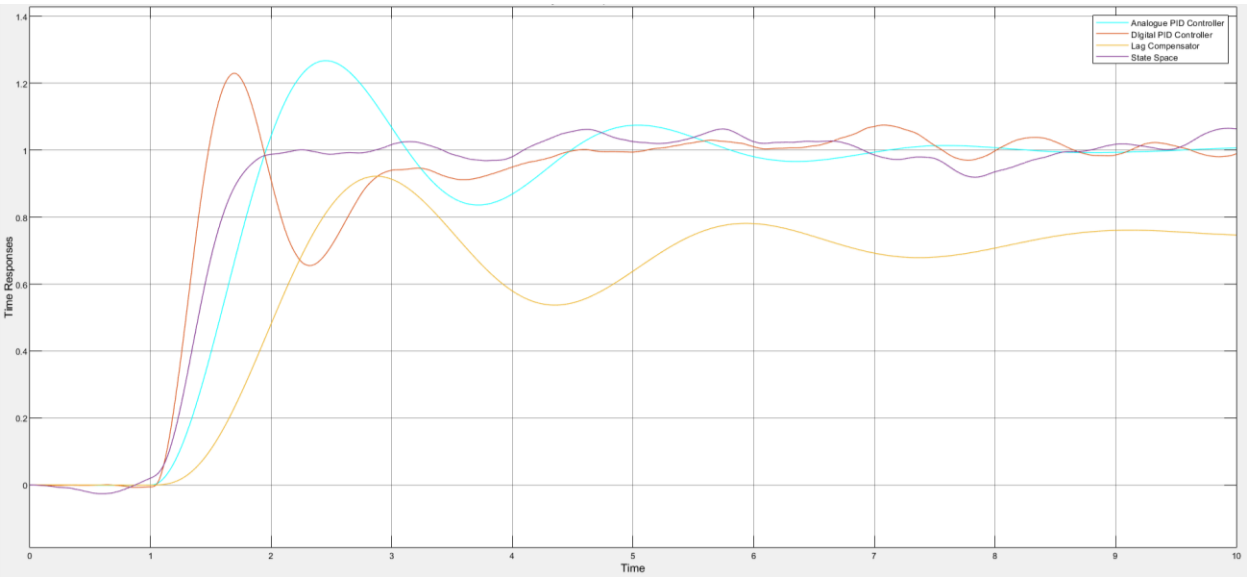


Figure 49: Time response when noise is introduced to 4 types of controllers

From figure 49, it shows that some controllers or compensators work well despite the noise put into the system. This is the case for the analogue PID controller and the lag compensator, which are two systems that are not software based. This is the main reason why the noise does not affect them very much. On the other hand, both the state space and digital PID controller are greatly affected by the noise. This may be because they are software-based controllers, therefore fluctuate with noise more than the other two.

7.2 Controller Performances

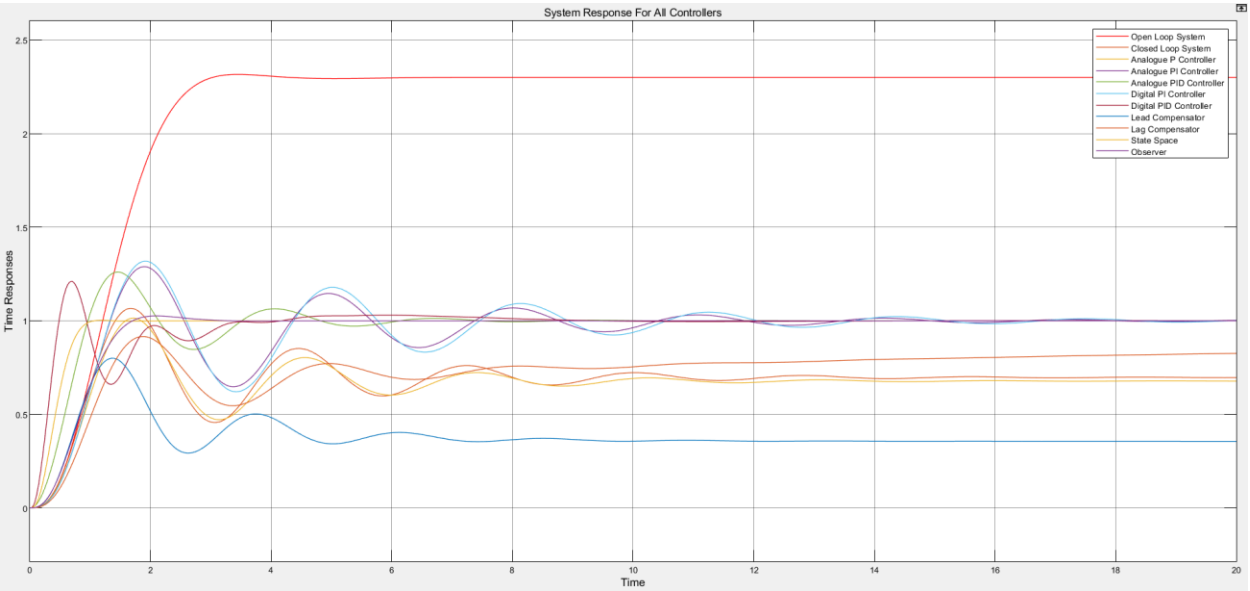


Figure 50: Time response for all controllers

8 Discussion

We can now observe every controlled system that was made during this project in one graph. As you can see from figure 50, the open loop and closed loop system response do not perform well in their time response and it shows when compared to all of the controlled and compensated systems. The open loop system is far too high and the closed loop system is far too low, both therefore have too high of a steady state error. This is also the case with the lead compensator. While it has a faster response time than the closed loop system, it has the lowest final value, hence one of the highest steady state errors. Other control methods improved the steady state error of the system, but some did not reach the required level such as the analogue P controller or the lag compensator. Both of these have similar time responses due to their characteristics.

When analysing control systems, it is important to first look at the steady state error of the system before observing the transient responses. Systems that did perform well in this regard are the analogue PI and PID controller, the digital PI and PID controller and the state space and observer systems. These all meet the steady state error requirements so other factors could determine which controllers performed better or worse.

All PI and PID controllers (analogue and digital) gave overshoots while the PI controllers gave slightly slower responses than the PID controllers. There is very little difference between these sets of controllers, with the digital and analogue PI controllers behaving similarly and the digital and analogue PID controllers behaving similarly. The preferred system depends on whether a digital or analogue controller is preferred. An advantage to the digital PI controller is that the sampling time does not greatly affect the output. This is good because the lower the sampling rate, the more power is required for the controller. We can make the sampling rate on the PI controller as high as possible that does not significantly impact the output and it will consume less power than the PID controller where a lower sampling rate is needed. Although, while saving power consumption is good and it saves money, for our scenario of a wind turbine, the power generated from the turbine is greatly dependent on the controller. While a digital controller with a high sample rate saves money in the short term, the power generated will not be as steady as the PID which is not ideal for a wind turbine as constant power output is desired.

In the graphs, we see that there is not much to separate the analogue and digital controller. The analogue controller has a slightly higher overshoot than the digital controller however they both reach their steady state values at roughly the same with analogue reaching the desired value first. Whether the client would want a digital or analogue PID controller depends on their circumstances. A digital PID controller is much easier to tune with variable tuning alongside other benefits like software integration. An analogue PID controller is harder to tune as real components and hardware is needed to tune it. However, the analogue controller is better for fast response times as there are limits to how fast a digital controller can respond. Furthermore, like previously discussed the digital controllers are restricted by their sampling rate so a low sampling rate is needed for accuracy.

Both digital and analogue PI and PID controllers perform well but neither were as accurate or had as fast of a settling time as the state space and observer systems. The state space system has miniscule overshoot while the observer system has a slightly larger but still very small amount of overshoot. The state space design performs well but is costly to measure with sensors, therefore the observer design may be more appealing to customers who want a cost efficient, well performing system.

While money is an important factor for many clients, most will be worried about performance and this is where these controllers shine. Both the state space and observer design reach their steady state value the fastest which will be extremely important to a wind turbine as there is a constant need to change the blade

angle depending on wind speed and direction. Furthermore, there are plenty of disturbances that may appear due to a wind turbine being prone to extreme weather and temperatures, therefore a controller would need to make the system respond well to disturbances and keep the output at the desired value. Because of this, the PID controllers stand out again as they return the system to a steady state at the desired value faster than the other controllers and with less variance. This variance is also an important factor as the other controllers have large overshoots and variance, if a disturbance could push the commanded speed of the rotor past its maximum capacity then this could cause damage to the turbine, particularly long term.

However, an issue with these controllers is when noise is put into the system, as seen in figure 49. To compensate for this, more expensive sensors are required to reduce the noise associated with sensors. As previously discussed, analogue systems like the compensators or the PID controllers do not have this issue and therefore may be more valuable to a customer. The control method to be used would depend on the system requirements.

9 Conclusion

During the course of this project, the solutions to designing a control method to optimize wind turbine performance, have been found. P, PI and PID analogue controllers were designed as well as digital PI and PID controllers. Along with these, lead and lag compensators as well as a state space model. These different control methods have been analysed and compared to determine the optimum controller for a wind turbine. This optimum controller would need to be safe, consistent, and efficient as wind turbines must have steady power output and a cautious response to any disturbances. Therefore, out of the controllers discussed in this project, the controller closest resembling this optimum design would be either the analogue PID controller or the state space model. The analogue PID controller deals with desired values with minimal overshoot and a fast response time. While it is not as fast as the state space model, a customer may be looking for a system not dependent on software in order to avoid issues that come with that like noise. However, the state space has a faster settling time and a smaller overshoot so may be more applicable to complex systems where accuracy is crucial. The best controller depends on a variety of factors like cost, upkeep, time of season and location. Ultimately, the best controller should be decided by the client.

10 References

[1] Sang-Hoon Kim, *Electric Motor Control*, Pg 39-93. 2017

[2] MathWorks Help Centre, *"What are State Space Models?"*, 2020 [online] available from: https://uk.mathworks.com/help/ident/ug/what-are-state-space-models.html [accessed 22/01/21]

11 Appendix

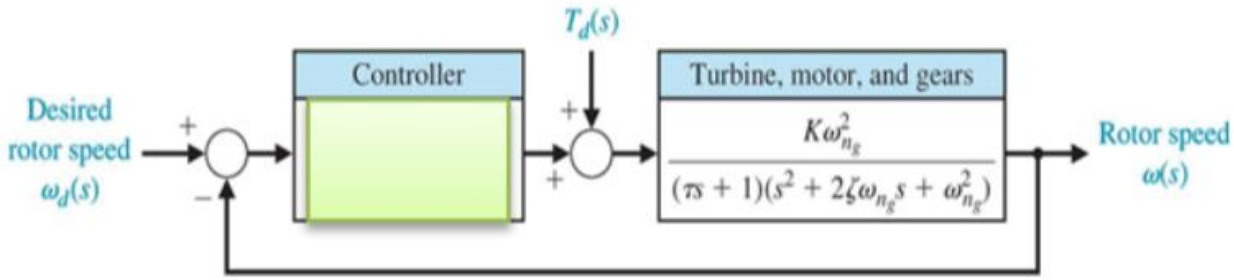
Electronic Logbook

W	Date	Work	Files	H
6	02/11/20	Obtained Transfer Function from parameters given to me in case study sheet. Created OLTF block diagram on Simulink.	OL_System.slx	1/2
6	05/11/20	Performed Linear Analysis on OLTF, obtained step and impulse response and pole-zero map.	OL_LAT.mat	1
6	06/11/20	Obtained values for step 5 on case study sheet. Created closed loop transfer function block diagram on Simulink, performed linear analysis to obtain step and impulse diagrams and pole-zero map. Obtained values from step 10 on case study sheet.	CL_System.slx CL_LAT.mat	3
7	12/11/20	Built block diagram to find Kcr and Pcr parameters. Through trial and error, found parameters. Used Ziegler-Nichols tables to convert values into PID form 1 and 2 parameters	Test_Kcr_Pcr.slx Find_Form_1.m Find_Form_2.m	4
7	15/11/20	Created block diagrams for P, PI and PID controllers, built code to plot graph showing all 3 outputs on one graph.	PID_outputs.slx Plot_P_PI_PID.m	2
8	16/11/20	Changed block diagrams to include disturbance, built code to plot graph and observed how each controller responded to disturbance.	PID_outputs_dist.slx Plot_P_PI_PID_dist.m	3
8	17/11/20	Built code to convert continuous PID into discrete (digital) PID. Built a block diagram with digital PID and zero order holds. Built code to plot digital PI and PID controller on same graph. Added disturbance to block diagram, built code to plot outputs with disturbance on graph.	c2d_converter.m Dig_PID_Outputs.slx Plot_dig_PI_PID.m Dig_PID_Outputs_dist.slx Plot_dig_PI_PID_dist.m	10
8	18/11/20	Built code to plot all controller outputs on same graph. Changed sampling time to observe how it effected output on graphs for PI and PID digital controllers.	all_outputs.slx all_outputs_dist.slx PI_T.slx PID_T.slx PI_T_dist.slx PID_T_dist.slx plot_dig_PI_T.m plot_dig_PID_T.m plot_dig_PI_T_dist.m plot_dig_PID_T_dist.m	12
-	08/01/21	Built Lead compensator, observed how a and T values changed time response.	Finding_Kc.slx Lead_Compensator_Test.slx	15

			Lead_Test.m Lead_Compensator_Test.slx	
-	12/01/21	Built lag compensator, observed how b and T values changed time response.	Lag_Compensator_Test.slx Lag_Test.m T and b relationship.xlsx	
-	14/01/21	Built state space model, observed how pole placements affect time response.	State_Space_Test_K.slx State_Space.m State_Space_Model.slx	
-	15/01/21	Built Observer model	Observer_Poles_Test.m Observer_design.slx	
-	16/01/21	Built block diagram to test controllers with responses. Built block diagram with all controllers to show time responses on one graph.	Noise_all.slx All_output.slx	

Monday, 2nd November 2020

For this assignment I made an electronic logbook to record all information and data gained from each lab session I had.



My plant model parameters:

$K = 2.3$ $\omega_{ng} = 1.9$ $\tau = 0.7$ $\xi\omega_{ng} = 1.2$

The open loop transfer function can be calculated using the parameters given:

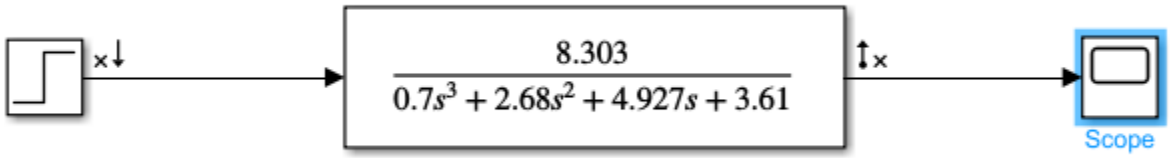
$$OLTF = \frac{K\omega_{ng}^2}{(\tau s+1)(s^2 + 2(\xi\omega_{ng})s + \omega_{ng}^2)} = \frac{(2.3)(1.9^2)}{\big((0.7)s+1\big)(s^2+2(1.2)s+1.9^2)}$$

$$OLTF = \frac{8.303}{(0.7s+1)(s^2 + 2.4s + 3.61)} = \frac{8.303}{(0.7s+1)(s^2 + 2.4s + 3.61)}$$

$$OLTF = \frac{8.303}{(0.7s^3 + 2.68s^2 + 4.927s + 3.61)}$$

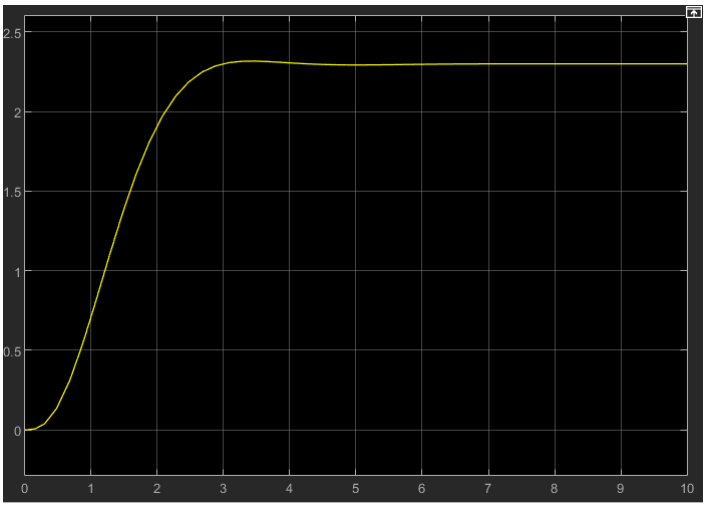
This is the transfer function that will be used in my open loop system.

Build Open-Loop transfer function in Simulink

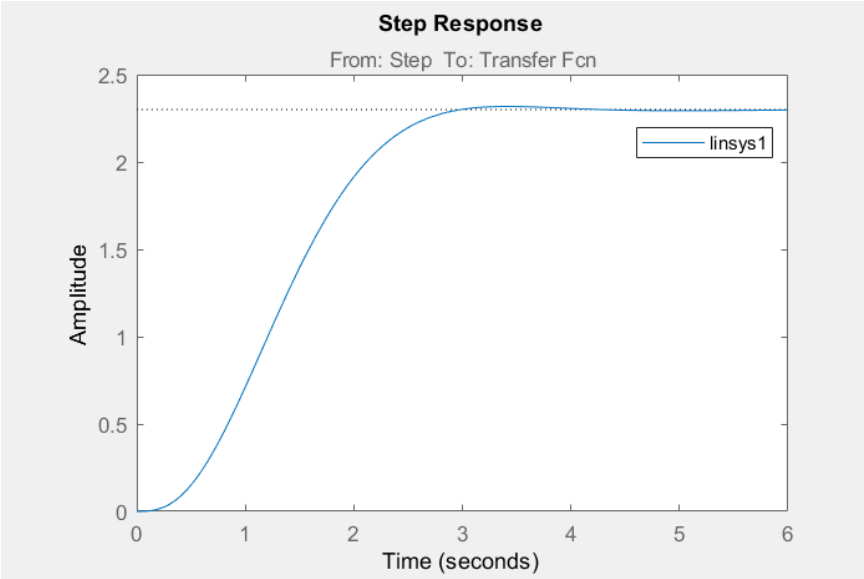


Thursday, 5th November 2020

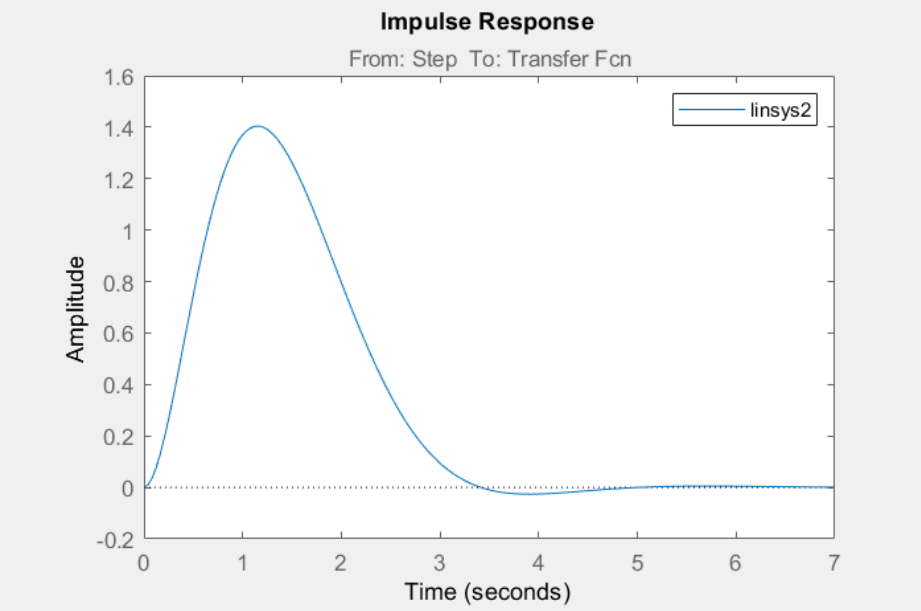
The scope shows the following graph which shows the response of the system from a step function.



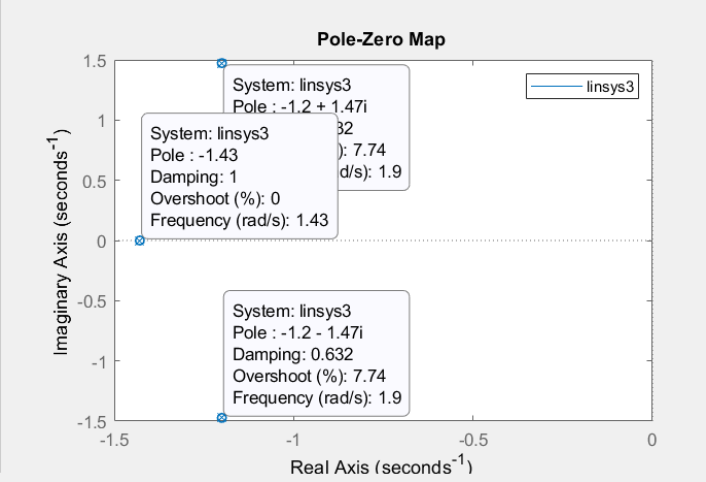
The linear analysis tool gives us this graph showing the output of the system in response to a step function.

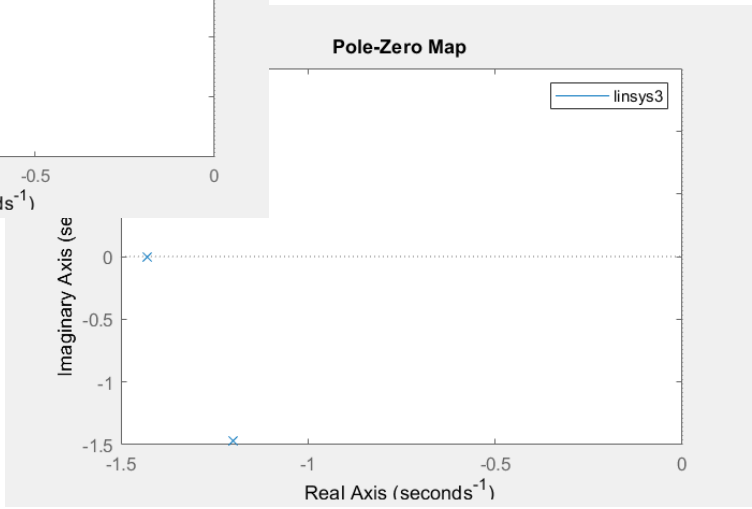


This is the response shown for an impulse function.



