EE 4360 Final Project

Spring May 28th, 2021

Drew Schmidt

A04451613

Section I. Description

PID control stands for Proportional Integral Derivative control. It is a controller that uses feedback from the plant's output to minimize error in a system. A systems error is fed into this PID controller then the controller outputs an input for the plant that forces the plant to produce an output that eliminates error. One common example of a PID controlled system is your car's cruise control. As the driver you set a desired speed for your car to stay at. Then when you encounter environmental or road condition such as wind or an incline in the road surface the car naturally will want to slow down. The PID controller sees the difference in your desired speed and your cars actual speed then sends an input to the engine telling your car to increase speed. This eliminates the error between desired speed and actual speed and your car travels at the speed you set it at. One example of a system without a PID controller is a dishwasher. When you turn on your dishwasher it goes through the same wash cycle no matter the number of dishes in the washer. This is because there is no feedback in a dishwasher it is an open loop system.

Design Outline:

1. First I will determine the current parameters of my system and define variables such as percent overshoot, settling time, and peak time. I will also define what is required of my system by the end of the project.

2. Verify plant poles by analyzing bode plots in MATLAB and SPICE.

3. Verify the uncompensated step response by analyzing the step response in MATLAB and SPICE. As well as find the physical component values needed to design the circuit.

4. Analyze the root locus of my system. This will verify poles and gain values to maintain a desired percent overshoot. Determine where the dominant pole pair should be to obtain the requirements for my system.

5. Implement an PI zero and pole pair. This will eliminate the steady state error in my system and give a more accurate response.

6. Design the PD portion of the PID controller to bend my systems root locus through the desired dominant pole pair. This will be done by calculating an extra zero to give the system an overall angle calculation (equation will be shown in report) equal to an odd multiple of 180 degrees.

7. All results will be recorded and compared in MATLAB and SPICE before moving to the tuning phase.

8. I will choose an output of my system then change it so that the tuned system's output will match the output I changed. Tunning research will be done and my system will be tuned to meet the requirements.

9. Comparisons between the untuned and tuned systems will be made to see if my system met the requirements.

Section II. The Plant

II.1 Plant Transfer Function and Design Targets

- Poles: s = 50.265, 62.832, 157.08 rad/s
- G(s) = $\frac{496097}{(s+50.265)(s+62.832)(s+157.08)}$ EQ(1)
    - W_n = 496097 = (50.265)(62.832)(157.08) EQ(2)
- %OS = 40%
- T_P% improvement = 50%

II.2 Verification of Plant Poles

Since the poles were given in radians per second, we can use the equation $\boldsymbol{\omega = \frac{1}{RC}}$ EQ(3) to find our resistor and capacitor values for each pole. By selecting an arbitrary resistance of 100k ohms for each pole I can find the capacitance values by solving the equation for C.

	ω (Rad/s)	R (Ohms)	C (Farads)
Pole 1	50.265	100k	198.95n
Pole 2	62.832	100k	159.15n
Pole 3	157.08	100k	63.67n

Table 1: The capacitance values in this table were calculated using the equation discussed in the paragraph above. Each frequency, resistance, and capacitance represents the corresponding pole in the left column.

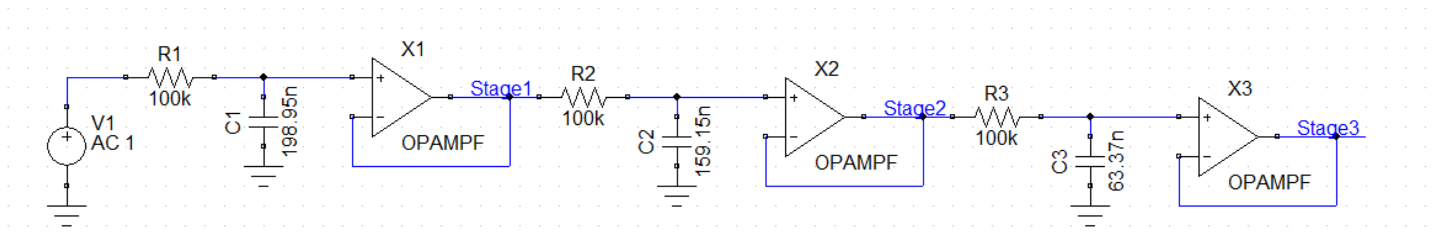


Figure 1: This schematic is my plant in TopSpice with an AC voltage source with a magnitude of one volt. All components values are labeled.

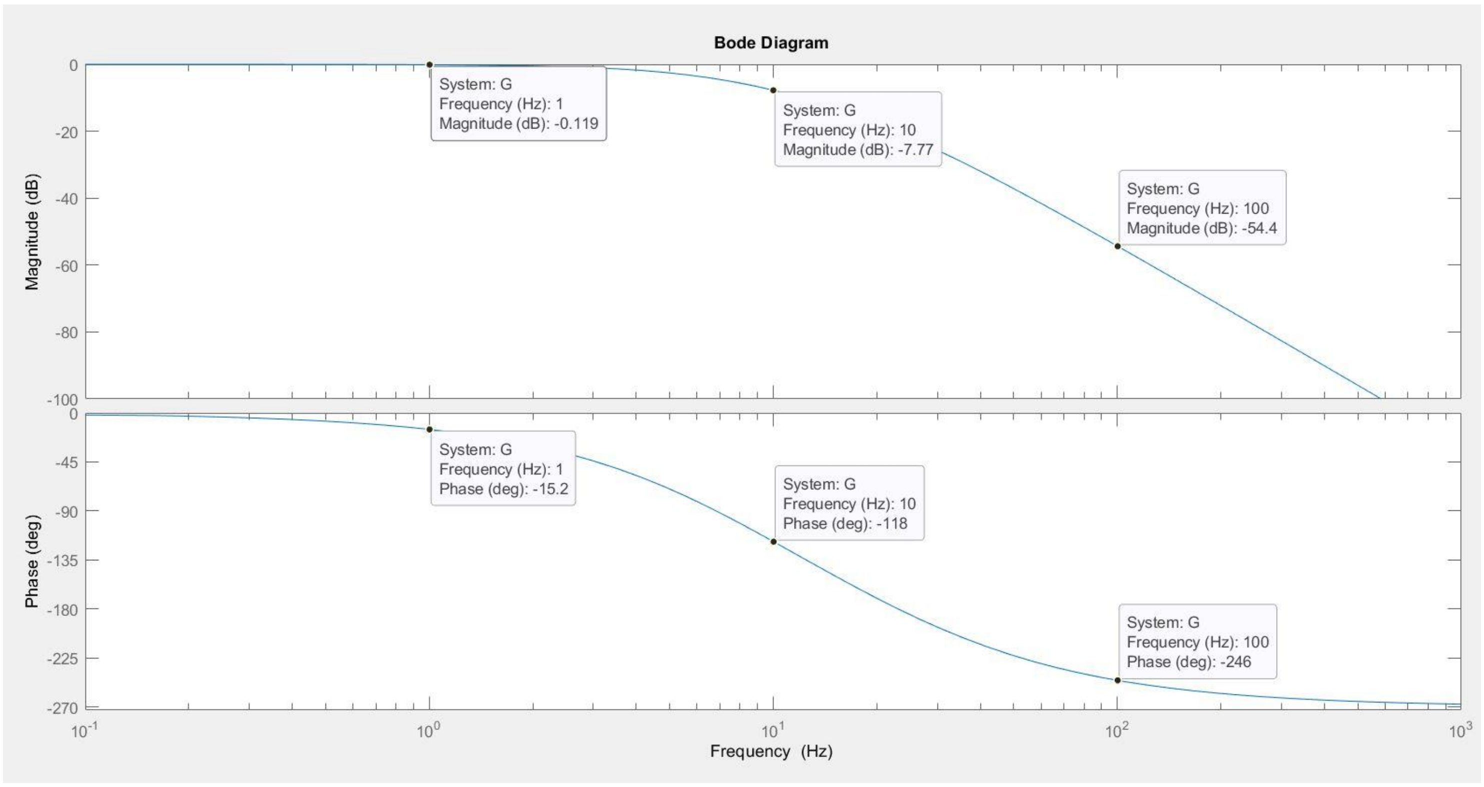


Figure 2: The systems Bode plot in MATLAB with 3 Data points. Notice the frequency is in Hertz and all axes are labeled properly.

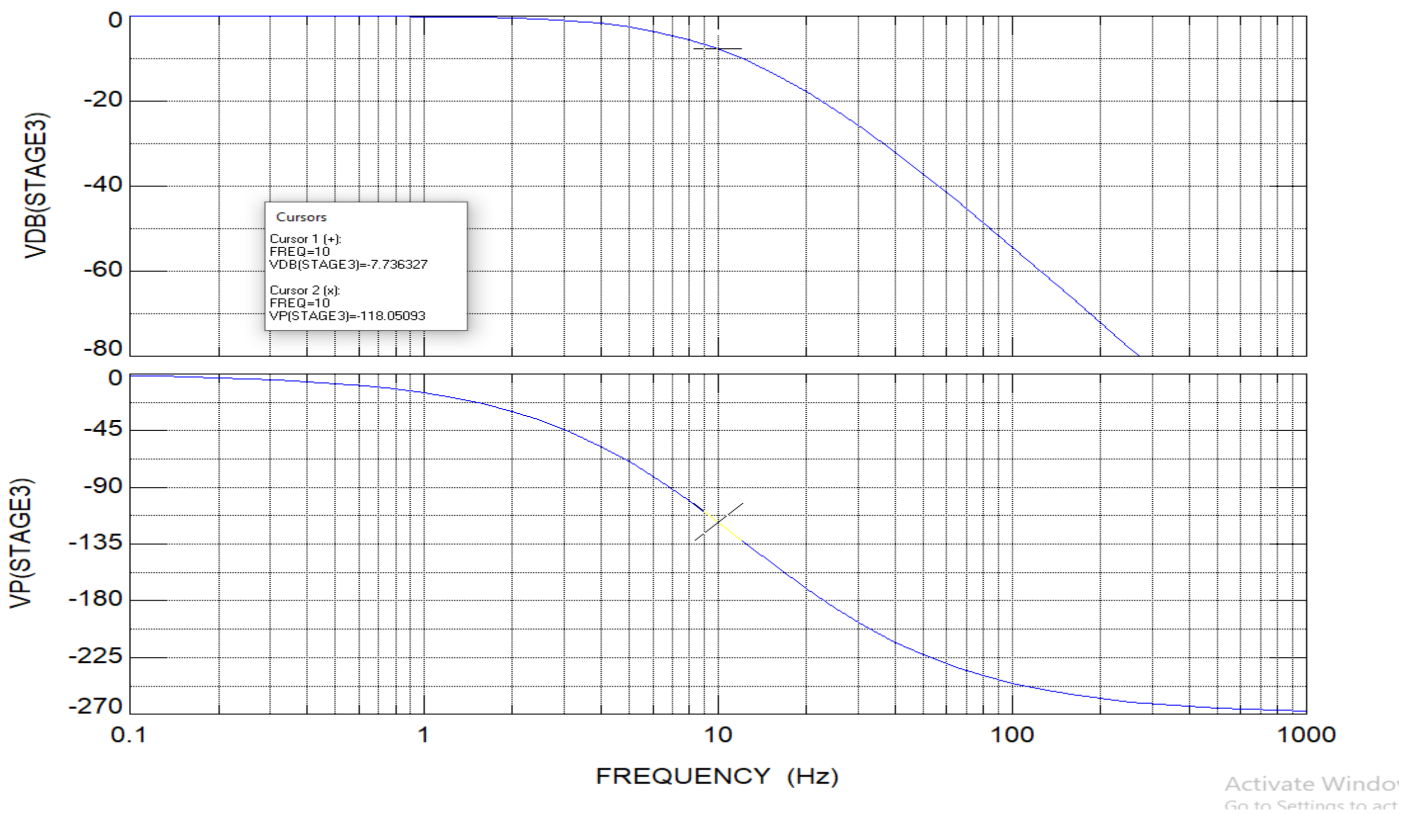


Figure 3: The system's Body plot in TopSpice is show above with the same axis parameters as Figure 2. I was only able to obtain one data point in TopSpice because I was not able to add multiple cursors. That being said I was able to confirm all of the data points values and they are almost the exact values as MATLAB.

II.3 Verification of Uncompensated Step Response

$$\text{Transfer Function} = \frac{496097}{(s+50.265)(s+62.832)(s+157.08)} \quad \text{EQ(1)}$$