Pole-zero graph given by linear analysis tool





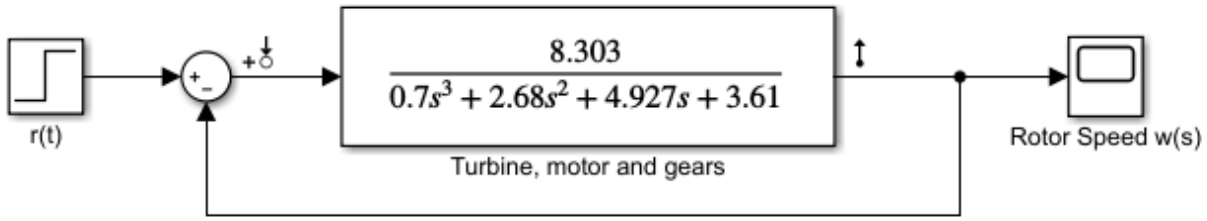
Friday, 6th November 2020

Using S=stepinfo(linsys1) I obtained values for question 5 on case study sheet:

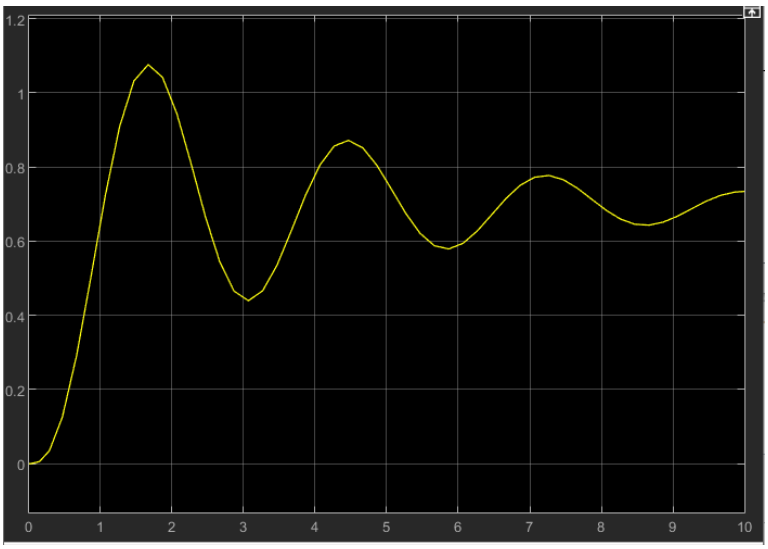
t_r = 1.632s t_s = 2.698s OP% = 0.77%

```
>> S=stepinfo(linsys1)

S =

  struct with fields:

       RiseTime: 1.6321
   SettlingTime: 2.6977
    SettlingMin: 2.0910
    SettlingMax: 2.3177
      Overshoot: 0.7709
     Undershoot: 0
           Peak: 2.3177
       PeakTime: 3.4155
```

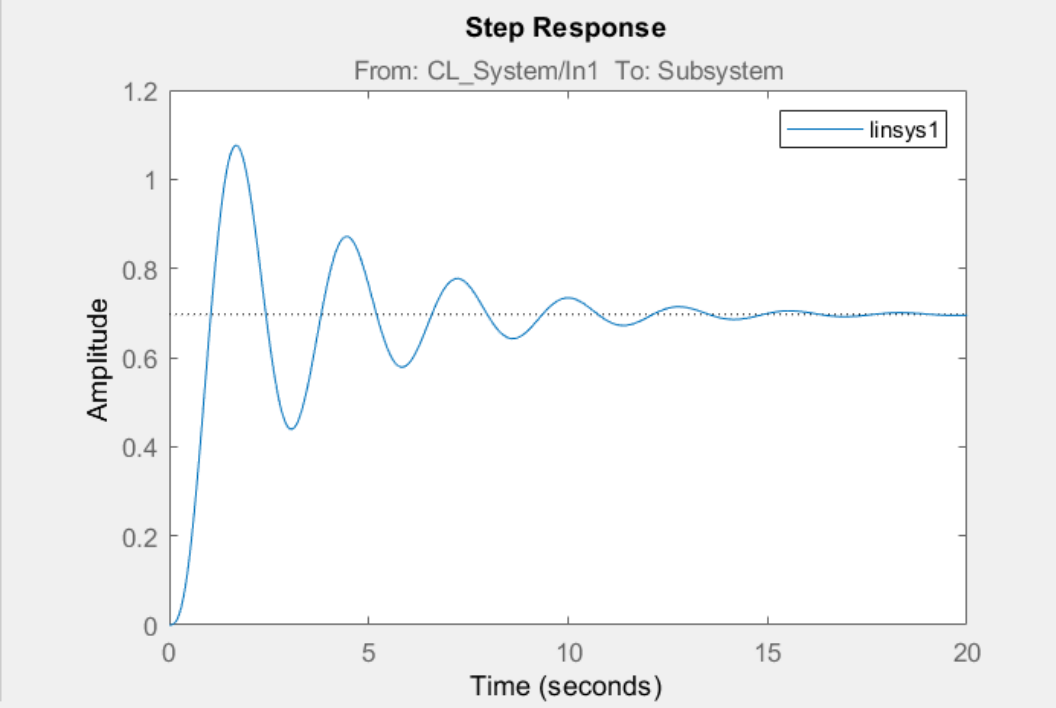
Closed loop system:



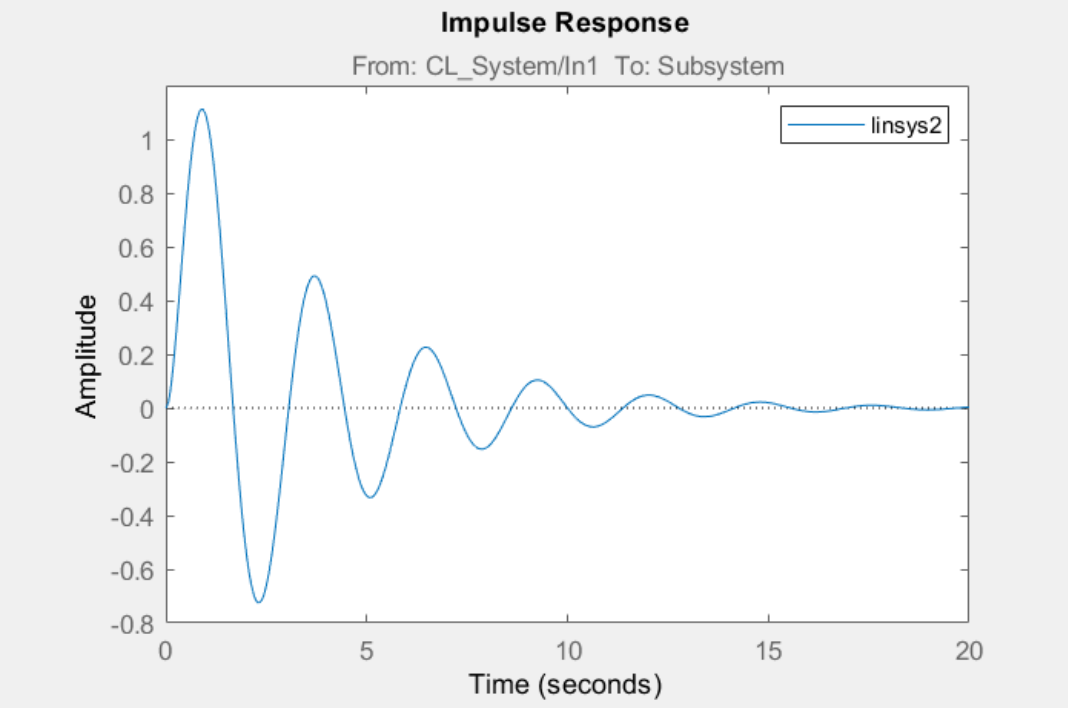
Closed-Loop System output on the scope:



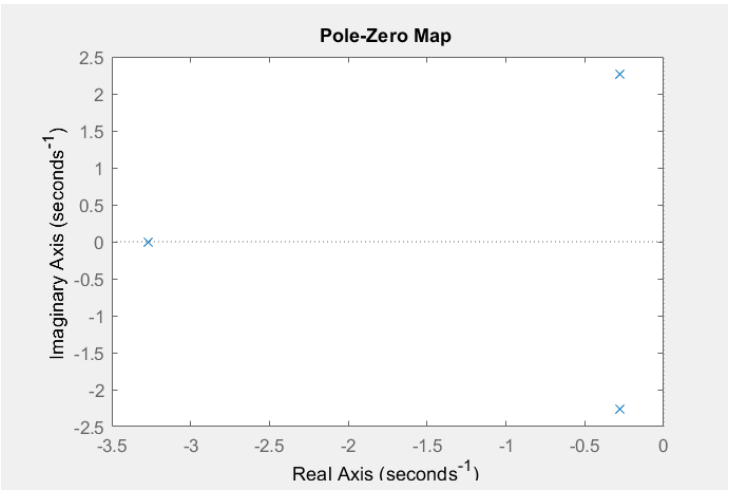
Step response through linear analysis tool:

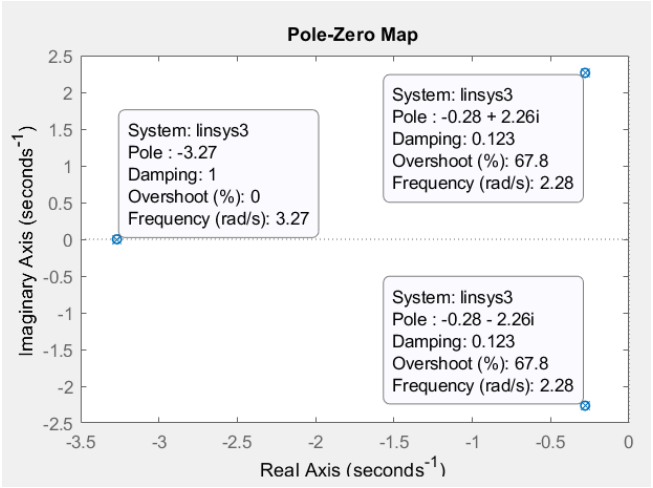


Impulse response:



Pole-zero map:





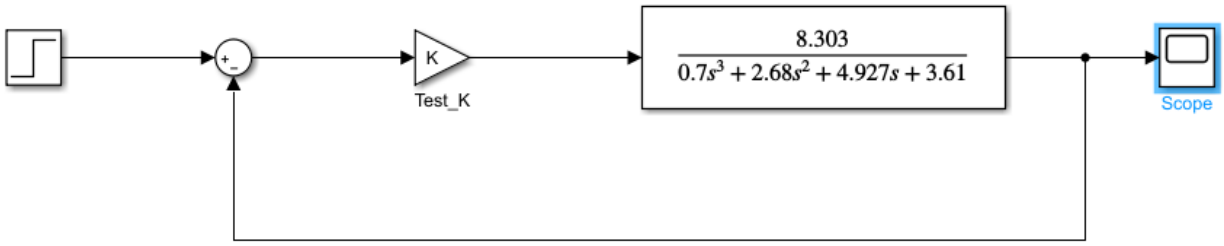
Through stepinfo function, obtained parameters from question 10 on case study sheet:

t_r = 0.6074s t_s = 13.04s OP% = 54.36%

```
>> S=stepinfo(linsys1)

S =

  struct with fields:

       RiseTime: 0.6074
   SettlingTime: 13.0440
    SettlingMin: 0.4393
    SettlingMax: 1.0759
      Overshoot: 54.3631
     Undershoot: 0
           Peak: 1.0759
       PeakTime: 1.6627
```

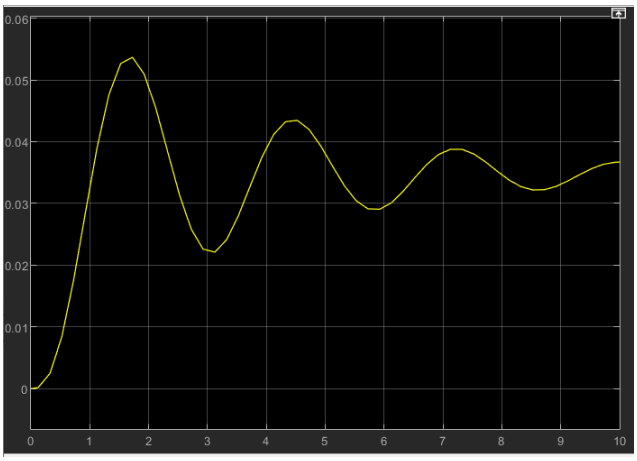
Thursday, 12th November 2020

Finding values for K_{cr} and P_{cr}

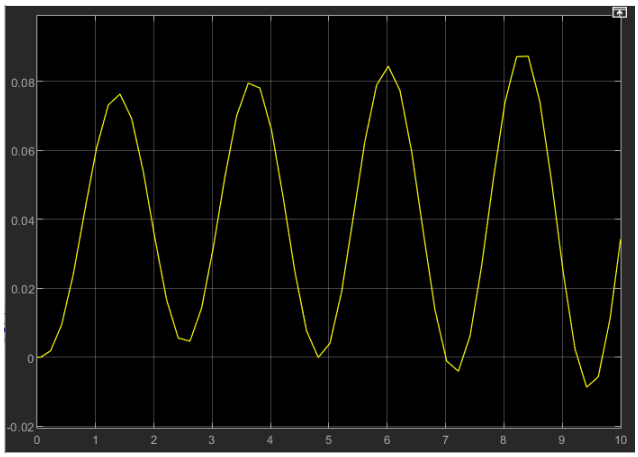


Trying values of K until we find critical value:

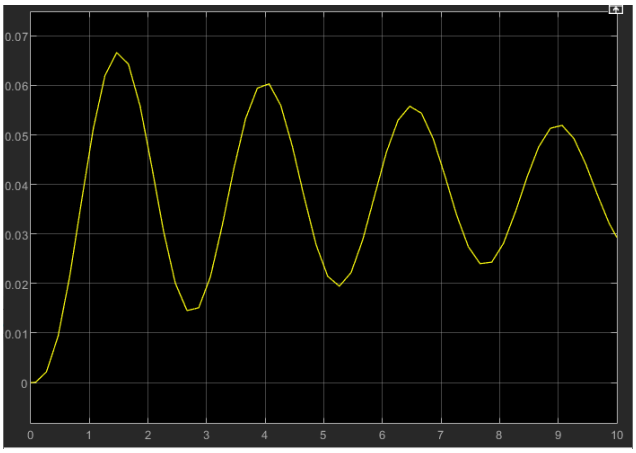
K=1:



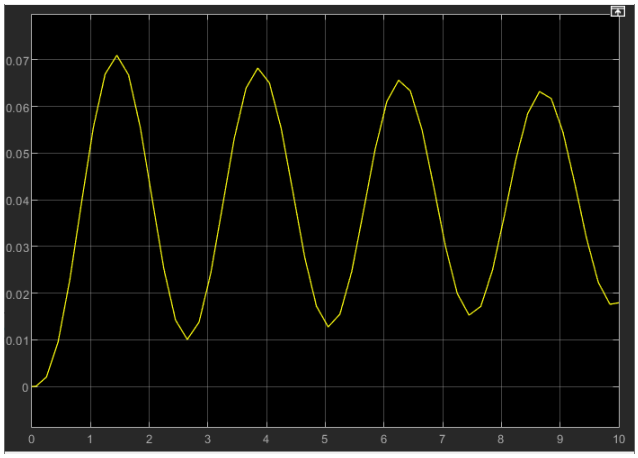
K=2:



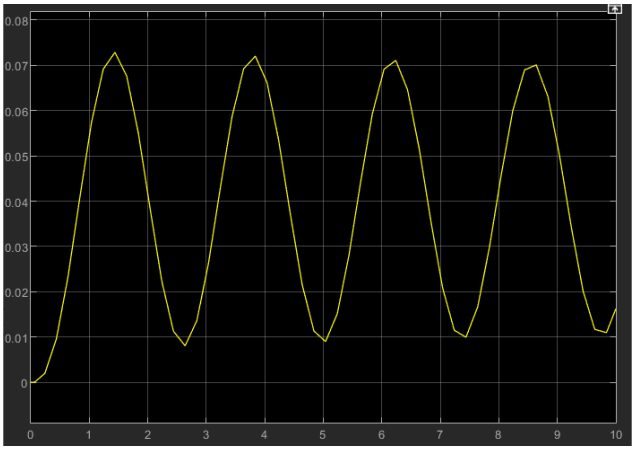
K=1.5:



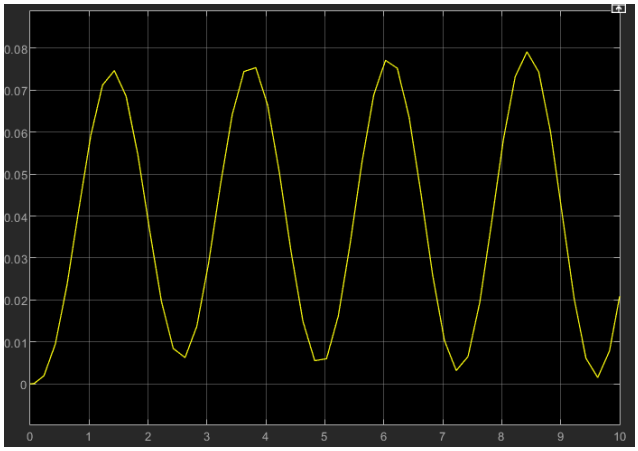
K=1.7



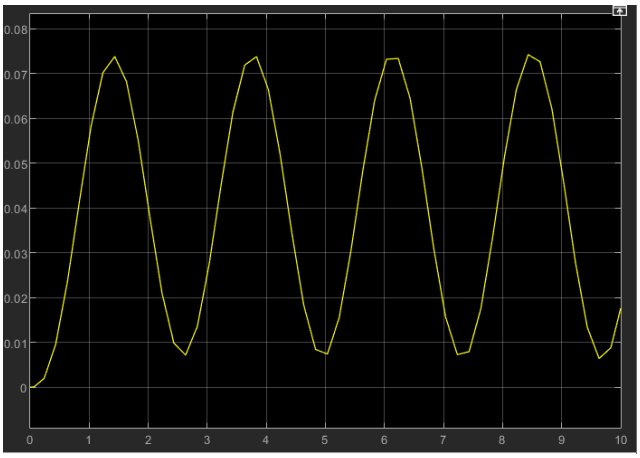
K=1.8:



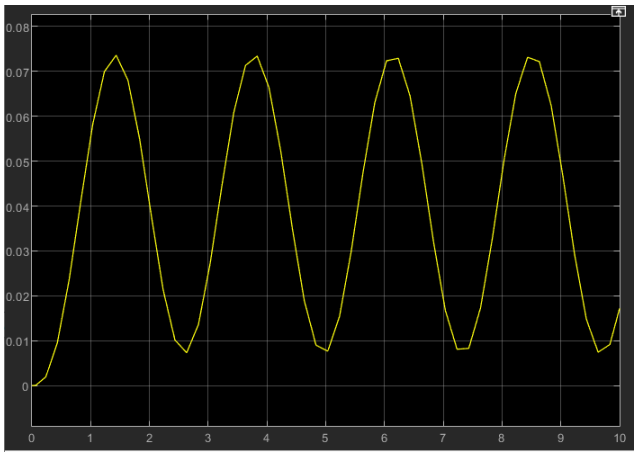
K=1.9:



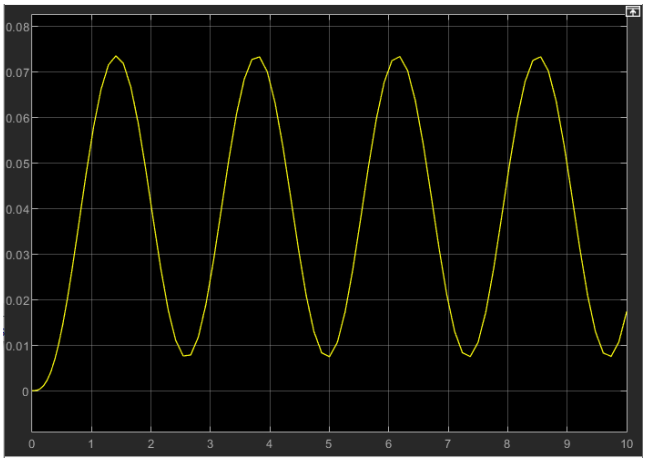
K=1.85:



K=1.84

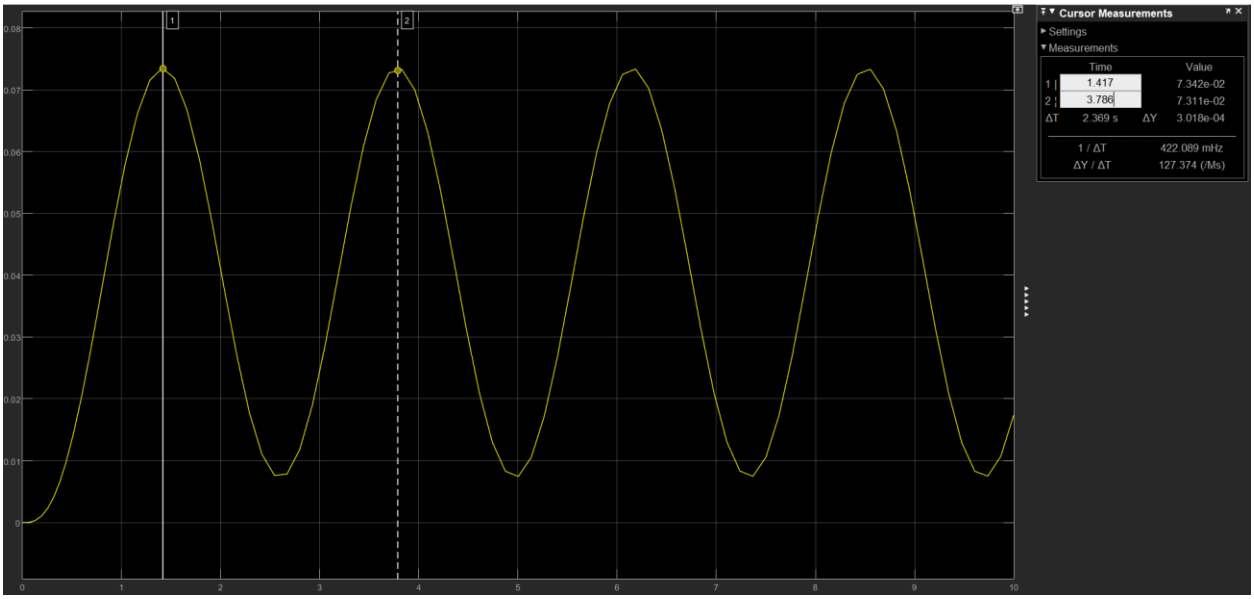


K=1.83:



Close between values of 1.83 or 1.84, chose 1.83 as it was slightly more consistent throughout waveform.

Showing the period between two peaks (P_{cr}):



$K_{cr} = 1.83$ $P_{cr} = 2.369$

Type of controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

PID Form 1:

Type of controller	K_p	T_i	T_d
P	0.915	∞	0
PI	0.8235	1.9742	0
PID	1.0980	1.1845	0.2961

$$P = K_p \qquad I = \frac{K_p}{T_i} \qquad D = K_p T_d$$

PID Form 2:

Type of controller	K_p	T_i	T_d
P	0.915	∞	0
PI	0.8235	0.4171	0
PID	1.0980	0.9270	0.3251

To calculate these values used in the tables above, I used a MATLAB script to convert the K_{cr} and P_{cr} values into the form 1 and form 2 P, PI and PID formulas.

Form 1:

```
% Find parameters Kp, Ti and Td for P, PI and PID
controllers based on Z-N
% tuning rule
Kcr=1.83;
Pcr=2.369;

% for P controller:
P_Kp=0.5*Kcr;

% for PI controller:
PI_Kp=0.45*Kcr;
PI_Ti=(1/1.2)*Pcr;

% for PID controller:
PID_Kp=0.6*Kcr;
PID_Ti=0.5*Pcr;
```

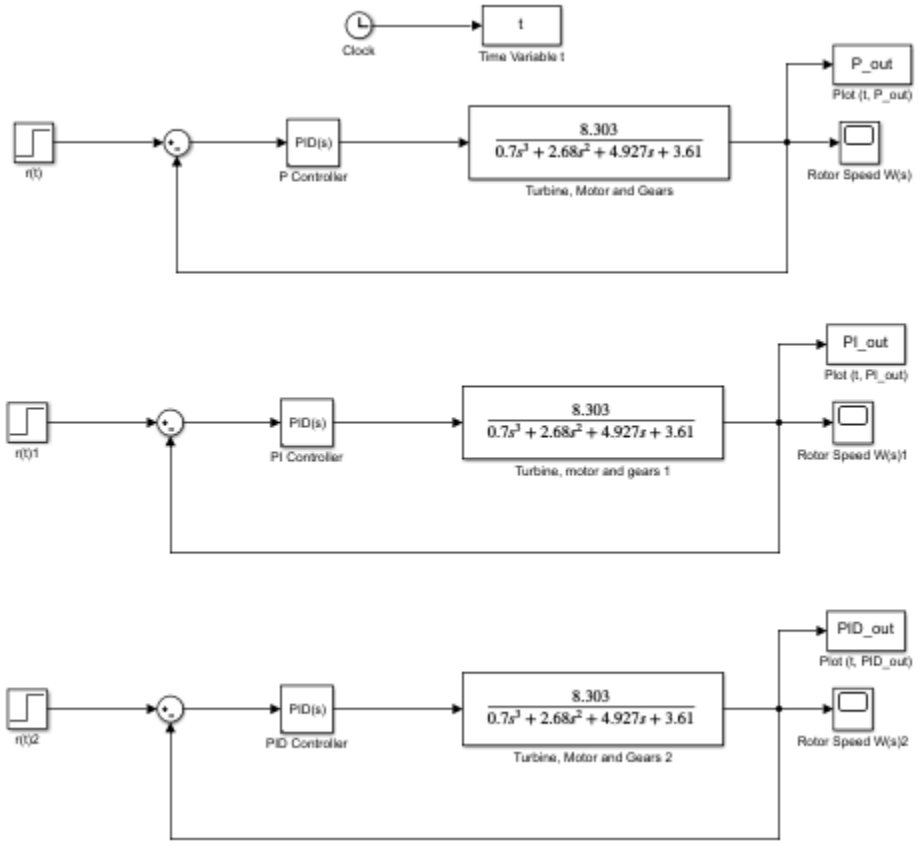
```
PID_Td=0.125*Pcr;
```

Form 2:

```
% from Kp, Ti and Td of P, PI and PID controllers we can
find P, I and D
% parameters of form 2 P, PI and PID controllers

% for P controller:
P_P=P_Kp;

% for PI controller:
PI_P=PI_Kp;
PI_I=PI_Kp/PI_Ti;

% for PID controller:
PID_P=PID_Kp;
PID_I=PID_Kp/PID_Ti;
PID_D=PID_Kp*PID_Td;
```

Sunday, 15th November 2020

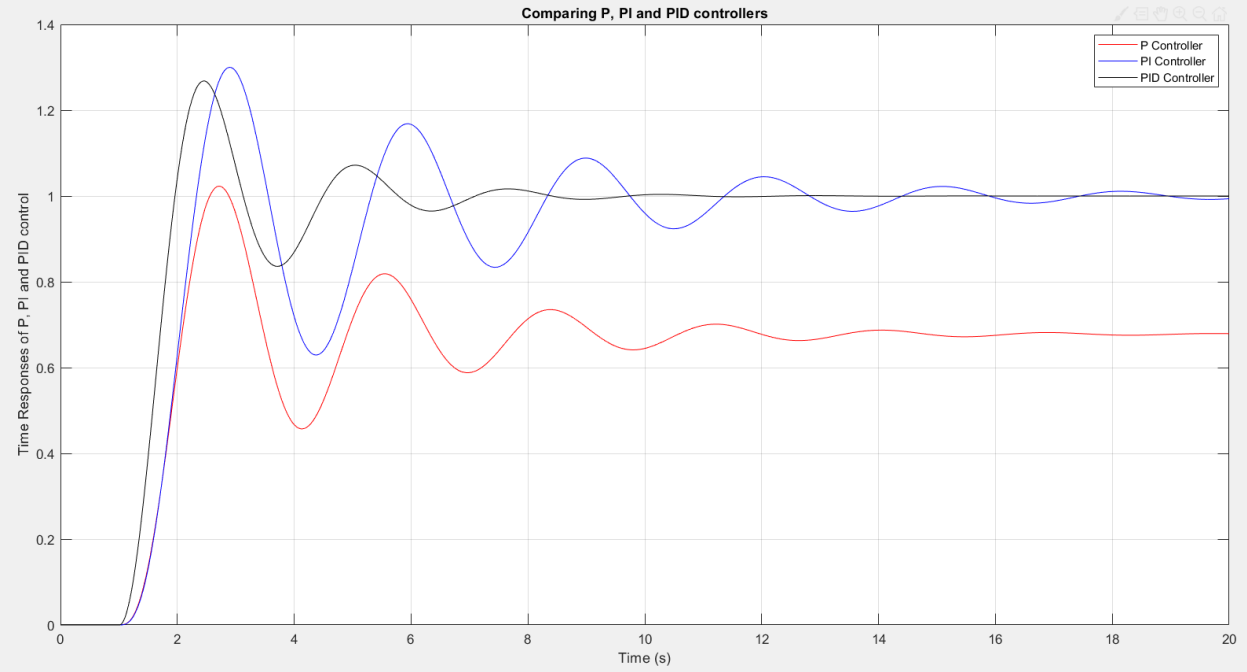
Screenshots of testing the outputs of the P, PI and PID controllers



MATLAB code used to plot graphs:

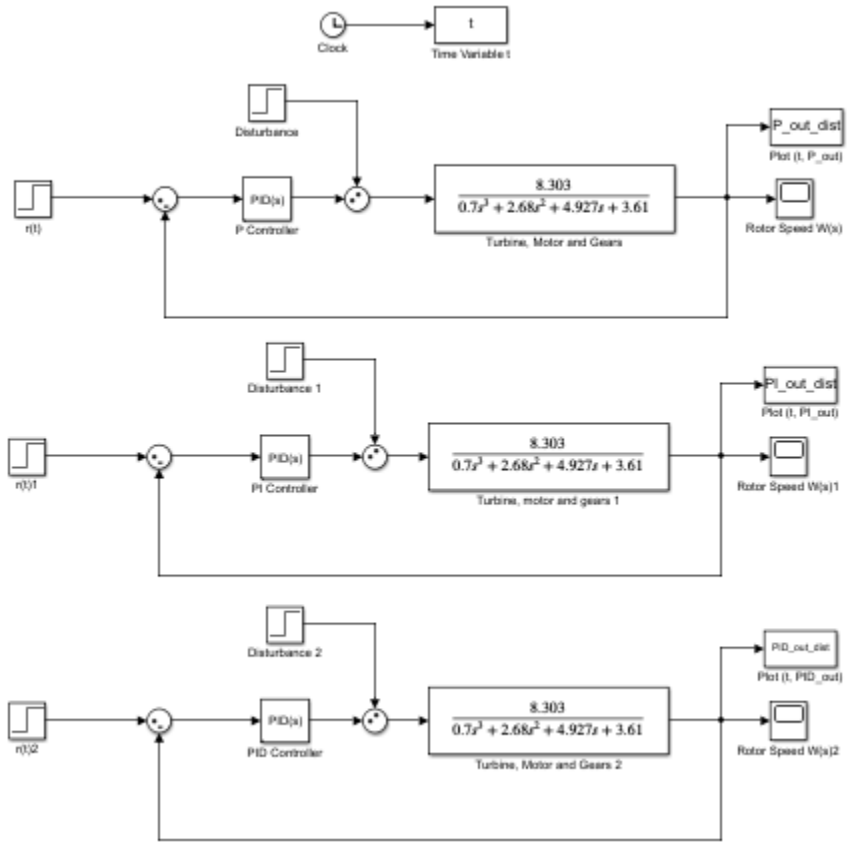
```
plot(t, P_out, 'r-', t, PI_out, 'b-', t, PID_out, 'k-');
grid;
xlabel('Time (s)');
ylabel('Time Responses of P, PI and PID control');
title('Comparing P, PI and PID controllers');
```

Graph comparing all controllers:



Monday, 16th November 2020

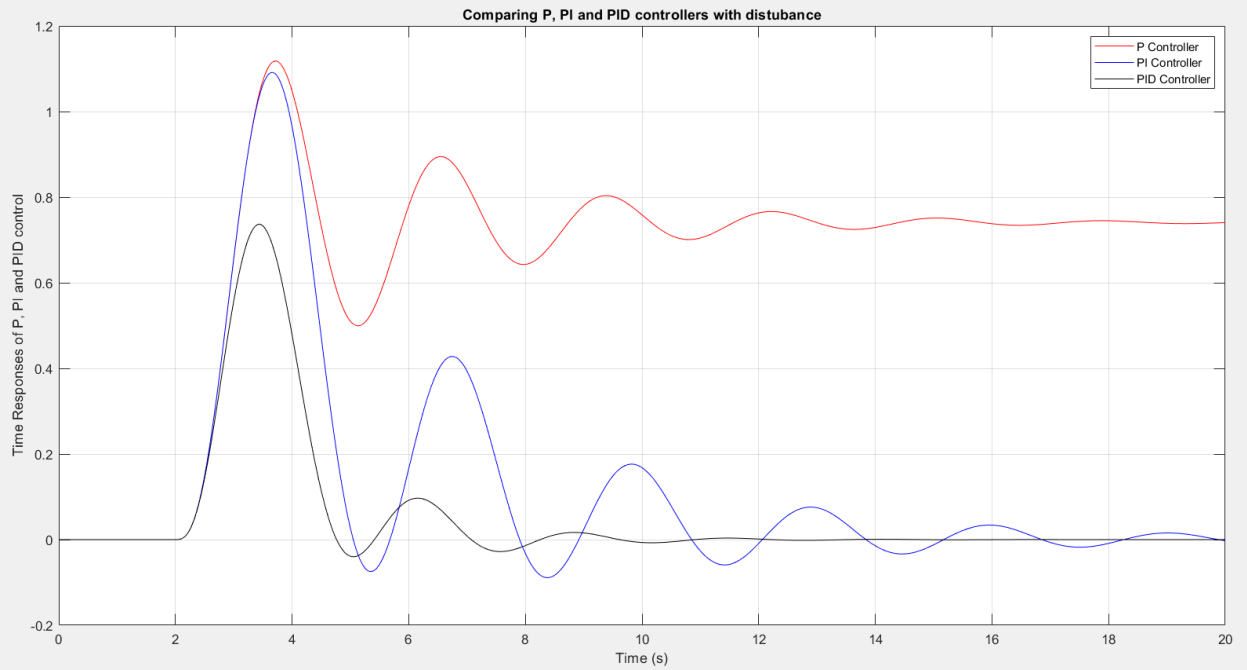
Adding Disturbance to PID controllers:



MATLAB script used to plot graph showing all PID outputs with disturbance

```
plot(t, P_out_dist, 'r-', t, PI_out_dist, 'b-', t,
PID_out_dist, 'k-');
grid;
xlabel('Time (s)');
ylabel('Time Responses of P, PI and PID control');
title('Comparing P, PI and PID controllers with
distubance');
```

Graph showing disturbance effects on PID controllers



Tuesday, 17th November 2020

$$PID\ Controller\ =\ G_c(s)\ =\ P + I\frac{1}{s} + D\frac{1}{\frac{1}{N}+\frac{1}{s}}\ =\ \frac{(P + D\ x\ N)s^2\ +\ (P\ x\ \ N\ +\ I)s\ +\ I\ x\ N}{s^2\ +\ sN\ +\ 0}$$

$$PI\ Controller =\ G_c(s)\ =\ \frac{Ps^2\ +\ (P\ x\ \ N\ +\ I)s\ +\ I\ x\ N}{s^2\ +\ sN\ +\ 0}$$

MATLAB script converting continuous to discrete:

```
% this script converts the transfer function of the PI and
PID controllers
% from continuous to discrete systems:
N=100;
PID_num=[PID_P+PID_D*N PID_P*N+PID_I PID_I*N];
PI_num=[PI_P PI_P*N+PI_I PI_I*N];
PID_den=[1 N 0];

PID_sysc=tf(PID_num,PID_den);
PI_sysc=tf(PI_num,PID_den);

%convert using c2d:
T=0.05;
[PID_Gcz]=c2d(PID_sysc,T,'zoh');
[PI_Gcz]=c2d(PI_sysc,T,'zoh');
```

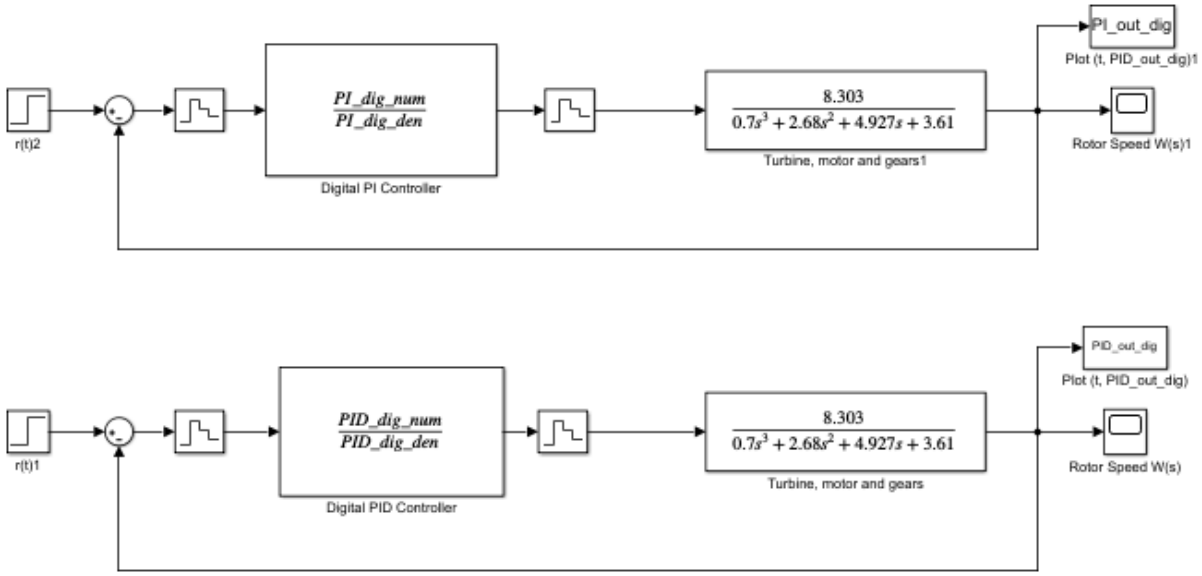
Obtained digital PI and PID Transfer Function:

```
>> PID_Gcz

PID_Gcz =

  33.61 z^2 - 66.09 z + 32.52
  ---------------------------
   z^2 - 1.007 z + 0.006738

Sample time: 0.05 seconds
Discrete-time transfer function.

>> PI_Gcz

PI_Gcz =

  0.8235 z^2 - 0.8082 z + 0.005408
  --------------------------------
      z^2 - 1.007 z + 0.006738

Sample time: 0.05 seconds
Discrete-time transfer function.
```

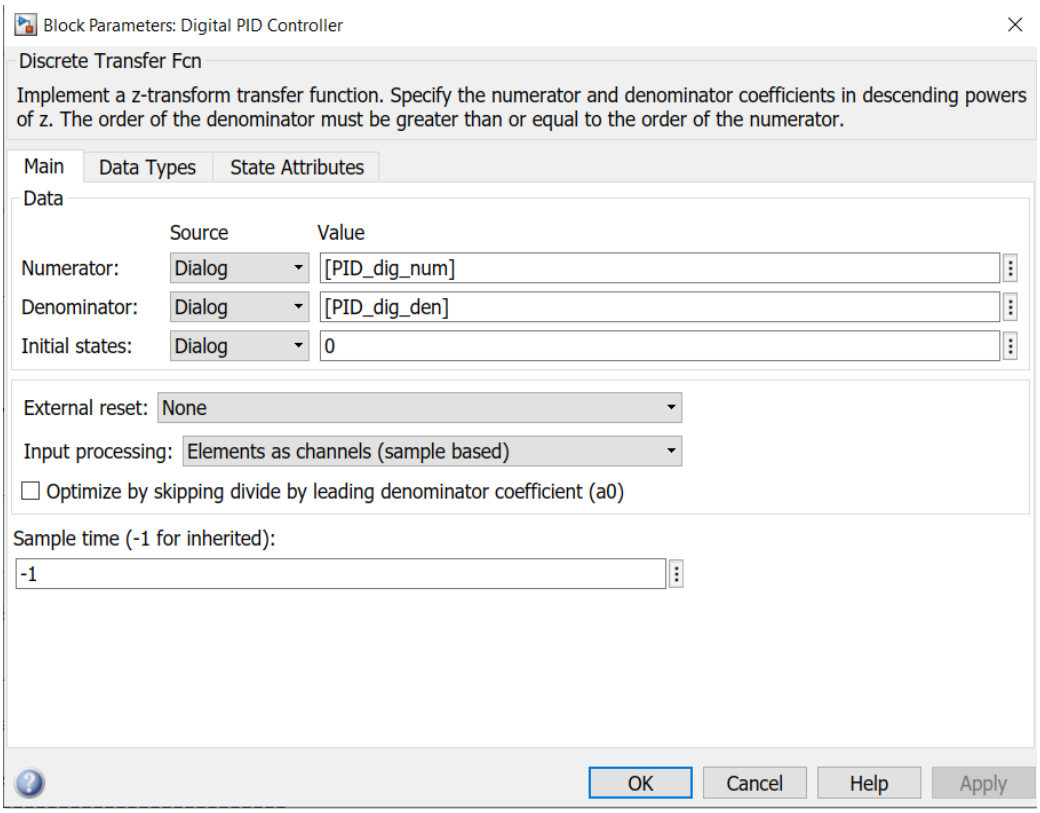
MATLAB Script converting continuous to discrete, putting numerator and denominator in separate array to be used for function in block diagram.

```
% this script converts the transfer function of the PI and
PID controllers
% from continuous to discrete systems:
N=100;
PID_num=[PID_P+PID_D*N PID_P*N+PID_I PID_I*N];
PI_num=[PI_P PI_P*N+PI_I PI_I*N];
PID_den=[1 N 0];

PID_transf=tf(PID_num,PID_den);
PI_transf=tf(PI_num,PID_den);

%convert using c2d:
[PID_dig_num,PID_dig_den]=tfdata(c2d(PID_transf,T,'zoh'),'v
');
[PI_dig_num,PI_dig_den]=tfdata(c2d(PI_transf,T,'zoh'),'v');
```

Block Diagram showing digital PI and PID controllers:



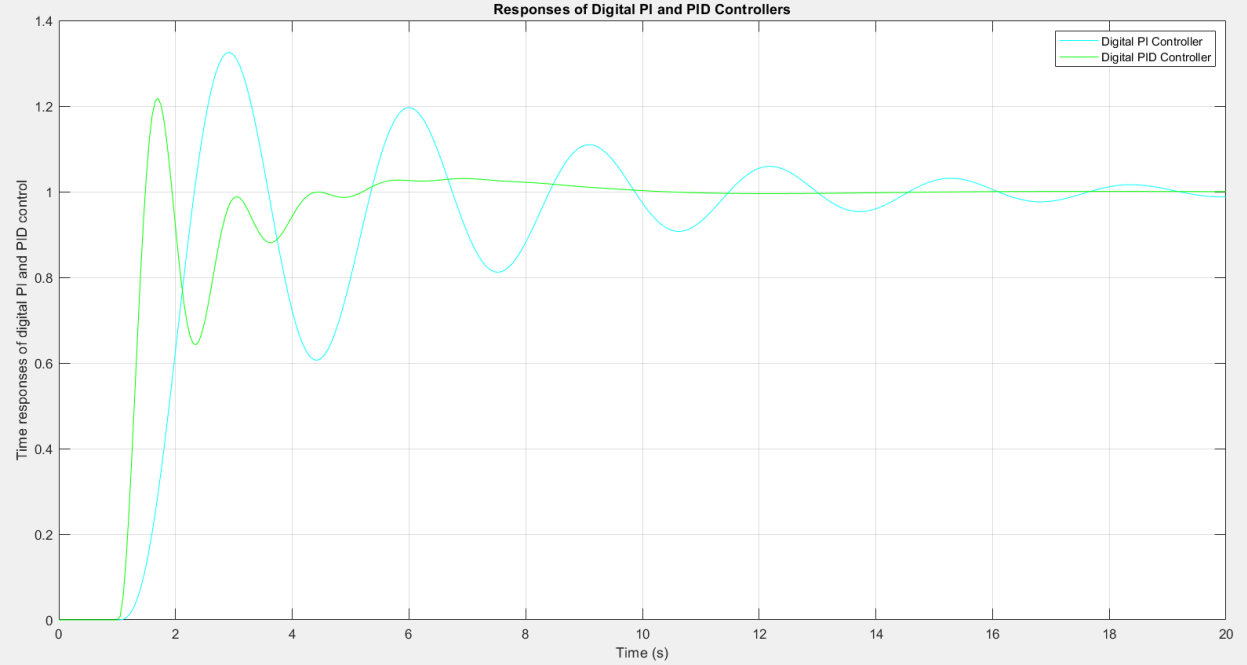
In discrete transfer function settings, I placed the numerator and denominator functions instead of the numbers, that way if I want to change the sampling time, I only need to do it once through MATLAB.



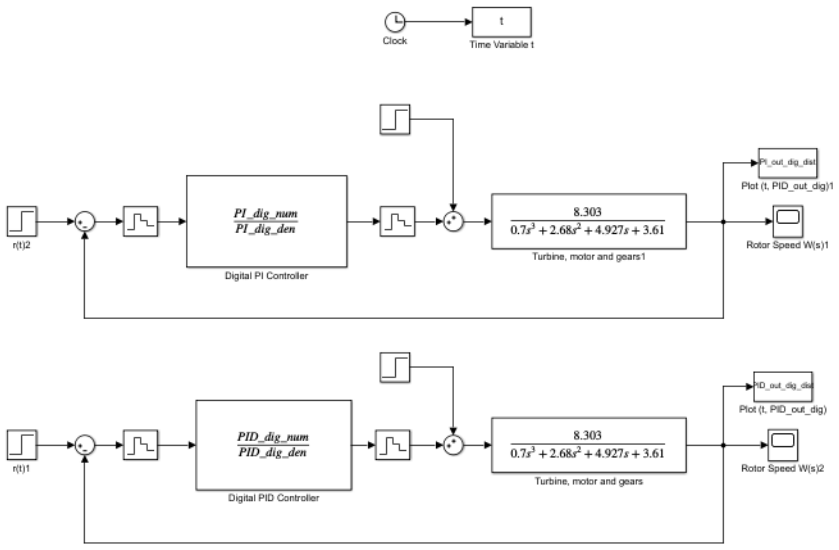
MATLAB script for plotting digital PI and PID controllers:

```
plot(t, PI_out_dig, 'c-', t, PID_out_dig, 'g-');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PI and PID control');
title('Responses of Digital PI and PID Controllers');
```

Graph showing digital PI and PID responses:



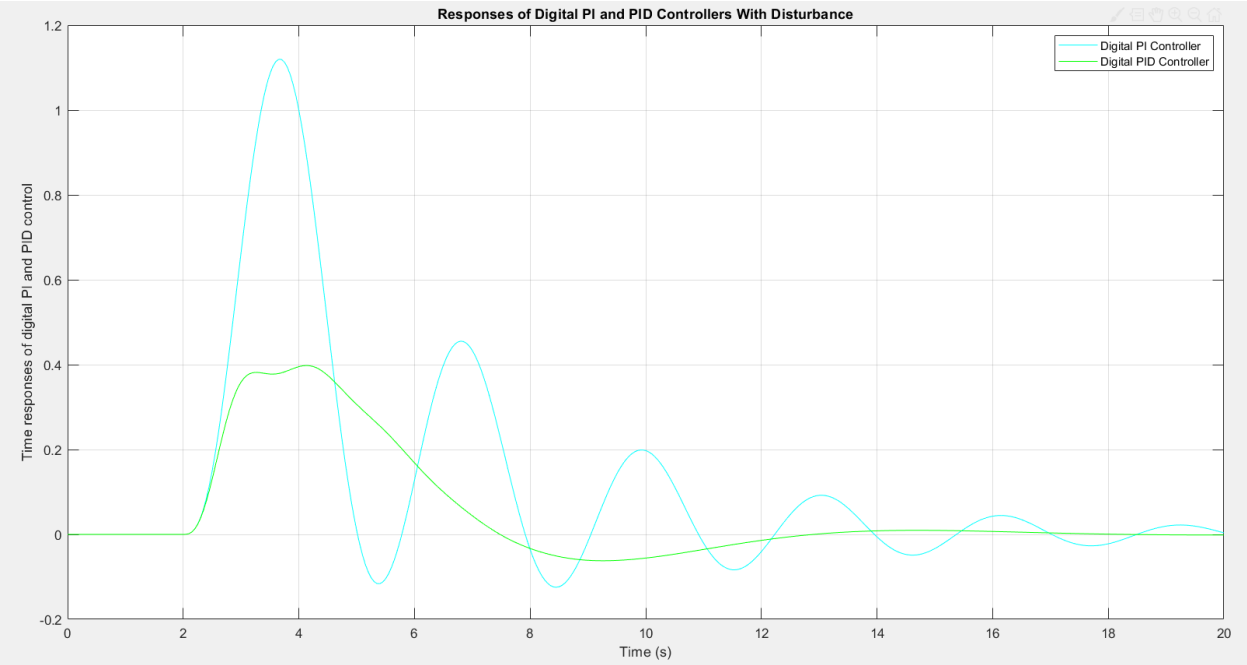
Digital PI and PID block diagram with disturbance:



MATLAB script to plot graph:

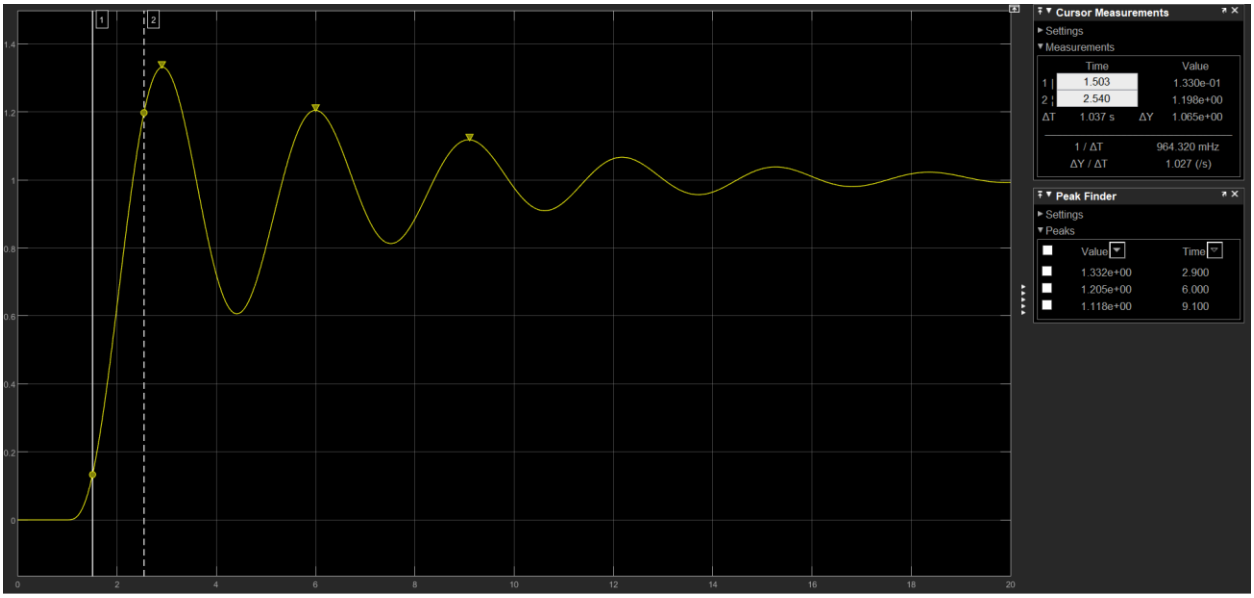
```
plot(t, PI_out_dig_dist, 'c-', t, PID_out_dig_dist, 'g-');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PI and PID control');
title('Responses of Digital PI and PID Controllers With
Disturbance');
```

Graph showing response of digital PI and PID controller with a disturbance



Wednesday, 18th November 2020

Finding Sampling time by finding t_r to calculate $T = t_r/n$ where $10 < n < 20$



n = 11 to 19:

t_r = 1.037s

T = 0.0546/0.0576/0.061/0.0648/0.0691/0.0740/0.0798/0.0864/0.0942

Try to plot graph showing all values for T

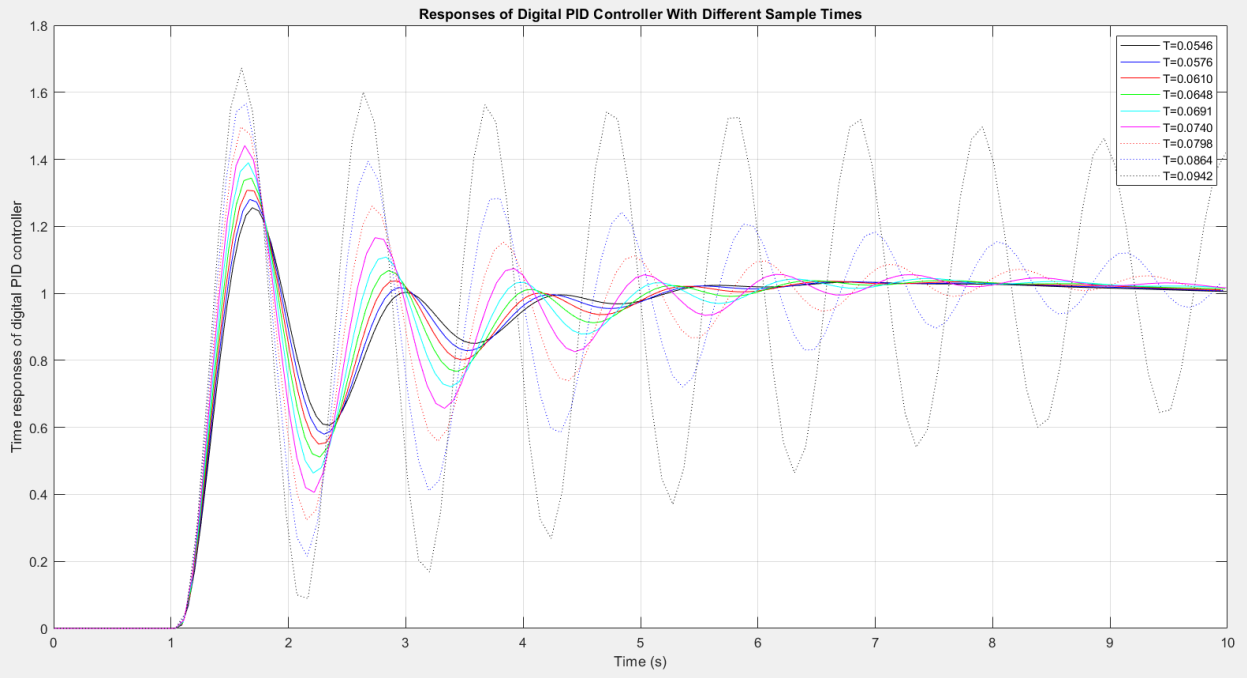
```
>> T=0.0546;
>> c2d_converter_2
>> T=0.0576

T =

    0.0576

>> c2d_converter_2
>> T=0.061;
>> c2d_converter_2
>> T=0.0648;
>> c2d_converter_2
>> T=0.0691;
>> c2d_converter_2
>> T=0.074;
>> c2d_converter_2
>> T=0.0798;
>> c2d_converter_2
>> T=0.0864;
>> c2d_converter_2
>> 0.0942;
>> c2d_converter_2
>> Plot_dig_PID_T
```

T	0.0942
T1_PI	*184x1 double*
T1_PID	*184x1 double*
T2_PI	*174x1 double*
T2_PID	*174x1 double*
T3_PI	*164x1 double*
T3_PID	*164x1 double*
T4_PI	*155x1 double*
T4_PID	*155x1 double*
T5_PI	*145x1 double*
T5_PID	*145x1 double*
T6_PI	*136x1 double*
T6_PID	*136x1 double*
T7_PI	*126x1 double*
T7_PID	*126x1 double*
T8_PI	*116x1 double*
T8_PID	*116x1 double*
T9_PID	*107x1 double*
t_t1	*184x1 double*
t_t2	*174x1 double*
t_t3	*164x1 double*
t_t4	*155x1 double*
t_t5	*145x1 double*
t_t6	*136x1 double*
t_t7	*126x1 double*
t_t8	*116x1 double*
t_t9	*107x1 double*
tout	*112x1 double*