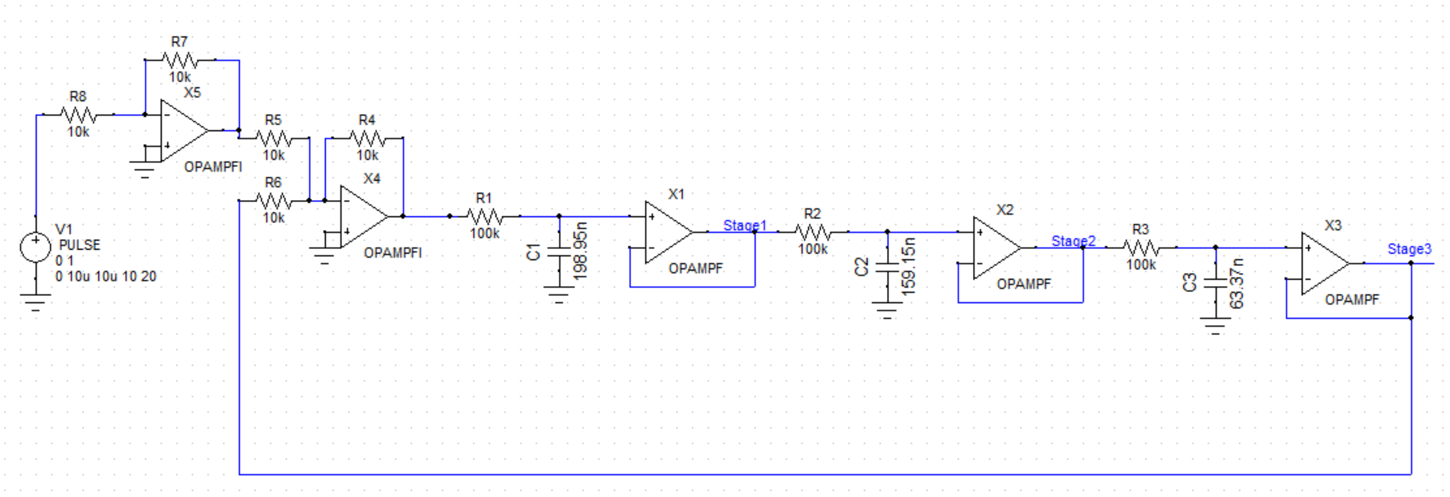
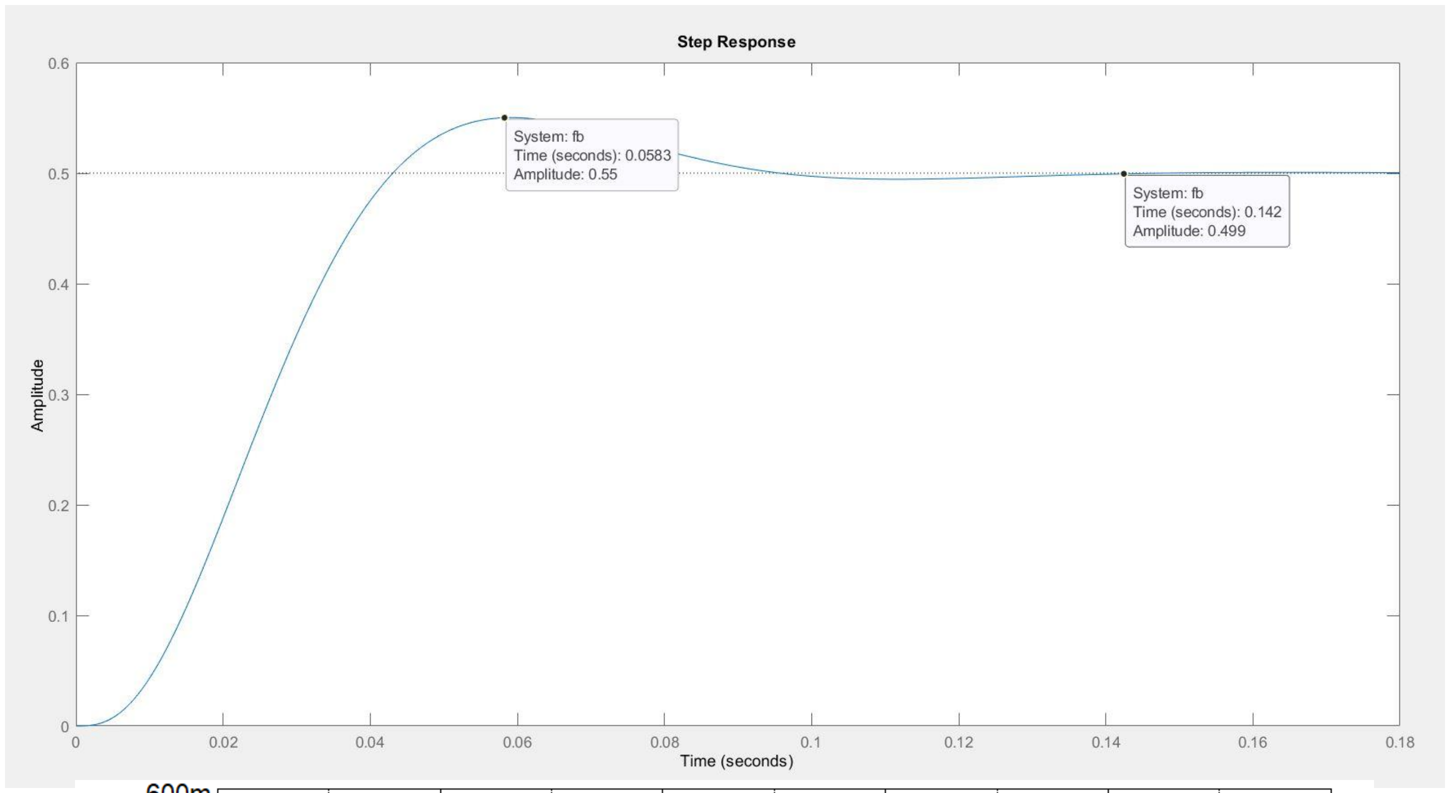
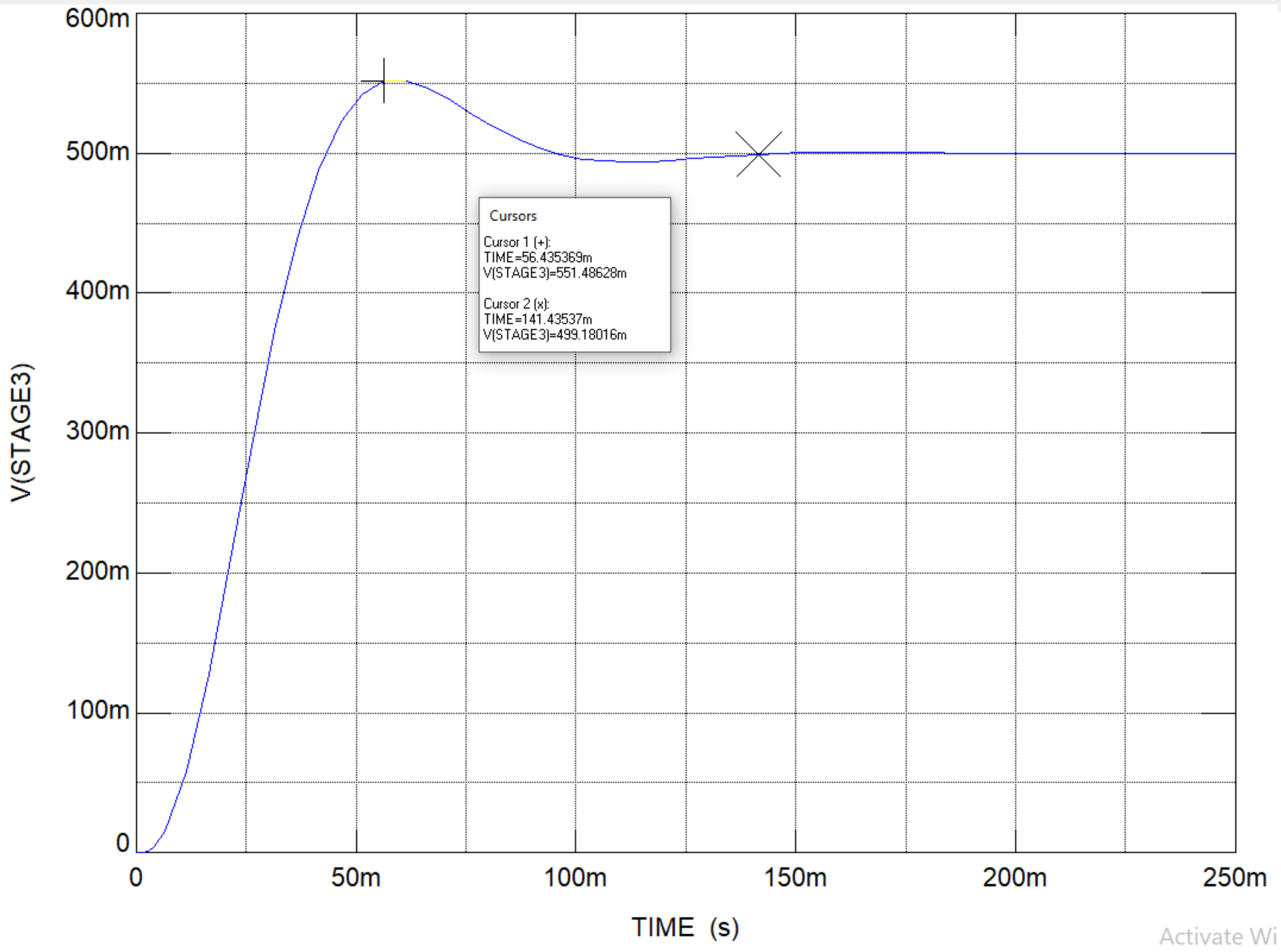


Figure 4: This schematic represents my system in TopSpice with a step input. All component values are labeled and I am using a pulse input.



```
System: fb
Time (seconds): 0.0583
Amplitude: 0.55
```

```
System: fb
Time (seconds): 0.142
Amplitude: 0.499
```



```
Cursors
Cursor 1 (+):
TIME=56.435369m
V(STAGE3)=551.48628m

Cursor 2 (x):
TIME=141.43537m
V(STAGE3)=499.18016m
```

Figure 5: This graphic of the two step input responses shows that in both TopSpice and MATLAB the rise time and settling time are roughly the same. The rise time occurs around 0.057s with a value of 0.55 volts

with a 1 volt input. The settling time occurs at 0.14s with a value of 0.5 volts. Since the peak is at an amplitude of 0.55v and the settling time is at 0.5v there is a 10% overshoot in the system.

Uncompensated Step Response Comparison		
	MATLAB	**SPICE**
T_s	0.142s	.0141s
T_p	0.0583s	0.0564s
%OS	10.2%	10.4%

Table 2: The results for the step response in MATLAB and TopSpice are shown. There is a 1.9% difference between the two %OS values. The points were taken as closely as possible to eachother.

II.4 MATLAB Analysis of Uncompensated Plant

From the given target %OS I was able to calculate my zeta value by using the following equation.

$$\zeta = \frac{-\,ln(\%OS)}{\sqrt{\Pi^2+ln^2(\%OS)}} = 0.280 \text{ EQ(4)}$$

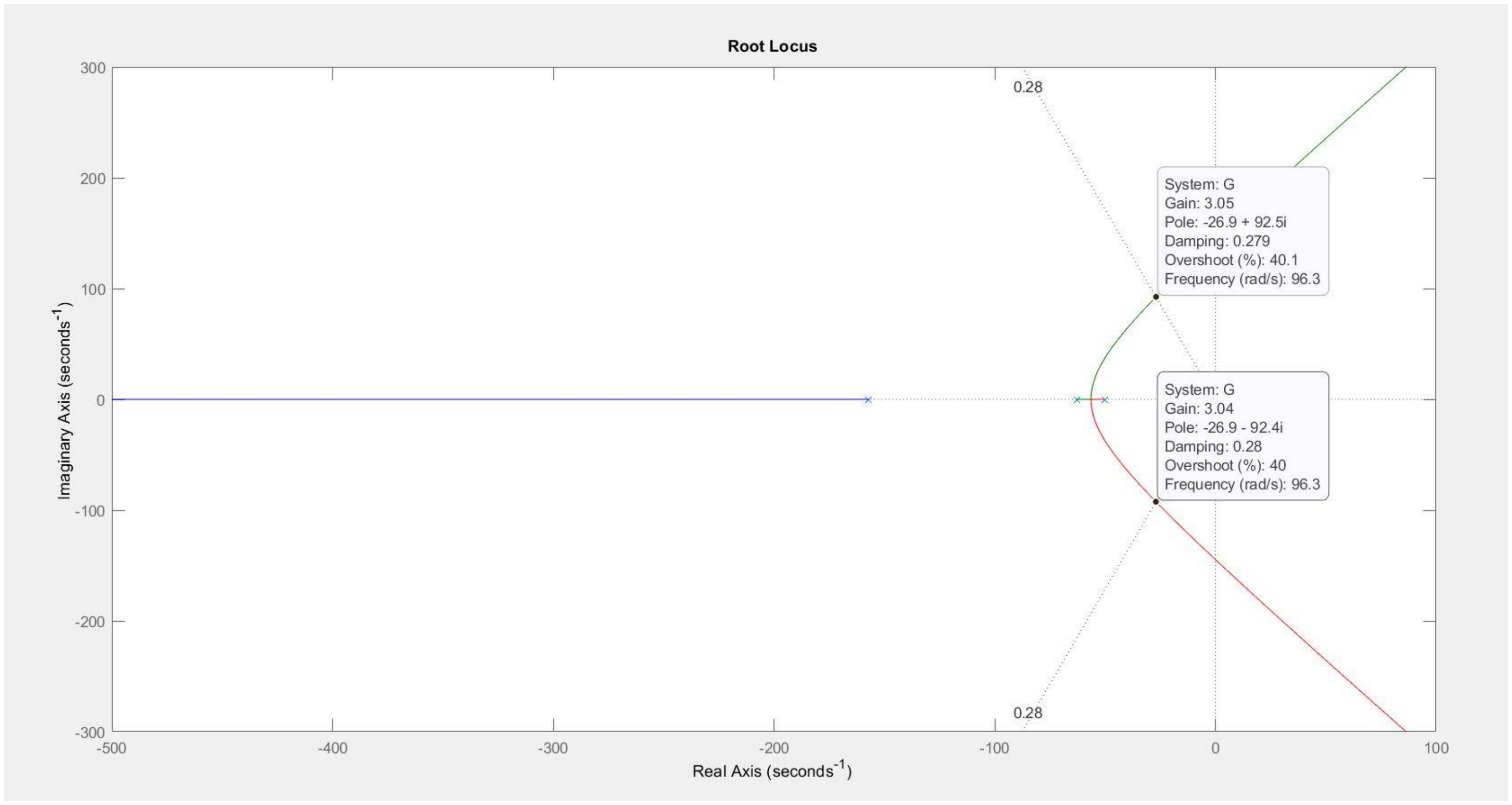
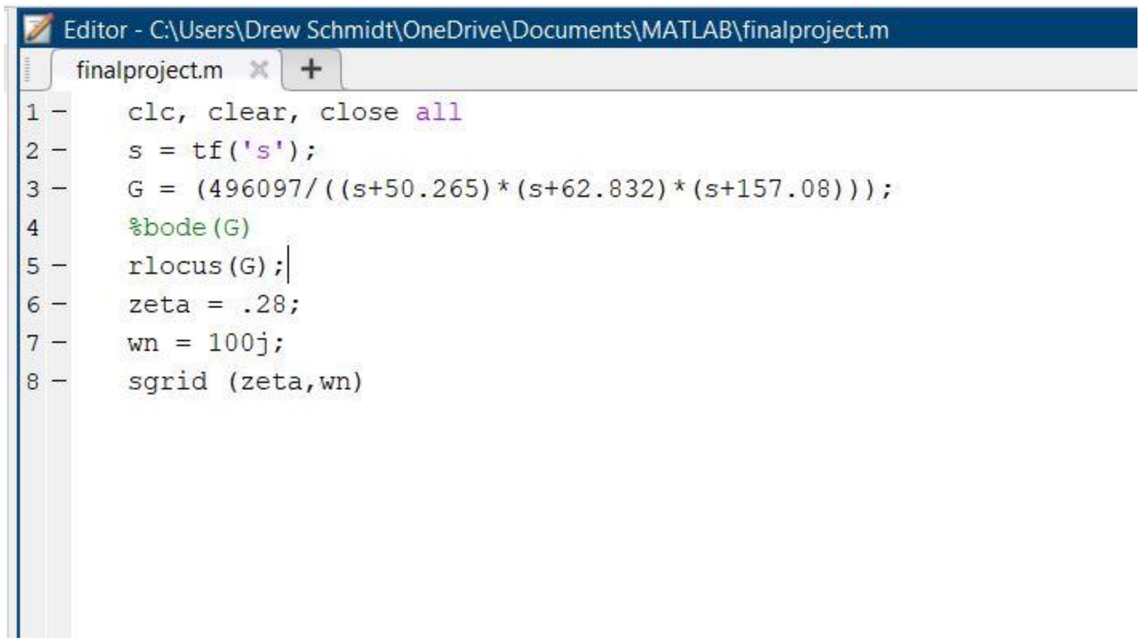


Figure 6: The root locus of my system along with the zeta values are plotted in the graph from MATLAB above. When my system has a gain of roughly 3.05 the desired %OS is achieved. The pole locations of my system are at s = -26.9 + j92.5 and s = -26.9 – j92.5.


Editor - C:\Users\Drew Schmidt\OneDrive\Documents\MATLAB\finalproject.m

finalproject.m

```
clc, clear, close all
s = tf('s');
G = (496097/((s+50.265)*(s+62.832)*(s+157.08)));
%bode(G)
rlocus(G);
zeta = .28;
wn = 100j;
sgrid (zeta,wn)
```

Figure 7: This is the code I wrote in MATLAB to generate the root locus of my system. I defined my transfer function as “G” and my zeta value then ran the “rlocus” function in MATLAB to plot the two on the same graph.