MATLAB Script for plotting different sample times:

```
plot(t_t1, T1_PID, 'k-', t_t2, T2_PID, 'b-', t_t3, T3_PID,
'r-', t_t4, T4_PID, 'g-', t_t5, T5_PID, 'c-', t_t6, T6_PID,
'm-', t_t7, T7_PID, 'r:', t_t8, T8_PID, 'b:', t_t9, T9_PID,
'k:');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PID controller');
title('Responses of Digital PID Controller with Different
Sample Times');
```

Graph showing different sample times affecting PI controller reaction to step input

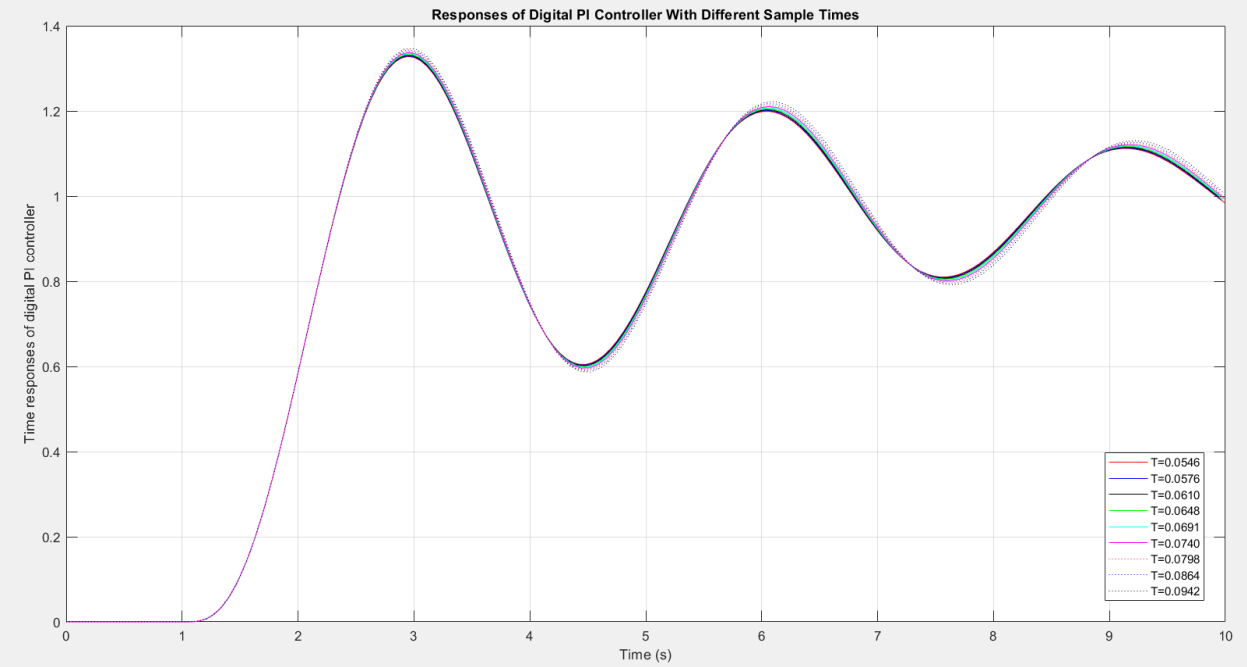


<u>Doing the same for other controllers and with disturbances</u>

MATLAB Script for plotting PI controller with different sample times:

```
plot(t_t1, PI_T1, 'r-', t_t2, PI_T2, 'b-', t_t3, PI_T3, 'k-
', t_t4, PI_T4, 'g-', t_t5, PI_T5, 'c-', t_t6, PI_T6, 'm-',
t_t7, PI_T7, 'r:', t_t8, PI_T8, 'b:', t_t9, PI_T9, 'k:');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PI controller');
title('Responses of Digital PI Controller With Different
Sample Times');
```

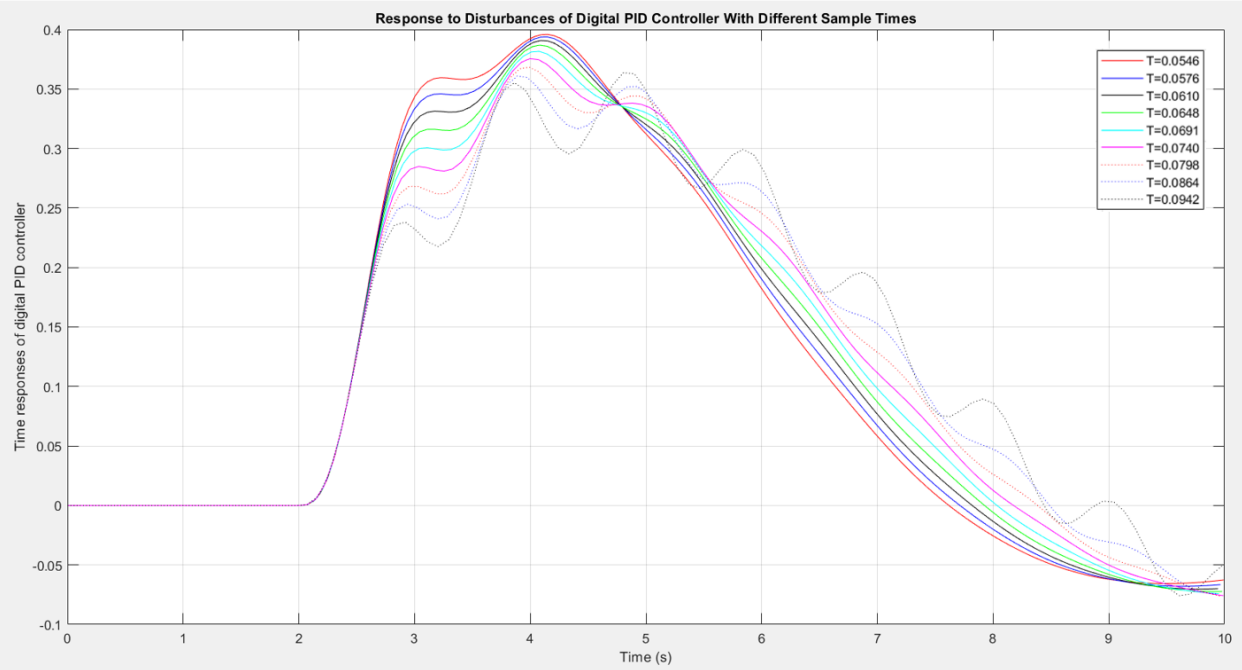
Graph showing different sample times affecting PI controller reaction to step input



MATLAB Script for plotting PID controller responding to disturbance:

```
plot(t_d1, DT1_PID, 'r-', t_d2, DT2_PID, 'b-', t_d3,
DT3_PID, 'k-', t_d4, DT4_PID, 'g-', t_d5, DT5_PID, 'c-',
t_d6, DT6_PID, 'm-', t_d7, DT7_PID, 'r:', t_d8, DT8_PID,
'b:', t_d9, DT9_PID, 'k:');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PID controller');
title('Response to Disturbances of Digital PID Controller
With Different Sample Times');
```

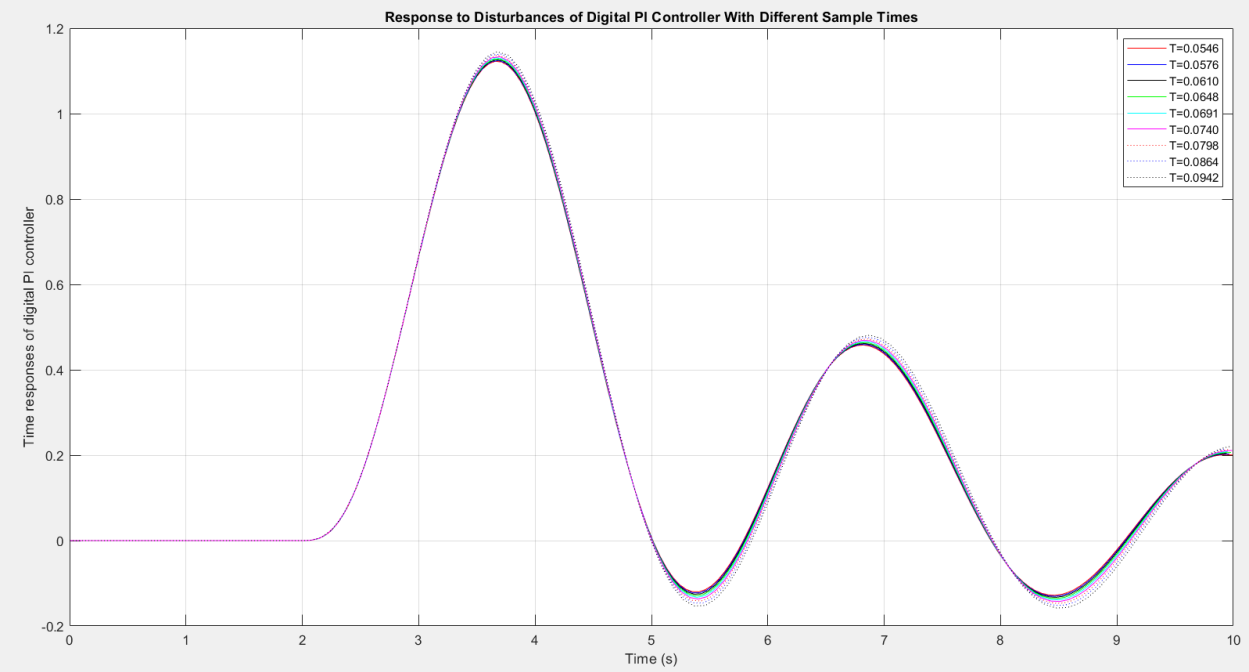
Graph showing sample times affecting PID controller reaction to disturbance



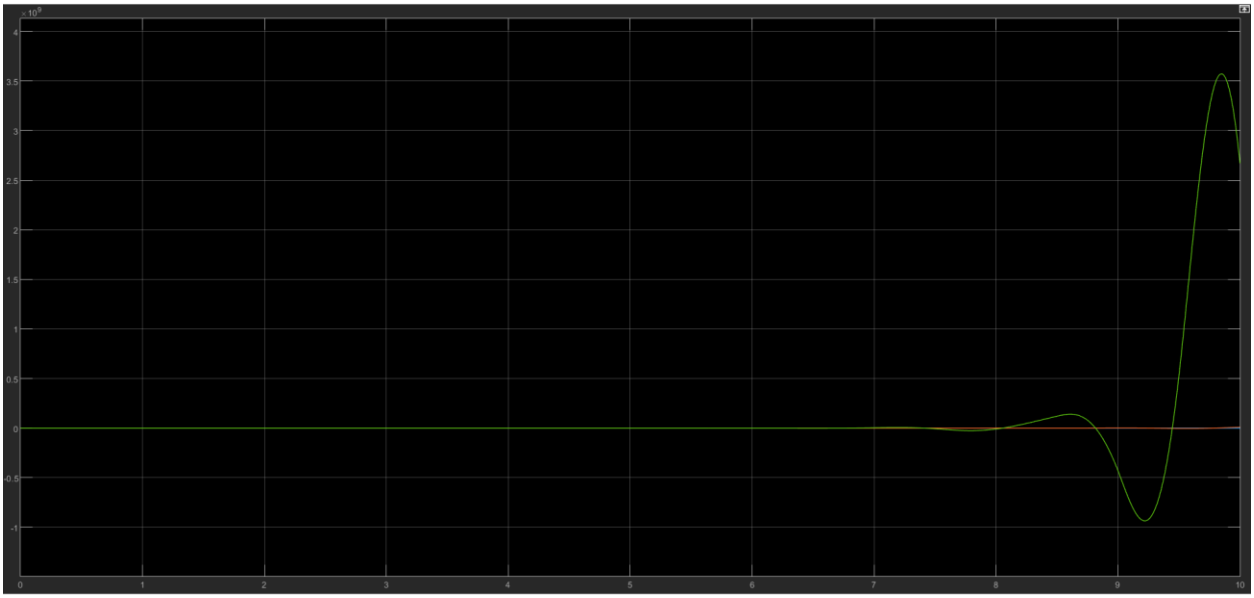
MATLAB code to plot digital PI controller response to disturbance.

```
plot(t_d1, DT1_PI, 'r-', t_d2, DT2_PI, 'b-', t_d3, DT3_PI,
'k-', t_d4, DT4_PI, 'g-', t_d5, DT5_PI, 'c-', t_d6, DT6_PI,
'm-', t_d7, DT7_PI, 'r:', t_d8, DT8_PI, 'b:', t_d9, DT9_PI,
'k:');
grid;
xlabel('Time (s)');
ylabel('Time responses of digital PI controller');
title('Response to Disturbances of Digital PI Controller
With Different Sample Times');
```

Graph showing sample times affecting PI controller reaction to disturbance

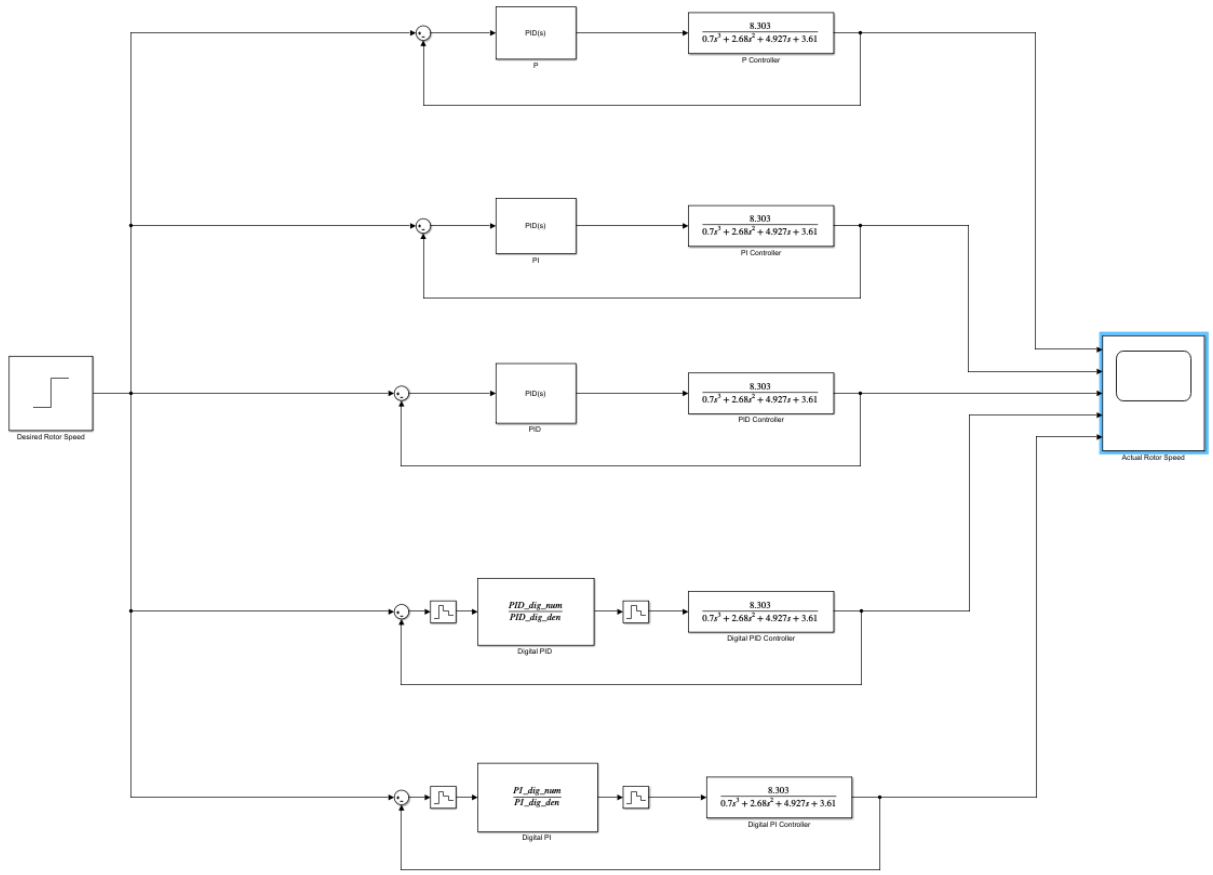


Issues from using T=0.01, 0.05, 0.1, 0.2, 0.5:

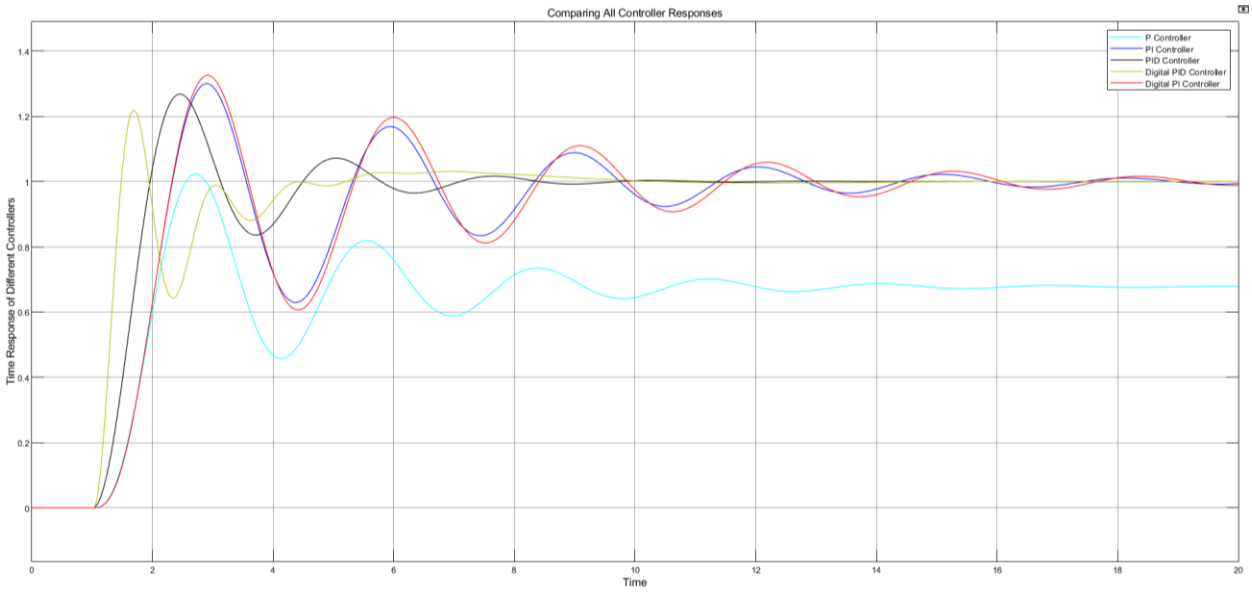


Incorrect range of values!

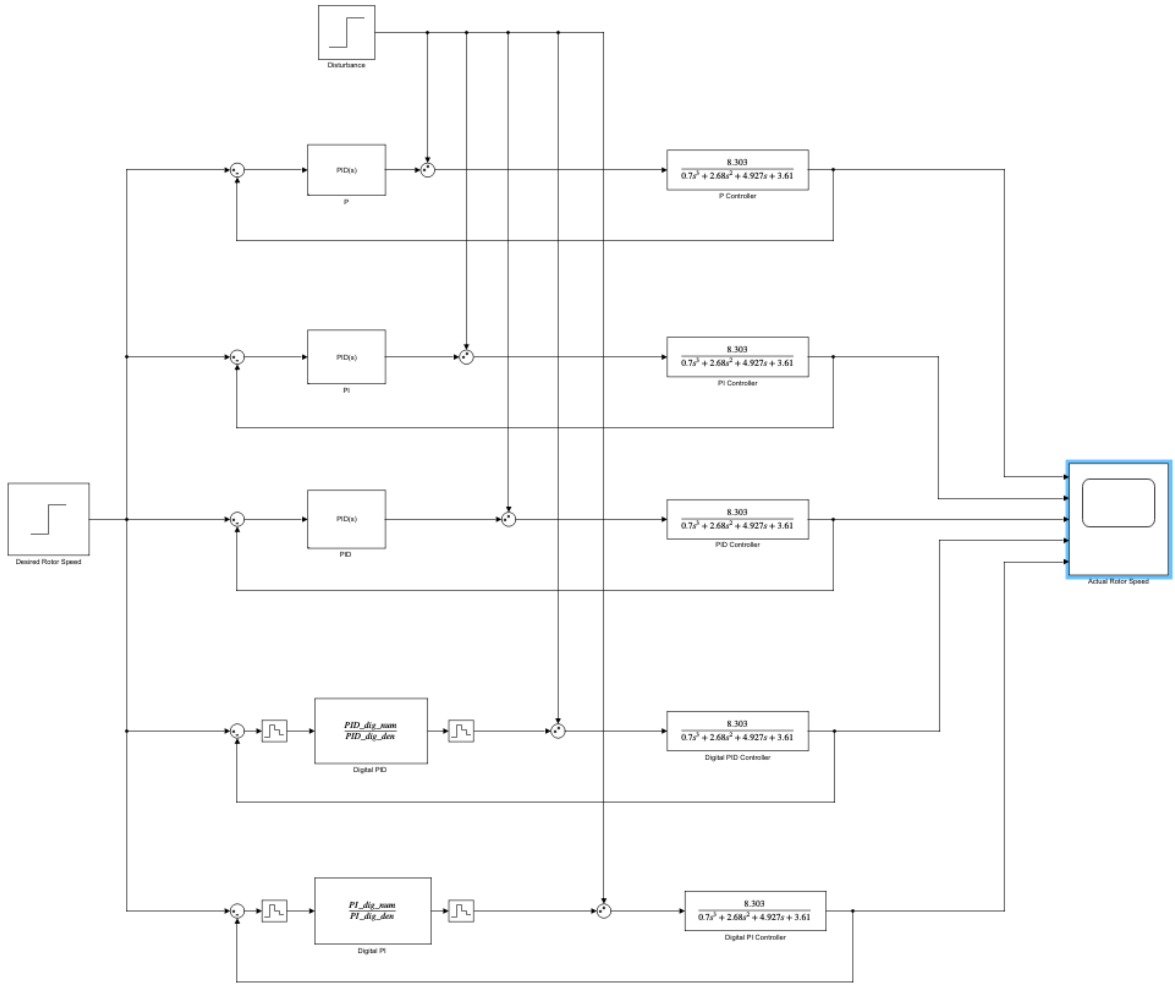
Block Diagram to compare all Controllers



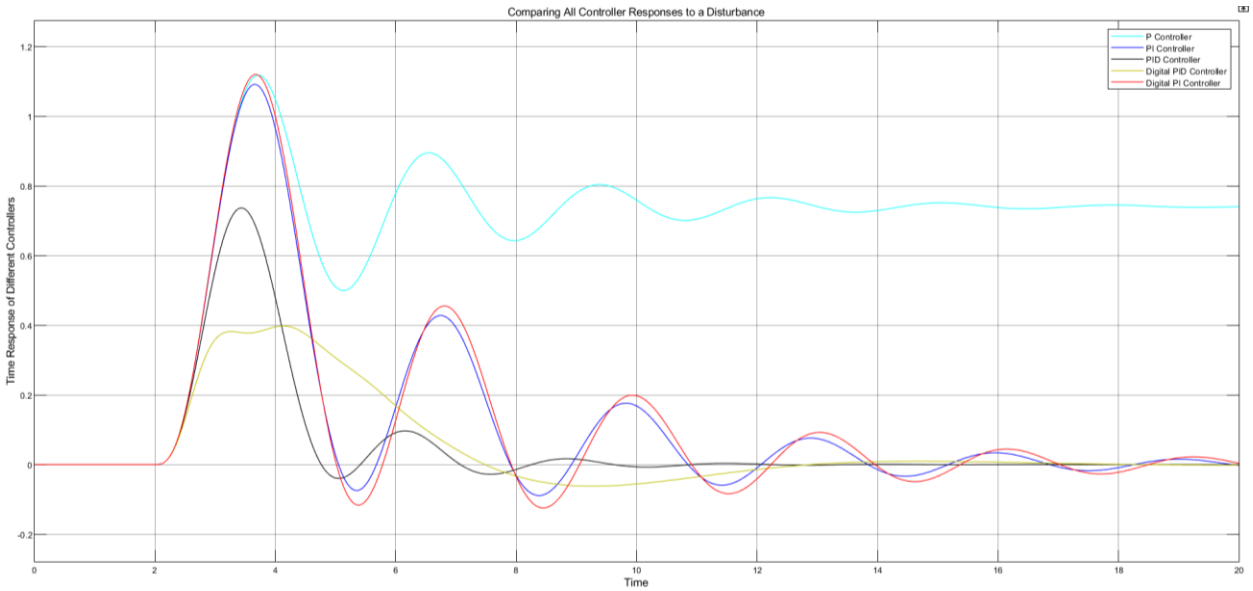
Graph Showing all Controller Responses



Block Diagram to compare all Controller Responses to disturbance



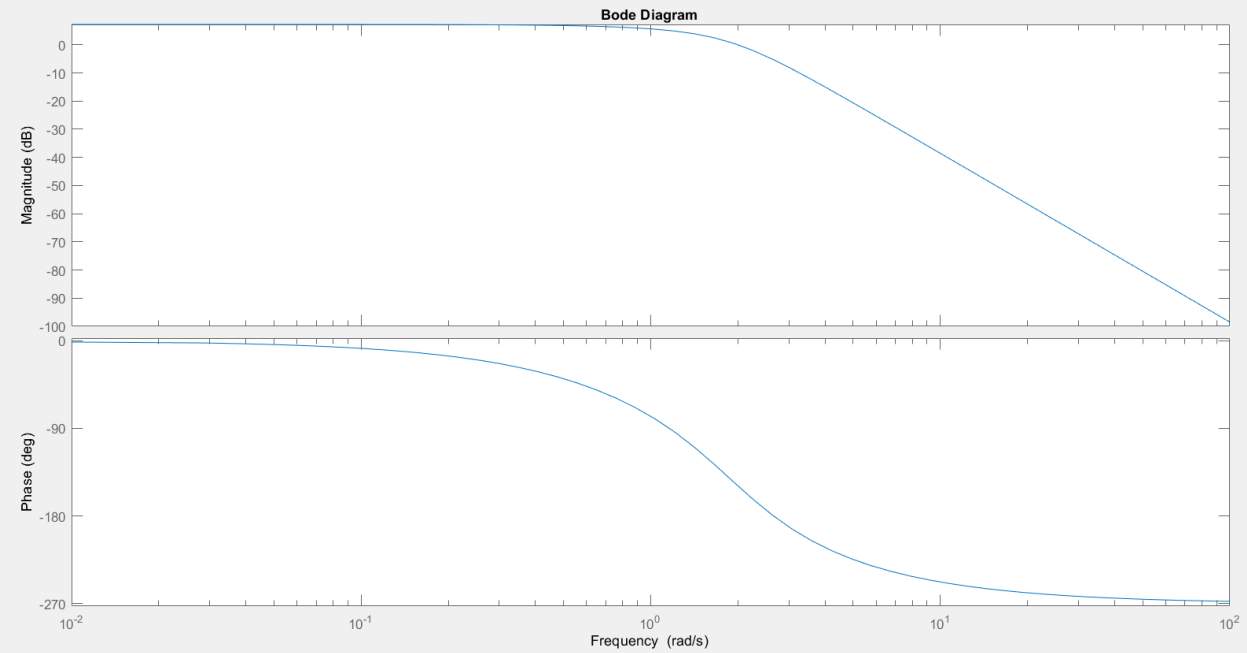
Graph showing responses to disturbance



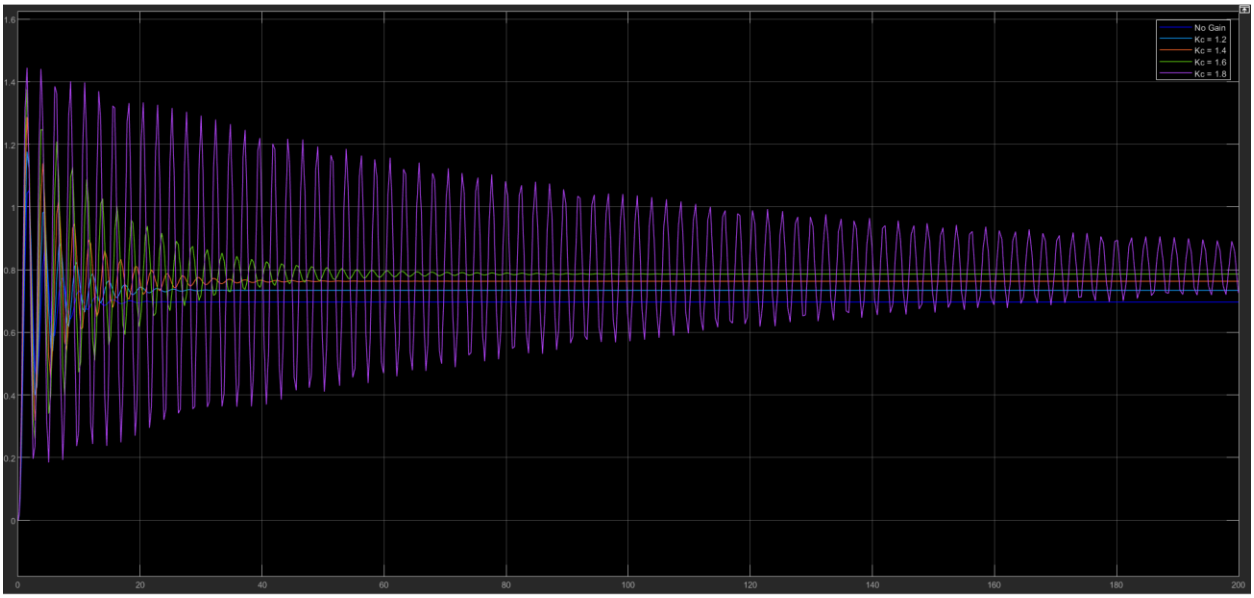
Friday, 8th January 2021

For final report, must design phase lead compensator.