Section III. PI Design

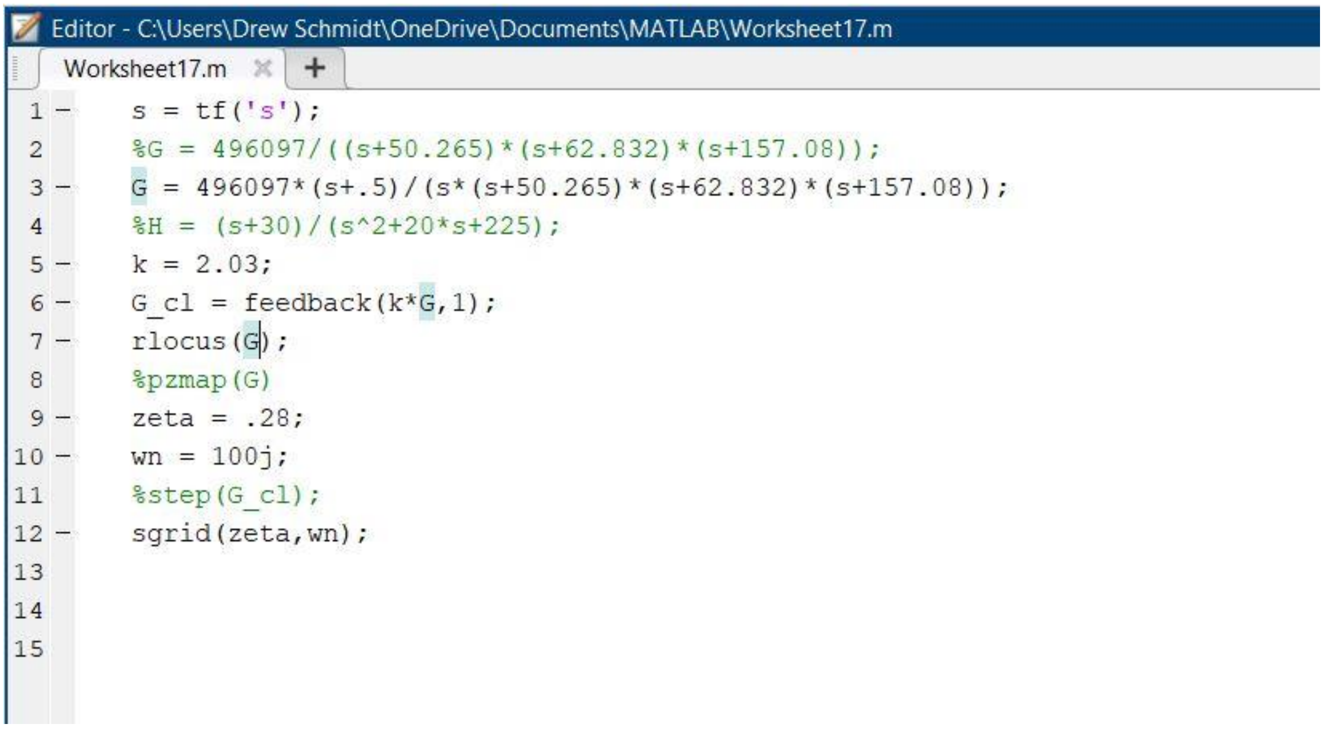
III.1 Function of PI Controller

A Proportional Integral or PI controller is a closed loop system designed to minimize the error in a systems actual output vs a desired output. The proportional part of the controller is to set the system to a desired response time for time to peak (Tp) and settling time (Ts). This value can be found by analyzing a systems closed loop root locus. For a desired percent overshoot (%OS) in a system you will have a set zeta value over laying the root locus. Along that zeta line there is a point that represents the desired Tp and Ts values. This point will have a gain factor (k) and this value is the proportional component of the controller. For my system I need to choose a k value that is on the zeta line representing 40%OS and has a Tp improvement of 50%.

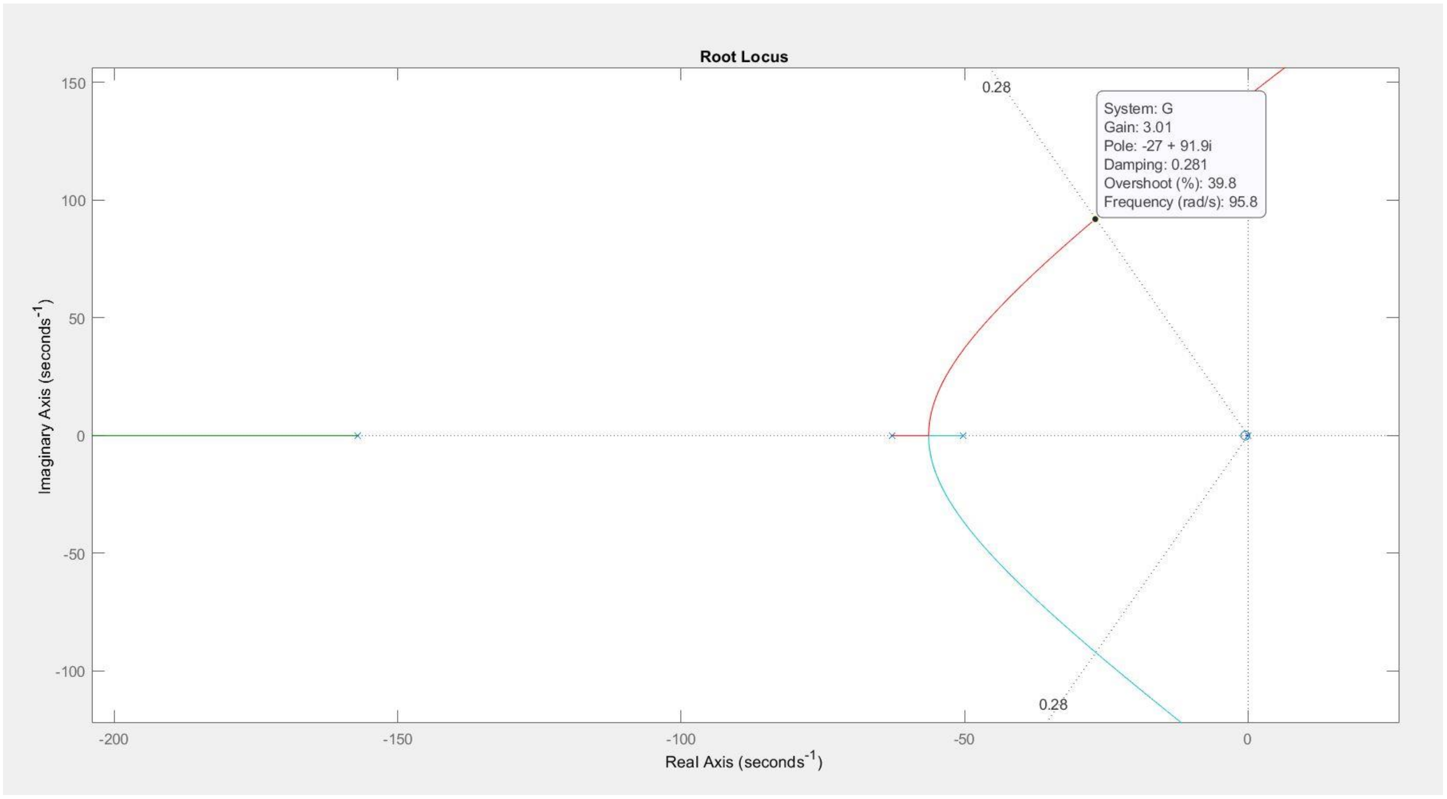
The integral term is used to improve the steady state error of the system. Since there is no pole at (s=0) there will be a non-zero steady state error for a step input. This means the system will never truly have zero error and will constantly be trying to compensate for the error. Which ends up being a waist of power and resources that in the long run will wear the system out faster. By adding the integral portion of the controller we are placing a pole at (s=0) to eliminate the steady state error. When this pole is placed the root locus of the system is changed. In order to readjust the system so that the root locus goes through the desired point on the zeta line, we must add a zero. It is important for the zero to have a value so that the angle from the positive x axis balances the following equation: **Σ(zeros) - Σ(poles) = (2k+1)180** degrees EQ(5). Where k is an integer value.

III.2 Design of PI Controller

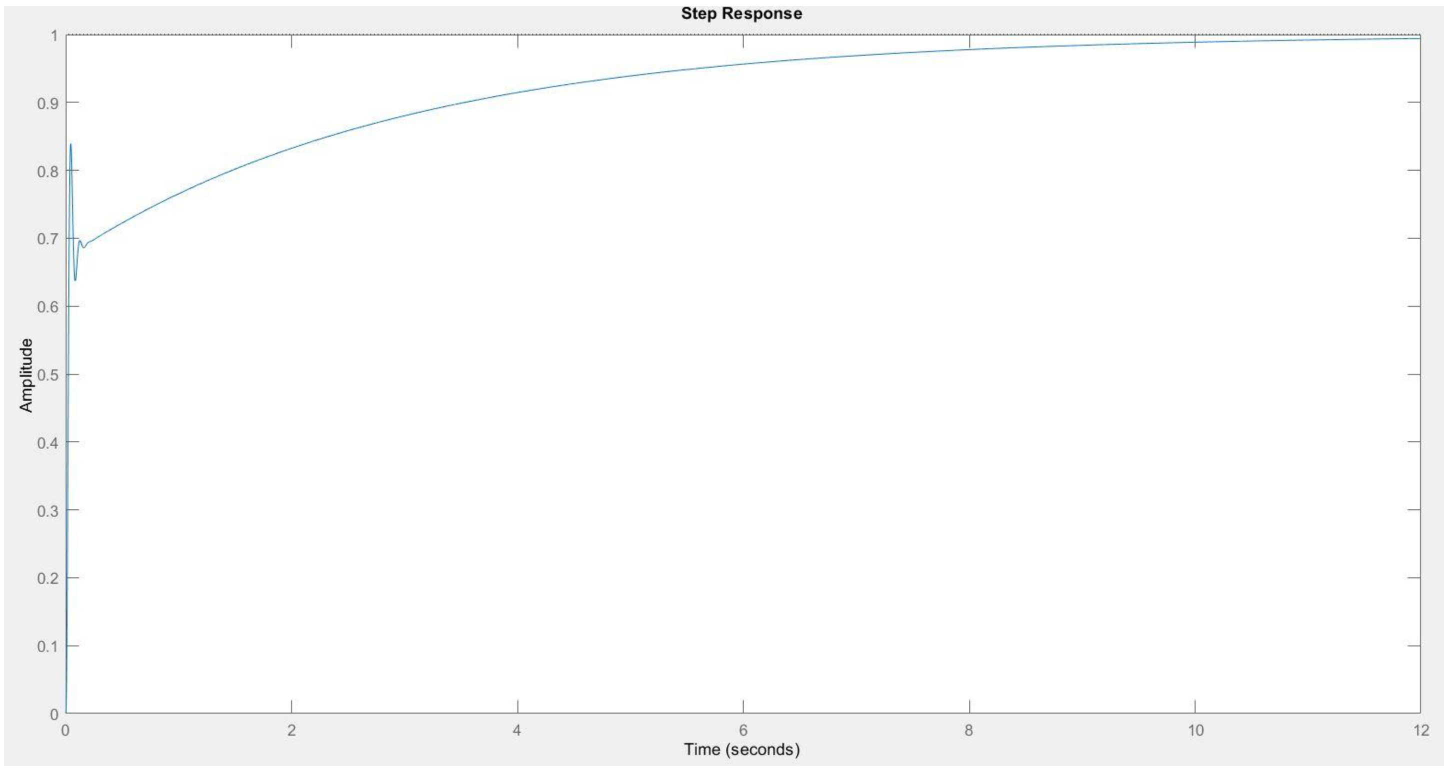
In MATLAB I determined the k value for my system is 3.03 because that is the value at which my system intersects the desired zeta value. After adding a pole at the origin and a zero close to the origin my equation came out as follows: $G(s) = \frac{(3.03)*(s+.5)*496097}{s*(s+50.265)*(s+62.832)*(s+157.08)}$ EQ(6)

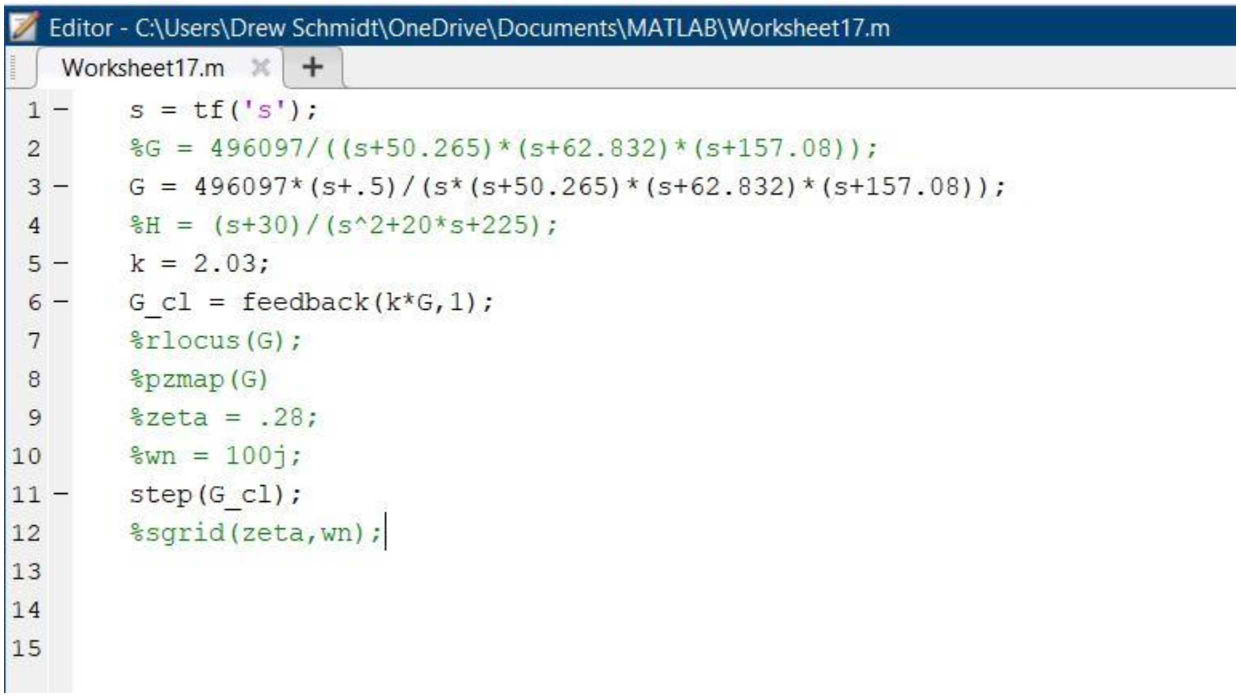

Editor - C:\Users\Drew Schmidt\OneDrive\Documents\MATLAB\Worksheet17.m
Worksheet17.m

```
s = tf('s');
%G = 496097/((s+50.265)*(s+62.832)*(s+157.08));
G = 496097*(s+.5)/(s*(s+50.265)*(s+62.832)*(s+157.08));
%H = (s+30)/(s^2+20*s+225);
k = 2.03;
G_cl = feedback(k*G,1);
rlocus(G);
%pzmap(G)
zeta = .28;
wn = 100j;
%step(G_cl);
sgrid(zeta,wn);
```



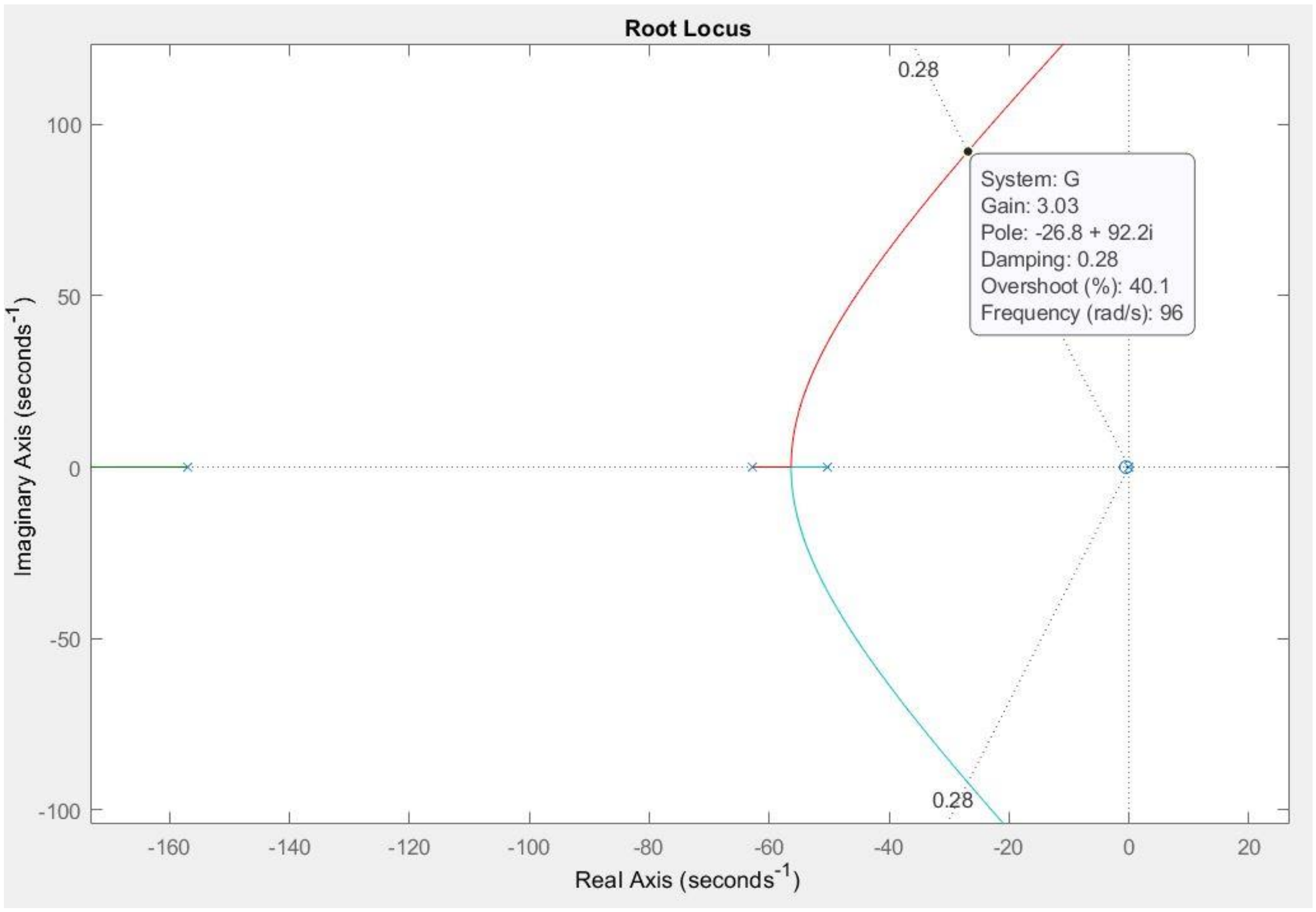
Figures 8 and 9: Show my open loop root locus code and graph after making the PI controller adjustments. As you can see there is not much of a change in the open loop root locus. This is because the pole and zero are close enough to cancle eachother out.