Bode Diagram for G_p (without compensator)



Finding K_c by testing output waveforms with different gain values:

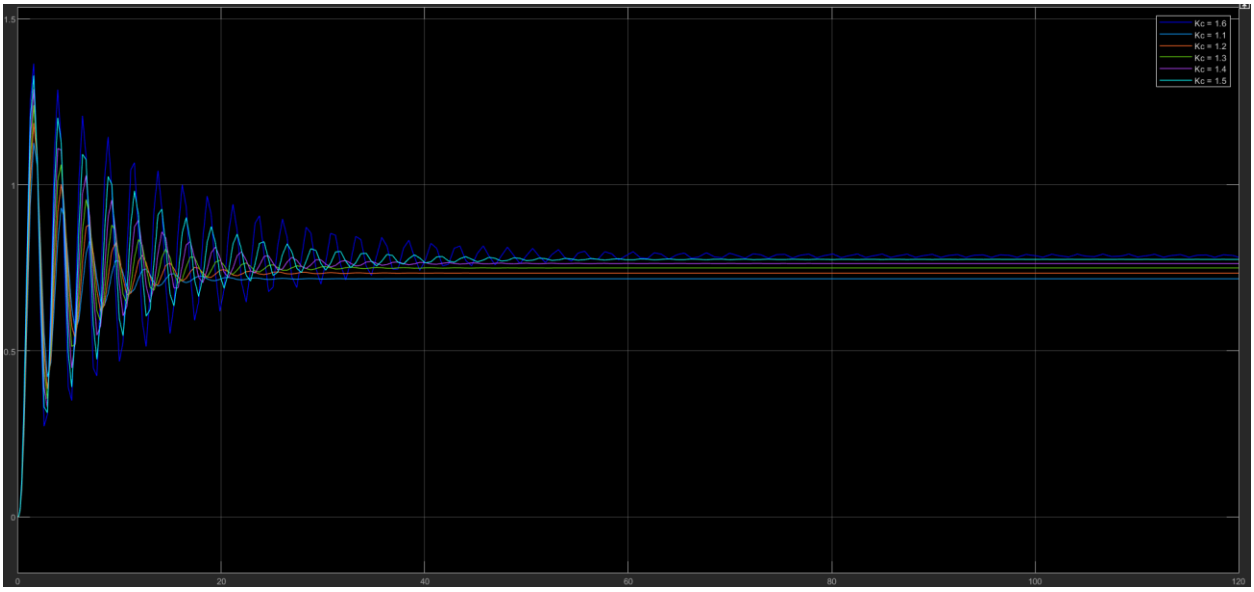


As gain increases:

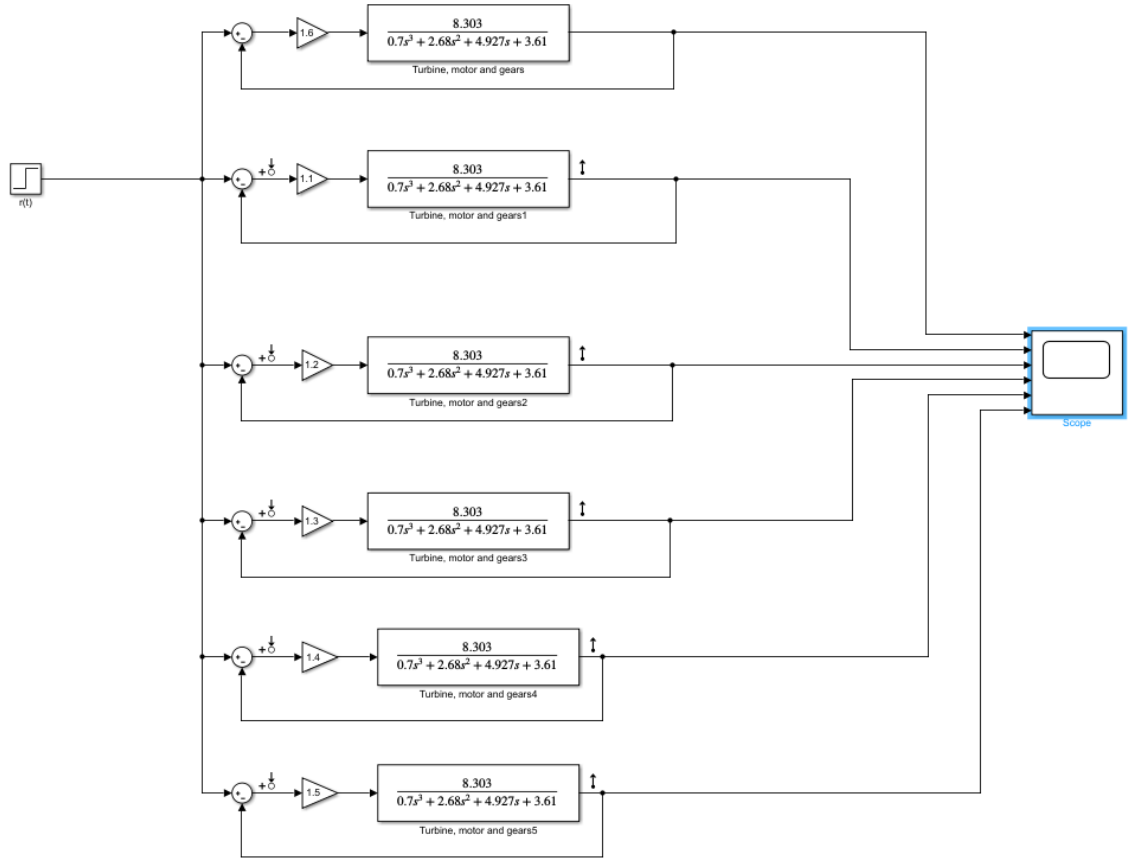
- Settling time increases
- Steady state error improves
- Rising time decreases

Settling time too large for Kc > 1.6

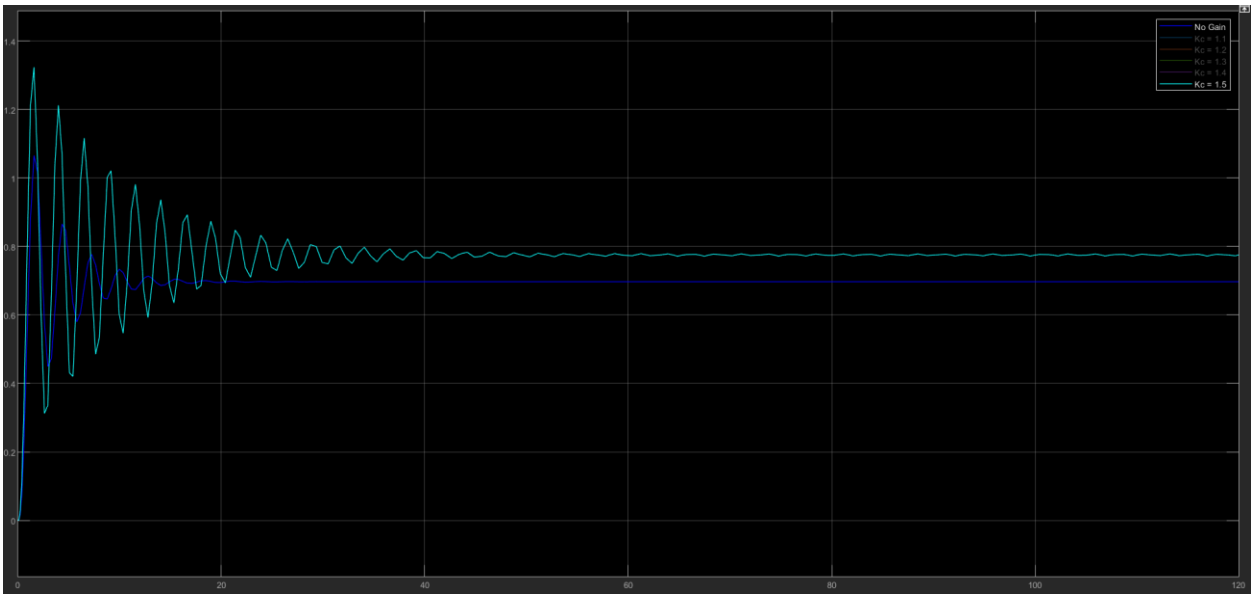
Try 1 < K < 1.7:



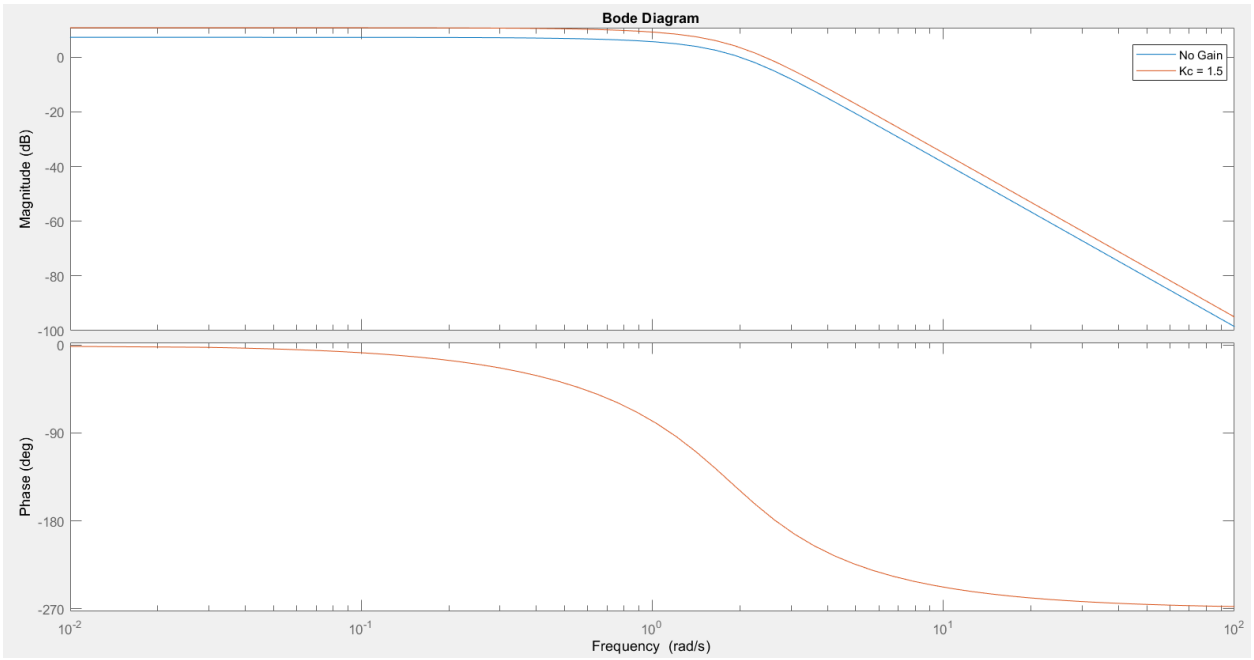
Using this block diagram on Simulink:



1.5 is an ideal value for Kc as it improves the steady state error by decreasing the gap between the input value and the value when the system reaches a steady state. It does this while not letting the settling time to be too large with the system settling within 60 seconds, compared to around 20 without the gain added. If Kc > 1.5, the settling time is over 100 seconds which is far too large for this type of system.



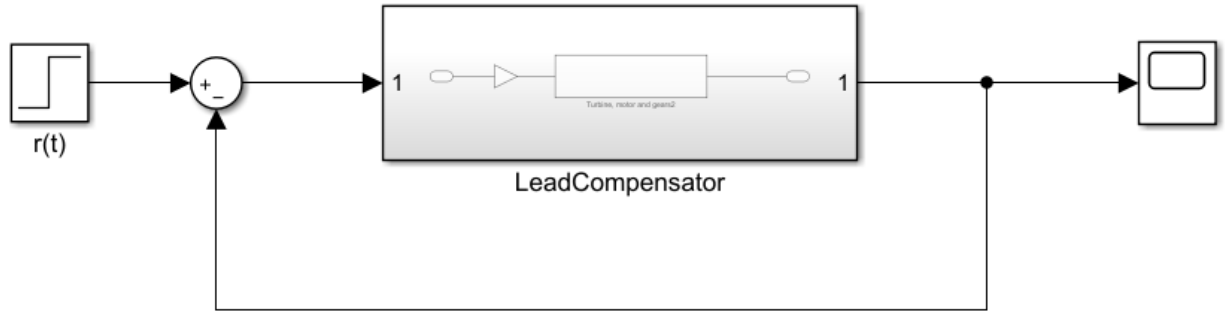
This value for Kc improves the steady state error while also increasing the gain on the Bode plot:



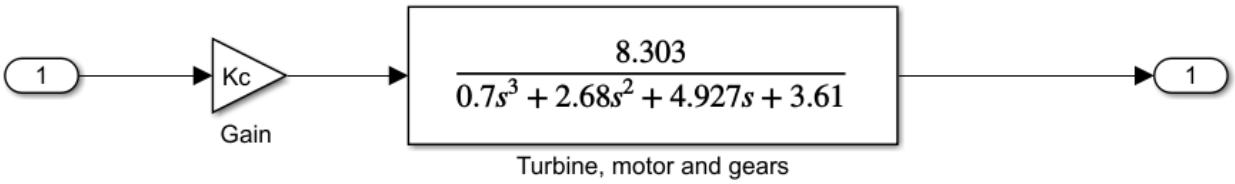
Using the function:

```
margin((linearize('Lead_Compensator_Test','Lead_Compensator_Test/LeadCompensator')))
```

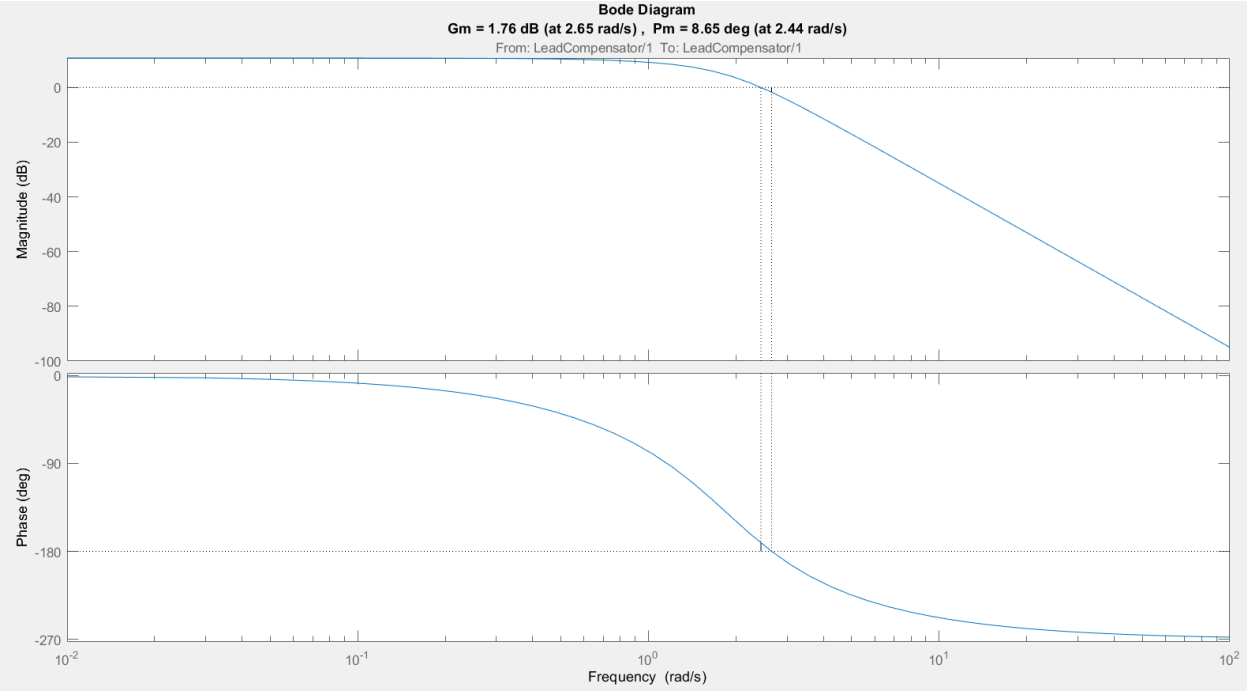
We can find the gain and phase margin of the system automatically by calling the slx file (file with block diagram) and the subsystem within that file containing the lead compensator.



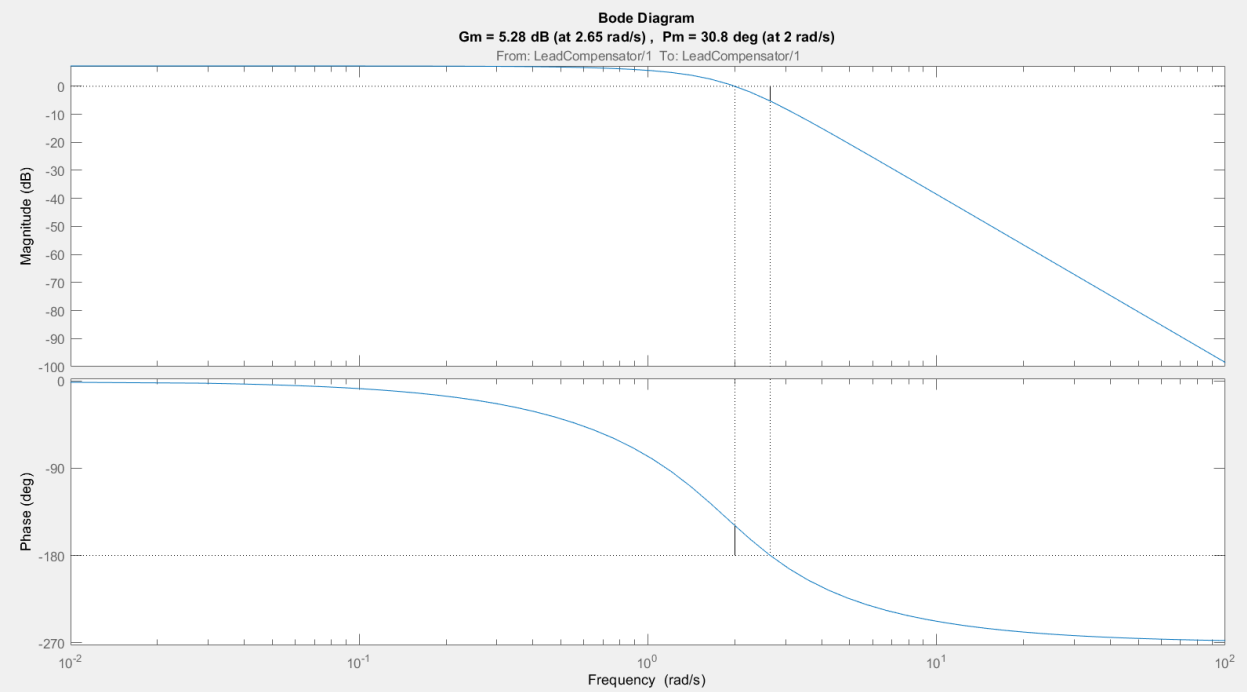
The subsystem consists of:



By entering the line of code above, the following figure opens, showing the gain and phase margin.



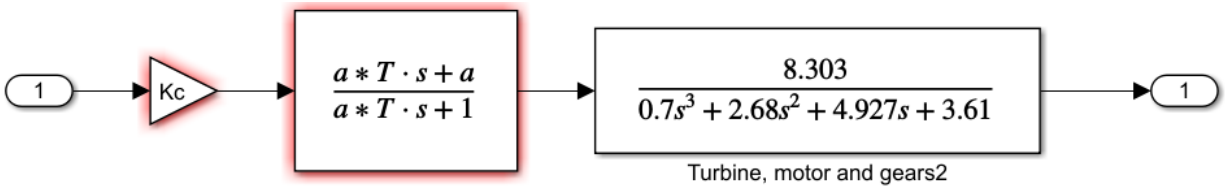
As seen, a gain of 1.5 added to the system gives a gain margin of 1.76 dB and a phase margin of 8.65 deg. This is opposed to a gain margin of 5.28 dB and a phase margin of 30.8 deg. As expected, the increase in gain, decreased the stability margin.



The dynamic part of the compensator should be designed so that the lost stability margin is recovered.

$$G_c(s) = K_c \frac{\alpha(Ts+1)}{\alpha Ts+1}$$

- Where K_c = 1.5
- Find α and T values to improve stability margins.
- 0.05 < α < 1



Since we know a (alpha) must be between 0.05 and 1, we can see how alpha will affect the stability margins while keeping T constant. When a = 1, T = 1, Kc =1.5, the bode plot will be the same as that with just the gain. We can calculate the value of a that is needed for the particular phase margin we would like. In this design, we need the phase margin to be ideally 65 deg but no less than 45 deg.

Using the equation:

$$sin\varphi_m = \frac{1 - \alpha}{1 + \alpha}$$

65 deg phase margin: a = 0.049

45 deg phase margin: a = 0.171

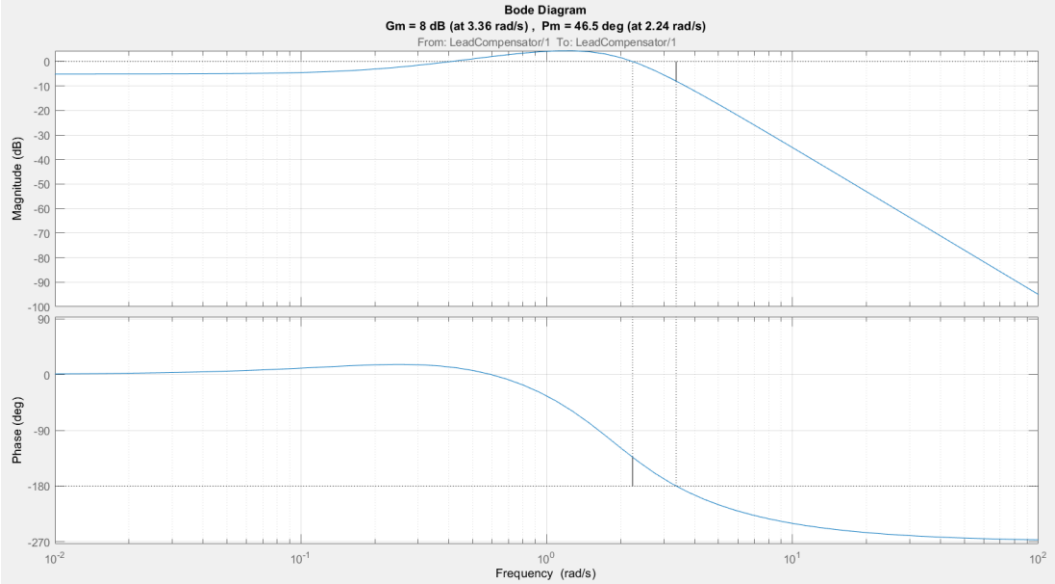
Therefore, the value of a should be between 0.049 and 0.171 for the phase margin to be in the correct region.