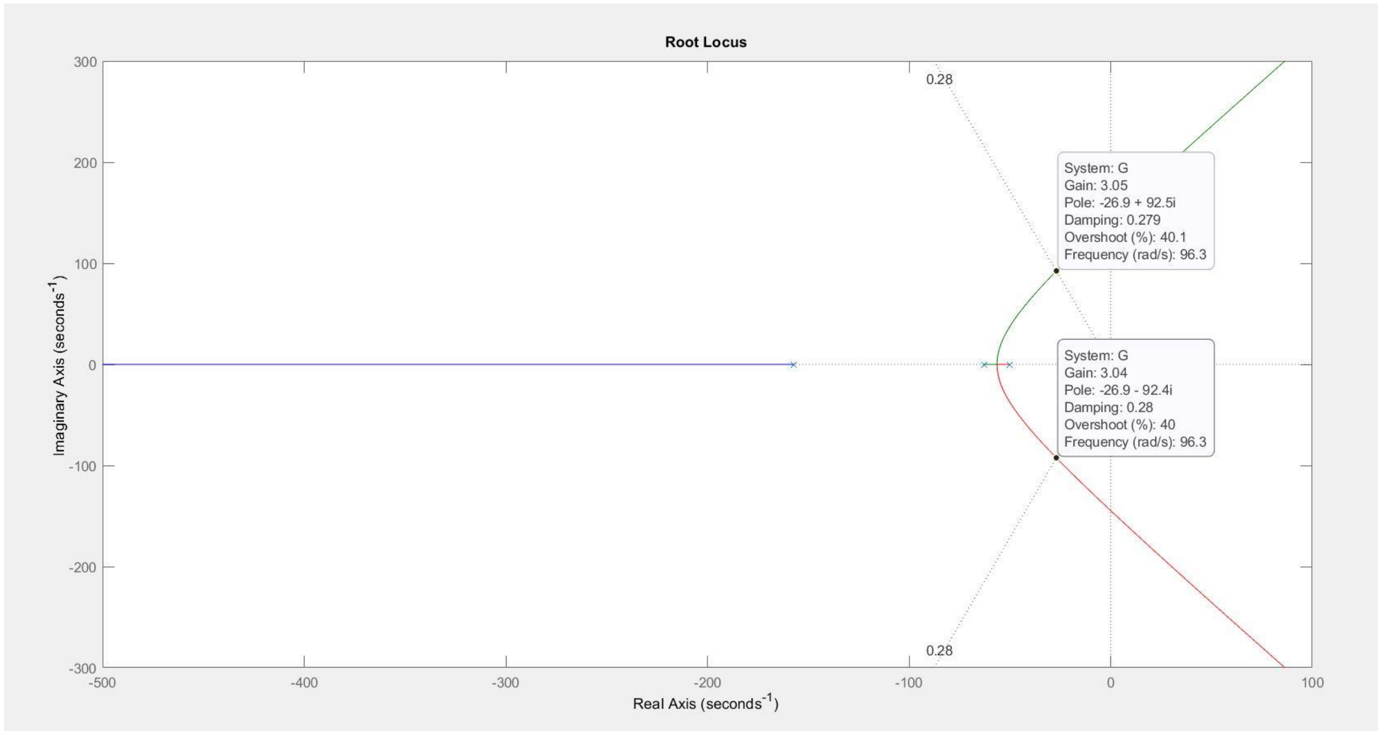



Editor - C:\Users\Drew Schmidt\OneDrive\Documents\MATLAB\Worksheet17.m
Worksheet17.m

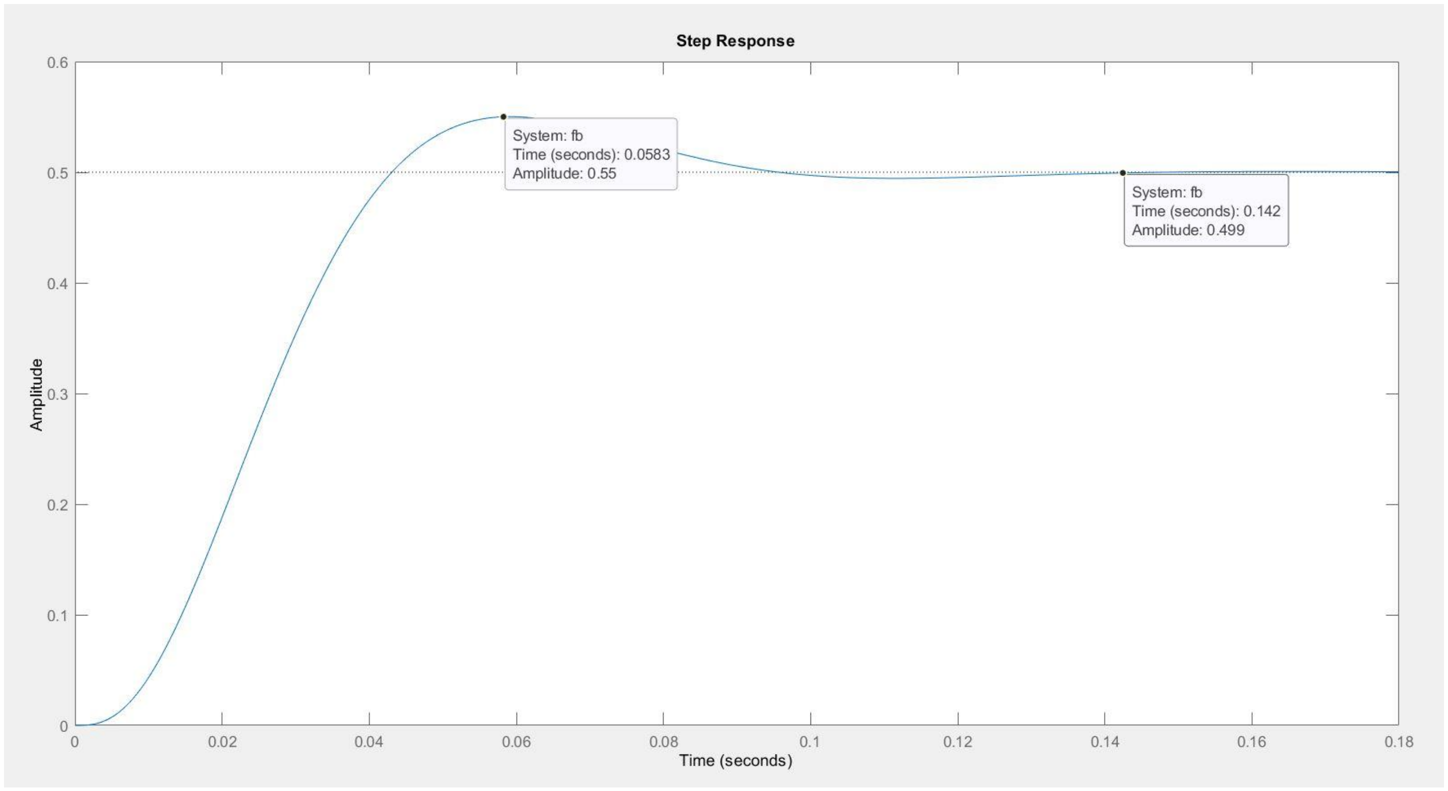
```
s = tf('s');
%G = 496097/((s+50.265)*(s+62.832)*(s+157.08));
G = 496097*(s+.5)/(s*(s+50.265)*(s+62.832)*(s+157.08));
%H = (s+30)/(s^2+20*s+225);
k = 2.03;
G_cl = feedback(k*G,1);
%rlocus(G);
%pzmap(G)
%zeta = .28;
%wn = 100j;
step(G_cl);
%sgrid(zeta,wn);
```

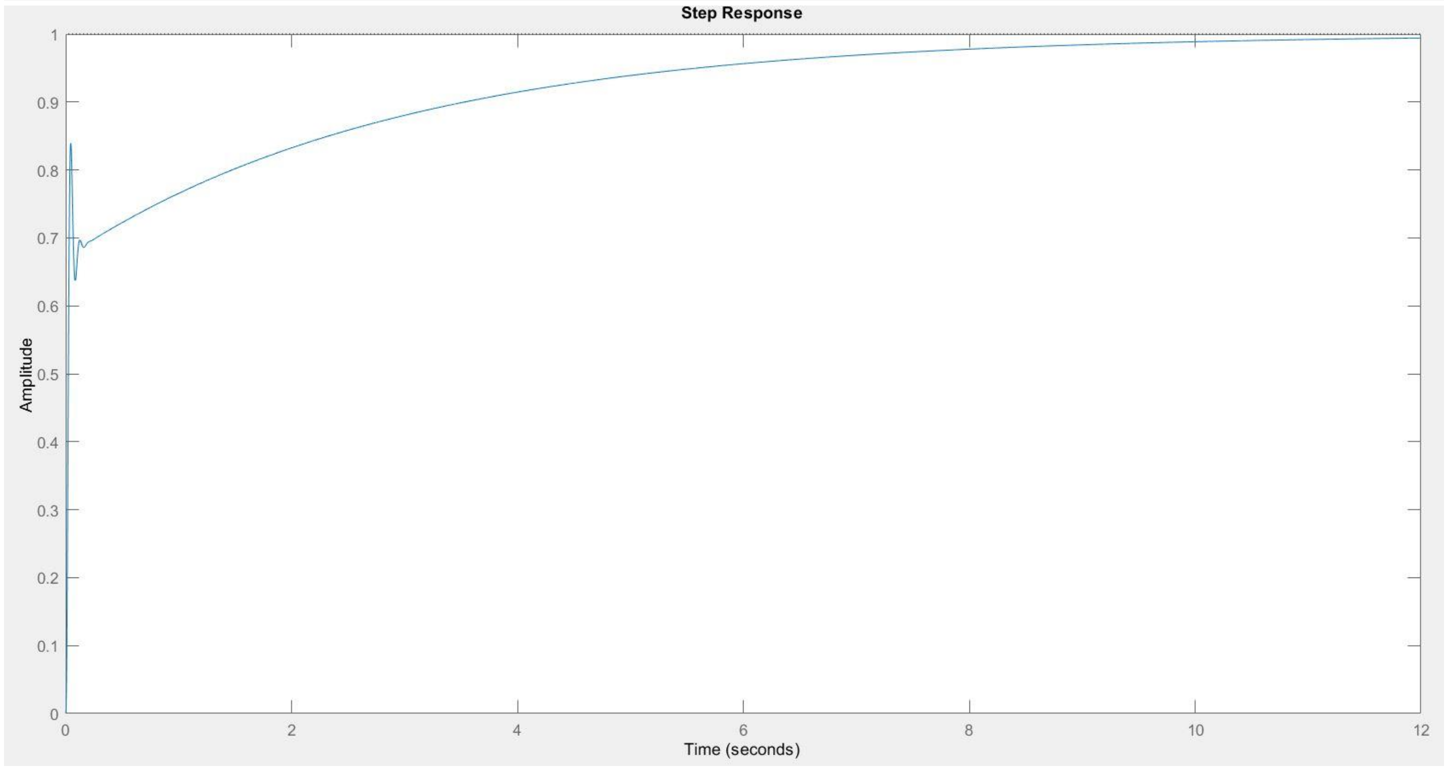
Figures 10 and 11: Here you can see the step response and code of the closed loop system after the PI controller adjustments. The step response has major differences from the previous step response. One is the response time is much slower. This is because the pole and zero are so close to the origin. But as you can see there is no steady state error now.





Figures 12 and 13: There is almost no change in the open loop root locus. The top figure shows the compensated system and the bottom shows the uncompensated system. The pole and zero cancle eachohter out so no change was expected in the root locus.





Figures 14 and 15: The top is the uncompensated step response while the bottom is the compensated response. Both figures represent the closed loop systems. The step for the bottom looks to have a slower response but there is no steady state error in the system. As you can see there is still an adjustment that needs to be made in order to have a more desirable response.

Table 2: Comparison of Dominant Pole Location		
	Pre PI	**Post PI**
Pole Location	-26.9 (+-) j92.5	-26.8 (+-) j92.2
K value	3.05	3.01

Table 2: There is no big change in the pole locations because the pole and zero that were added to the system cancel each other out because the distance between them is so small. The purpose of the PI controller is to drive the steady state error to zero.

Section IV. PID Design

IV.1 PD Design

Table 3: Pole and Zero Locations after PI	
Zeros	**Poles**
-0.5	0
	-50.265
	-62.832
	-157.08

Table 3: This shows the location of the poles and zeros in the system after the PI compensation.

The equation used to find Time to peak is $\boldsymbol{Tp = \pi/|Im|}$ In order improve the time to peak by 50% we need to increase the magnitude of the imaginary component of our pole by 50% when increasing 92.2 by 50% we get **Im = j138.6.**

Since I know the imaginary component of the desired point and I also know it is on the zeta line I can find the real component of the point with this equation: **Real Component =** $\boldsymbol{\frac{138.6}{\tan\left(\cos^{-1}(0.28)\right)}}$ **= 40.43** EQ(7)

This puts my desired point at: **s = 40.43 (+-) j138.6**. Using the equation:

Σ(zeros) - Σ(poles) = (2k+1)180 degrees EQ(5) I was able to determine the system needs a zero at **s = -227.08** in order to curve the root locus and achieve 50% improvement in time to peak.