Saturday, 9th January 2021

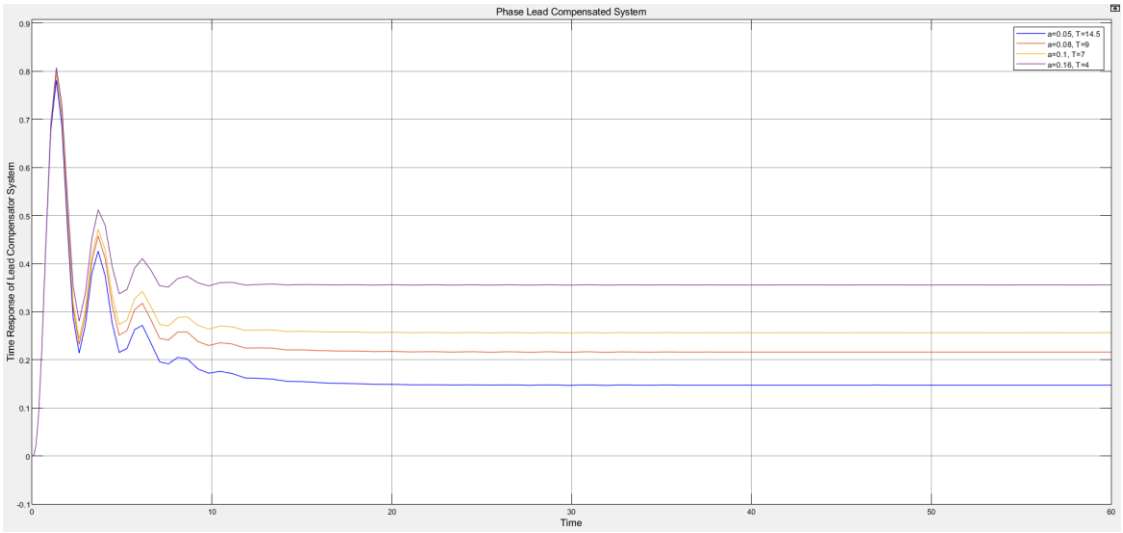
Must find value of a and T for lead compensator.

- T=14.5, a=0.05
- T=9, a=0.08
- T=7, a=0.1
- T=4.025, a=0.16

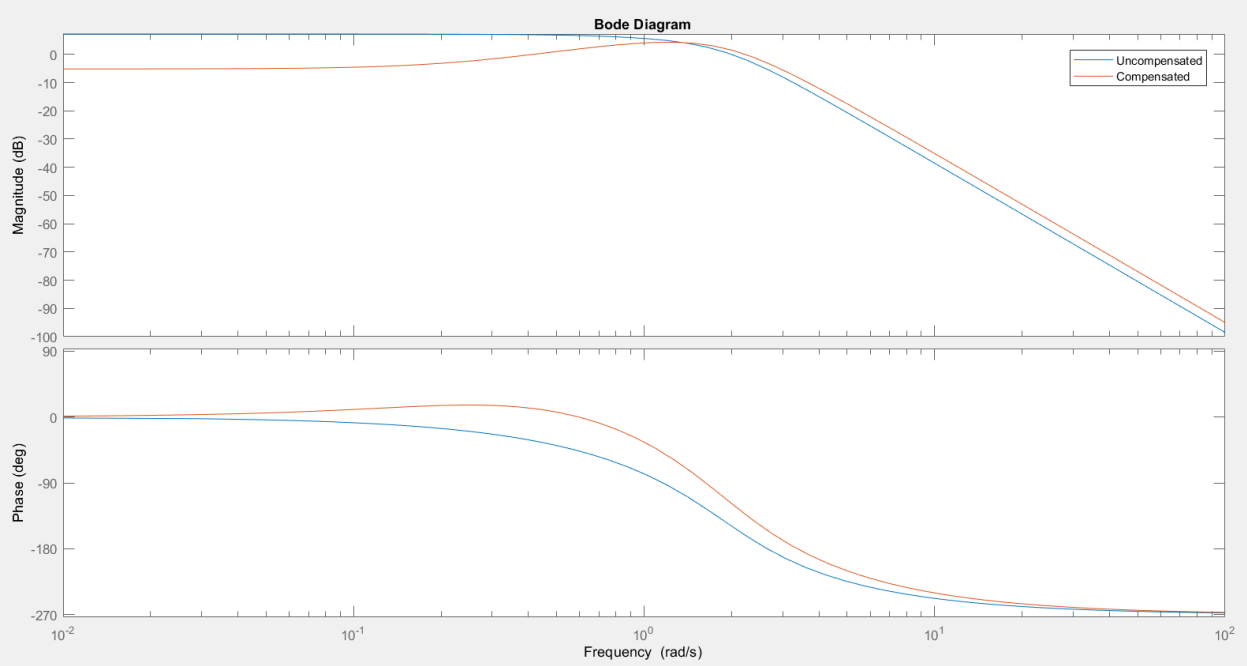
All of these values give the required stability margins, e.g. T=4.025, a=0.16:



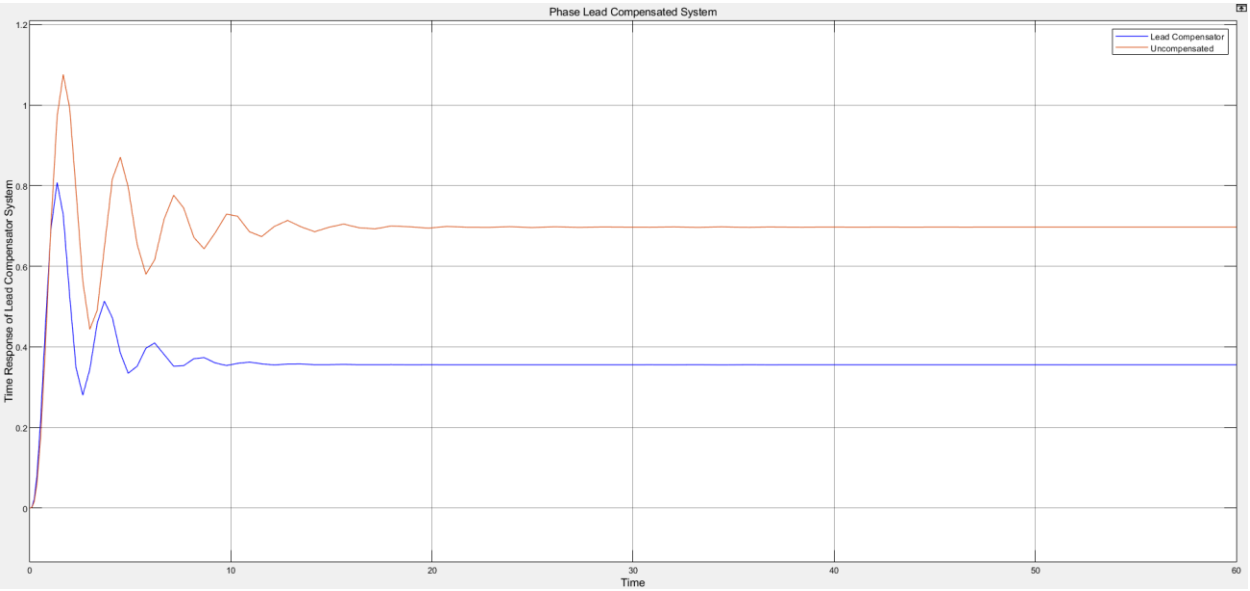
With these values, we have designed a lead compensator that meets the required stability margins of 8dB as the desired gain margin and a minimum of 45 deg for the minimum phase margin. Multiple combinations of a and T give required stability margins but let's see how these values affect time responses:



Bode plots of uncompensated and compensated system:



The Output Waveform for the entire system with and without the lead compensator:



- Final value is very low compared to desired input at 1.
- Settling time is faster
- Overshoot is lower

Tuesday, 12th January 2021

Design Phase Lag Compensator

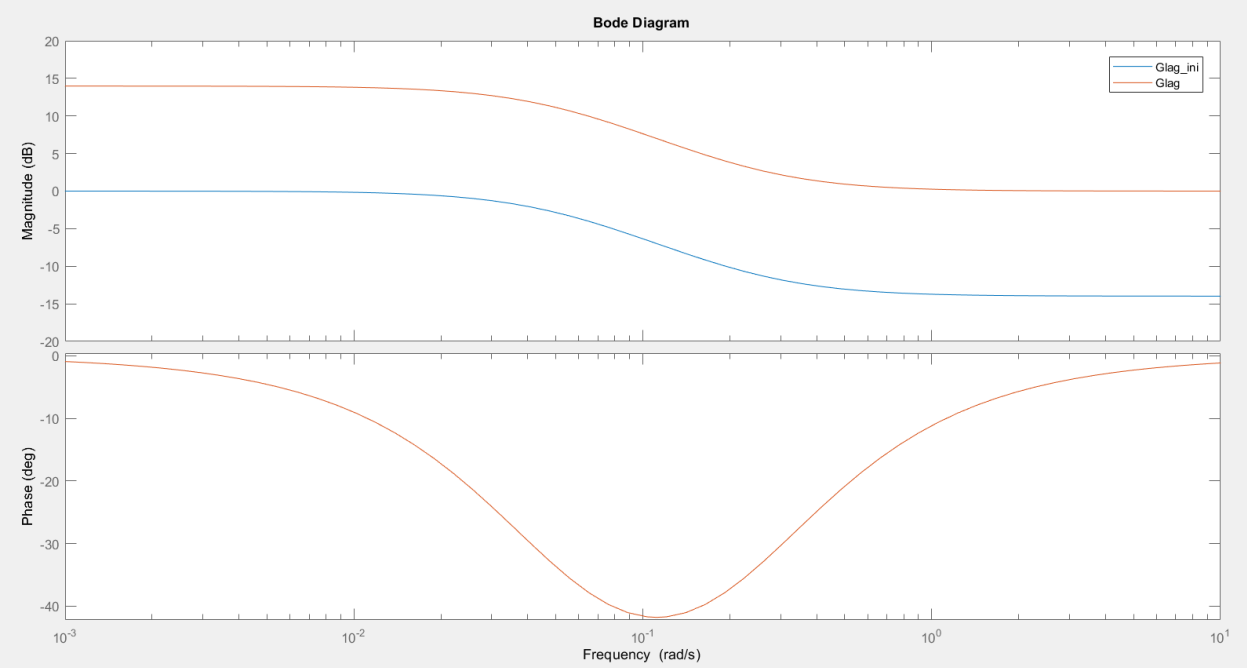
Add dynamic part first, then add gain.

$$G_{\text{lag}}(s) = K_c \frac{\text{b}(Ts+1)}{\text{b}Ts+1}$$

$$G_{\text{lag\_ini}}(s) = K_c \frac{Ts+1}{\text{b}Ts+1}$$

Set T=4, b =5

Find bode plot for $G_{\text{lag\_ini}}(s)$ and $\text{G}_{\text{lag}}(\text{s})$ and check to see if gain is reduced by 20log(b)



Gain is reduced by 20log(b).

Investigate how b and T values change gain and phase margin as well as time response.

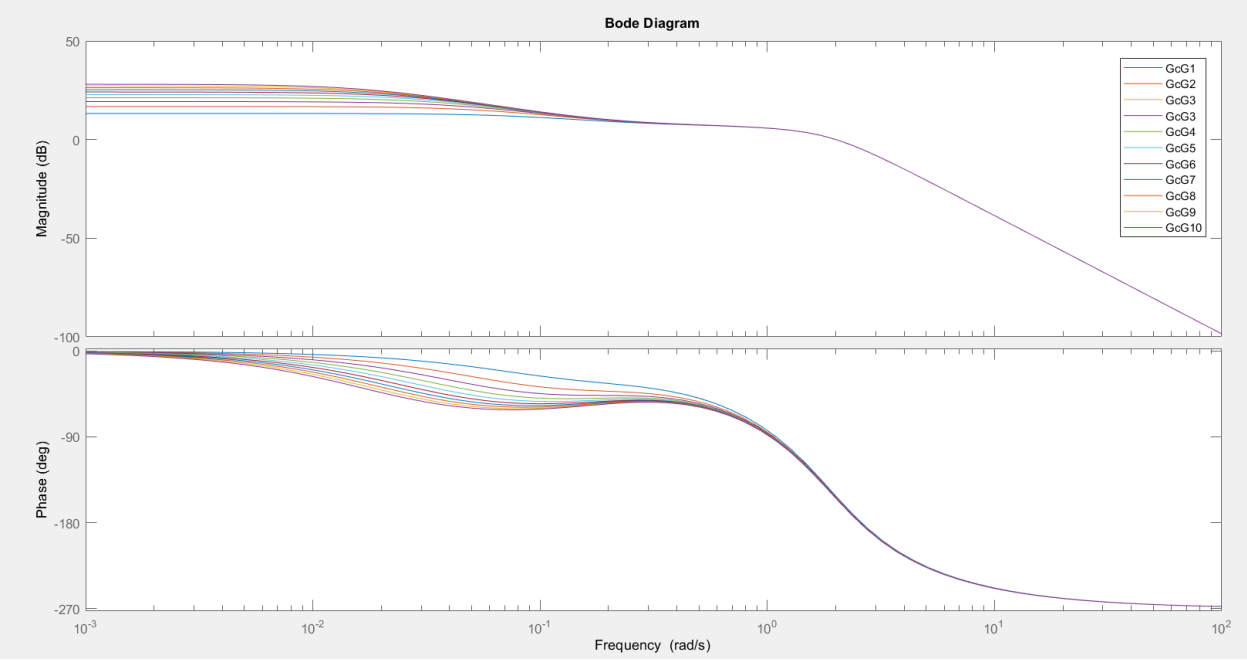
1 < b < 12

Using Script:

```
T=5;
b1=2;b2=3;b3=4;b4=5;b5=6;b6=7;b7=8;b8=9;b9=10;b10=11;

Glag1=(b1*T*s+b1)/(b1*T*s+1);
Glag2=(b2*T*s+b2)/(b2*T*s+1);
Glag3=(b3*T*s+b3)/(b3*T*s+1);
Glag4=(b4*T*s+b4)/(b4*T*s+1);
```

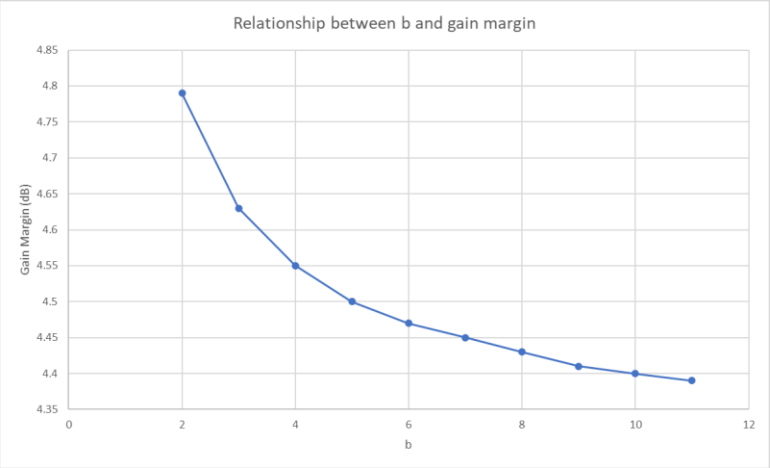
```
Glag5=(b5*T*s+b5)/(b5*T*s+1);
Glag6=(b6*T*s+b6)/(b6*T*s+1);
Glag7=(b7*T*s+b7)/(b7*T*s+1);
Glag8=(b8*T*s+b8)/(b8*T*s+1);
Glag9=(b9*T*s+b9)/(b9*T*s+1);
Glag10=(b10*T*s+b10)/(b10*T*s+1);

GcG1=Glag1*sys1;
GcG2=Glag2*sys1;
GcG3=Glag3*sys1;
GcG4=Glag4*sys1;
GcG5=Glag5*sys1;
GcG6=Glag6*sys1;
GcG7=Glag7*sys1;
GcG8=Glag8*sys1;
GcG9=Glag9*sys1;
GcG10=Glag10*sys1;

bode(GcG1,GcG2,GcG3,GcG3,GcG4,GcG5,GcG6,GcG7,GcG8,GcG9,GcG10)
```

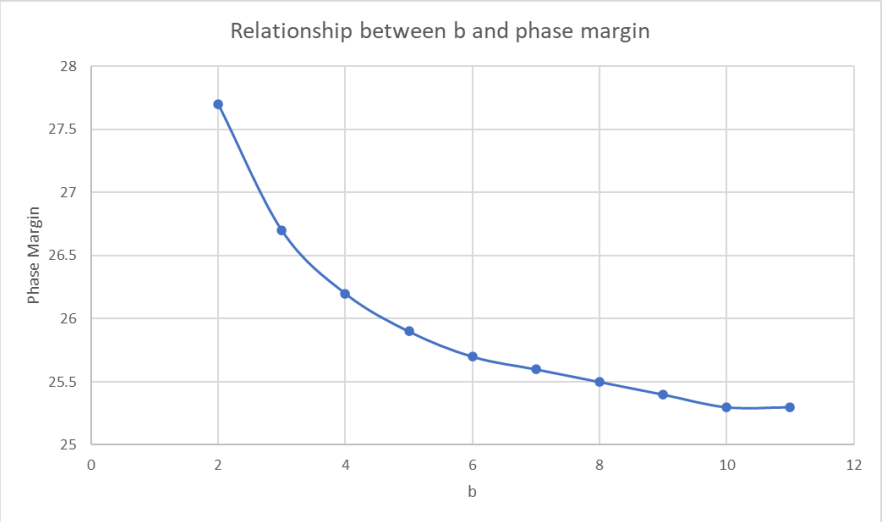
Gives following Bode plot



They all give different gain and phase margins too:

Value of b	Gain Margin (dB)	Phase Margin (deg)
2	4.79	27.7
3	4.63	26.7
4	4.55	26.2
5	4.5	25.9
6	4.47	25.7
7	4.45	25.6
8	4.43	25.5
9	4.41	25.4
10	4.4	25.3
11	4.39	25.3





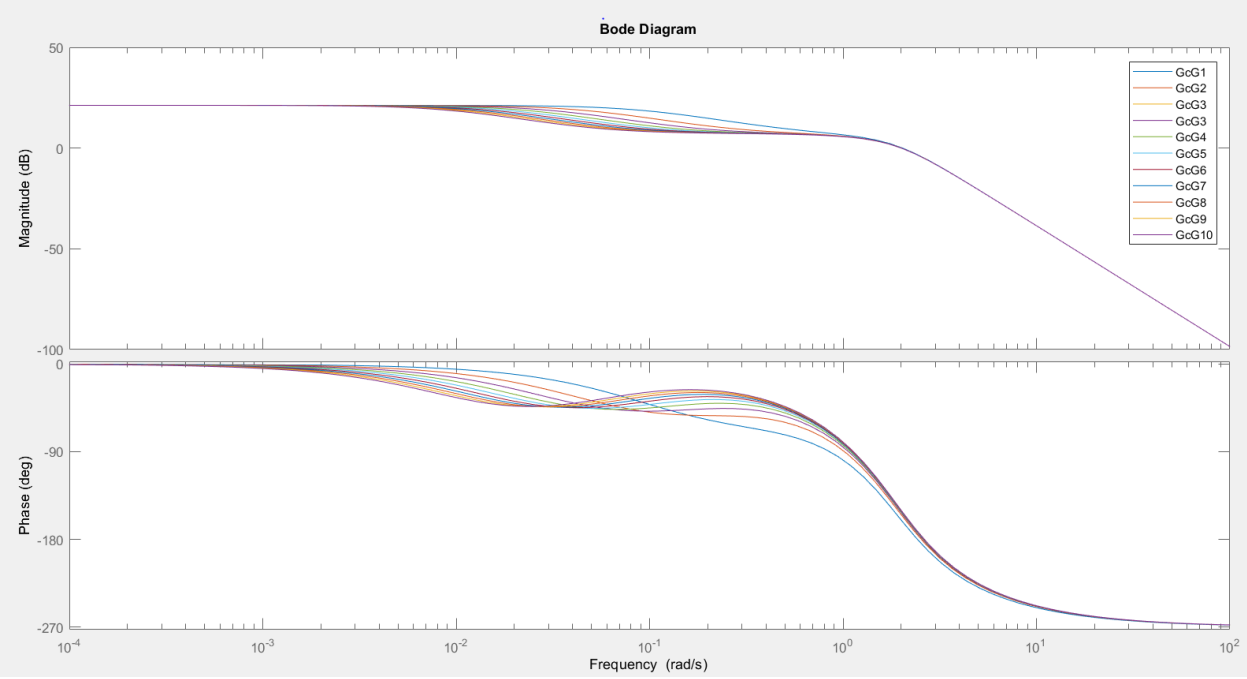
We can see that as b increases, the rate at which gain and phase margins decrease.

Let's test how T affects gain and phase margins.

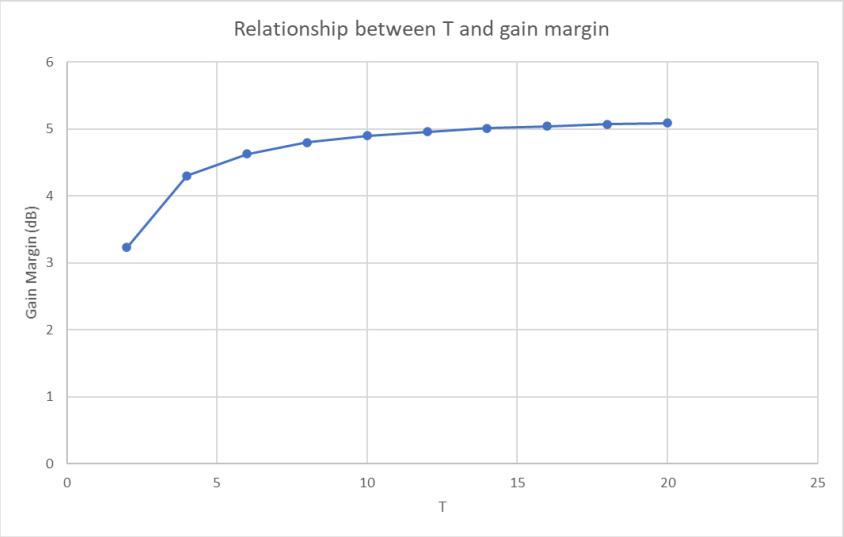
```
b=5;
T1=2;T2=4;T3=6;T4=8;T5=10;T6=12;T7=14;T8=16;T9=18;T10=20;

Glag1=(b*T1*s+b)/(b*T1*s+1);
Glag2=(b*T2*s+b)/(b*T2*s+1);
Glag3=(b*T3*s+b)/(b*T3*s+1);
Glag4=(b*T4*s+b)/(b*T4*s+1);
Glag5=(b*T5*s+b)/(b*T5*s+1);
Glag6=(b*T6*s+b)/(b*T6*s+1);
Glag7=(b*T7*s+b)/(b*T7*s+1);
```

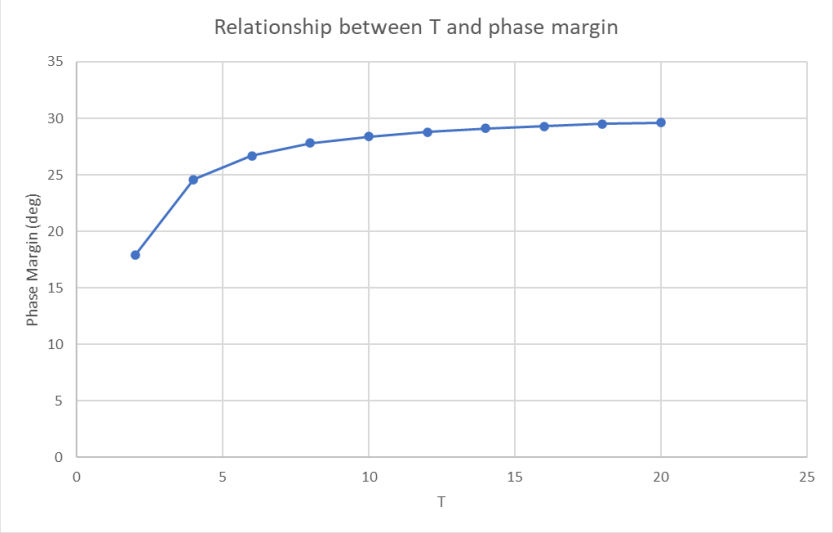
```
Glag8=(b*T8*s+b)/(b*T8*s+1);
Glag9=(b*T9*s+b)/(b*T9*s+1);
Glag10=(b*T10*s+b)/(b*T10*s+1);

GcG1=Glag1*sys1;
GcG2=Glag2*sys1;
GcG3=Glag3*sys1;
GcG4=Glag4*sys1;
GcG5=Glag5*sys1;
GcG6=Glag6*sys1;
GcG7=Glag7*sys1;
GcG8=Glag8*sys1;
GcG9=Glag9*sys1;
GcG10=Glag10*sys1;

bode(GcG1,GcG2,GcG3,GcG3,GcG4,GcG5,GcG6,GcG7,GcG8,GcG9,GcG1
0)
```

Gives the following Bode plot:

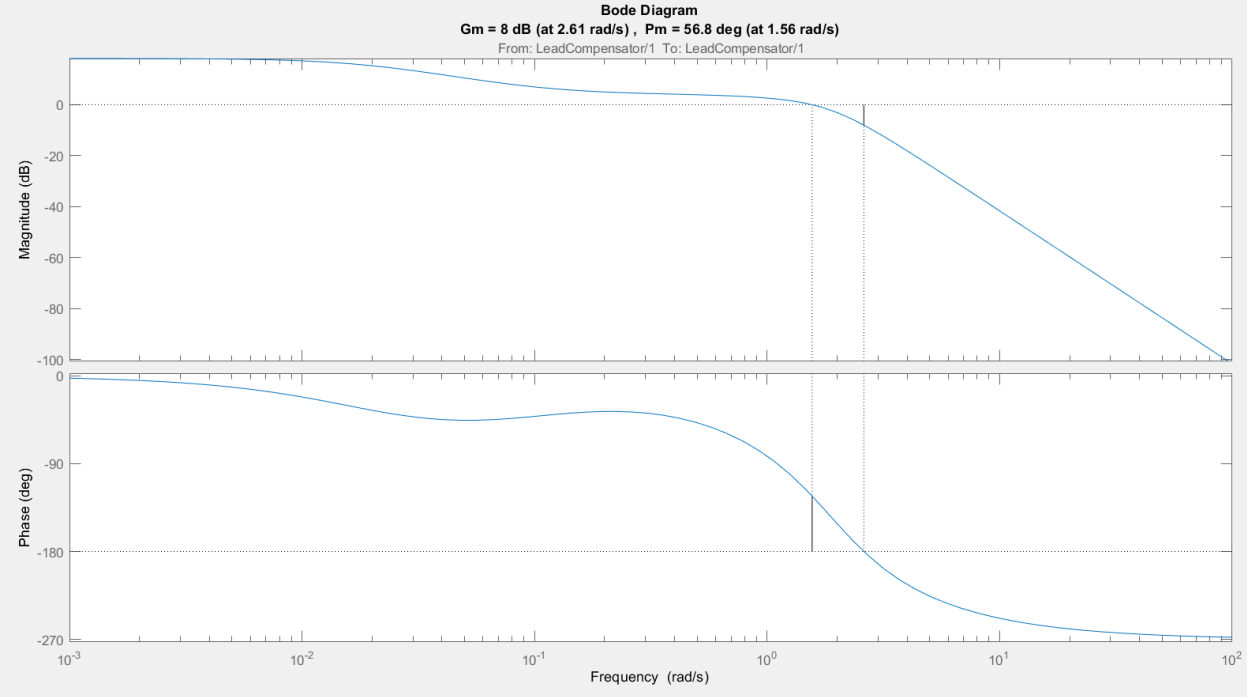


Value of T	Gain Margin (dB)	Phase Margin (deg)
2	3.23	17.9
4	4.3	24.6
6	4.63	26.7
8	4.8	27.8
10	4.9	28.4
12	4.96	28.8
14	5.01	29.1
16	5.04	29.3
18	5.07	29.5
20	5.09	29.6

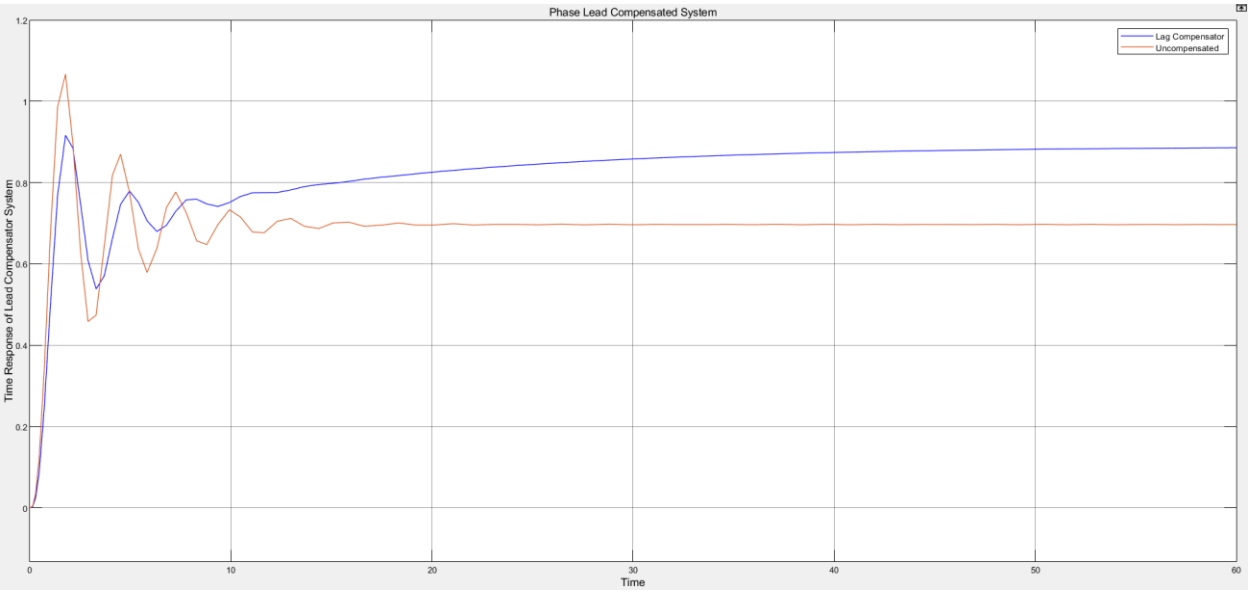




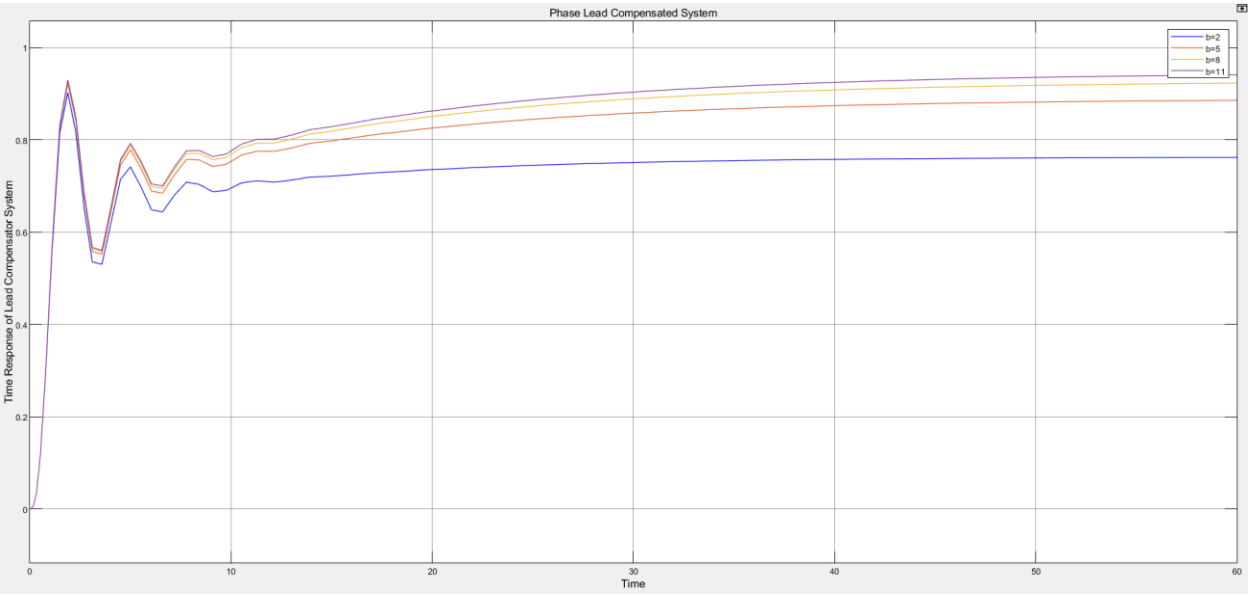
When Kc=0.7, b=5, T=10



Which gives this time response:



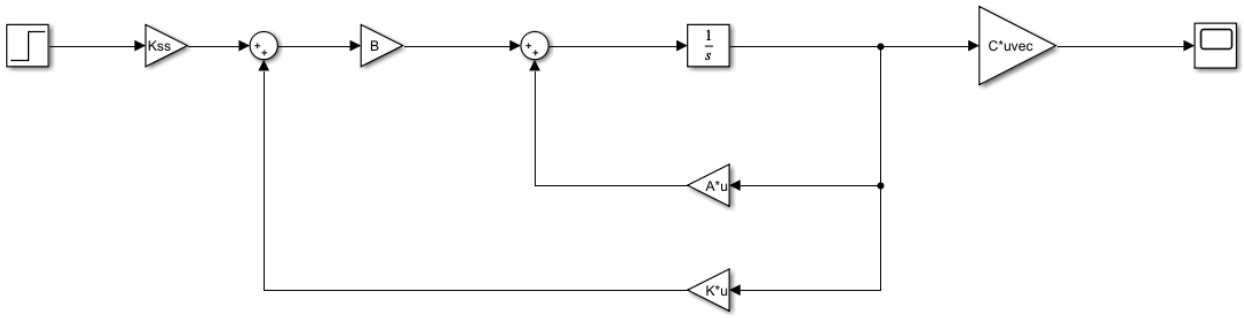
Investigate how b affects time response:



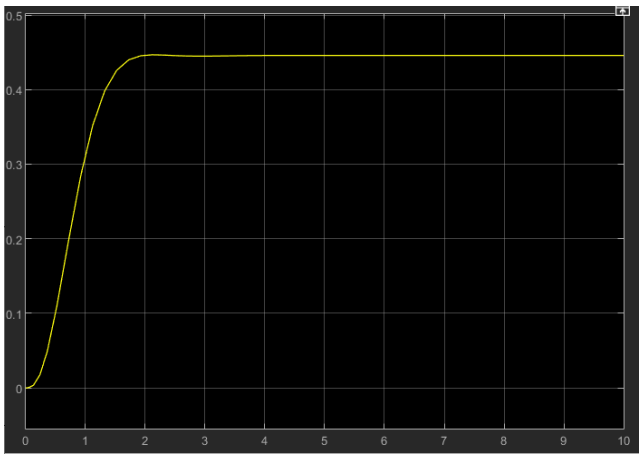
Thursday, 14th January 2021

MATLAB script showing process of state space design:

```
% define numerator and denominator for plant transfer
function
num=[8.303];
den=[0.7 2.68 4.927 3.61];
% find A,B,C,D values using converter
[A, B, C, D]=tf2ss(num,den);
% find open loop system poles
eig(A);
% define complex number i to be used for Cp
i=sqrt(-1);
% assign closed loop system poles Cp
Cp=[-2.2+2.4731i; -2.2-2.4731i; -2.4286+0i];
% obtain matrix K
K=place(A,B,Cp);
% check that routes for closed loop system are as desired
eig(A-B*K);
% adjust Kss to meet steady state error requirements
Kss=1;
```

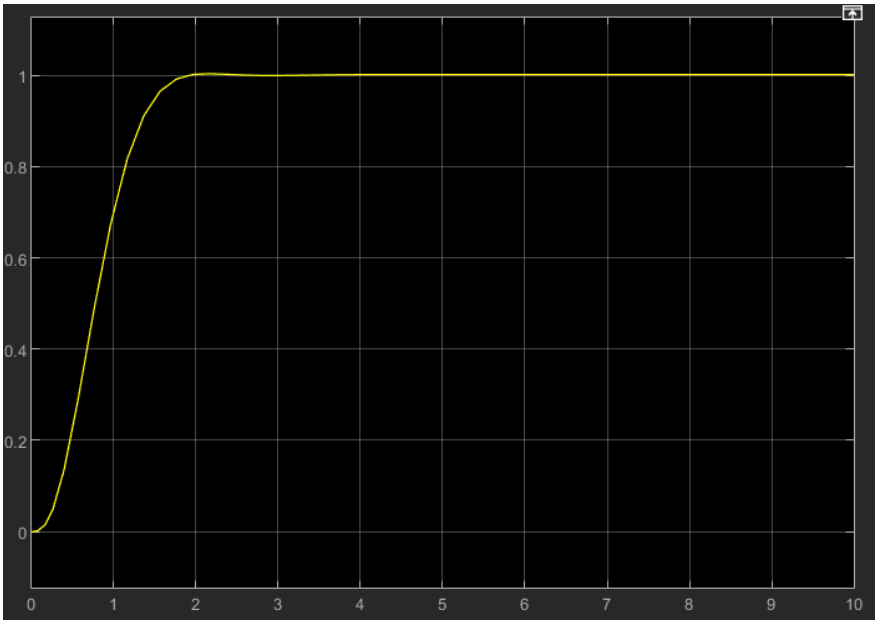


Time Response of that designed system:



- Time response of this design had a final value of 0.445.
- Can reduce steady state error by changing Kss value.
- 1/0.445 = 2.247

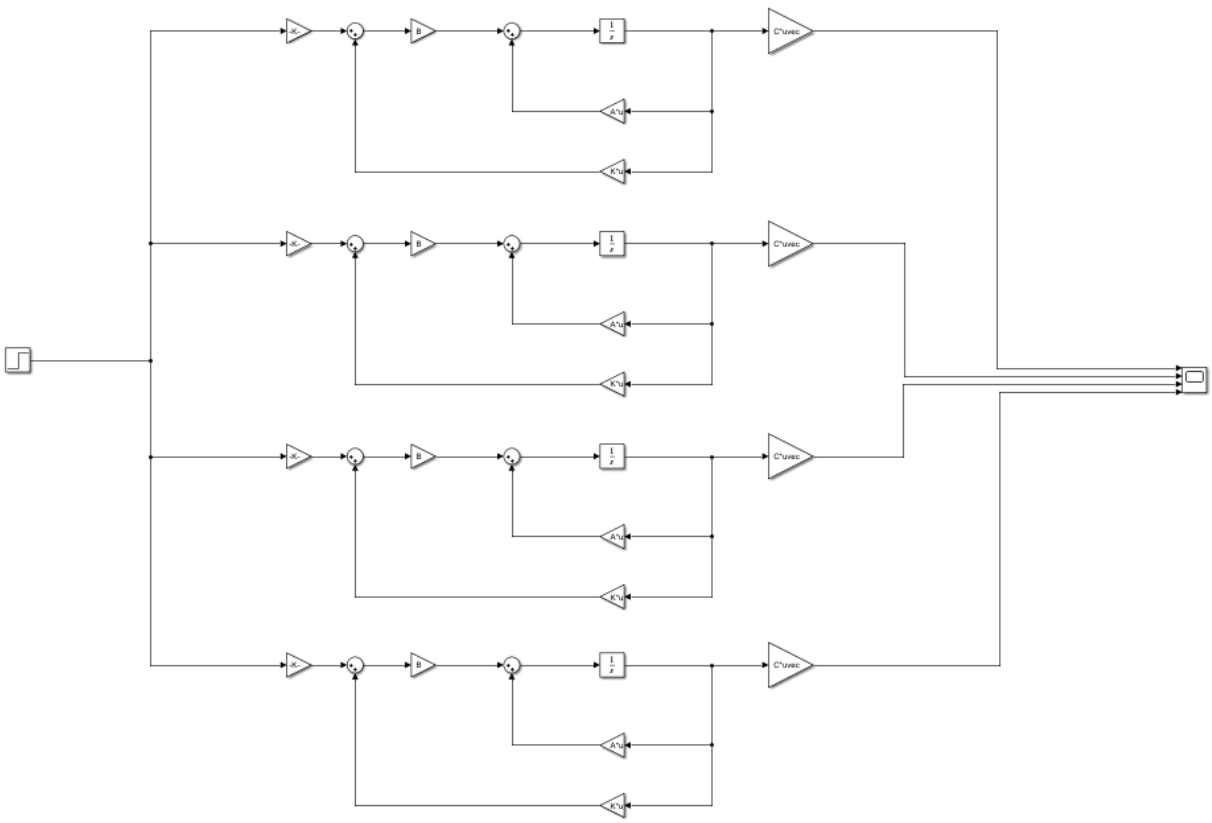
Time response of closed loop system:



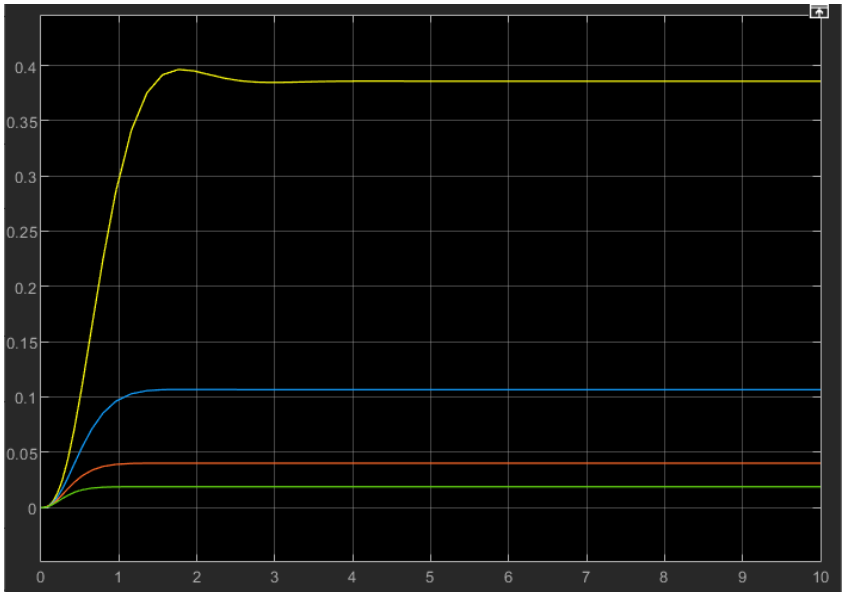
Investigate how pole placement effects time response by changing Cp matrix:

```
Cp1=[-2+2.5i; -2-2.5i; -3];
K1=place(A,B,Cp1);

Cp2=[-4+2.5i;-4-2.5i;-5];
K2=place(A,B,Cp2);

Cp3=[-6+2.5i;-6-2.5i;-7];
K3=place(A,B,Cp3);

Cp4=[-8+2.5i;-8-2.5i;-9];
K4=place(A,B,Cp4);
```

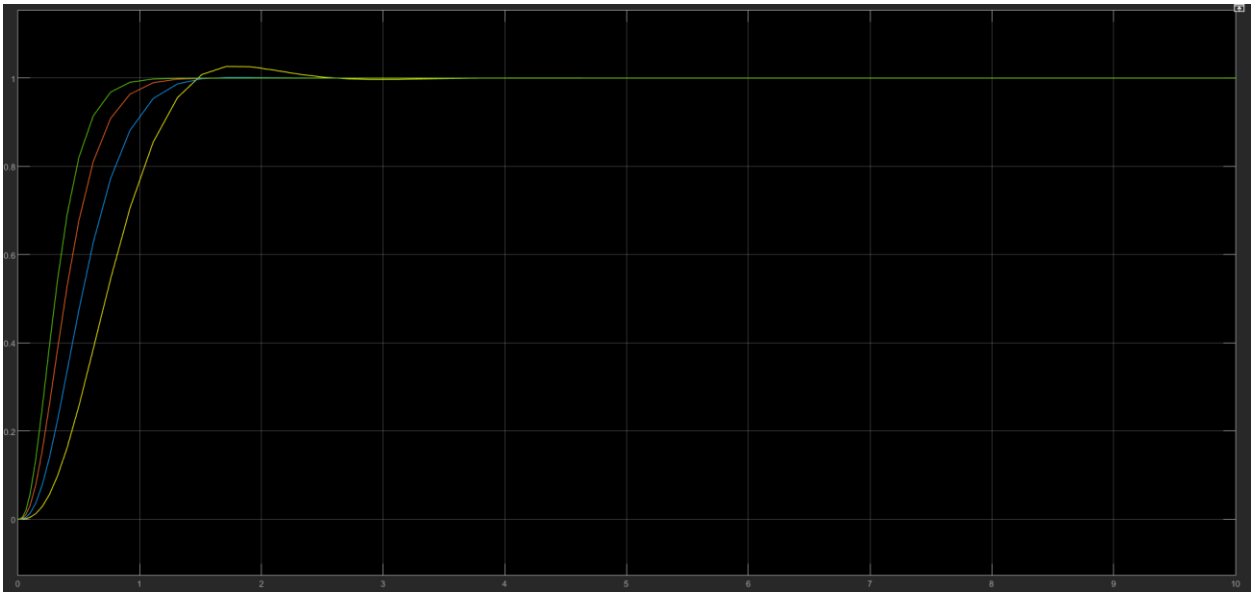
Using the following block diagram to test multiple state space designs at once:



Gives the following time response:



Adjusting Kss for each system so that each has zero steady state error:



Further you move the poles to the left, the faster the response but the higher the Kss gain needs to be. Therefore, there may not be enough speed in the actuators to generate the required response due to it taking more gain to move the poles further from their starting point.

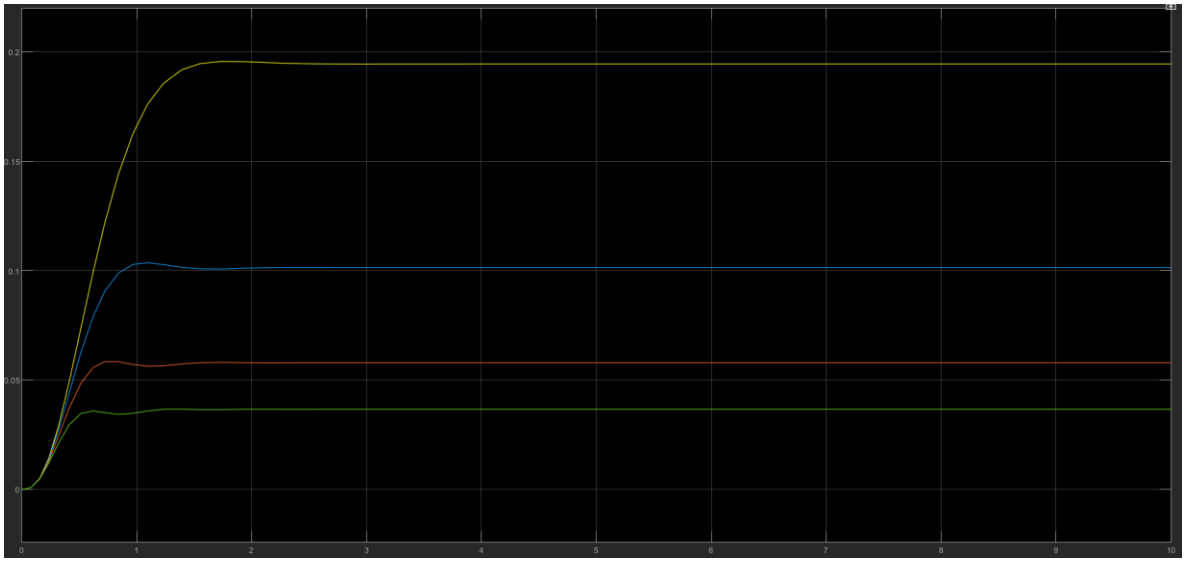
This is shown by the value of Kss changing for each system:

2.59, 9.38, 24.93, 53.3

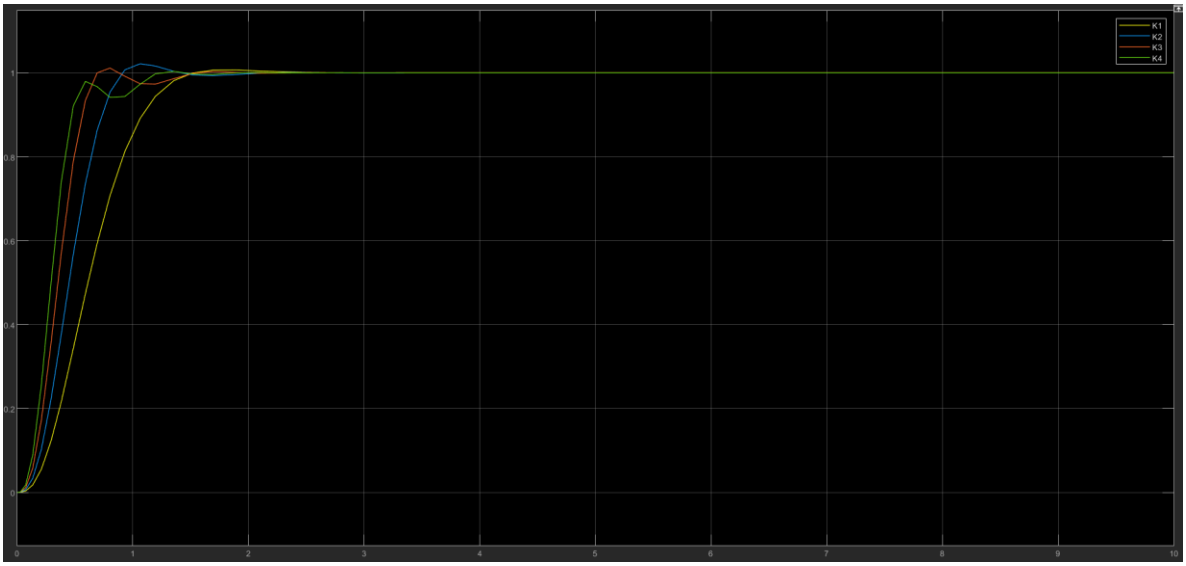
Moving the poles up or down also has an impact on the time response.

```
Cp1=[-3+2.5i;-3-2.5i; -4];
K1=place(A,B,Cp1);
Cp2=[-3+4.5i;-3-4.5i;-4];
K2=place(A,B,Cp2);
Cp3=[-3+6.5i;-3-6.5i;-4];
K3=place(A,B,Cp3);
Cp4=[-3+8.5i;-3-8.5i;-4];
K4=place(A,B,Cp4);
```

Gives this time response:



Adjusting Kss so each system has the correct steady state error



The value of Kss also changes for each system when the poles move up or down.

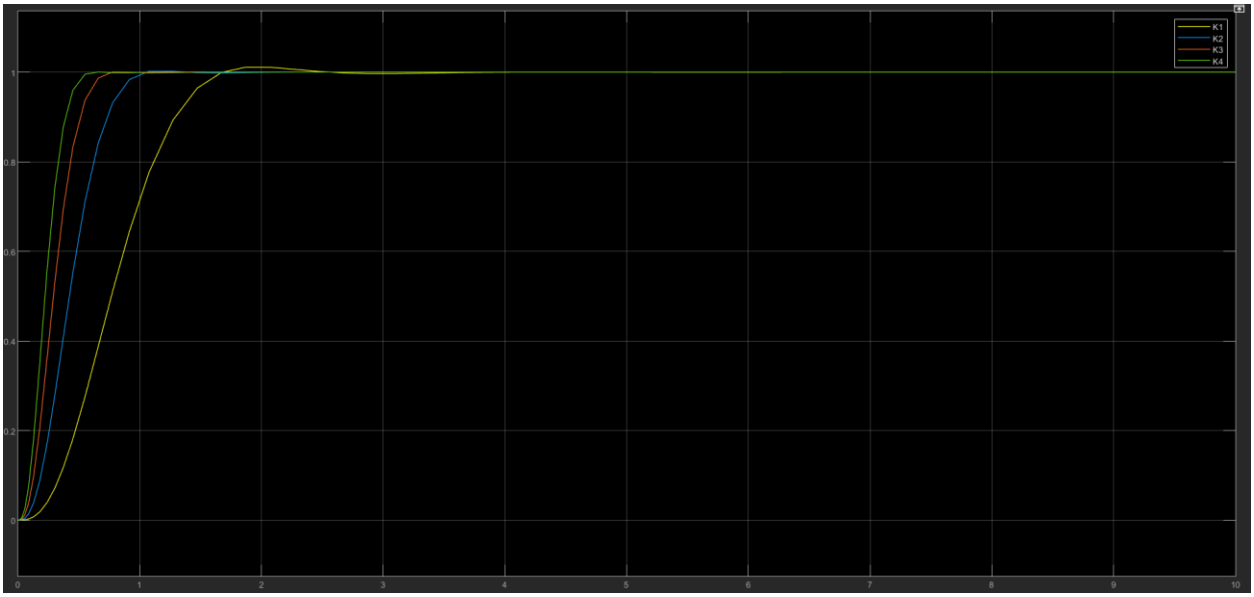
5.14, 9.86, 17.28, 27.4

The closer the poles are to the real axis, the slower the response, however, the higher gain is needed.