Our new system with the PID controller implemented is:

$$G(s) = \frac{(0.0303)*(s+227.08)*(s+.5)*496097}{s*(s+50.265)*(s+62.832)*(s+157.08)} \text{ EQ(8)}$$

The PID transfer function without the plant is:

$$G_{PID}(s) = \frac{(s+.5)*(s+227.08)}{s} \text{ EQ(9)}$$

$$G_{PI}(s) = \frac{(s+.5)}{s} \text{ EQ(10)}$$

$$G_{PD}(s) = (s+227.08) \text{ EQ(11)}$$

VI.2 PID Verification in MATLAB

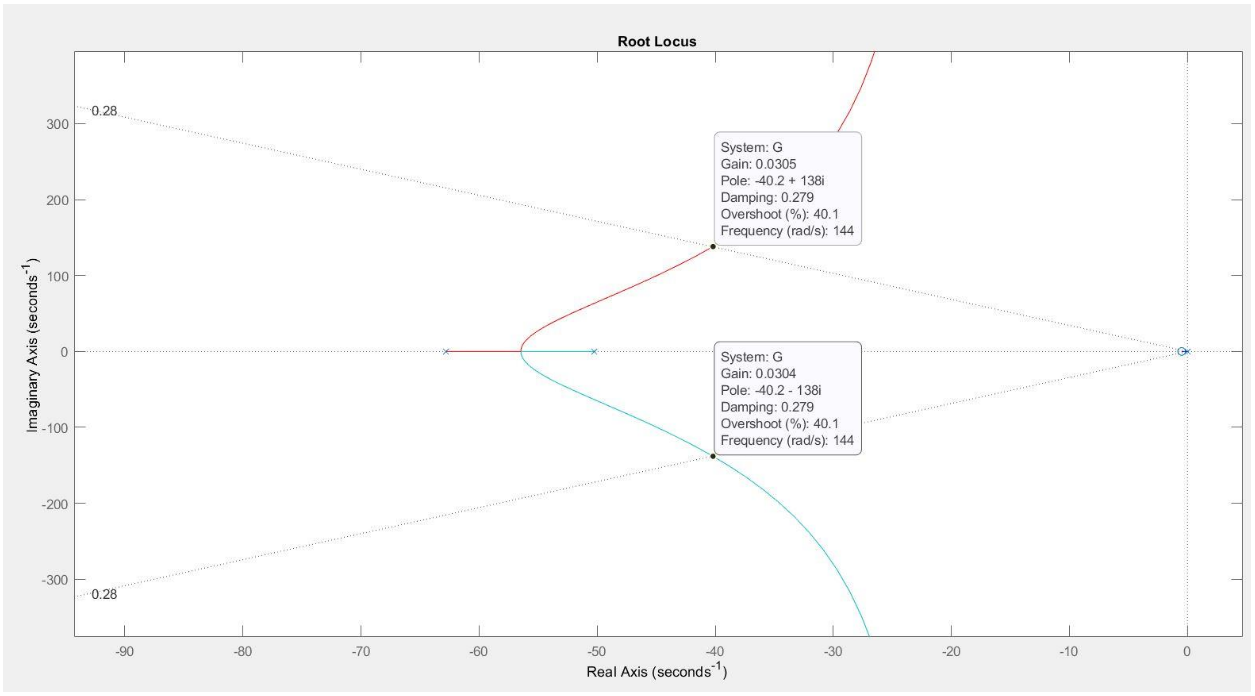


Figure 16: This is the system's root locus after the PID controller is implemented. The dominant poles are at **s = 40.2 (+-) j138**. Since the dominant poles are now shifted we can see that the system is adjusted to have a 50% improved time to peak with the same 40%OS.

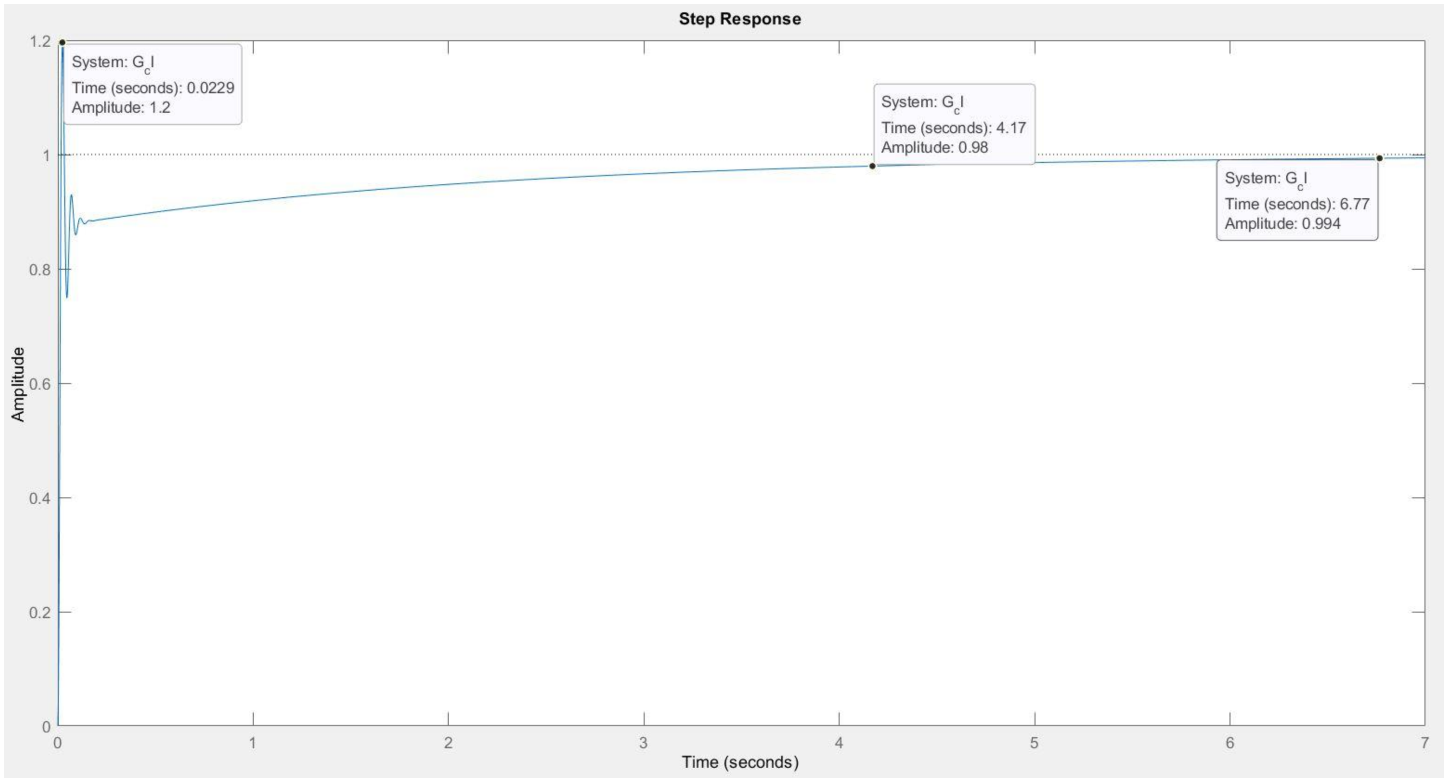


Figure 17: The system's step response after implementing the PID controller is shown. The settling time is 4.17 seconds with a peak time at 0.0229 seconds. The step input error is 0.006 according to the point plotted above.

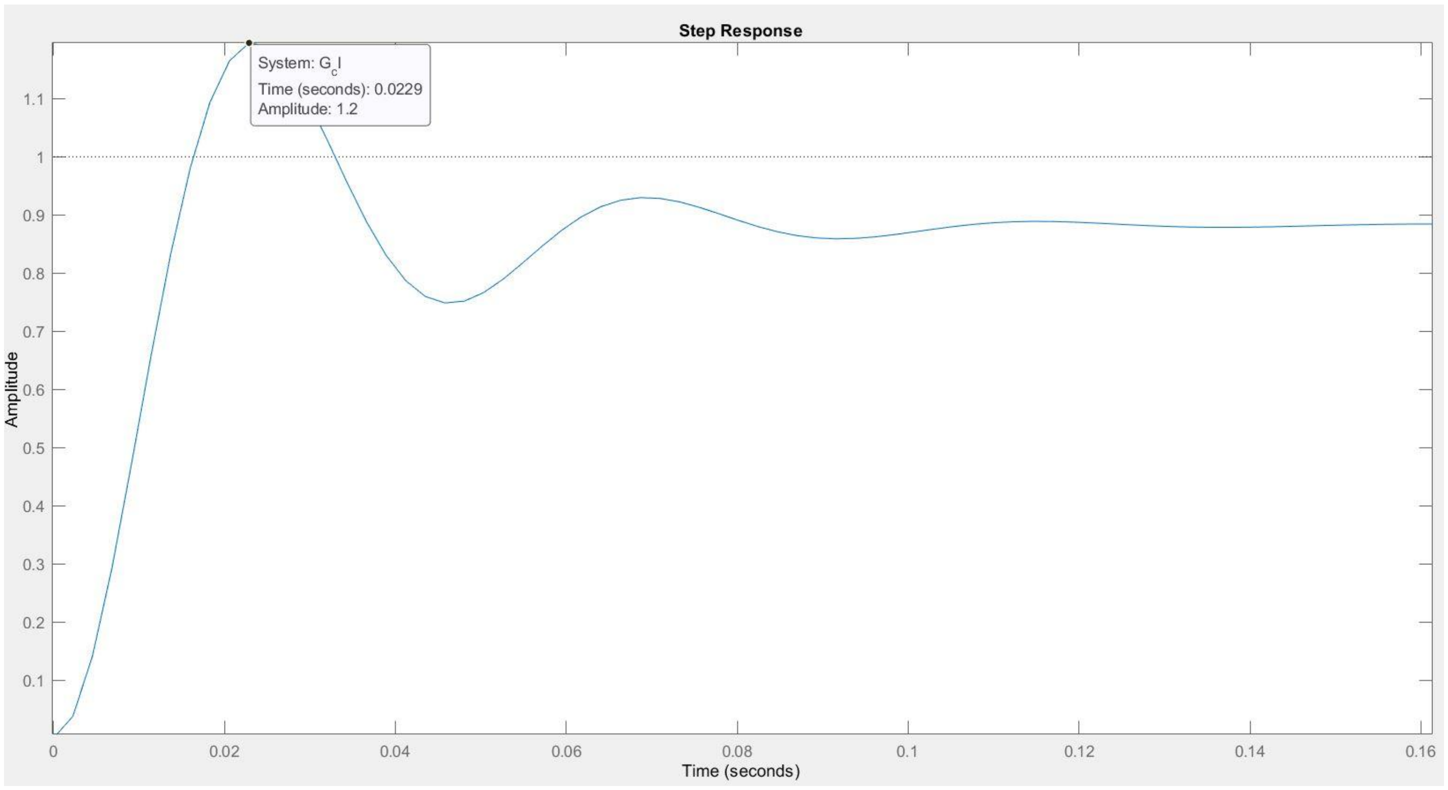


Figure 18: Closed loop step response of system with PID controller with a short time scale. This highlights the time to peak value better than the long time scale image.

VI.3 PID Verification in Spice

Using the PI and PD equations shown above along with the PID gain value of 0.0303 I derived the equation $G_{PID}(s) = \frac{0.0303*(s+.5)*(s+227.08)}{s} = \frac{0.0303*(s^2+227.58s+90.8)}{s}$ EQ(12). The coefficients from this equation are used as the k values for the op amps in the top spice circuit.

Table 4: Gain and Component Calculation				
Component	**Equation**	**Gain**	**(Resistor/Capacitor)$_s$**	**(Resistor/Capacitor)$_f$**
Proportional	$K_1 = \frac{-R_f}{R_s}$	6.896	100k ohms	689.6k ohms
Integral	$K_2 = \frac{1}{RC}$	2.751	100k ohms	3.635 µF
Derivative	$K_3 = RC$	0.0303	303 nF	100k ohms

Table 4: This table shows the values that will be used in the top spice design and how I arrived to those values.

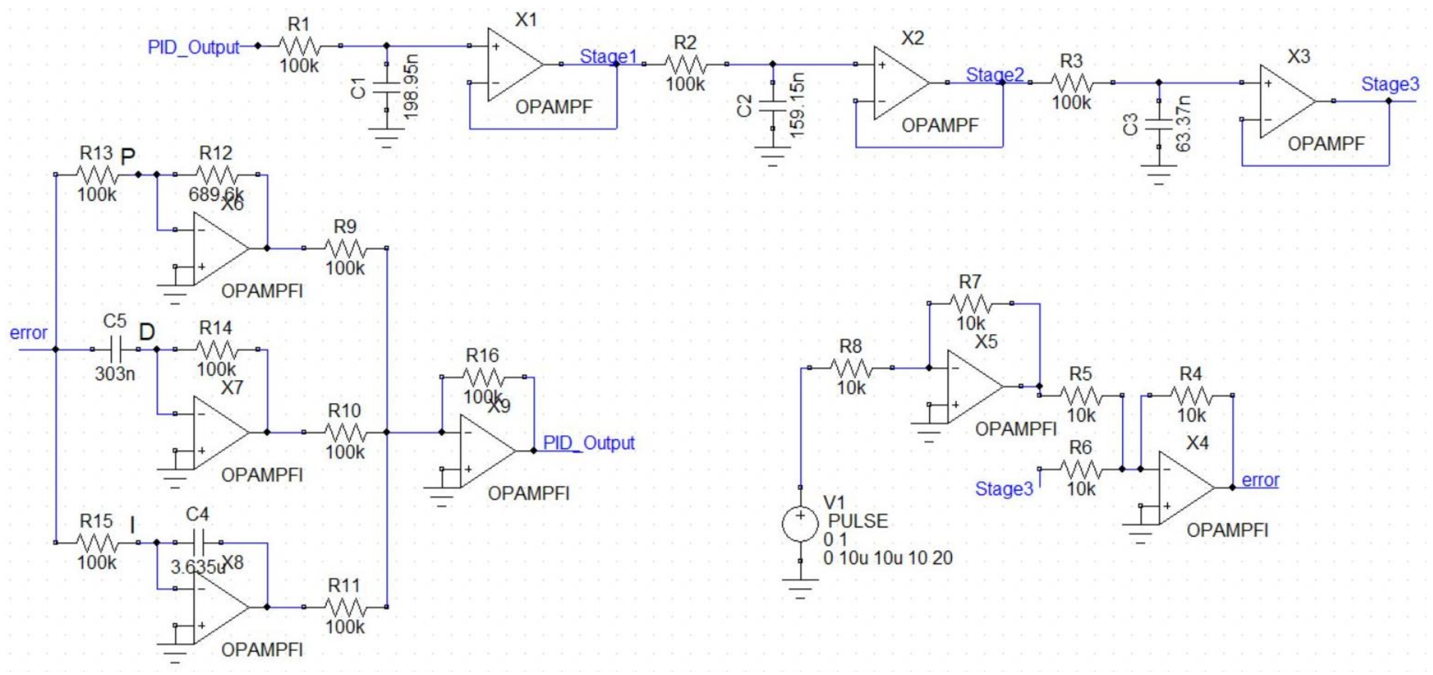
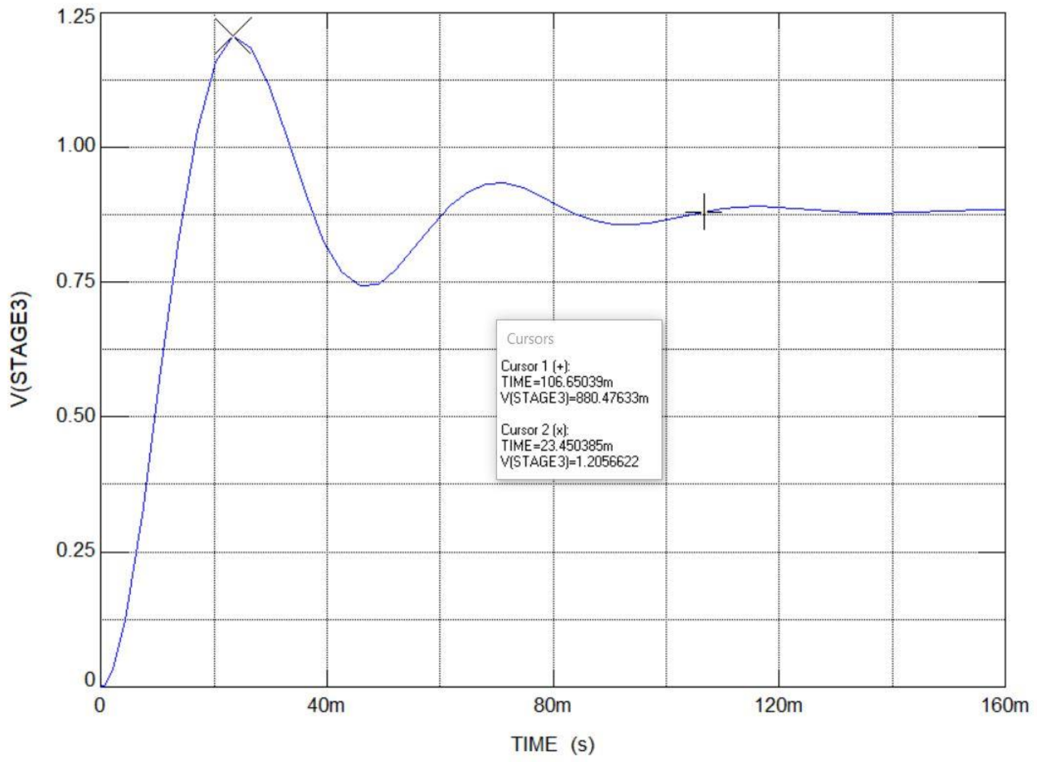


Figure 19: This is my system with the PID controller and subtracter designed in top spice. The system output is at stage 3 of the plant. The PID controller is labeled so that it is easy to tell which segment is proportional, integral, and derivative.



Cursors
Cursor 1 (+):
TIME=106.65039m
V(STAGE3)=880.47633m

Cursor 2 (x):
TIME=23.450385m
V(STAGE3)=1.2056622

Figure 20: This is the short time step response of the system with PID controller in top spice. As you can see the system has a peak time of 23.45m seconds with a voltage of approximately 1.2 volts. The %OS can be

seen by dividing the two voltages. In this figure the %OS is 37%. The results are almost identical to the results obtained in MATLAB.

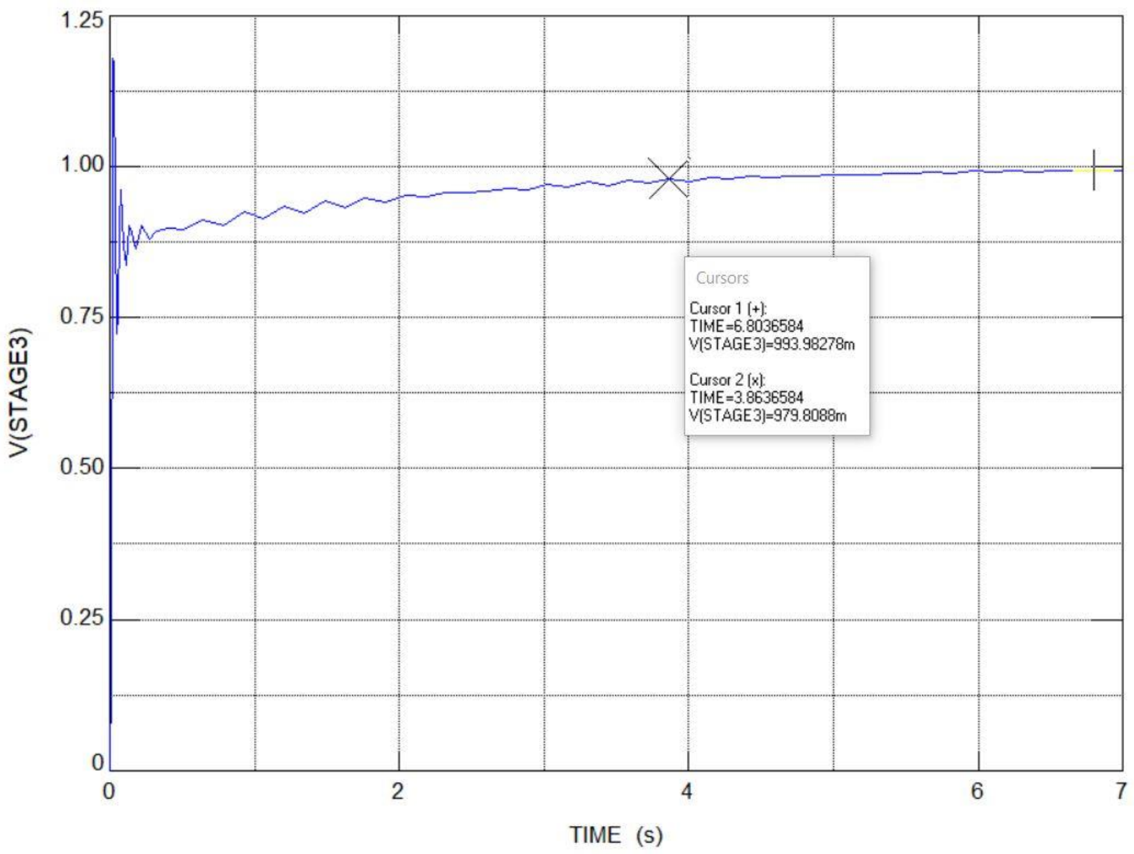


Figure 21: This is the long step response of my system with PID controller in top spice. The time cursors show similar results to MATLAB. Figures 20 and 21 verify the results of my system in MATLAB.

IV.4 PID Tuning

After spending some time researching PID controller tuning methods I was able to find various methods including but not limited to manual tuning, Ziegler-Nichols, Tyreus-Luyben, and Cohen-Coon. All tuning methods include manipulating the PID equation to achieve a desired result. Since the project already required me to adjust the peak time and %OS I've decided to adjust the settling time. I want to decrease my settling time by 50%. According to MATLAB the current settling time of my system (within 2% of the final value) is 4.17 seconds. This means my new settling time will be half of 4.17 which is 2.09.

In class we discussed that adjusting the zero location for the PI term will vary the rate of the systems response. Using the equation for settling time $\boldsymbol{T_s = \frac{4}{R_e}}$ EQ(13) we can set T_s equal to 2.09 and solve for R_e. When doing this I got a value of 1.18 for R_e. Since I want the system to be faster than it currently is value of R_e has to be further way from the jω axis than its current location of s = -0.5. In order to adjust it to the correct value I took the difference between 0.5 and 1.18 then added half of that value to the current 0.5 zero. That gave me a system function that looked like this:

$$G(s) = \frac{(0.0303)*(s+0.84)*(s+227.08)*496097}{s*(s+50.265)*(s+62.832)*(s+157.08)} \text{ EQ(14)}$$

This systems response had a settling time of 2.49 seconds which is closer to the desired 2.09 but not quite close enough. In order to tune the system even more I had to adjust the zero even further from the jω axis. By increasing the zero in increments of 0.02 I was able to achieve the desired result at s = -0.98. The equation is shown below.

$$G(s) = \frac{(0.0303)*(s+0.98)*(s+227.08)*496097}{s*(s+50.265)*(s+62.832)*(s+157.08)} \text{ EQ(15)}$$

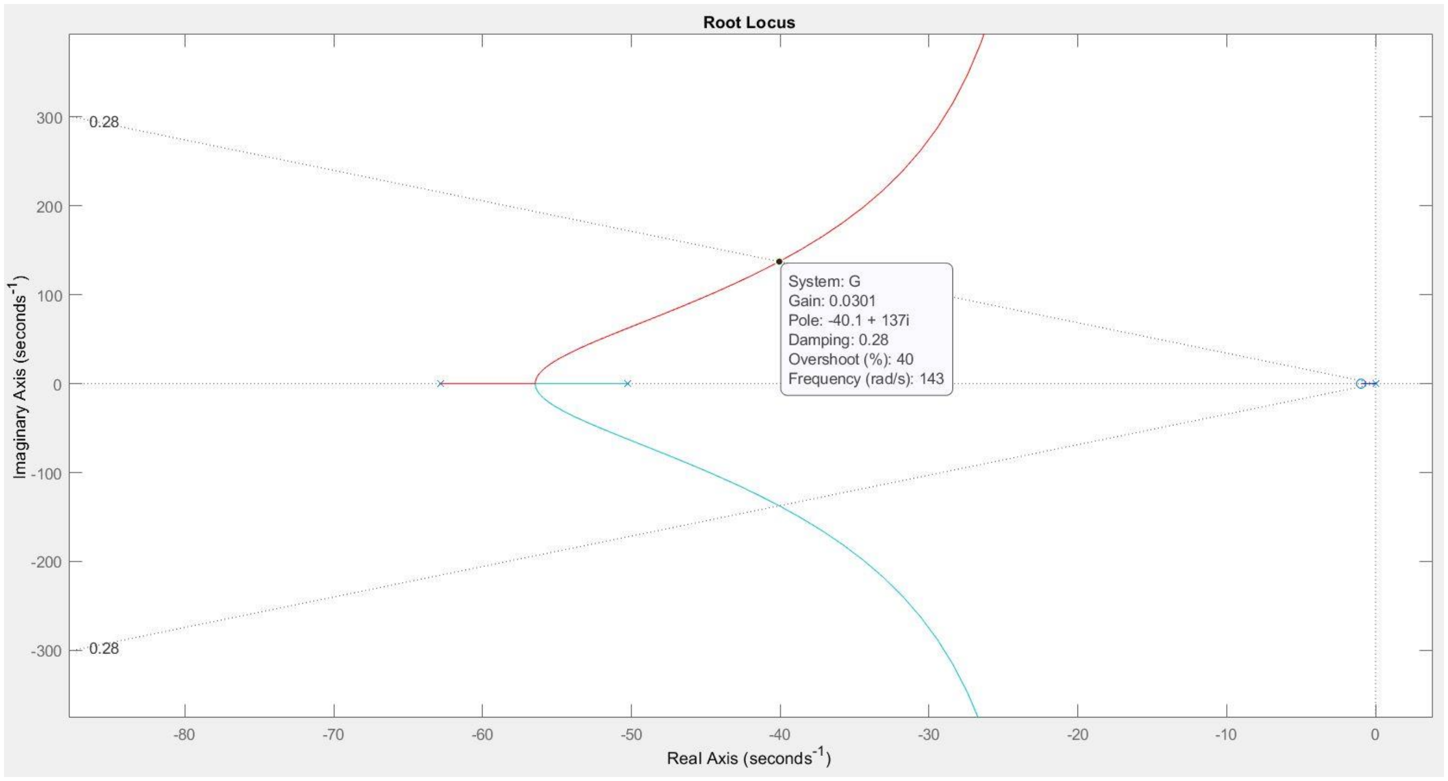


Figure 22: The tuned systems new root locus has poles at **s = -40.1 (+-) j137** which are very close to the untuned systems pole locations. Meaning the %OS and peak times will be very similar as well.

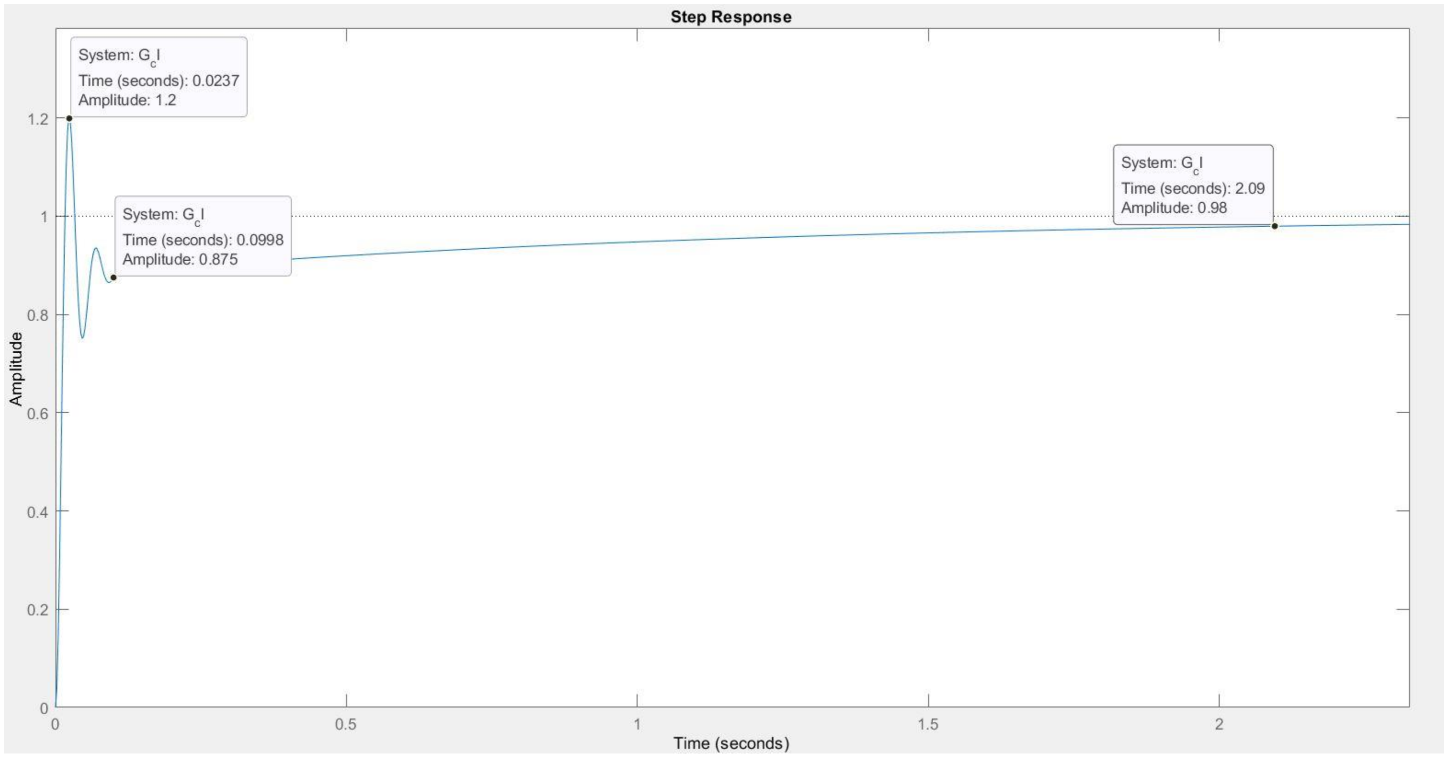


Figure 23: After tuning my PID controller the step response results are shown above. From the values labeled you can see the system has an improved T_s that is 2.09s. Also the %OS of this system is close to the target value of 40%. The %OS of this system is 37.1% after tuning.

Using the equation shown to complete table 4 I incorporated the new PI zero and derived the new gain values. $G_{PID}(s) = \frac{0.0303*(s+0.98)*(s+227.08)}{s} = \frac{0.0303*(s^2+228.06s+222.54)}{s}$ EQ(16). The coefficients from this equation are used as the k values for the op amps in the top spice circuit.

Table 5: Gain and Component Calculation				
Component	**Equation**	**Gain**	**(Resistor/Capacitor)$_s$**	**(Resistor/Capacitor)$_f$**
Proportional	$K_1 = \frac{-R_f}{R_s}$	6.91	100k ohms	691k ohms
Integral	$K_2 = \frac{1}{RC}$	6.743	100k ohms	1.483 µF
Derivative	$K_3 = RC$	0.0303	303 nF	100k ohms

Table 5: This table shows the values that correspond to the tuned PID system. The K values are the gains of the OP Amps for the Proportional, Integral, and Derivative portions of the circuit.

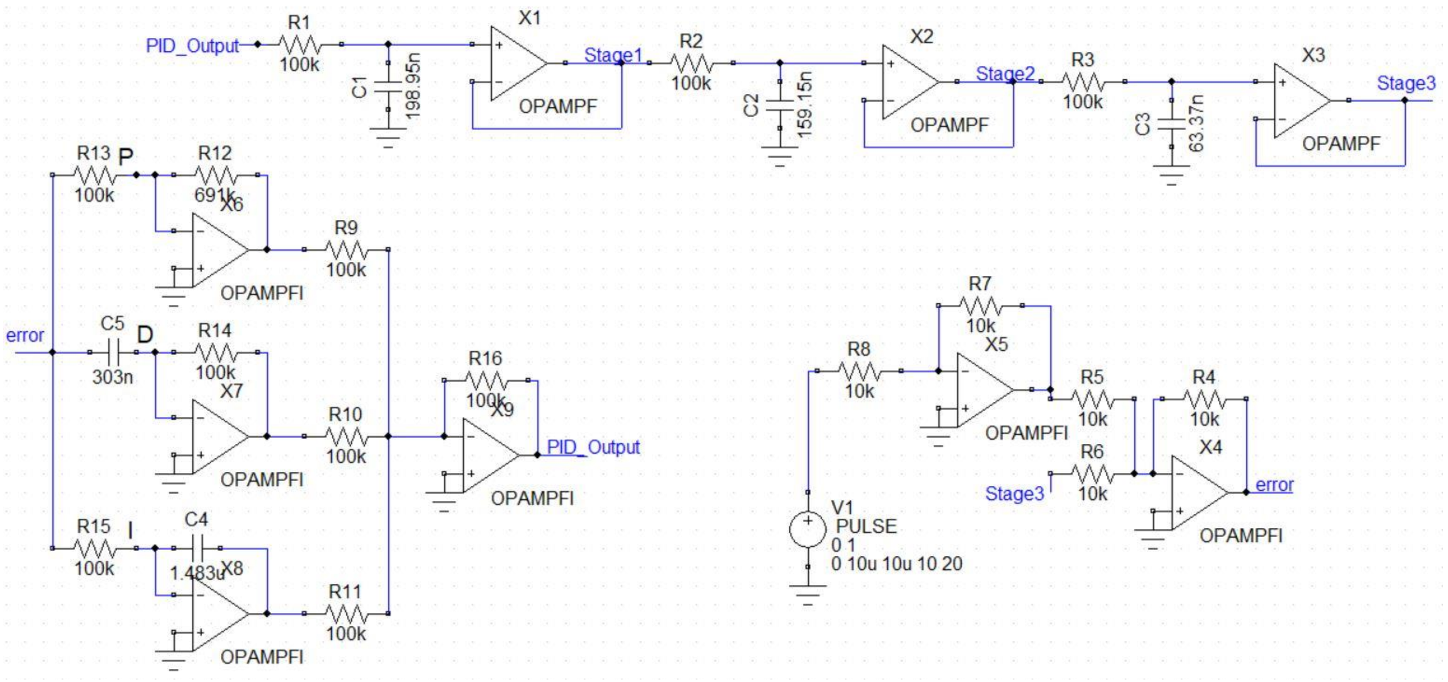


Figure 24: This is the schematic in top spice of my tuned system. You can see the gain values have been compensated for since the P and I portions of my controller have different component values.

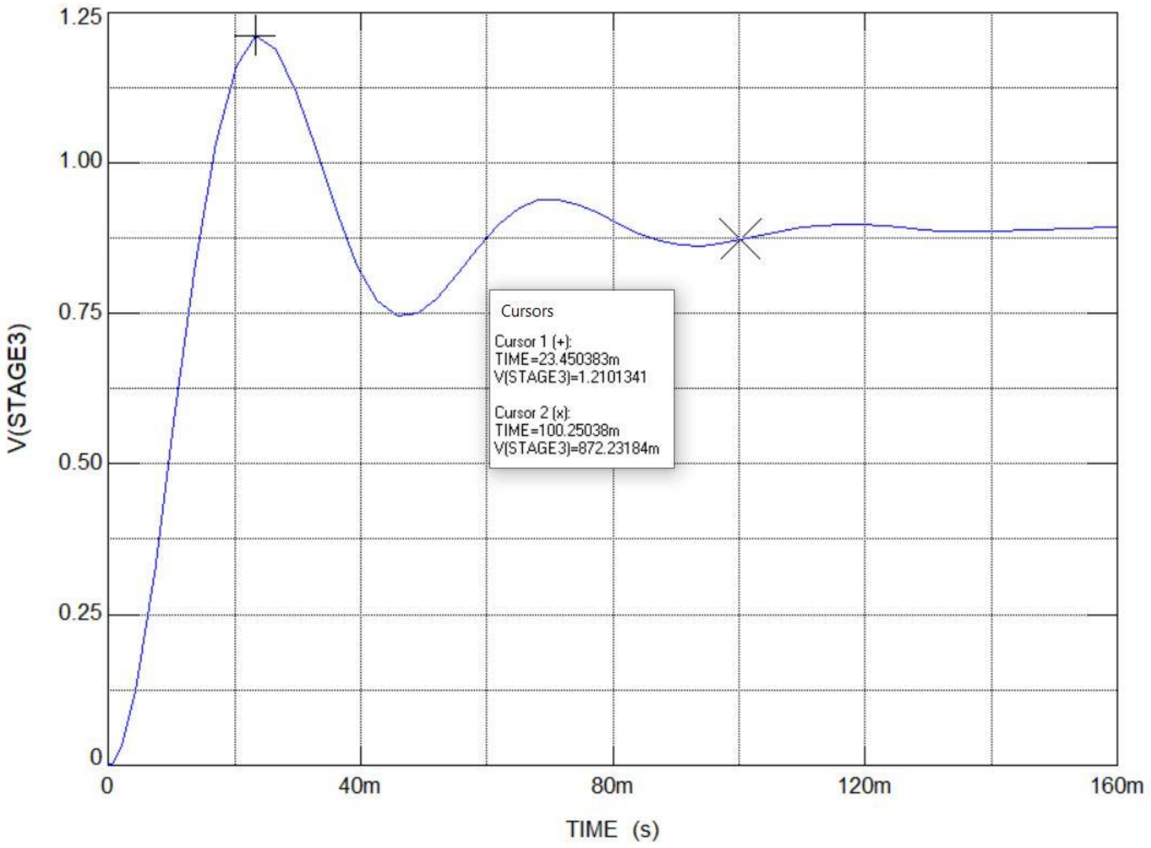


Figure 25: This is my tuned system's step response in top spice. From this short time span figure you can tell the peak value is 1.21 volts giving the systems a %OS of 38.7%.

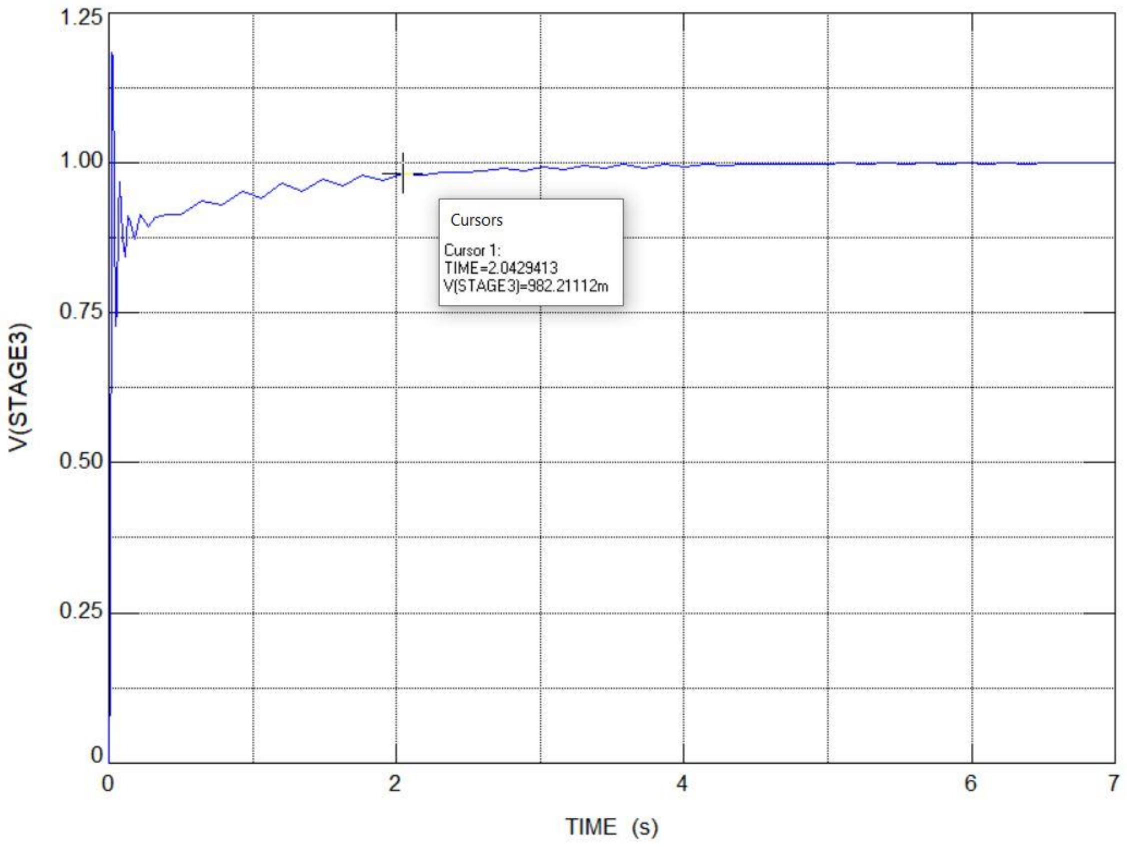
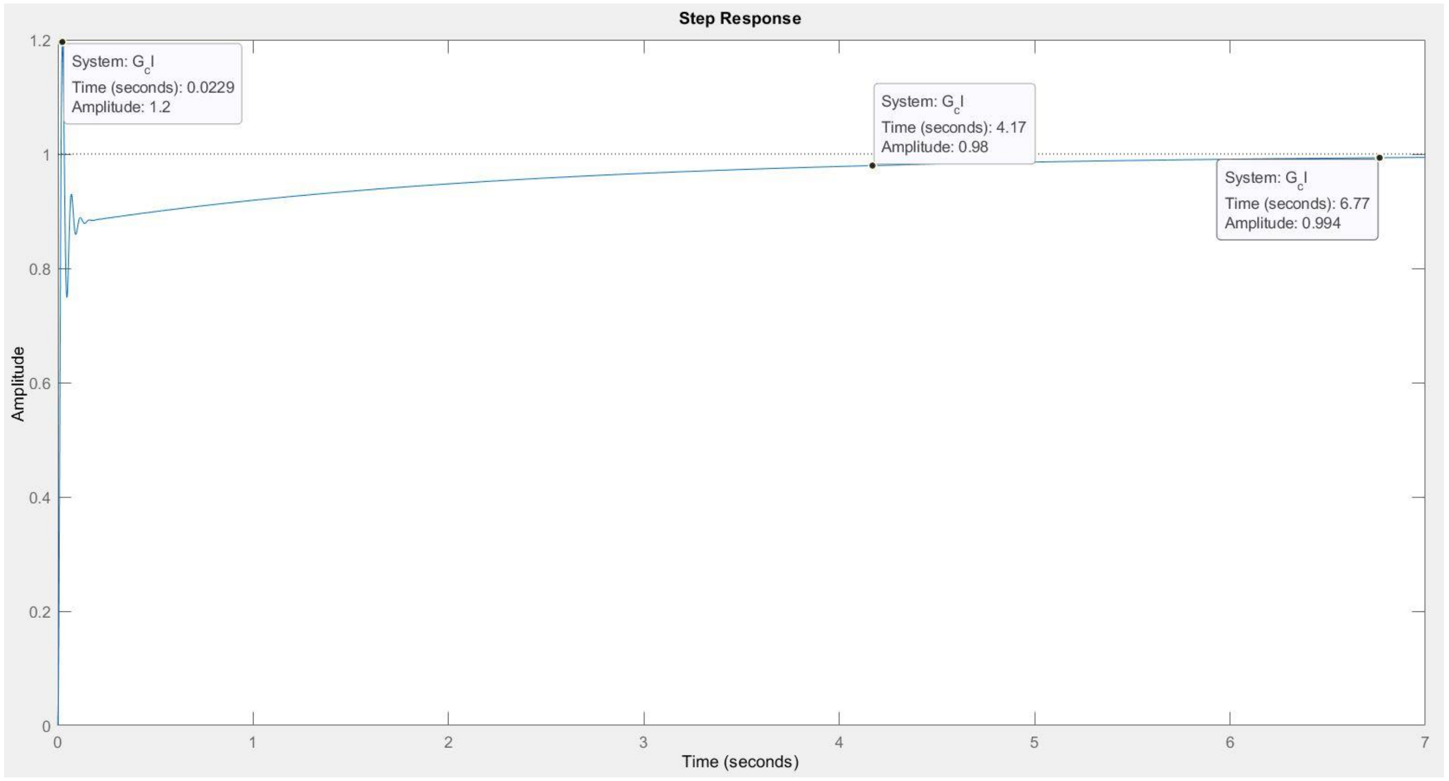
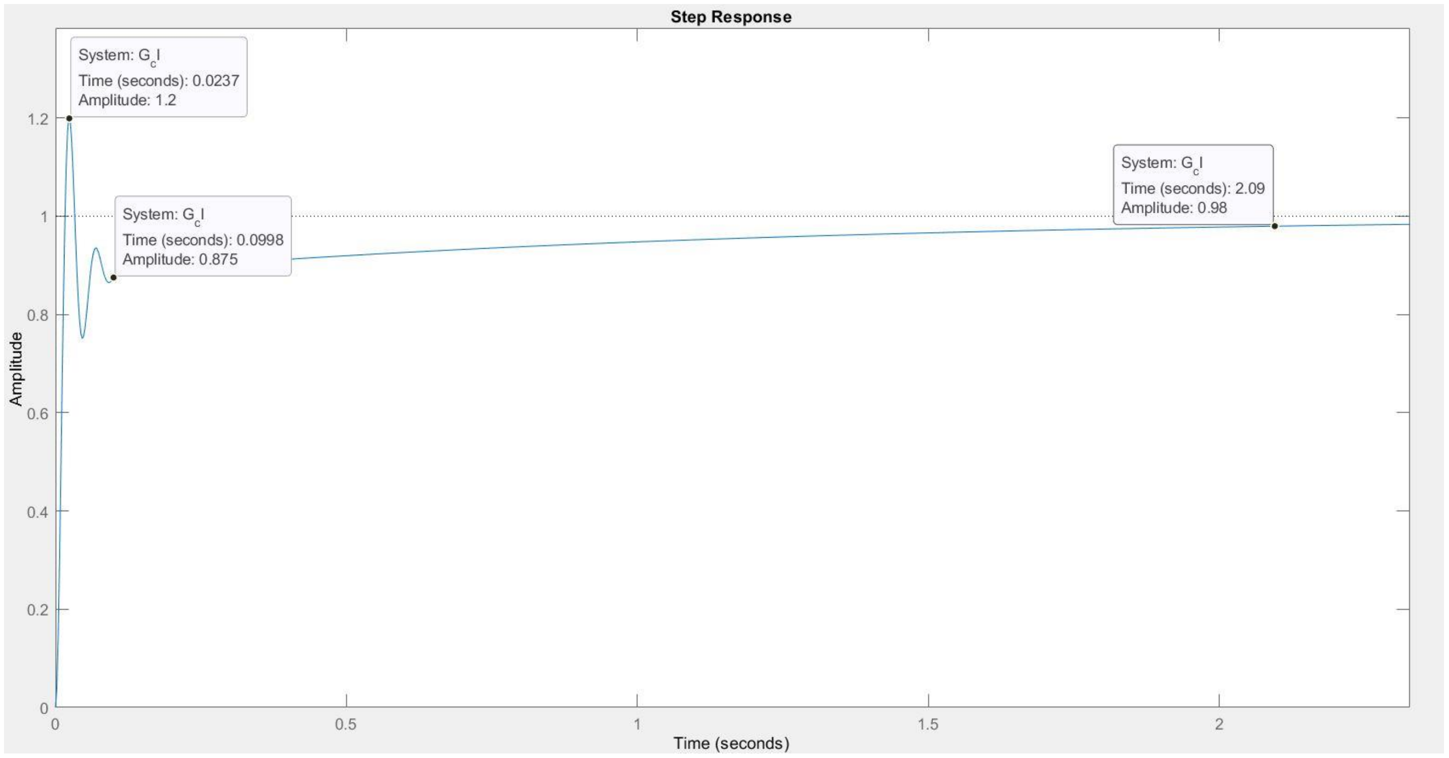


Figure 26: The system has a settling time of 2.04 seconds in top spice.

IV.5 PID Performance Comparison

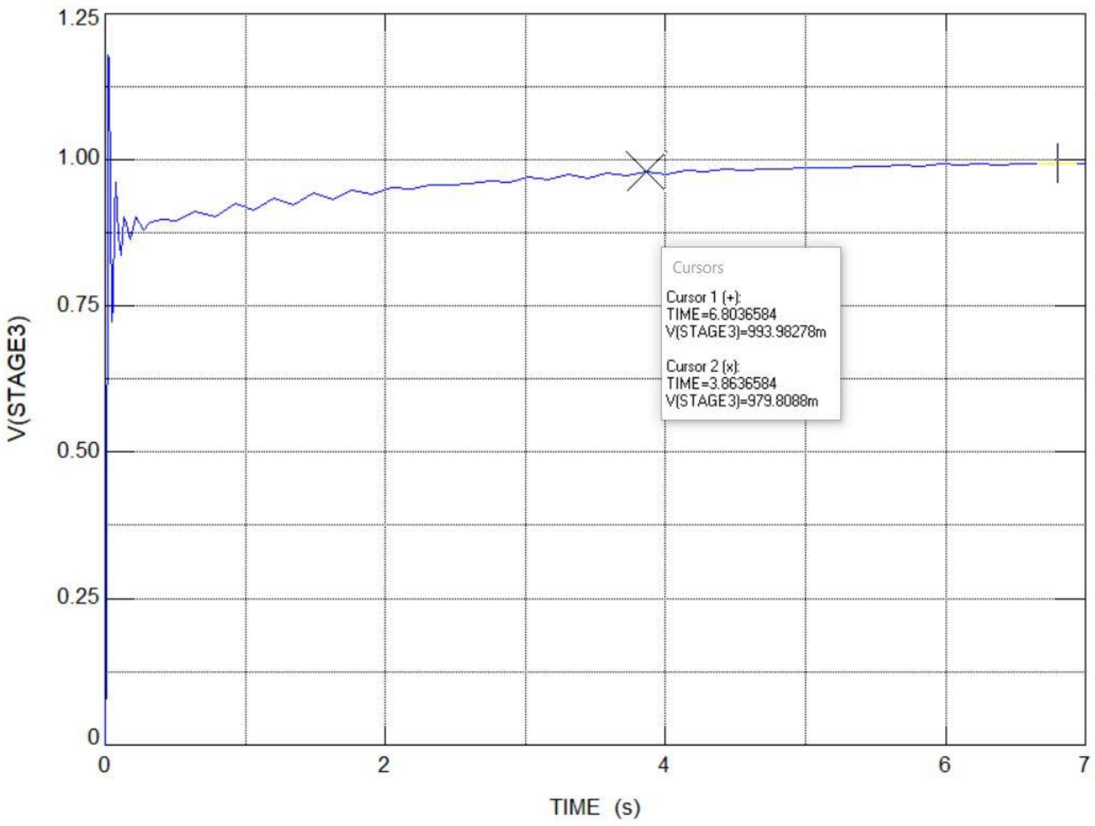
Original PID is on the top and tuned PID is on the bottom.





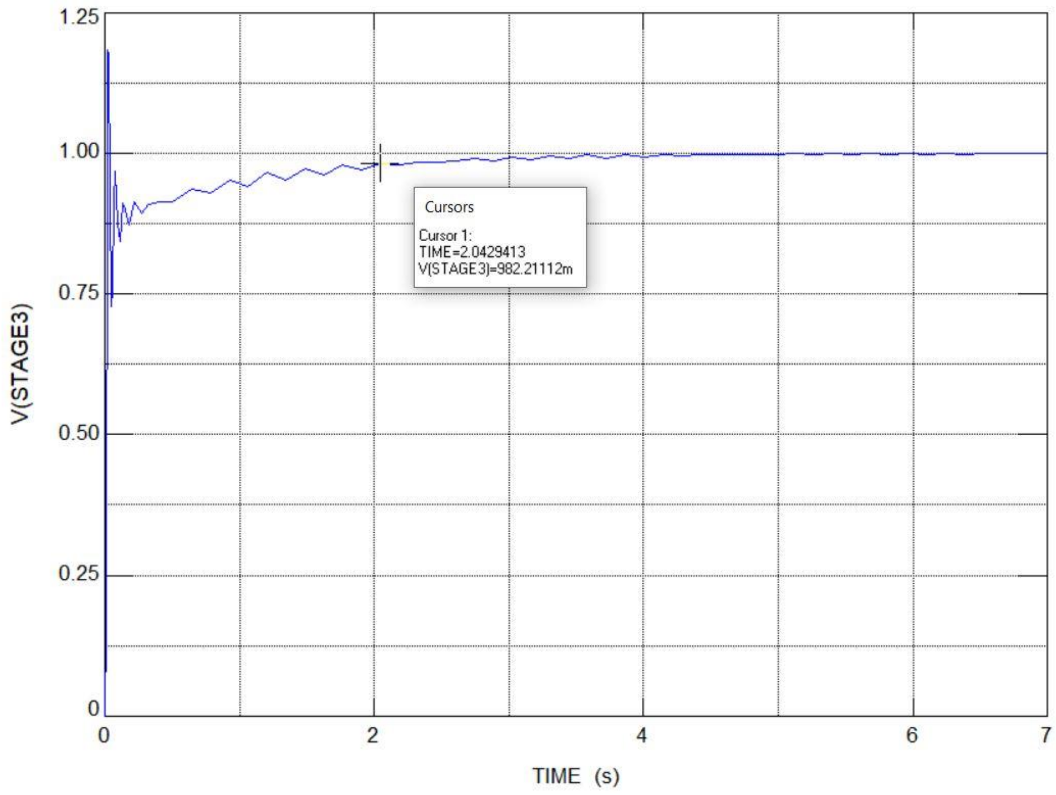
Figures 27 and 28: The data tips show that the peak value and times are virtually the same for both the tuned and untuned systems. Where the voltage reaches 0.98v is where the systems settling time is and as you can see the tuned system has half the settling time as the untuned. Meaning I was successful in tuning my system.

Original PID is on the top and tuned PID is on the bottom.



Cursor 1 (+):
TIME=6.8036584
V(STAGE3)=993.98278m

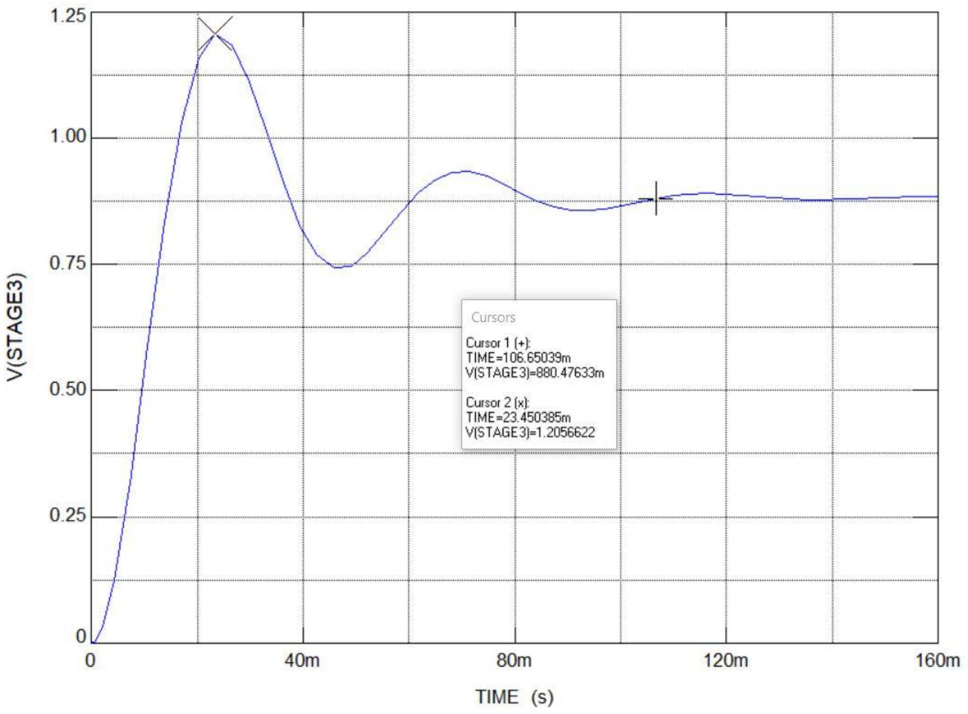
Cursor 2 (x):
TIME=3.8636584
V(STAGE3)=979.8088m



Cursor 1:
TIME=2.0429413
V(STAGE3)=982.21112m

Figures 29 and 30: Here you can see the systems have different settling times. The original PID has a settling time of 3.86s while the tuned system has a settling time of 2.04s. The goal was to cut settling time in half. In this case the tuned PID is 1.89 times faster than the untuned system.

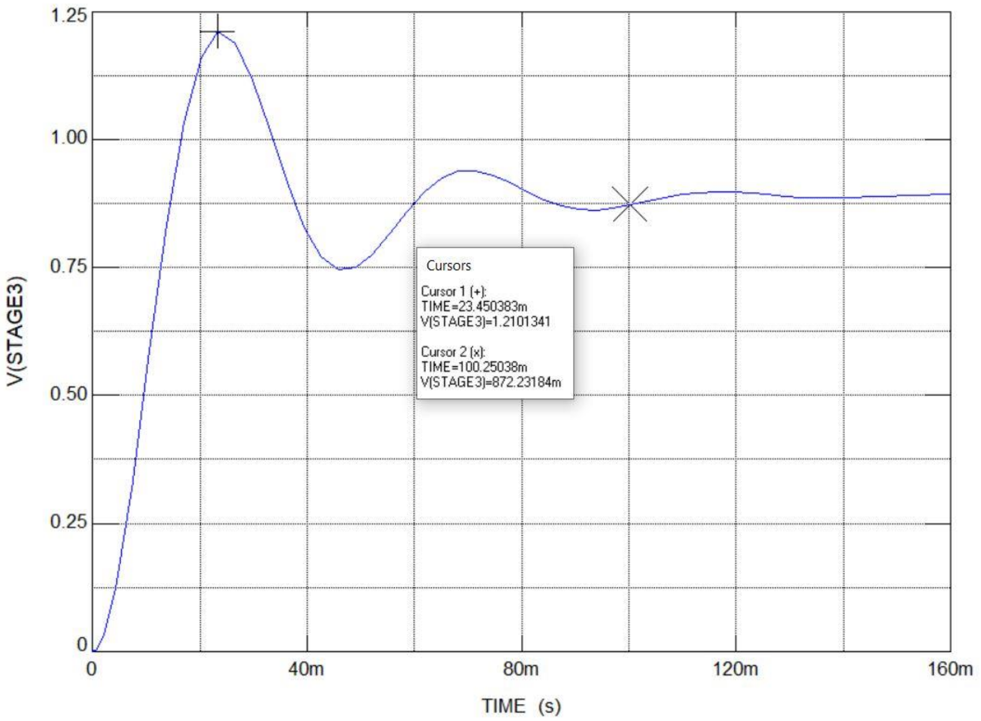
Original PID is on the top and tuned PID is on the bottom



Cursors

Cursor 1 (+):
TIME=106.65039m
V(STAGE3)=880.47633m

Cursor 2 (x):
TIME=23.450385m
V(STAGE3)=1.2056622



Cursors

Cursor 1 (+):
TIME=23.450383m
V(STAGE3)=1.2101341

Cursor 2 (x):
TIME=100.25038m
V(STAGE3)=872.23184m

Figures 31 and 32: The peak values for the untuned and tuned PID systems are 1.205v and 1.21v respectively. These graphics also show the %OS's for the systems are 36.9% for the untuned and 38.7% for the tuned PID. The tuned system has a closer value to the target 40% OS and has a 3.3% margin of error. The tuned system has a more accurate faster performance than the untuned system.

PID Performance Comparison				
	MATLAB Original	MATLAB After Tuning	SPICE Original	SPICE After Tuning
T_p (s)	0.0229	0.0237	0.0235	0.0235
T_s (s)	4.17	2.09	3.86	2.04
%OS	38%	37.1%	36.9%	38.7%

Table 6: In both cases the settling time became nearly half which was the goal of the tuning. The peak time was almost unchanged in both cases. In the MATLAB simulation the %OS decreased after tuning while in the Spice simulation the %OS increased after tuning. If the application required better peak time or %OS the gain could be increased in both cases to reach a desired goal.

V. Conclusions and Discussion

V.1 PID control has many advantages. A controlled system will help eliminate any error between a desired result and an actual result. This is good for systems because it minimizes resources while decreasing any extra wear on equipment. Also, it takes the human element out of having to control a system which can minimize error or hazards. One example that comes to mind is SpaceX being able to reuse their rocket thrusters. Using feedback systems to land a rocket thruster, which is a job much to dangerous for a person, saves the company millions of dollars on equipment costs. For open loop systems this is not possible because feedback is not incorporated meaning the system is going to preform the same task no matter the workload or environmental conditions.

The disadvantages of having a PID control are cost in terms of equipment complexity and making sure the system is calibrated properly. PID controllers have more hardware components making them more expensive than most open loop systems. It is also important to make sure the system is calibrated properly. If a sensor is feeding information to a controller the controller better have the right output to the plant so that it minimizes error instead of maximizes it.

V.2 Some of the PID tuning methods I researched included manual tuning, Ziegler-Nichols, Tyreus-Luyben, and Cohen-Coon. During this project I used a manual tuning method by adjusting the existing PI zero to achieve a faster system. The tuning method worked very good for this project considering my final results which are in table 6.

PID Performance Comparison

	MATLAB Original	MATLAB After Tuning	SPICE Original	SPICE After Tuning
T_p (s)	0.0229	0.0237	0.0235	0.0235
T_s (s)	4.17	2.09	3.86	2.04
%OS	38%	37.1%	36.9%	38.7%

Table 6: In both cases the settling time became nearly half which was the goal of the tuning. The peak time was almost unchanged in both cases. In the MATLAB simulation the %OS decreased after tuning while in the Spice simulation the %OS increased after tuning. If the application required better peak time or %OS the gain could be increased in both cases to reach a desired goal.

V.3 The results of my step responses for MATLAB and Top Spice are shown in table 6 above.

Percent Error Between MATLAB and SPICE

	Before Tuning	**After Tuning**
T_p	2.6%	0.8%
T_s	8%	2.5%
%OS	3%	4.3%

Table 7: This graph shows the percent error between the two programs before and after tuning.

I believe the difference in data comes from each of their abilities to place data points. In MATLAB it was easy to place a data point at an exact value but in SPICE it had larger gaps between data point locations. The differences could also come from the fact that I had to round off the resistor and capacitor values before designing my schematic in SPICE. MATLAB has more accurate data because I based my desired settling time before tuning off of the MATLAB data.

V.4 A digital implementation would have to have a sampling frequency high enough to accurately evaluate the output of your system at all speeds. That being said, the faster the plant is operating the worse the approximations would be for the digital PID. Depending on the band and sampling frequency of your digital PID it may be more desirable for some systems than others. The analog PID would respond better to the modularity of the output due to its physical components but may be more complex to design depending on how the physical components are put together.

V.5 I learned why PID controllers are so desirable for systems as far a performance and conservation of resources goes. Also, I found it very interesting in learning all the ways to change a systems performance just by manipulating the root locus. I did not know how the physical circuitry of a PID controller looked before doing this project and it was surprisingly simple! Just a few op amps, resistors, and capacitors can turn a good system into a great one. I know it was not used in this project necessarily, but I also find it interesting how I now know how to derive a transfer function from a mechanical system and build a PID controller for the system. Very interesting stuff, I hope to use it in my career one day.