Testing how moving poles diagonally, effects time response

```
Cp1=[-2+2.5i;-2-2.5i; -2.5];
K1=place(A,B,Cp1);
Cp2=[-4+4.5i;-4-4.5i;-4.5];
K2=place(A,B,Cp2);
Cp3=[-6+6.5i;-6-6.5i;-6.5];
K3=place(A,B,Cp3);
Cp4=[-8+8.5i;-8-8.5i;-8.5];
K4=place(A,B,Cp4);
```

When Kss is adjusted for each design to meet steady state error requirement, we have the following time response:



Gains required are: 2.16, 13.76, 42.88, 97.66

Due to the fast response time, low amount of overshoot and low value of gain needed to reduce steady state error, I will choose the poles to be at [-4+4.5i, -4-4.5i, -4.5+0i].

As Zeta decreases, % overshoot increases

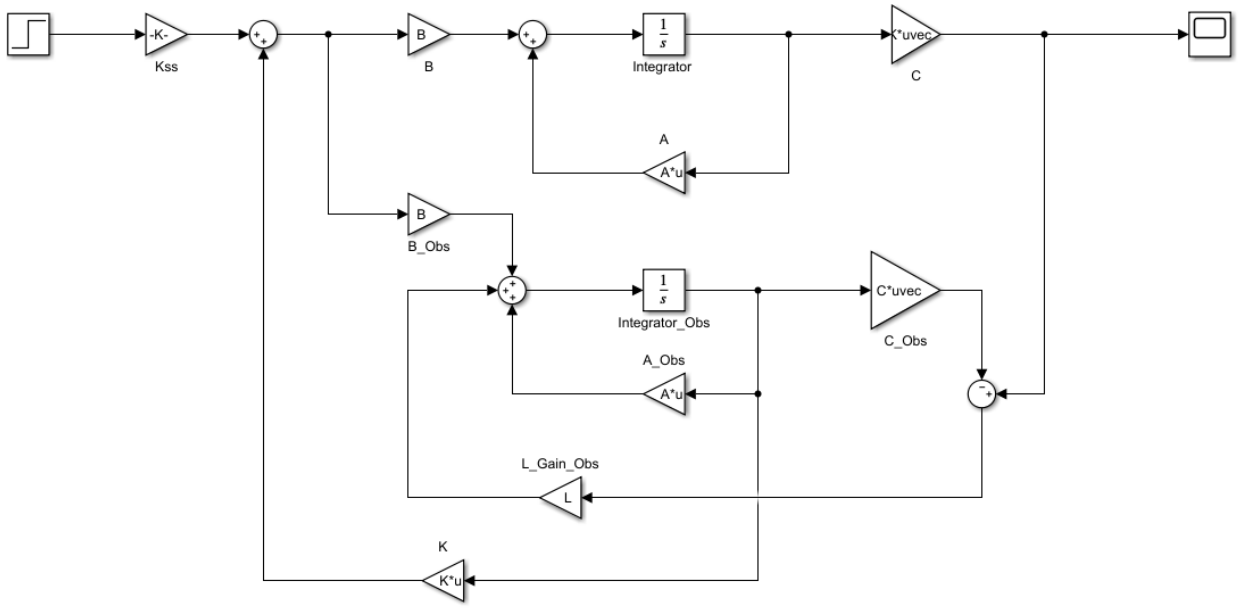
Friday, 15th January 2021

Observer Design

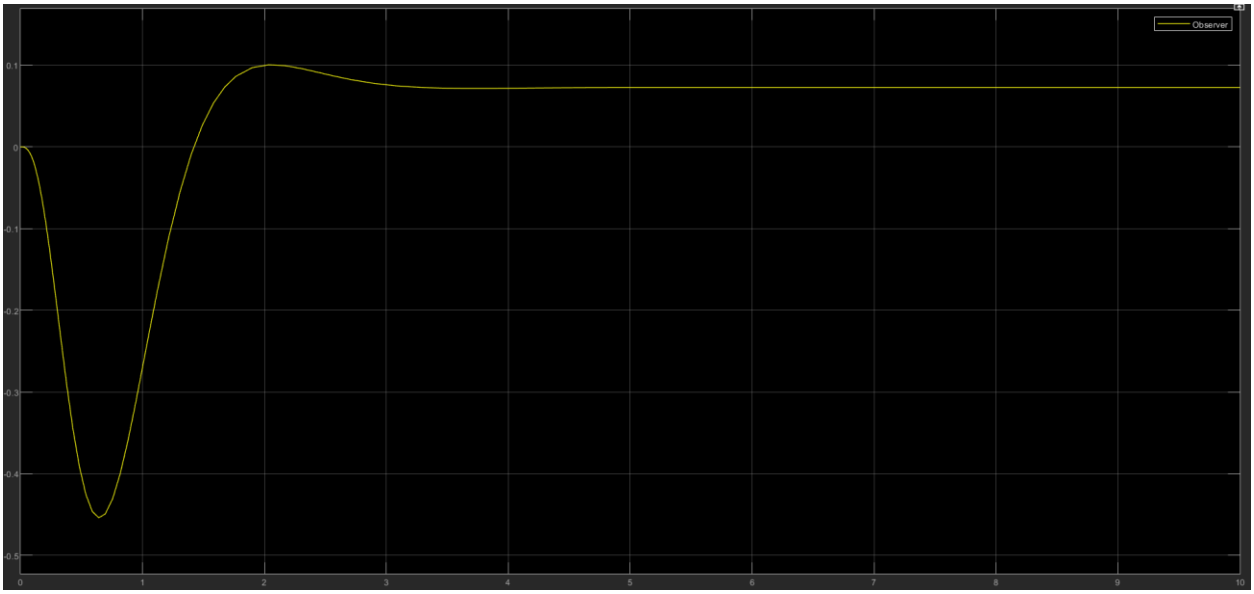
Matlab Script:

```
den=[0.7 2.68 4.927 3.61];
% find A,B,C,D values using converter
[A, B, C, D]=tf2ss(num,den);
% find open loop system poles
eig(A);
% define complex number i to be used for Cp
i=sqrt(-1);
% assign closed loop system poles Cp
Cp=[-4+4.5i; -4-4.5i; -4.5+0i];
% obtain matrix K
K=place(A,B,Cp);
% check that routes for closed loop system are as desired
eig(A-B*K);
% adjust Kss to meet steady state error requirements
Kss=1;
OBSp=[-9+6.5i; -9-6.5i; -12];
L=place(A,B,OBSp);
```

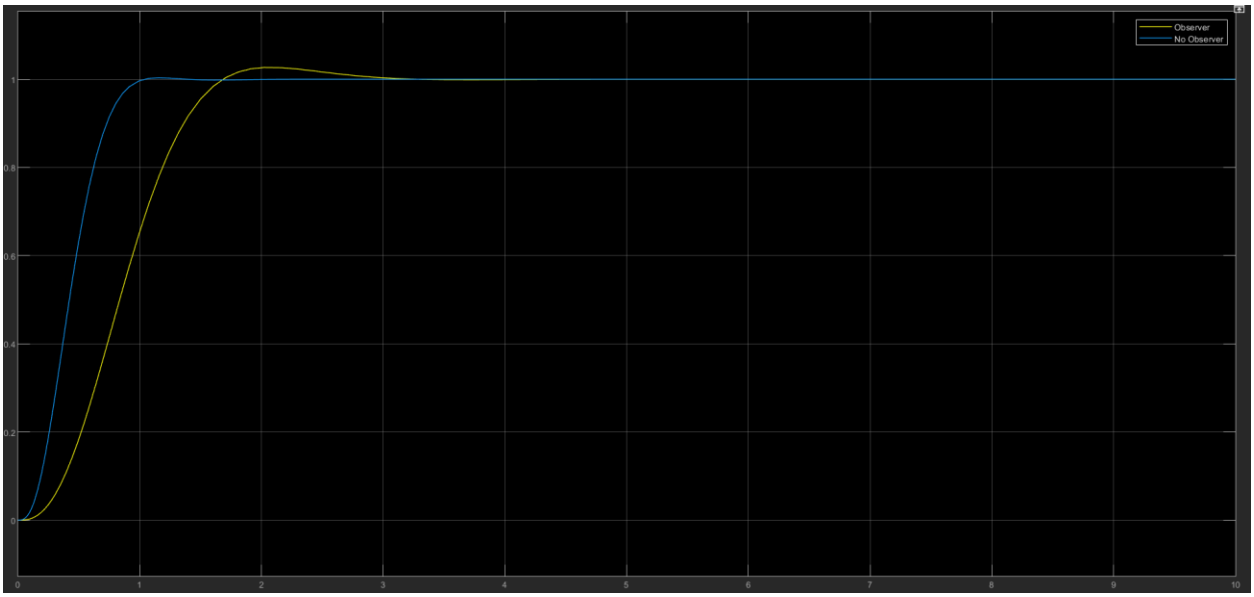
Block Diagram:



Gives following time response:



When Kss adjusted and compared to design without observer:

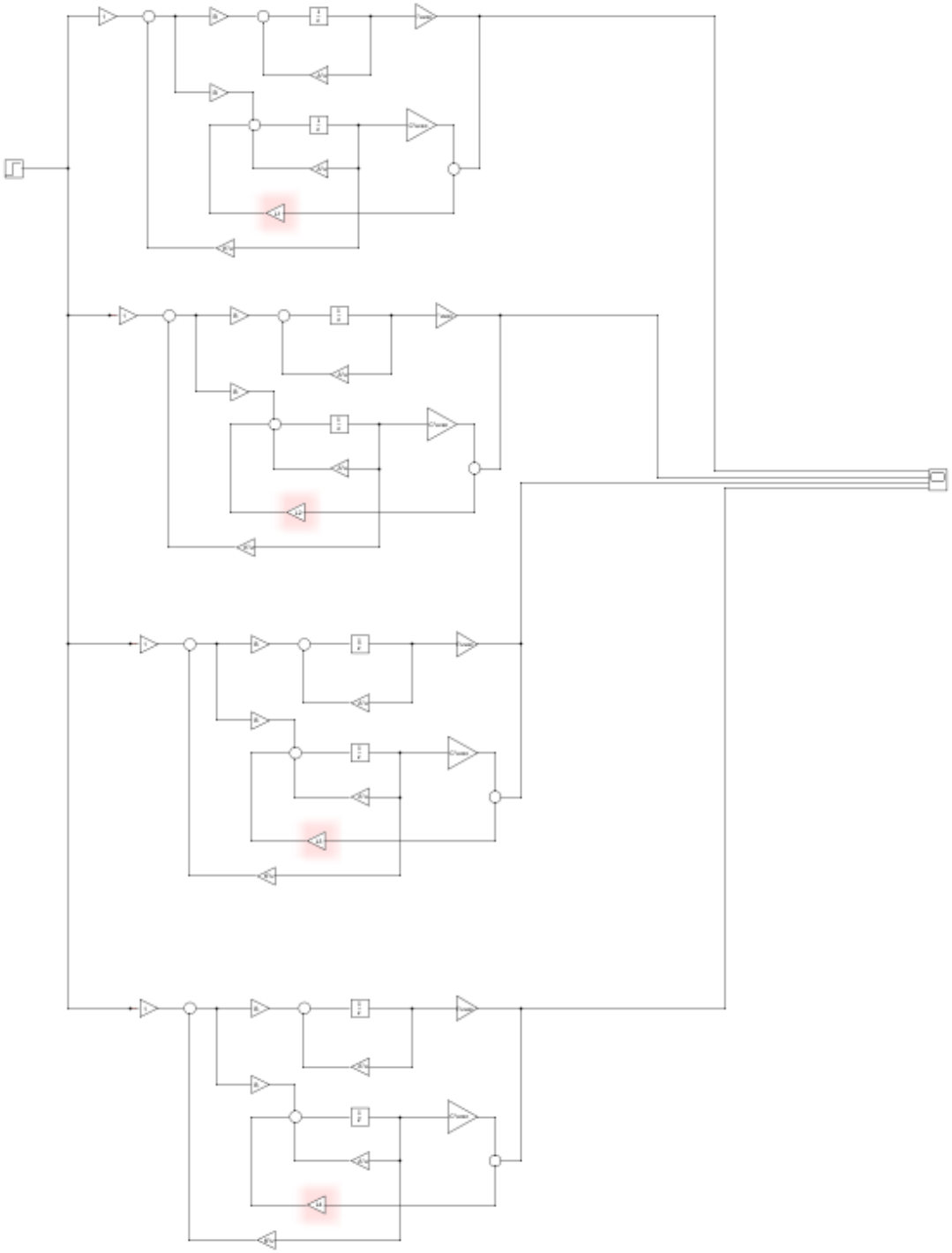


Observe how pole placement of observer design, changes time response:

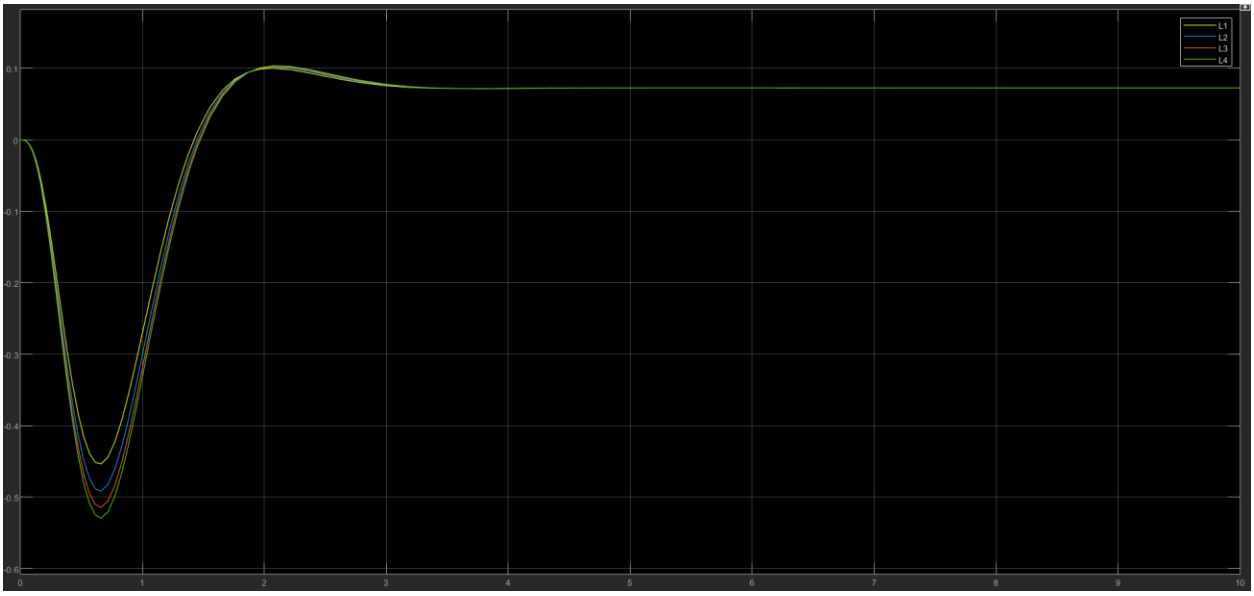
```
OBSp1=[-9+6.5i; -9-6.5i; -12];
OBSp2=[-12+9.5i; -12-9.5i; -15];
OBSp3=[-15+12.5i; -15-12.5i; -18];
```

```
OBSp4=[-18+15.5i; -18-15.5i; -21];
L1=place(A,B,OBSp1);
L2=place(A,B,OBSp2);
L3=place(A,B,OBSp3);
L4=place(A,B,OBSp4);
```

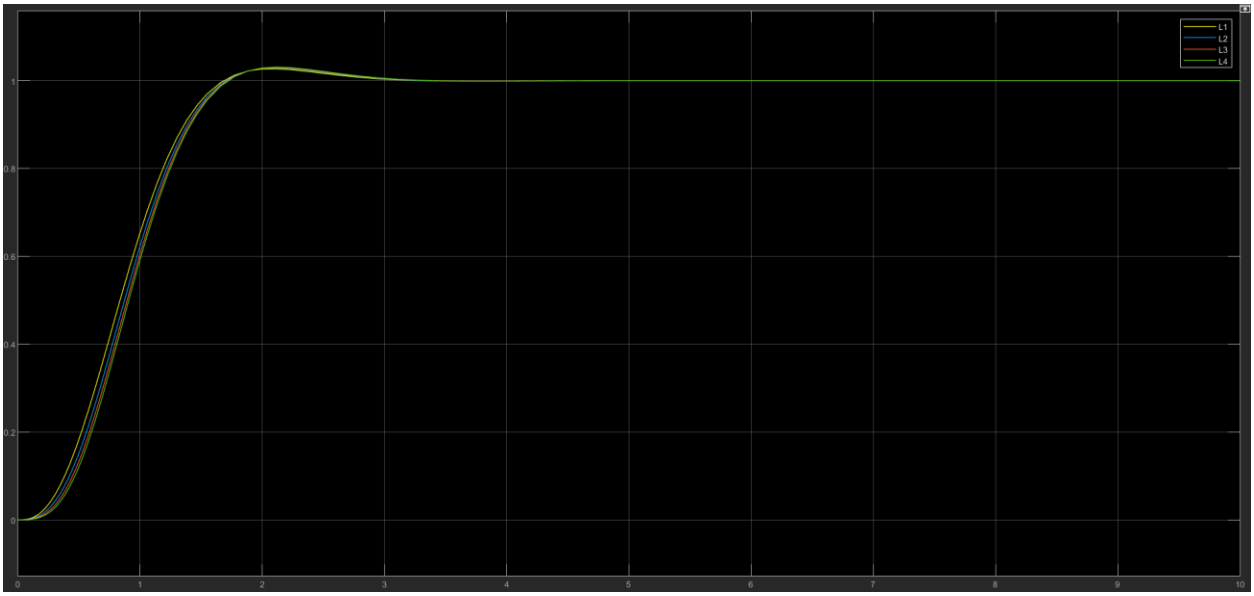
Block Diagram:



Gives the following time response:



When Kss Adjusted:

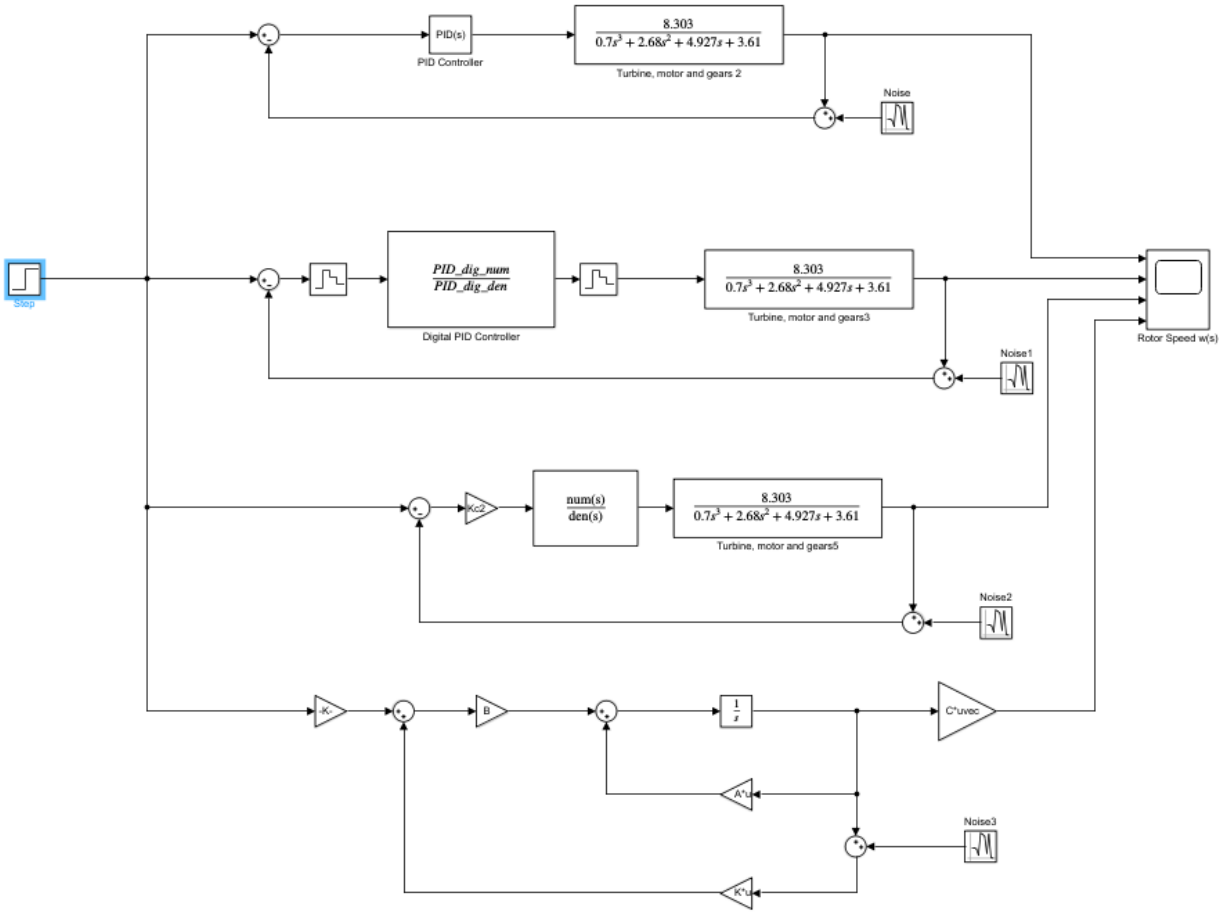


As can be observed, the pole placement of the observer design has minimal impact on the time response of the system, only improving time response slightly. Therefore, for my final design for the observer, I will place the poles at [-9+6.5i, -9-6.5i, -12+0i].

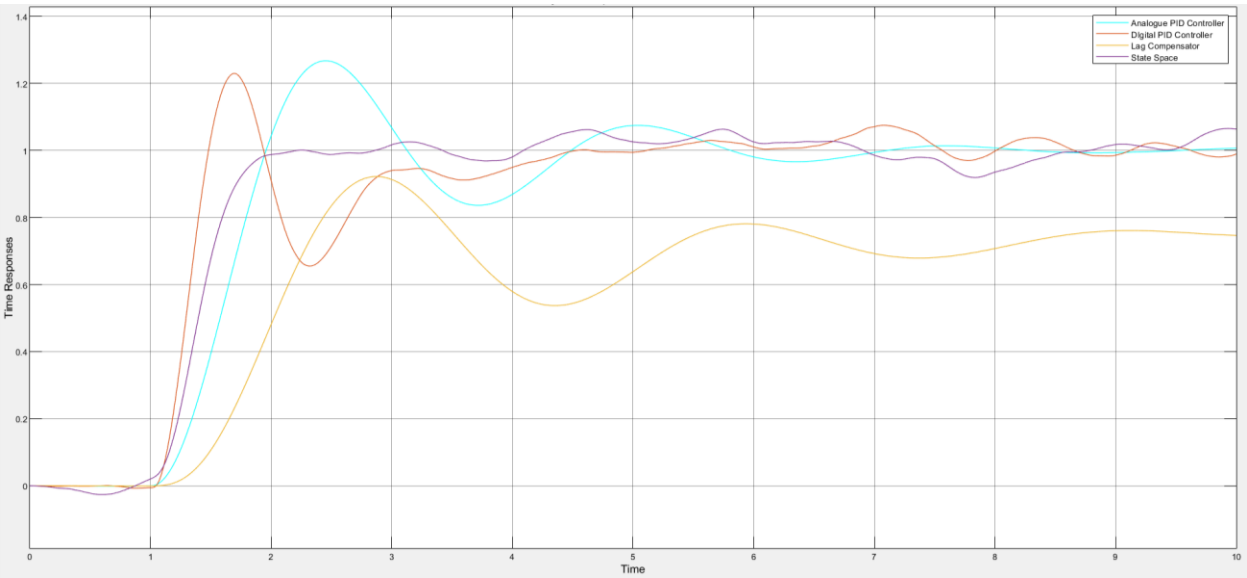
Friday, 16th January 2021

Add Noise to different controllers and observe effects

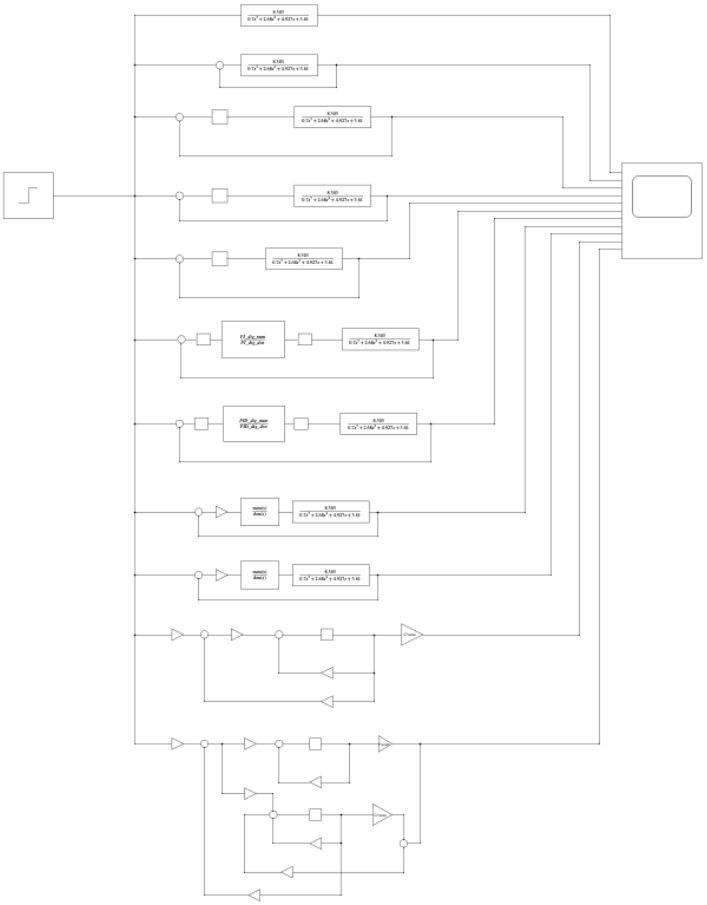
- Analogue PID controller
- Digital PID controller
- Lag Compensator
- State Space



Gave following time response:



Constructed following block diagram to test all controllers:



To give following output: