EE 4360 Final Project
Design & Simulation of a Tuned PID Controller

Fall, 2024

Brandon Markham A05191661

Contents

Section I: Description

PID controllers are feedback control loops widely used in industrial control systems to maintain desired performance. They continuously calculate the error, defined as the difference between a setpoint (the desired value) and a measured variable, and apply corrections using proportional, integral, and derivative actions. These controllers excel in managing processes with relatively linear behavior and predictable dynamics, such as maintaining temperature, regulating motor speed, or controlling position. However, they may struggle with highly non-linear, time-varying, or rapidly changing systems. A notable limitation is their sensitivity to noise, especially in the derivative term, which can amplify high-frequencies and lead to instability.

In this design project, I successfully implemented a PID controller that met or exceeded all target specifications. Through careful pole placement and systematic tuning using the Ziegler-Nichols method, we achieved a significantly improved system response. The final design demonstrated excellent performance, with a time to peak (Tp) of 26.49 ms, representing a 93.75% improvement over the target of 72.16 ms. The system maintained a percent overshoot of 9.18%, well within the specified maximum of 9.48%. The settling time (Ts) was optimized to 142.93 ms, and the final value achieved unity gain with perfect steady-state accuracy. Both MATLAB simulations and LTSPICE circuit implementations showed remarkable agreement, with final differences of less than 1.5% across all performance parameters, validating the robustness of our design. A table with the results can be seen below,

Table 0: PID Performance Comparison
MATLAB vs. LTSPICE Circuit Simulator

Parameter	Target	MATLAB	LTSPICE	%Δ
Tp	72.16ms	26.1 ms	26.49 ms	1.48%
Ts		143 ms	142.93 ms	-0.05%
%OS	≤ 9.48%	9.18%	9.18%	0%
FV		1.0	1.0	☑

Section II: Plant Poles

For this section of the design process the poles were verified using a cross-analysis method. Two Bode plots were generated. One using the Matlab Bode plot of the given transfer function and another Bode plot was obtained by building an equivalent circuit with op-amps, resistors and capacitors using the LTSPICE circuit simulator. The table below shows the project parameters along with the overshoot and improvement goals of the system to be designed.

Table 1: Project Parameters	
Parameter	**Goal**
Pole 1 (35)	35 rad/s || 5.57 Hz
Pole 2 (55)	55 rad/s || 8.75 Hz
Pole 3 (75)	75 rad/s || 11.94 Hz
Tp % Improvement	25%
%OS	9.48%

Table 1: Table listing the poles, their corresponding frequencies, and design goals for the PID

II.i: Transfer Functions

From the poles given by the instructor for the PID design prompt the following transfer functions represented by equations 1 and 2 were obtained.

$$G(s) = \frac{(35)(55)(75)}{(s+35)(s+55)(s+75)} \qquad [1]$$

$$\mathrm{G(s)} = \frac{144375}{(s+35)(s+55)(s+75)} \qquad [2]$$

II.ii: Frequency Conversion

Once the transfer functions were obtained the units of the poles were converted from ${}^{rad}/_{s}$ to frequency in Hz by using equation 3. Listed below are the poles and their respective frequencies after the conversion.

$$\mathrm{Hz} = \frac{\mathrm{rad/s}}{2\pi} \qquad [3]$$

From equation 3

$$P1 = 35\ {}^{rad}/_{s} \rightarrow P1 = 5.57\ Hz$$

$$P2 = 55\ {}^{rad}/_{s} \rightarrow P2 = 8.75\ Hz$$

$$P3 = 75\ {}^{rad}/_{s} \rightarrow P3 = 11.94\ Hz$$

II.iii: Magnitude Measurements

For both simulation's magnitudes were measured at three points. To obtain these points the following equations were used

$$P1 \div 2 \qquad [4.1]$$

$$P2 + \frac{P3-P2}{2} \qquad [4.2]$$

$$P3\ \times 2 \qquad [4.3]$$

1. Half the frequency of the lowest pole obtained using equations 3 and 4.1 yielded $\frac{35}{2} {rad}/_{s} = 2.79\ Hz.$

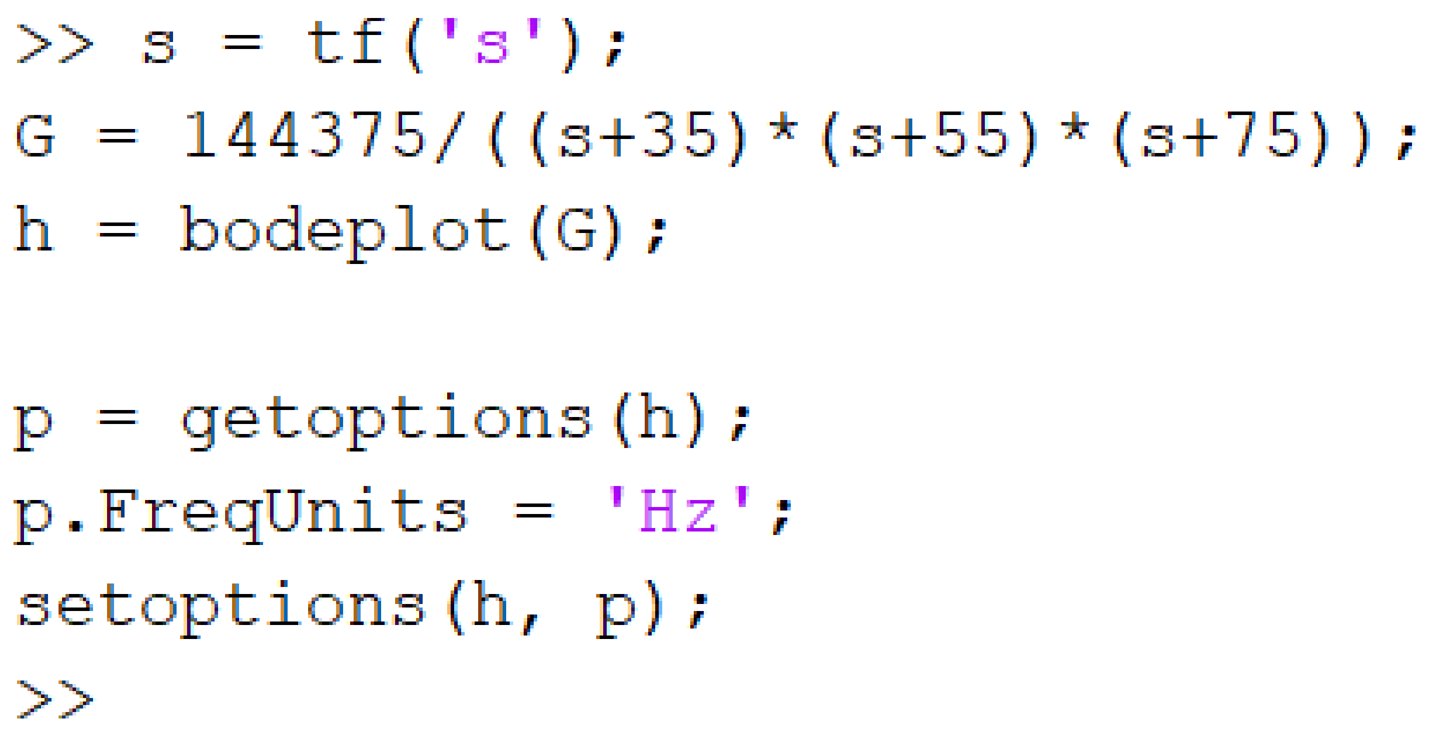
2. The Mid way between the 2nd and highest pole obtained using equations 3 and 4.2 yielded $55 + \frac{75-55}{2} = 65 {rad}/_{s} = 10.4\ Hz.$

3. Twice the frequency of the highest pole obtained using equations 3 and 4.3 yielded $75\ \times 2 = 150 {rad}/_{s} = 23.9\ Hz.$

If the magnitudes at these three points match between the Matlab and circuit simulator Bode plots, then the poles are correct.

II.iv: Matlab Bode Plots

The following code was used to generate a Bode plot in Matlab.



```
>> s = tf('s');
G = 144375/((s+35)*(s+55)*(s+75));
h = bodeplot(G);

p = getoptions(h);
p.FreqUnits = 'Hz';
setoptions(h, p);
>>
```

The Bode plots generated as well as the data tips used to measure their respective magnitudes are depicted in Figure 1, Figure 2, and Figure 3.

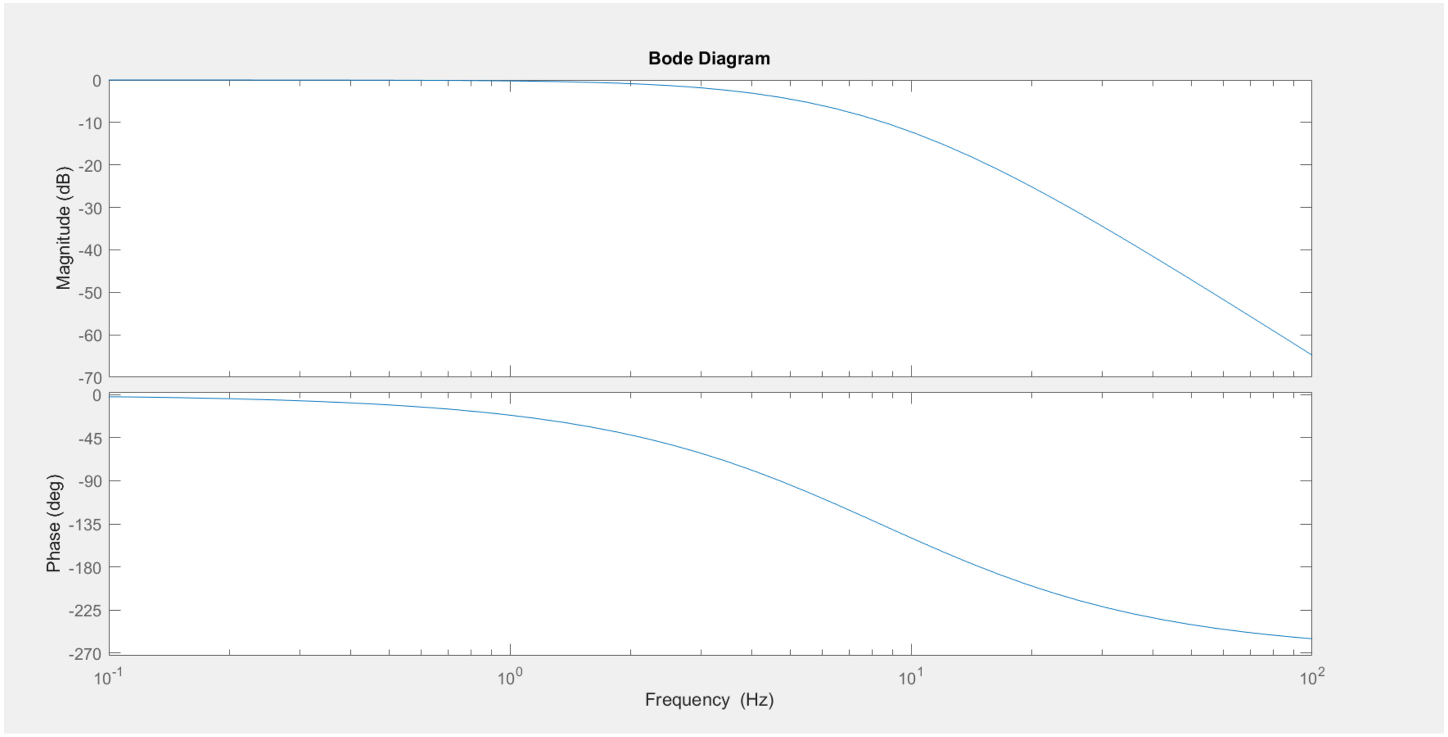


Figure 2: Matlab Bode plot of Plant.

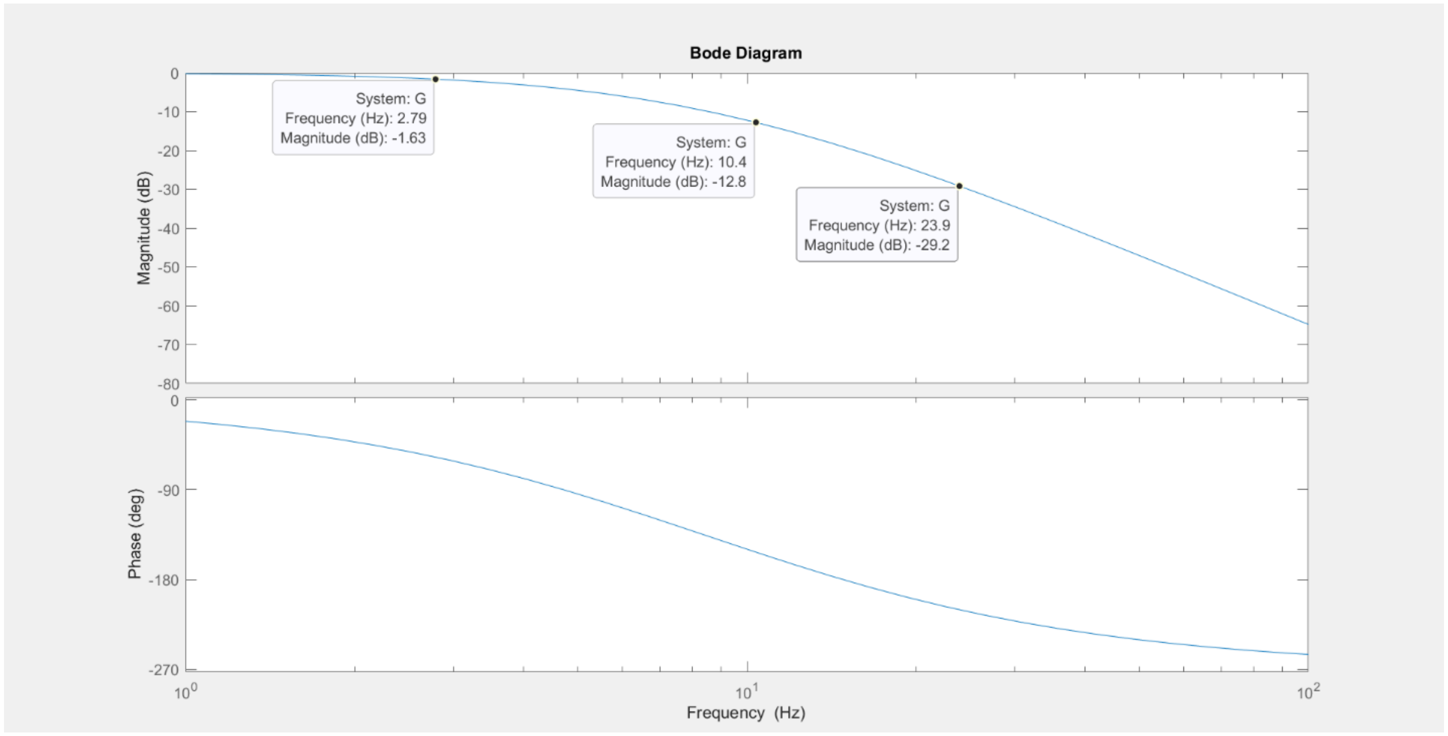


Figure 2: Matlab Bode plot of Plant with data tips at 2.79Hz measuring -1.63 dB, 10.40 Hz measuring -12.8 dB , and 23.9 Hz measuring -29.2 dB.

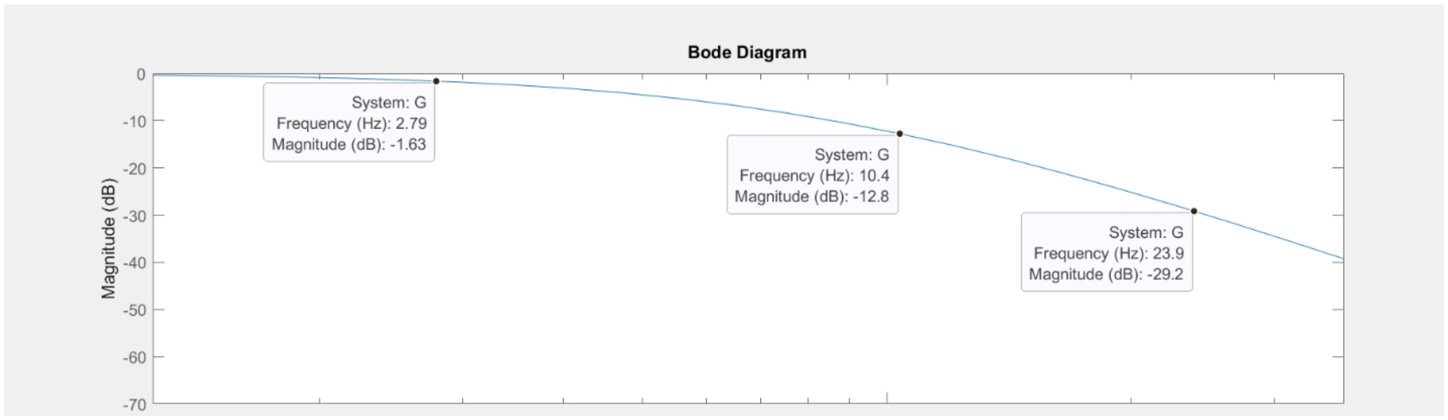


Figure 3: Close up of Matlab Bode plot of Plant with data tips at 2.79Hz measuring -1.63 dB, 10.40 Hz measuring -12.8 dB , and 23.9 Hz measuring -29.2 dB.

II.v: Circuit Simulator Bode Plots

For the second part of the analysis the poles were modeled in LTSPICE circuit simulator using op-amps, resistors and capacitors. Each pole is real with a magnitude of $\frac{1}{RC}$, on the negative real axis. A resistor of 10 kΩ was chosen and calculations for capacitance were derived using equations 5.a-c.

$$P1\ is\ 35\ {}^{rad}/_{s}\ so\ C1 = \frac{1}{(10k \times P1)} = 2.8571\ uF \qquad [5.a]$$

$$P2\ is\ 55\ {}^{rad}/_{s}\ so\ C1 = \frac{1}{(10k \times P2)} = 1.8181\ uF \qquad [5.b]$$

$$P3\ is\ 75\ {}^{rad}/_{s}\ so\ C1 = \frac{1}{(10k \times P3)} = 1.3333\ uF \qquad [5.c]$$

The circuit was constructed in the simulator and is depicted below in Figure 4.

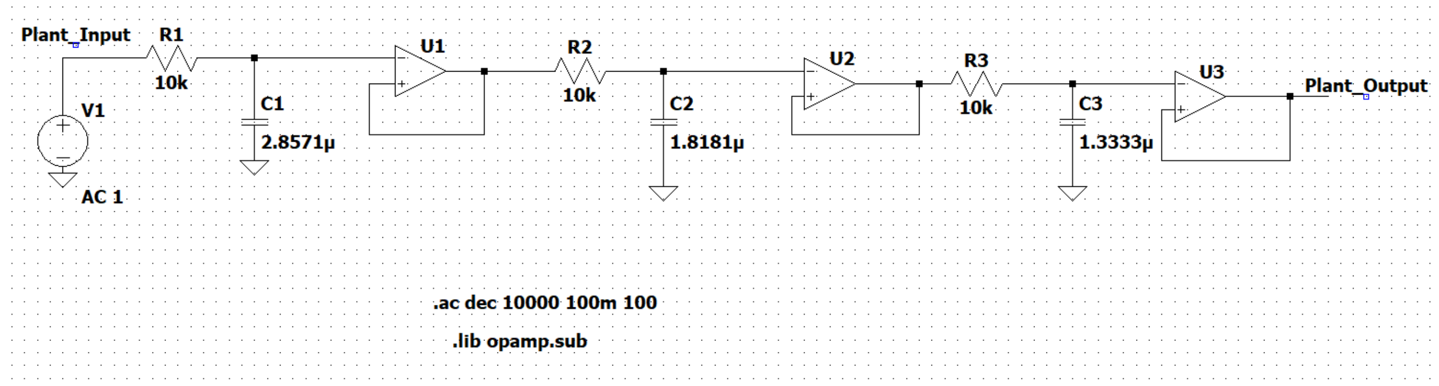
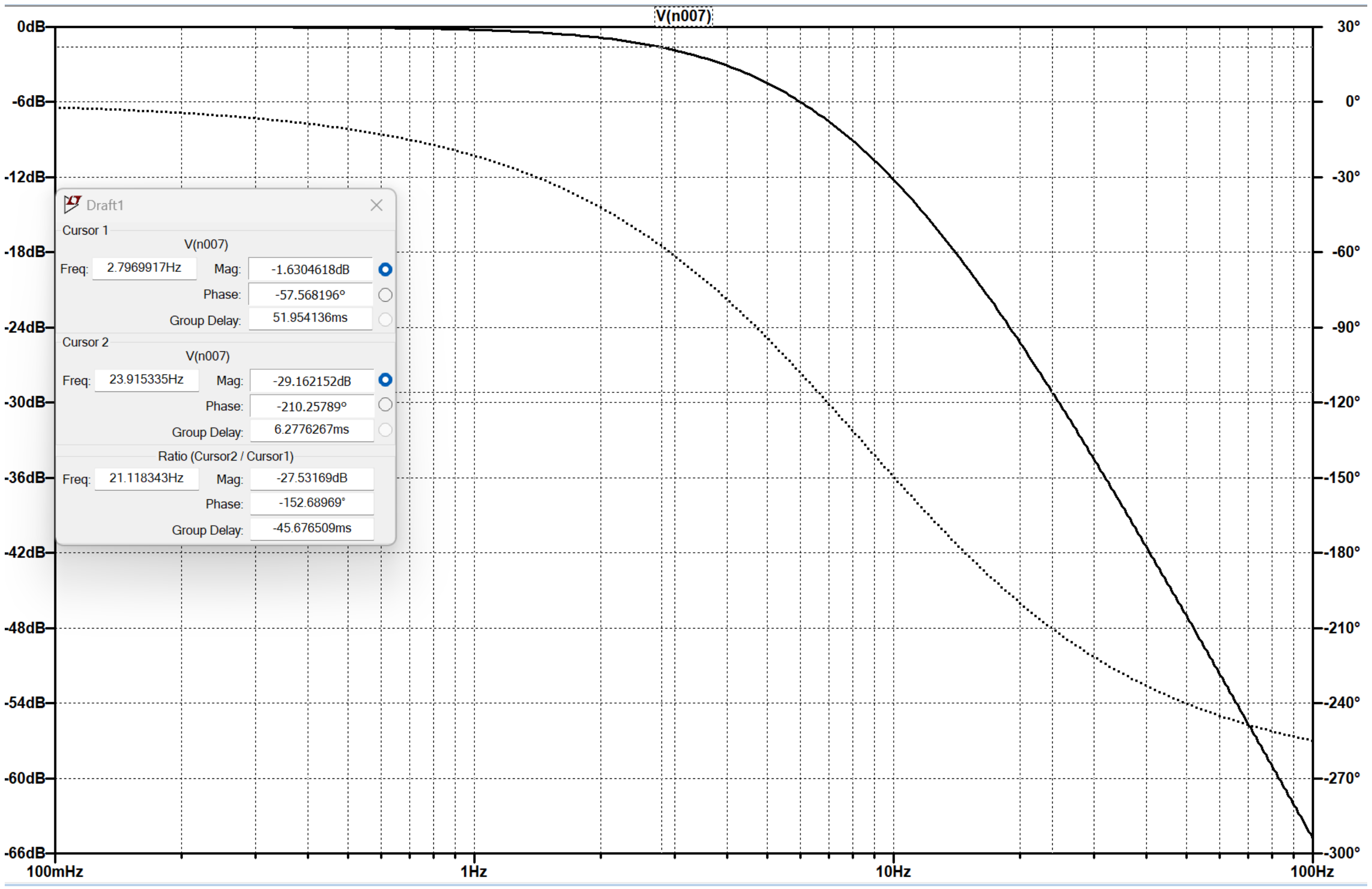


Figure 4: LTSPICE Circuit Simulation with 1V AC source connected to the plant.

To make the bode plots easier to read the figures containing the magnitudes were broken up into two separate figures. Figure 5.a contains the half the lowest frequency (2.79 Hz) and twice the highest frequency (23.9 Hz). Figure 5.b contains the midpoint between Pole 2 and Pole 3 (10.4 Hz).



Cursor 1

V(n007)

Freq: 2.7969917Hz Mag: -1.6304618dB

Phase: -57.568196°

Group Delay: 51.954136ms

Cursor 2

V(n007)

Freq: 23.915335Hz Mag: -29.162152dB

Phase: -210.25789°

Group Delay: 6.2776267ms

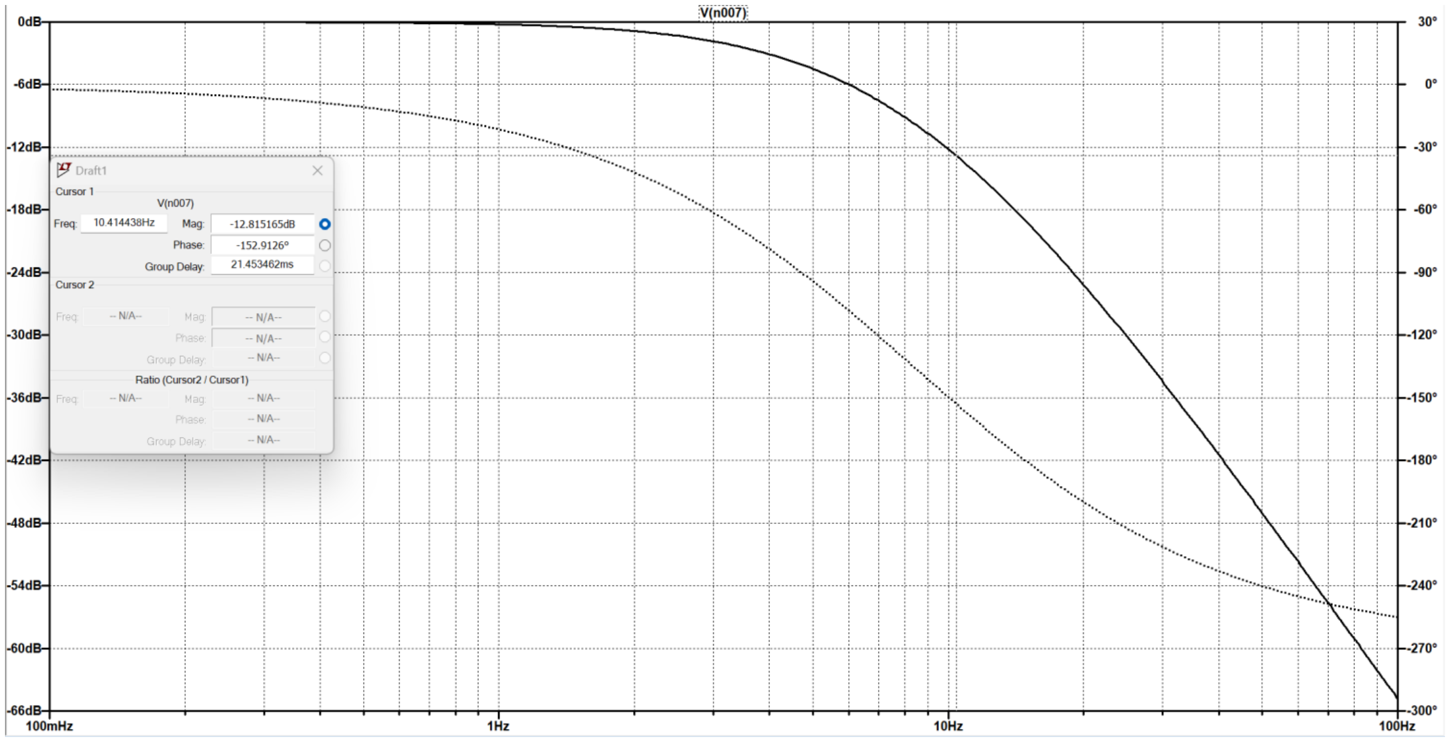
Ratio (Cursor2 / Cursor1)

Freq: 21.118343Hz Mag: -27.53169dB

Phase: -152.68969°

Group Delay: -45.676509ms

Figure 5A: LTSPICE Bode Plot with cursor 1 placed at 2.79 Hz measuring -1.63 dB and cursor 2 placed at 23.9 Hz measuring -29.16 dB.



Cursor 1

V(n007)

Freq: 10.414438Hz Mag: -12.815165dB

Phase: -152.9126°

Group Delay: 21.453462ms

Cursor 2

Freq: -- N/A-- Mag: -- N/A--

Phase: -- N/A--

Group Delay: -- N/A--

Ratio (Cursor2 / Cursor1)

Freq: -- N/A-- Mag: -- N/A--

Phase: -- N/A--

Group Delay: -- N/A--

Figure 5B: LTSPICE Bode Plot with cursor 1 placed at 10.4 Hz measuring -12.81 dB.

II.vi: Matlab vs. Simulator Comparison

After both Bode plots were generated and all three test magnitudes were measured on both plots, a percent difference calculation was performed to verify that the poles on our plant are correct. The difference found was negligible and within tolerance of expected differences that are to be anticipated using such analysis techniques thus validating that our poles of our plant are correct. The table containing these differences can be found below in Table 2.

Table 2: Magnitude Comparison, Matlab vs. Simulator			
Frequency, Hz	**Hz MATLAB Magnitude, dB**	**Simulator Magnitude, dB**	**% Δ**
2.79 Hz (35 rad/s)	-1.63	-1.63	0.00 %
10.4 Hz (55 rad/s)	-12.8	-12.81	0.08 %
23.9 Hz (75 rad/s)	-29.2	-29.16	0.13 %

Table 2: Percent Difference calculations between Bode plot magnitudes for Matlab and Circuit simulation.

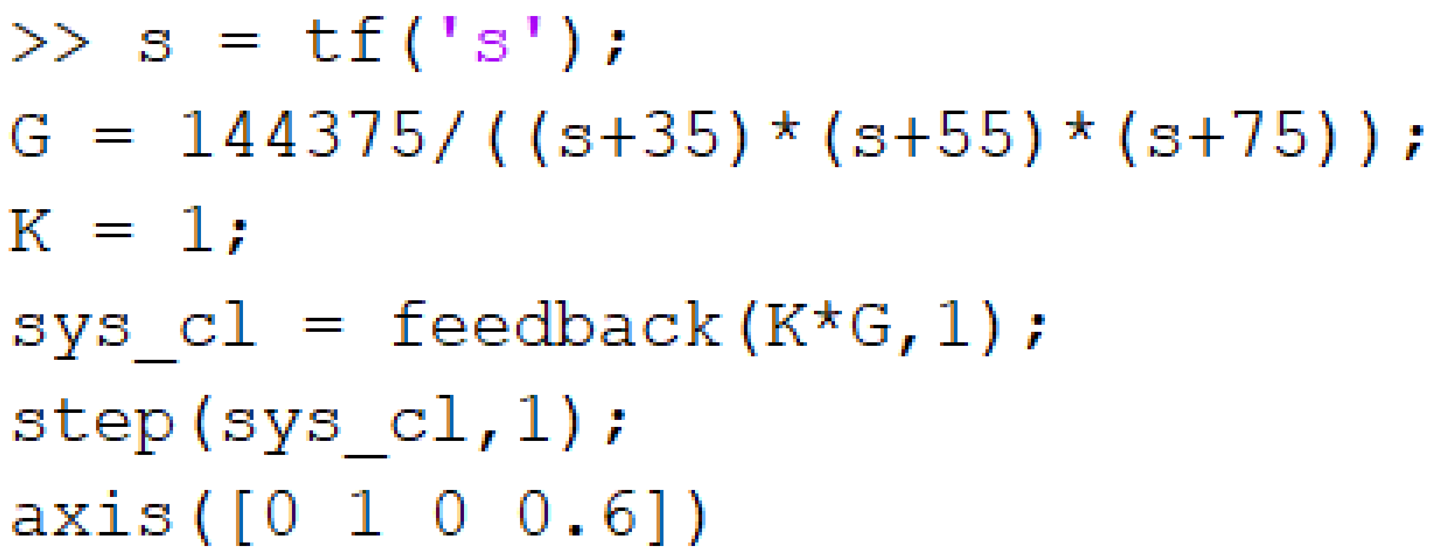
Section III: Step Response

For this section a step response was run in both Matlab and LTSPICE to verify that both simulations generate matching results. This was achieved by placing data points on both plots to measure the time to peak (Tp) and time to settle (Ts). These points were used in conjunction with equation 6 to calculate the percent overshoot (%OS). Once these values were obtained a percentage difference calculator was used to confirm that the difference between the values from both plots were less than 0.5%. The results from this analysis can be seen in Table 3.

$$\%OS = \frac{c_{max} - c_{final}}{c_{final}} \times 100 \qquad [6]$$

III.i: Matlab Transient Response

The following code was used to generate the transient response in Matlab.



```
>> s = tf('s');
G = 144375/((s+35)*(s+55)*(s+75));
K = 1;
sys_cl = feedback(K*G,1);
step(sys_cl,1);
axis([0 1 0 0.6])
```

The plots showing the transient response as well as the data tips placed at time to peak (Tp) and time to settle (Ts) can be seen on the next page in Figure 6, and Figure 7.

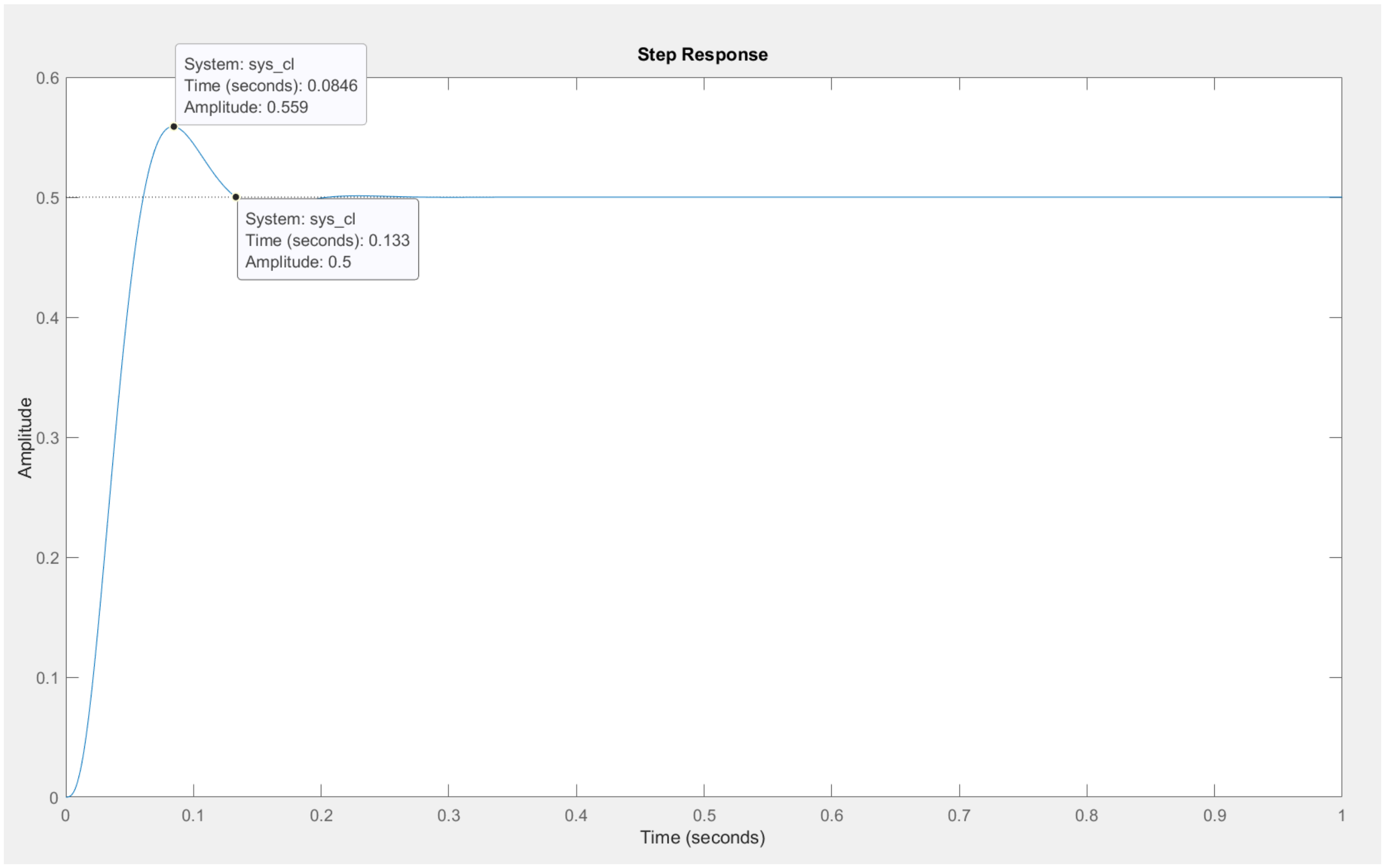


Figure 6: MATLAB transient response plot with data tips at Tp = 84.6 ms with an amplitude of 559 mV and Ts = 133 ms with an amplitude of 500 mV.

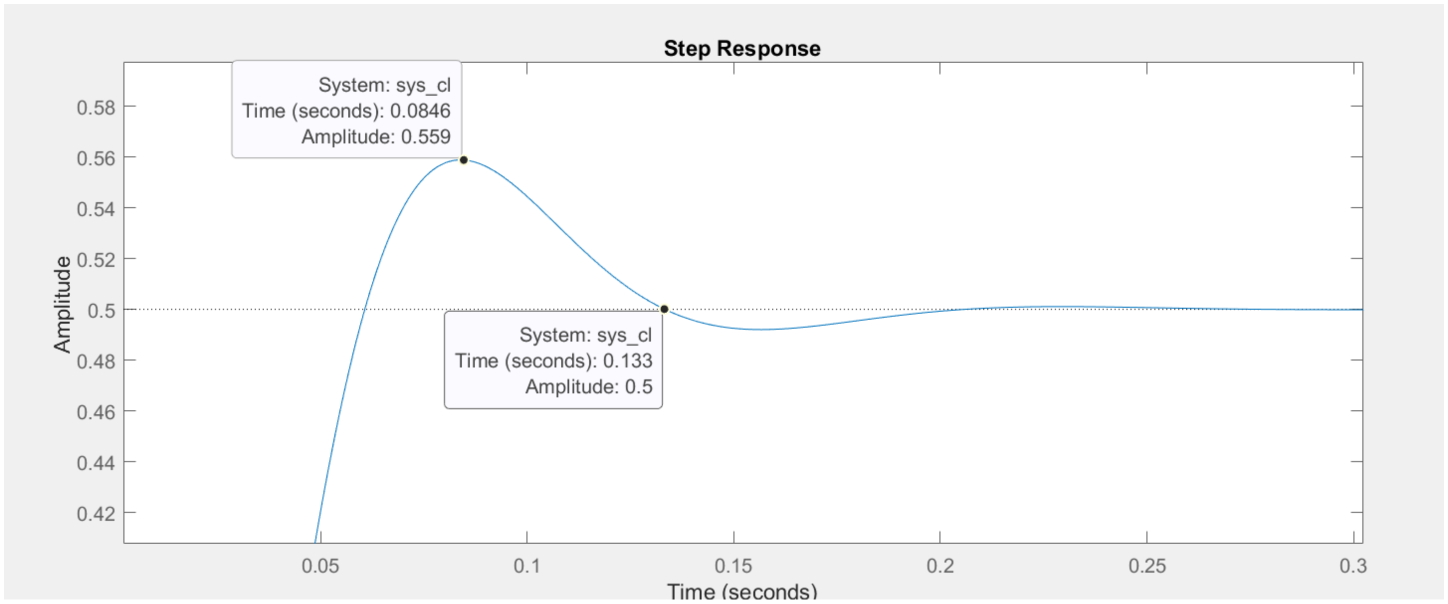


Figure 7: Close up of MATLAB transient response plot with data tips at Tp = 84.6 ms with an amplitude of 559 mV and Ts = 133 ms with an amplitude of 500 mV.

III.ii: Circuit Simulator Transient Response

The circuit depicted in Figure 8 was constructed in LTSPICE circuit simulator to generate the transient response depicted in Figure 9.

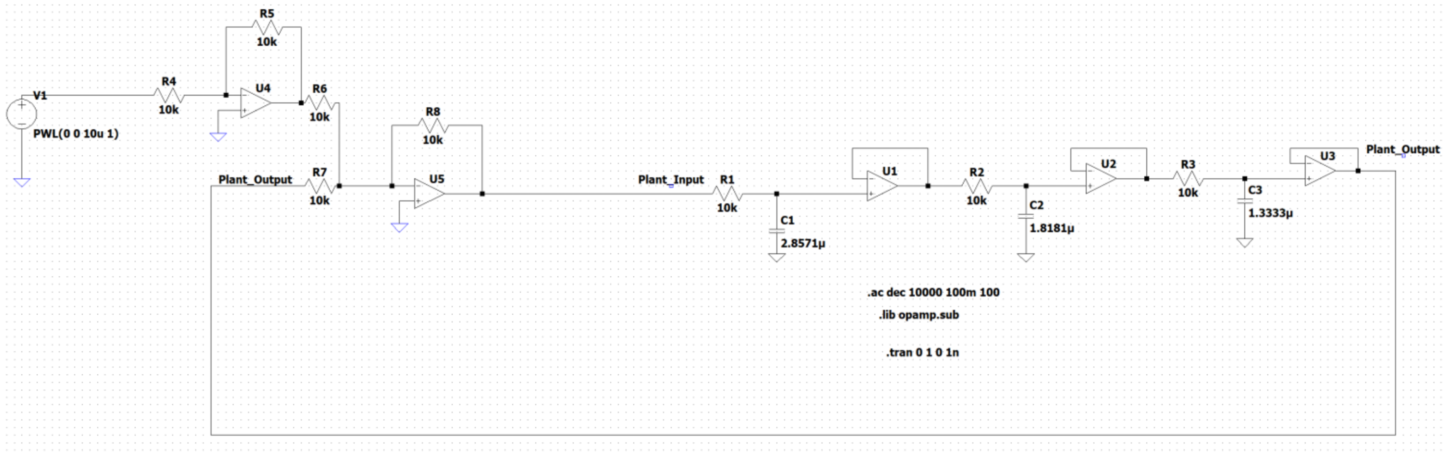
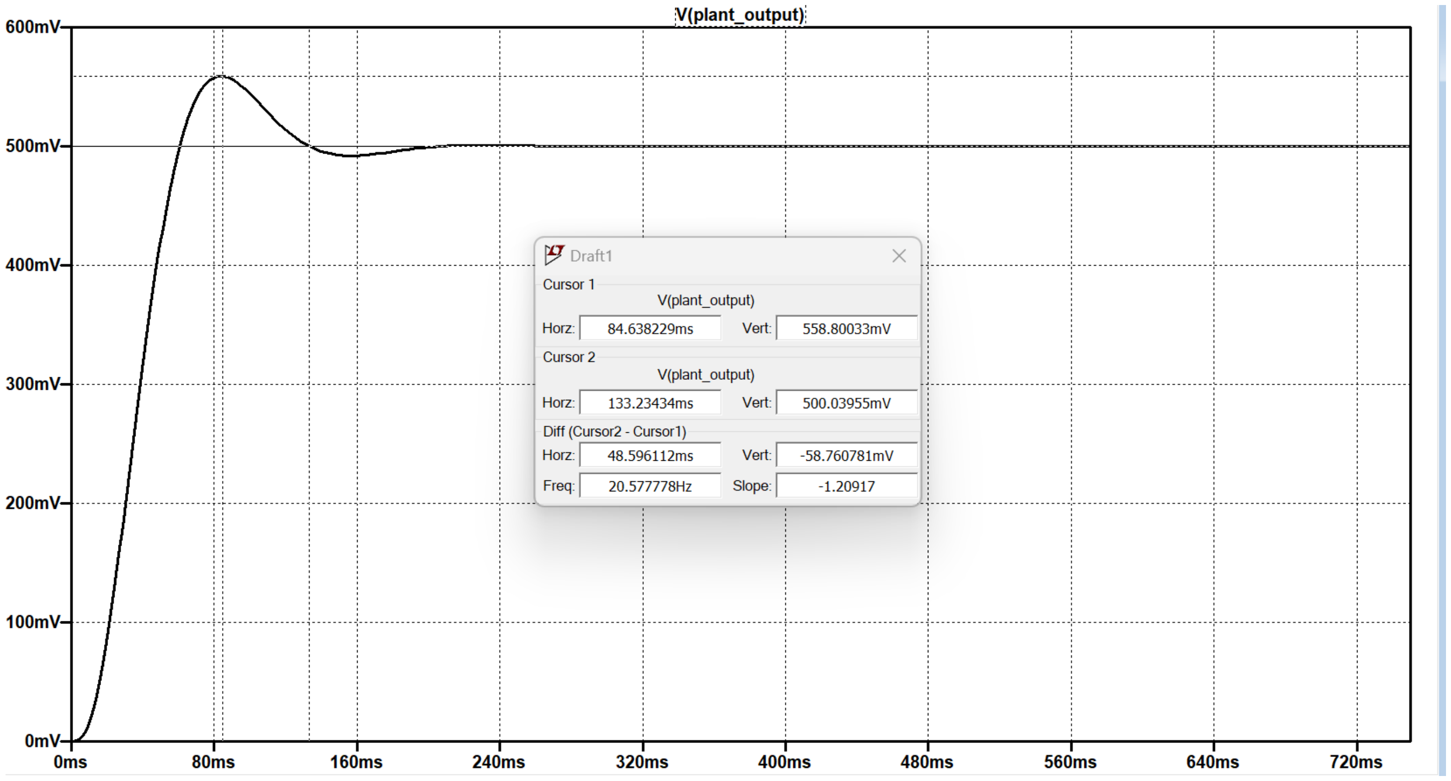


Figure 8: Schematic of LTSPICE circuit used to generate transient response plot where voltage was applied for one second from 0 v to 1v with a timestep of 10 us.



```
Cursor 1
            V(plant_output)
Horz: 84.638229ms      Vert: 558.80033mV
Cursor 2
            V(plant_output)
Horz: 133.23434ms      Vert: 500.03955mV
Diff (Cursor2 - Cursor1)
Horz: 48.596112ms      Vert: -58.760781mV
Freq: 20.577778Hz      Slope: -1.20917
```

Figure 9: Transient response plot generated from LTSPICE circuit simulation showing cursor 1 at the Tp occurring at 84.63 ms with an amplitude of 558.8 mV and cursor 2 at the Ts occurring at 133.24 ms with an amplitude of 500.03 mV.

III.iii: Matlab vs. Simulator Comparison

After running both simulations, a comparison was made between the parameters obtained from their respective plots. The results in Table 3 show differences ranging from 0.04% to 0.42%. This level of agreement between the two models is very good and acceptable for verification purposes.

Table 3: Step Response Comparison, Matlab vs. Simulator			
Parameter	**Matlab**	**Simulator**	**% Δ**
Tp	84.60 ms	84.63 ms	0.04 %
Ts	133.00 ms	133.23 ms	0.18 %
%OS	11.80 %	11.75 %	0.42 %
FV	500 mV	500 mV	0 %

Table 3: Percent Difference calculations between transient response parameters from Matlab and Circuit simulation.

Section IV: Plant Verification

For this section the plant was verified by assuming that the plant was a second order plant and then using the maximum overshoot value and equation 7 to calculate the value for zeta (ζ). After the value for zeta was calculated, a root locus analysis was performed in Matlab to determine the value of gain that would align with intercepts of the data lines in the Matlab plot. Once this gain value was obtained a step response was performed in Matlab and a corresponding circuit was constructed in LTSPICE where another step response analysis was performed. Time to peak, time to settle, and final values were extracted from both analyses and percentage difference calculations were performed. The results from these analyses can be found in table 4.

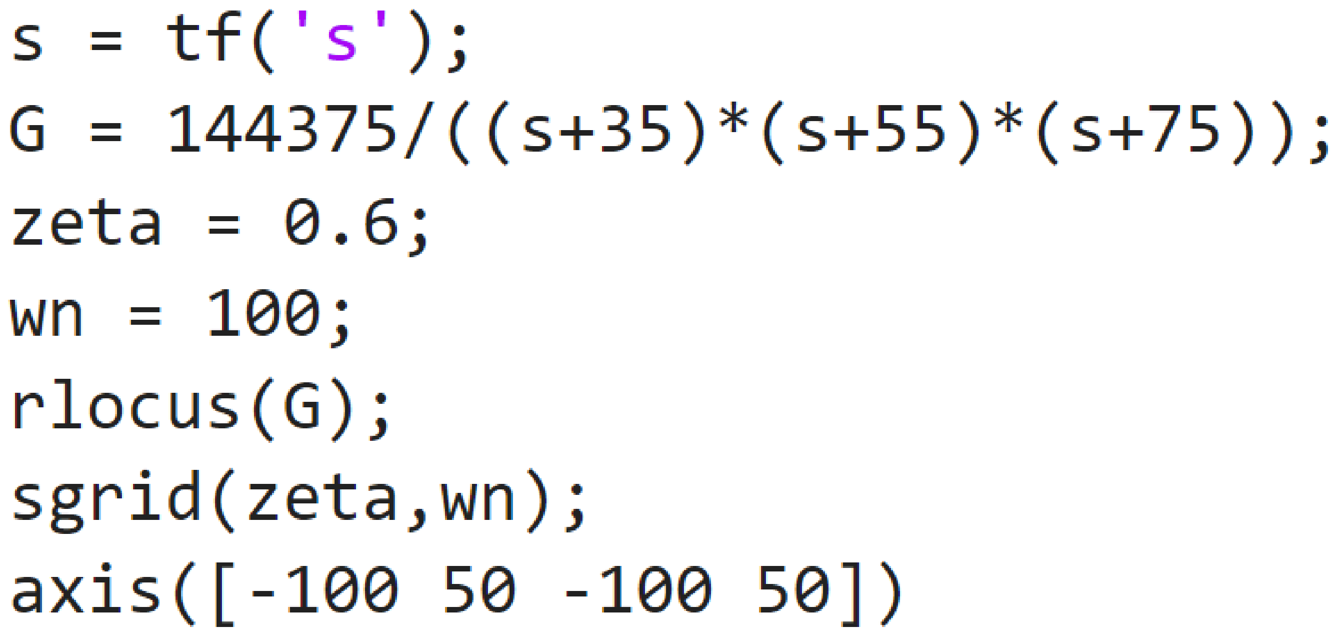
IV.i: Finding Value of ζ

The maximum overshoot determined by the project parameters is 9.48%. using equation 7 below to calculate value for ζ yields a value of 0.599966. We will round that value to 0.6 to run the anlysis.

$$\zeta = \frac{-\ln\left(\frac{\%OS}{100}\right)}{\sqrt{(\pi^2 + ln^2\left(\frac{\%OS}{100}\right))}} \qquad [7]$$

IV.ii: Root Locus in Matlab

To run the root locus analysis in Matlab a value of ω_n was used as to not interfere. The following code was used to perform the analysis on the plant and determine a value for gain that would align with the intercepts of the data lines



```
s = tf('s');
G = 144375/((s+35)*(s+55)*(s+75));
zeta = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
axis([-100 50 -100 50])
```

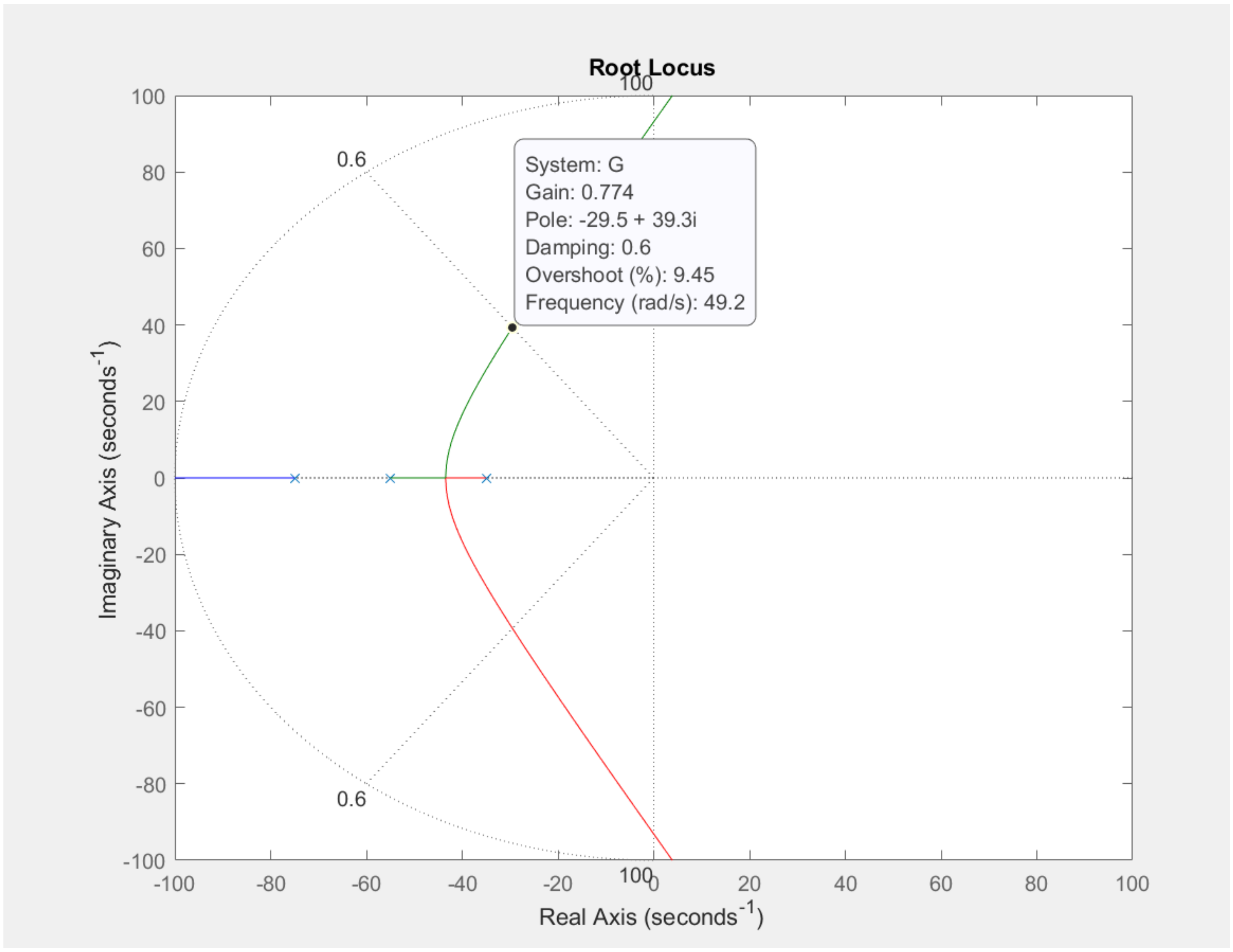
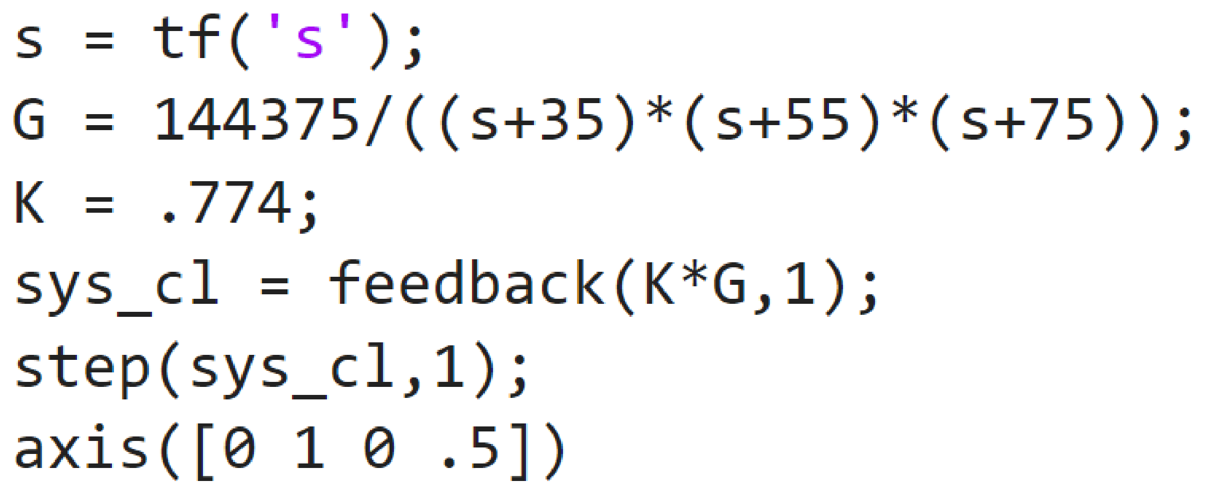


Figure 10: Root locus plot generated in MATLAB showing the value of gain where the data lines intersect to be 0.774.

IV.iii: MATLAB Step Response

After the value of gain was obtained using the root locus technique in MATLAB, the following code was used to generate a transient response. The results of this analysis can be seen in figure 11 and figure 12.



```
s = tf('s');
G = 144375/((s+35)*(s+55)*(s+75));
K = .774;
sys_cl = feedback(K*G,1);
step(sys_cl,1);
axis([0 1 0 .5])
```

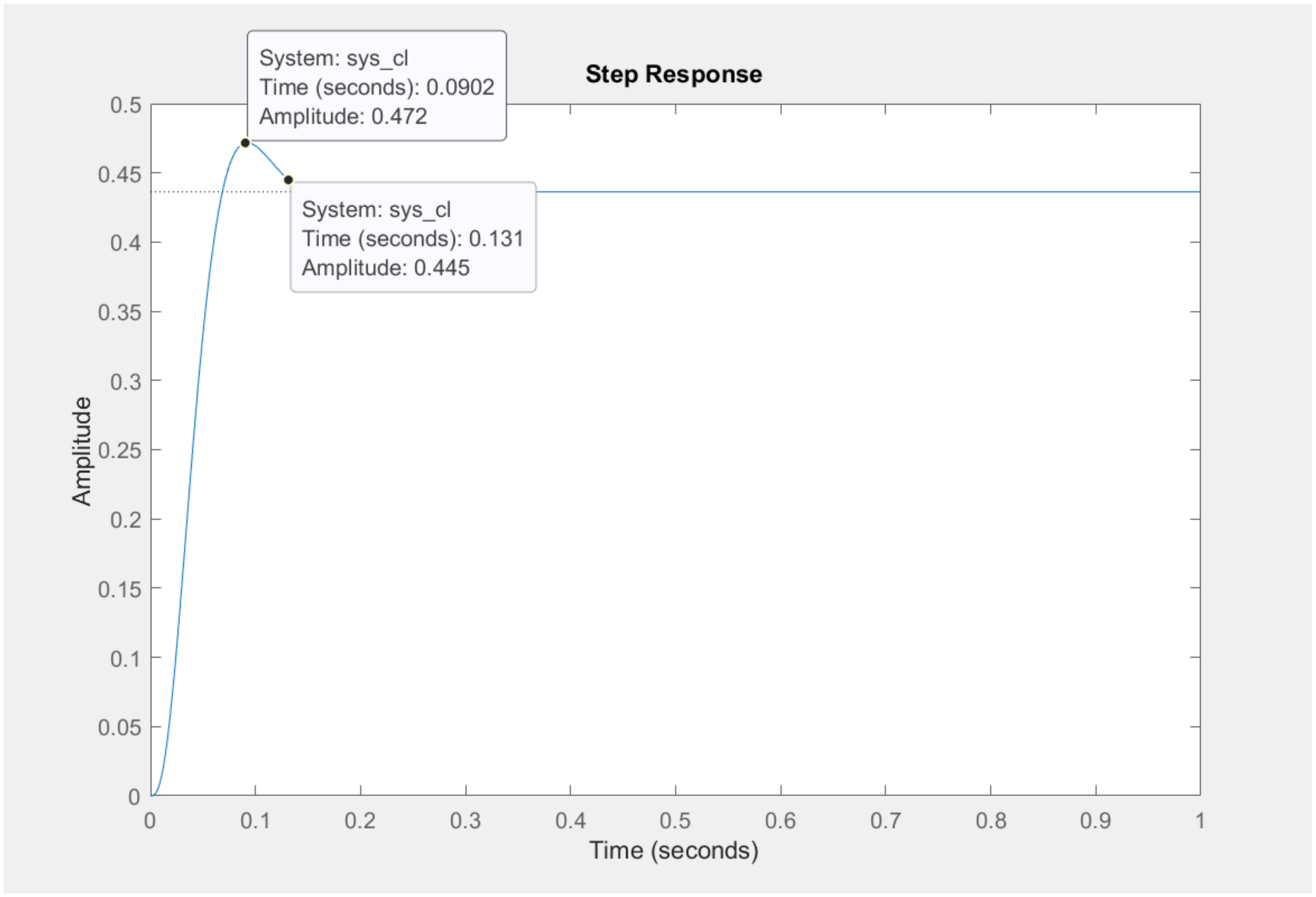


Figure 11: Transient response plot generated from Matlab showing data tip at the Tp occurring at 90.2 ms with an amplitude of 472 mV and another at the Ts occurring at 131 ms with an amplitude of 445 mV.

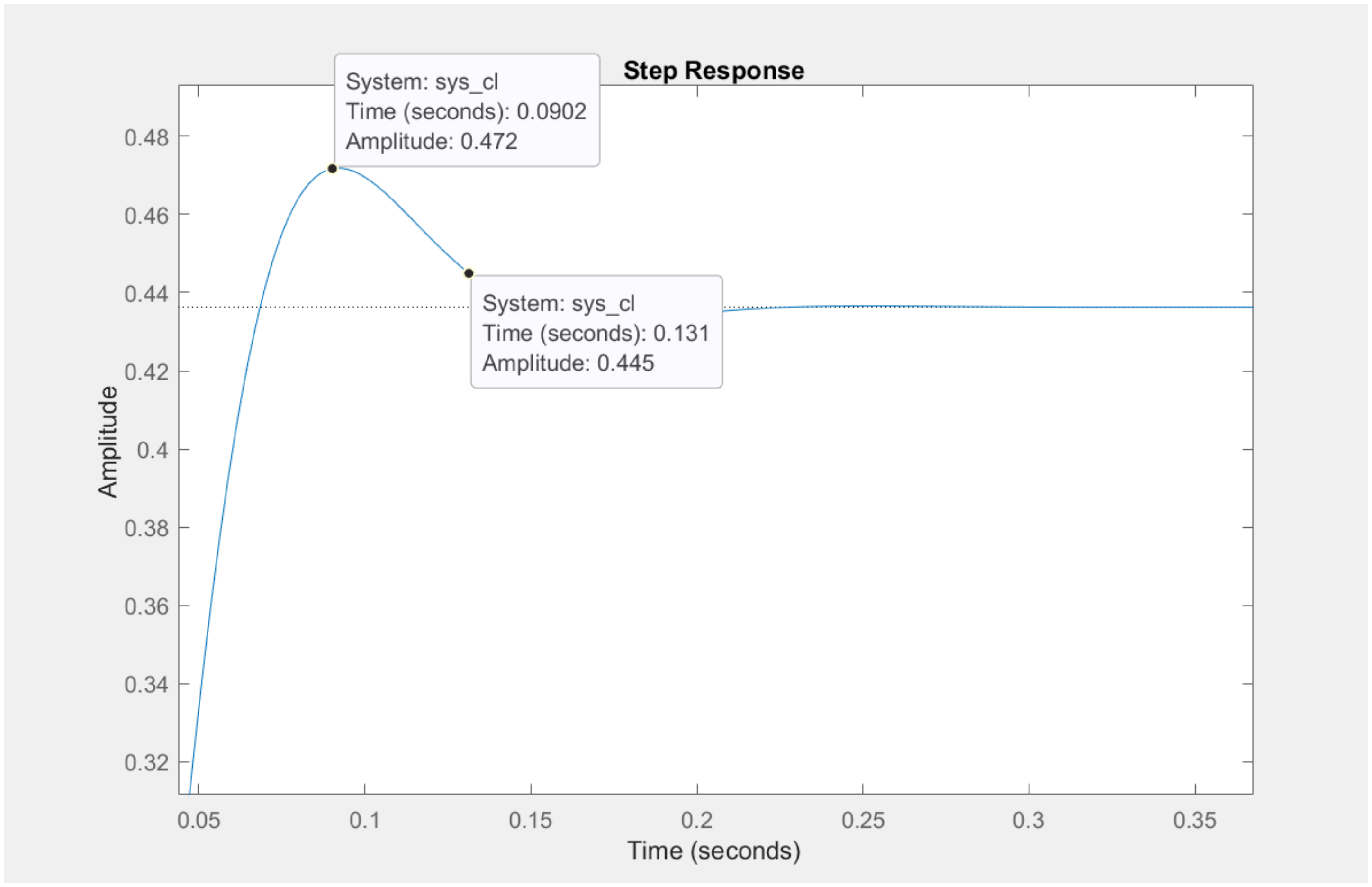
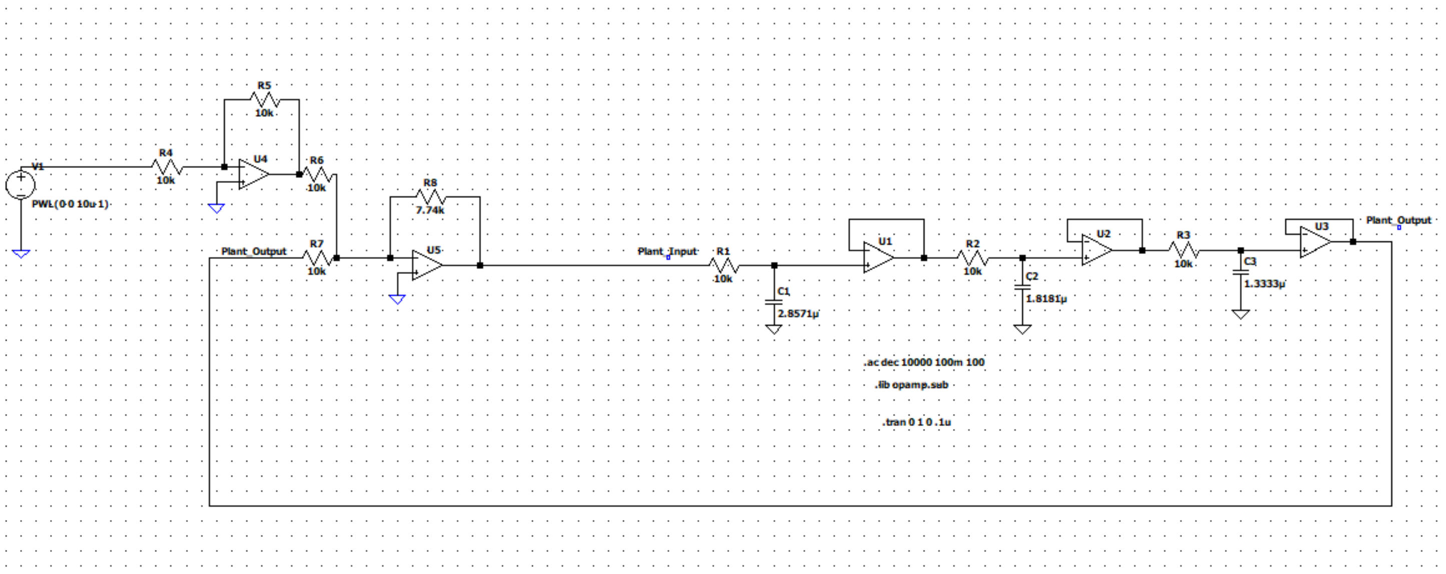


Figure 12: Close up of transient response plot generated from Matlab showing data tip at the Tp occurring at 90.2 ms with an amplitude of 472 mV and another at the Ts occurring at 131 ms with an amplitude of 445 mV.

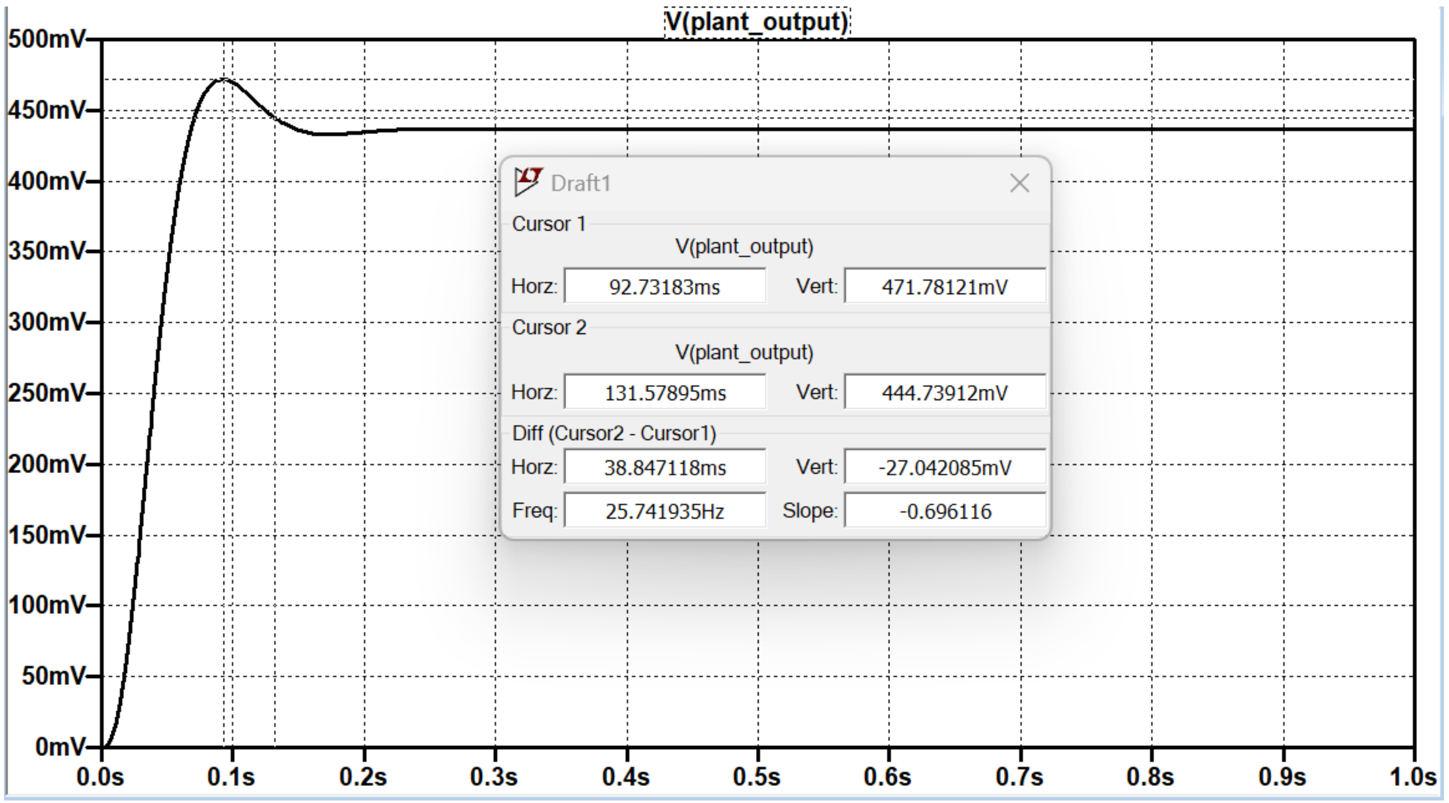
IV.iv: LTSPICE Step Response

The circuit from Section III was modified by multiplying the value of gain by the component value of R8 to yield a value of 7.74 kΩ. After the component value was changed, a step response plot was generated. The changes in this circuit can be seen in Figure 13. The results from the plot can be seen in Figure 14.



```
.ac dec 10000 100m 100
.lib opamp.sub
.tran 0 1 0 .1u
```

Figure 13: Schematic of LTSPICE circuit used to generate transient response plot where voltage was applied for one second from 0 v to 1v with a timestep of 10 us.



```
Cursor 1
              V(plant_output)
Horz:  92.73183ms        Vert:  471.78121mV
Cursor 2
              V(plant_output)
Horz:  131.57895ms       Vert:  444.73912mV
Diff (Cursor2 - Cursor1)
Horz:  38.847118ms       Vert:  -27.042085mV
Freq:  25.741935Hz       Slope: -0.696116
```

Figure 14: Transient response plot generated from LTSPICE circuit simulation showing cursor 1 at the Tp occurring at 92.73 ms with an amplitude of 471.78 mV and cursor 2 at the Ts occurring at 131.58 ms with an amplitude of 444.74 mV.

IV.v: MATLAB vs. Simulator Comparison

After running both simulations, a comparison was made between the parameters obtained from their respective plots. The results in Table 4 show differences ranging from 0.06% to 2.77%. This level of agreement between the two models is very good for all values except for the time to peak where a larger difference is observed.

Table 3: Step Response Comparison, MATLAB vs. Simulator

Parameter	MATLAB	Simulator	% Δ
Tp	90.20 ms	92.73 ms	2.77 %
Ts	131.00 ms	131.58 ms	0.44 %
%OS	6.07 %	6.08 %	0.16 %
FV	436 mV	436.28 mV	0.06 %

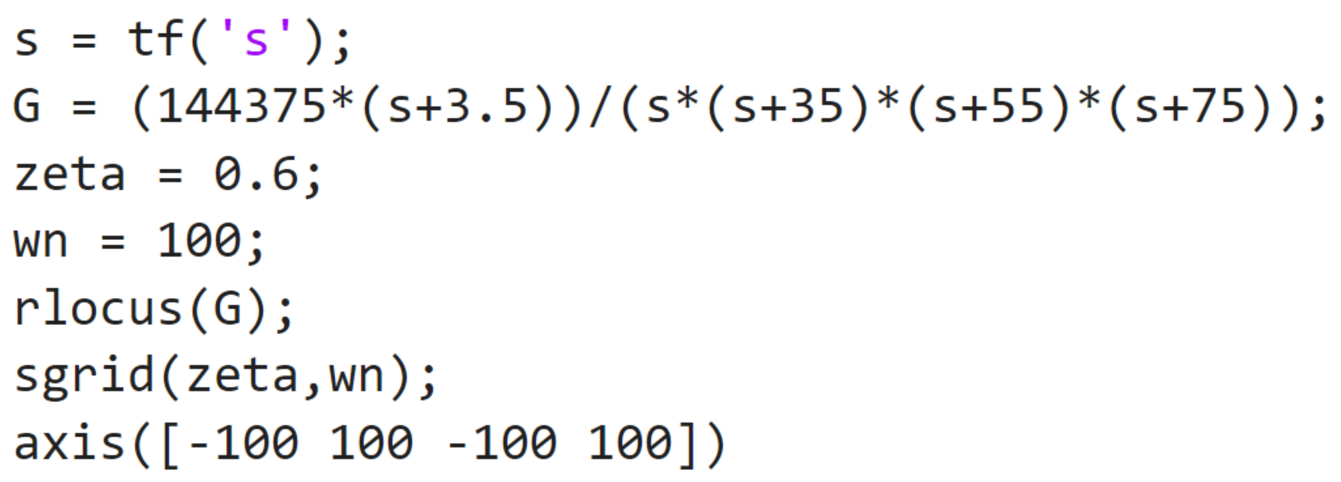
Table 4: Percent Difference calculations between transient response parameters from Matlab and Circuit simulation.

Section V: PD Calculations & Matlab Verification

To enhance system performance, a PID controller was implemented by adding integral and differential components, aligning with the project's goals. This involved inserting an integrator (a pole at the origin) and strategically placing a zero between the origin and the system's dominant pole to cancel out the integrator's effect. These adjustments significantly modified the system's step response. Additionally, a differential component was introduced to further optimize the root locus by aligning real and imaginary components to achieve the desired system behavior and meet the performance targets effectively

V.i: Implementing the Integrator

The following MATLAB code was used as a starting point to incorporate the integral component of our design by adding a pole at the origin and cancelling its effect by placing a zero 10% of the way between the origin and the most dominant pole. In this case the most dominant pole was -35 and the zero was placed at -3.5



```
s = tf('s');
G = (144375*(s+3.5))/(s*(s+35)*(s+55)*(s+75));
zeta = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
axis([-100 100 -100 100])
```

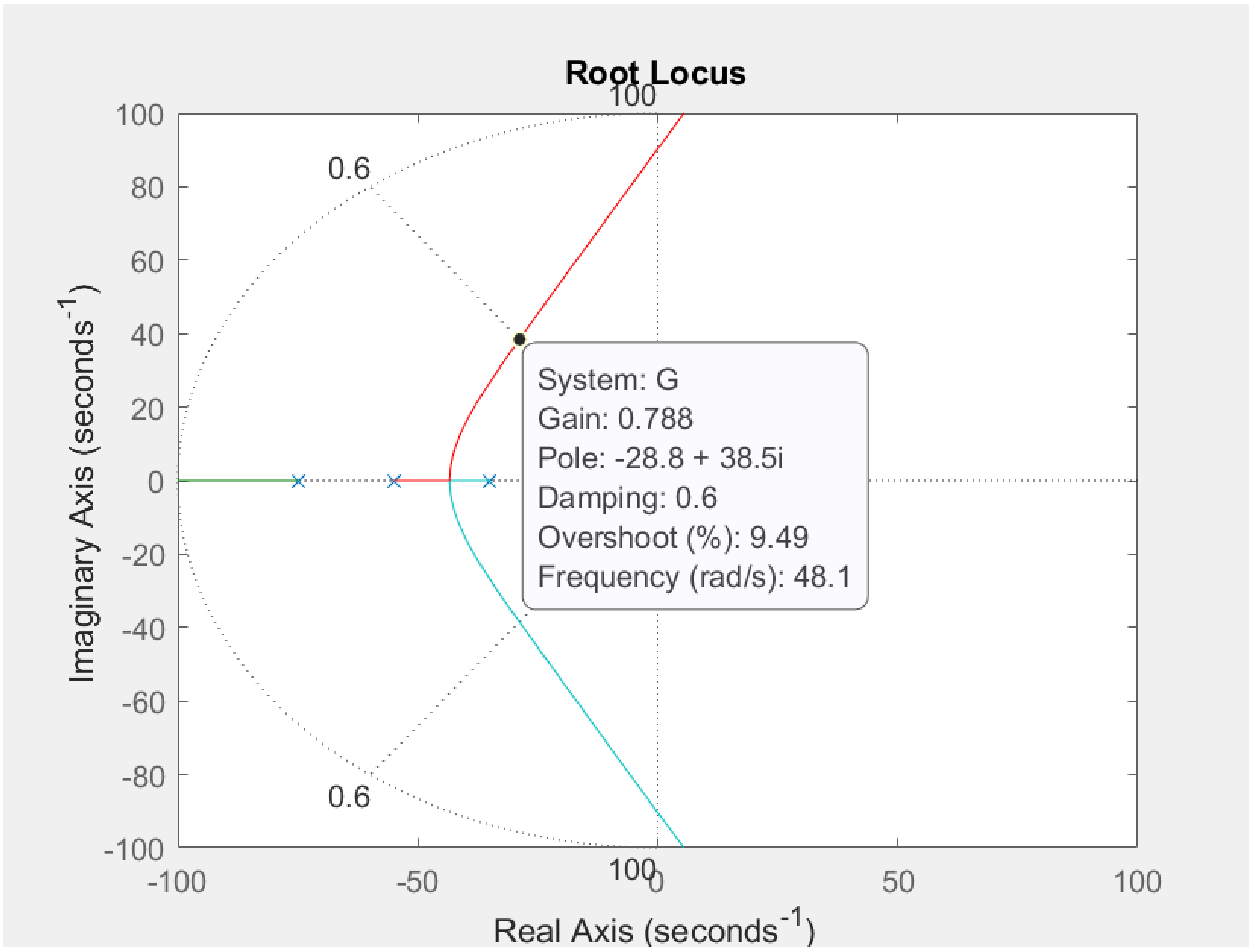


Figure 15: This figure depicts the root locus after the integral compensation was applied. Notice that the root locus intersects the ζ line at -28.8 ± j38.5. Gain K is at 0.788.

After the gain value K was obtained from the root locus the following MATLAB code was used to run a step response to cross analyze the system before and after PI compensation.

```
s = tf('s');
G = (144375*(s+3.5))/(s*(s+35)*(s+55)*(s+75));
K = .788;
sys_cl = feedback(K*G,1);
step(sys_cl,6);
```

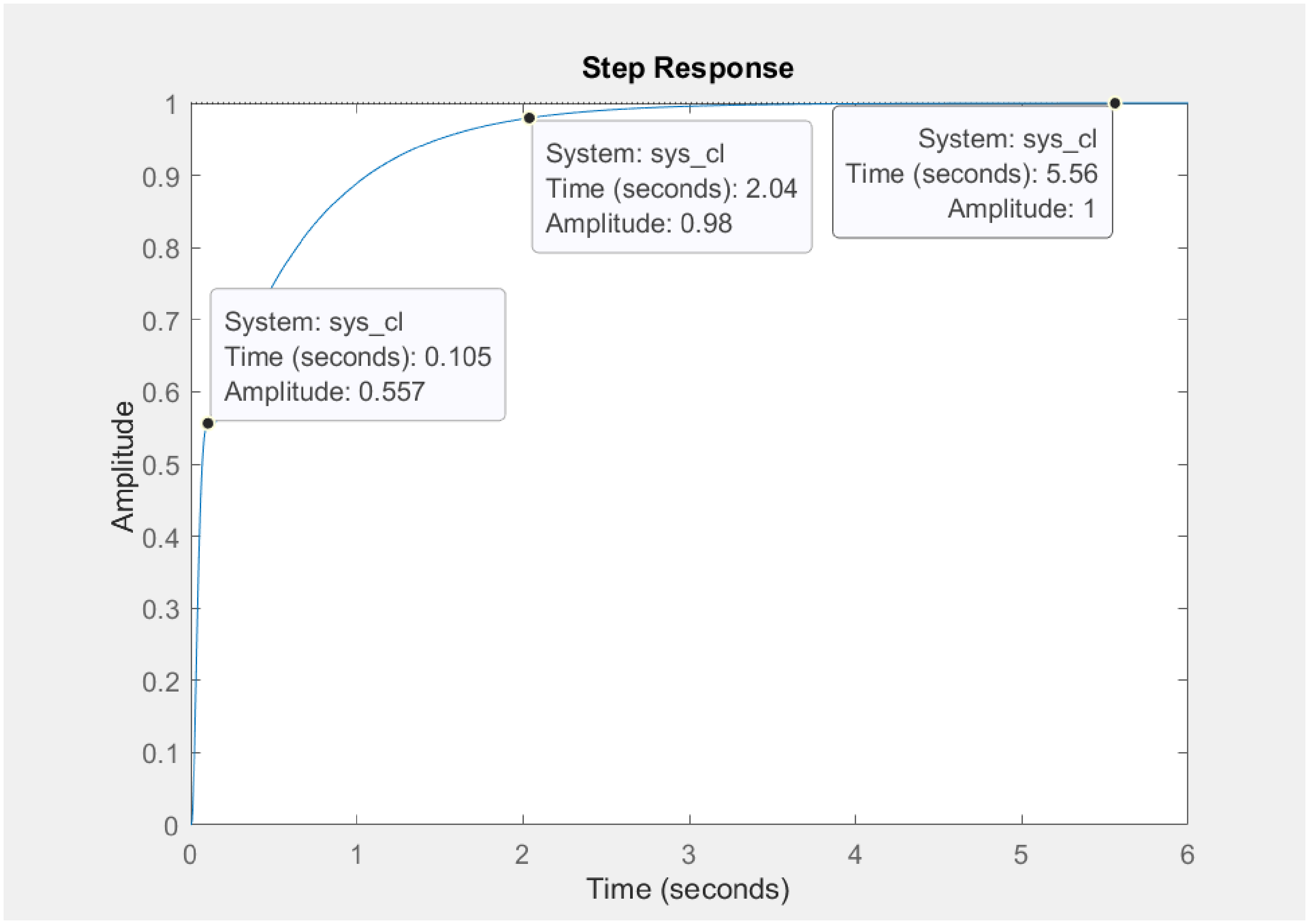


Fig 16: Step response with PI compensation and zero at -3.5 rad/s, and gain K = 0.788. Notice that $\boldsymbol{e_{step}(\infty) = 0, T_p = 105\ ms, T_s = 2.04\ s}$ ***and Final Value is 1.***

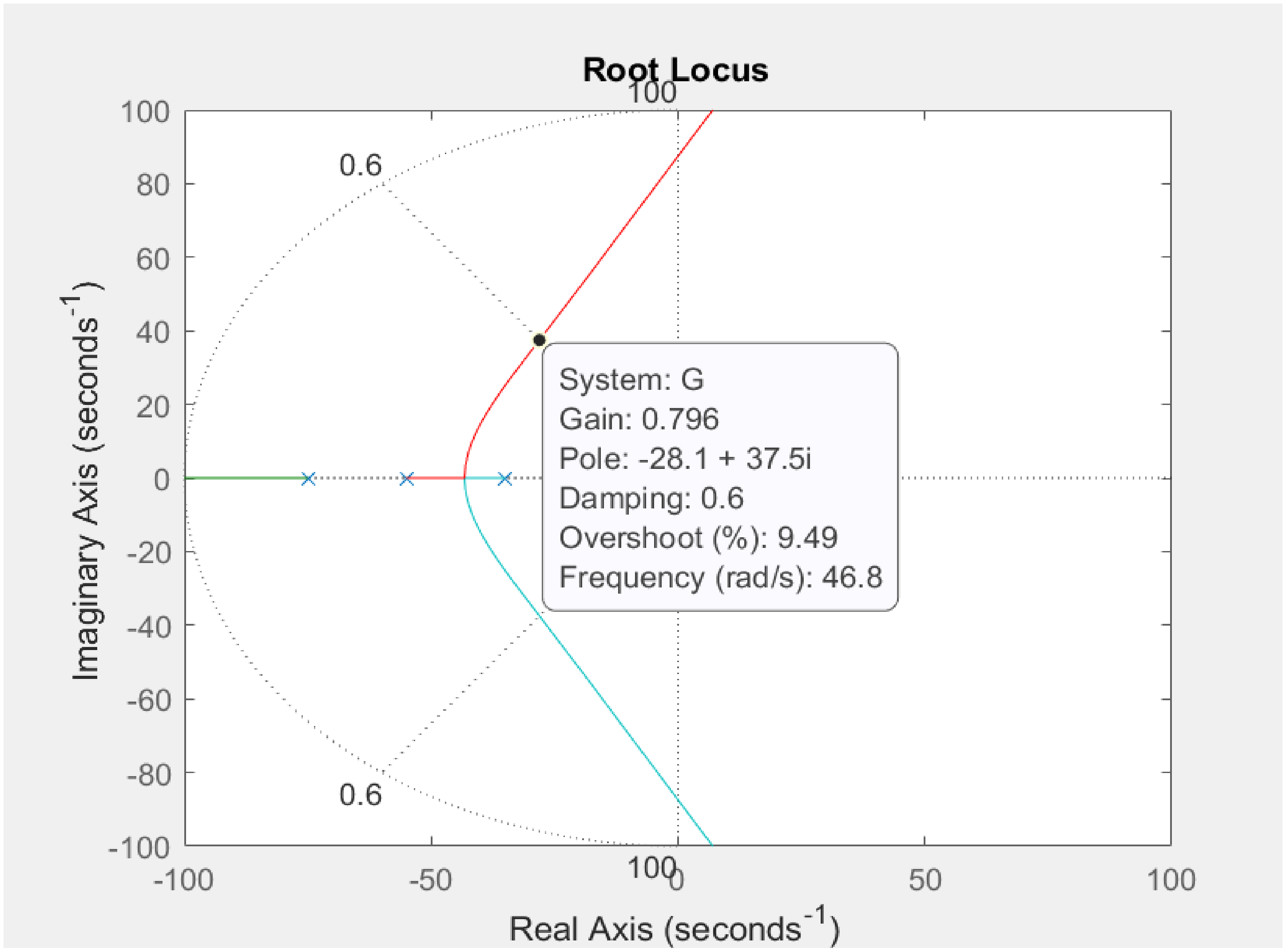
Table 5: Performance Comparison, P vs. PI Compensation at ζ = 0.6, K = 0.788 and PI Zero at -3.5 rad/s			
Parameter	**P Only**	**PI**	**%Δ**
T_P	90.20 ms	105 ms	15.16%
$Pole\ Location$	-29.5 $\pm$ j39.3	-28.8$\pm$ j38.5	-2.18%
T_S	131 ms	2.04 s	N/A
$\%OS$	6.07%	0%	N/A
FV	436 mV	1 V	☑

Table 5: Percent difference calculations between system before and after PI compensation with zero at 3.5 rad/s.

An increase in the time to settle indicates that the system has slowed down significantly. This can be adjusted by moving the zero farther from the origin.

The following MATLAB code was used to generate a root locus with the zero at -7 rad/s.

```
s = tf('s');
G = ((144375)*(s+7))/(s*(s+35)*(s+55)*(s+75));
zeta = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
axis([-100 100 -100 100])
```



System: G
Gain: 0.796
Pole: -28.1 + 37.5i
Damping: 0.6
Overshoot (%): 9.49
Frequency (rad/s): 46.8

Figure 17: This figure depicts the root locus after the integral compensation was applied with zero at -7 rad/s. Notice that the root locus intersects the ζ line at -28.1 ± j37.5. Gain K is at 0.796.

The following MATLAB code was used to generate a step response with zero at -7 rad/s and gain K = 0.796

```
s = tf('s');
G = ((144375)*(s+7))/(s*(s+35)*(s+55)*(s+75));
K = .796;
sys_cl = feedback(K*G,1);
step(sys_cl,1.26);
```

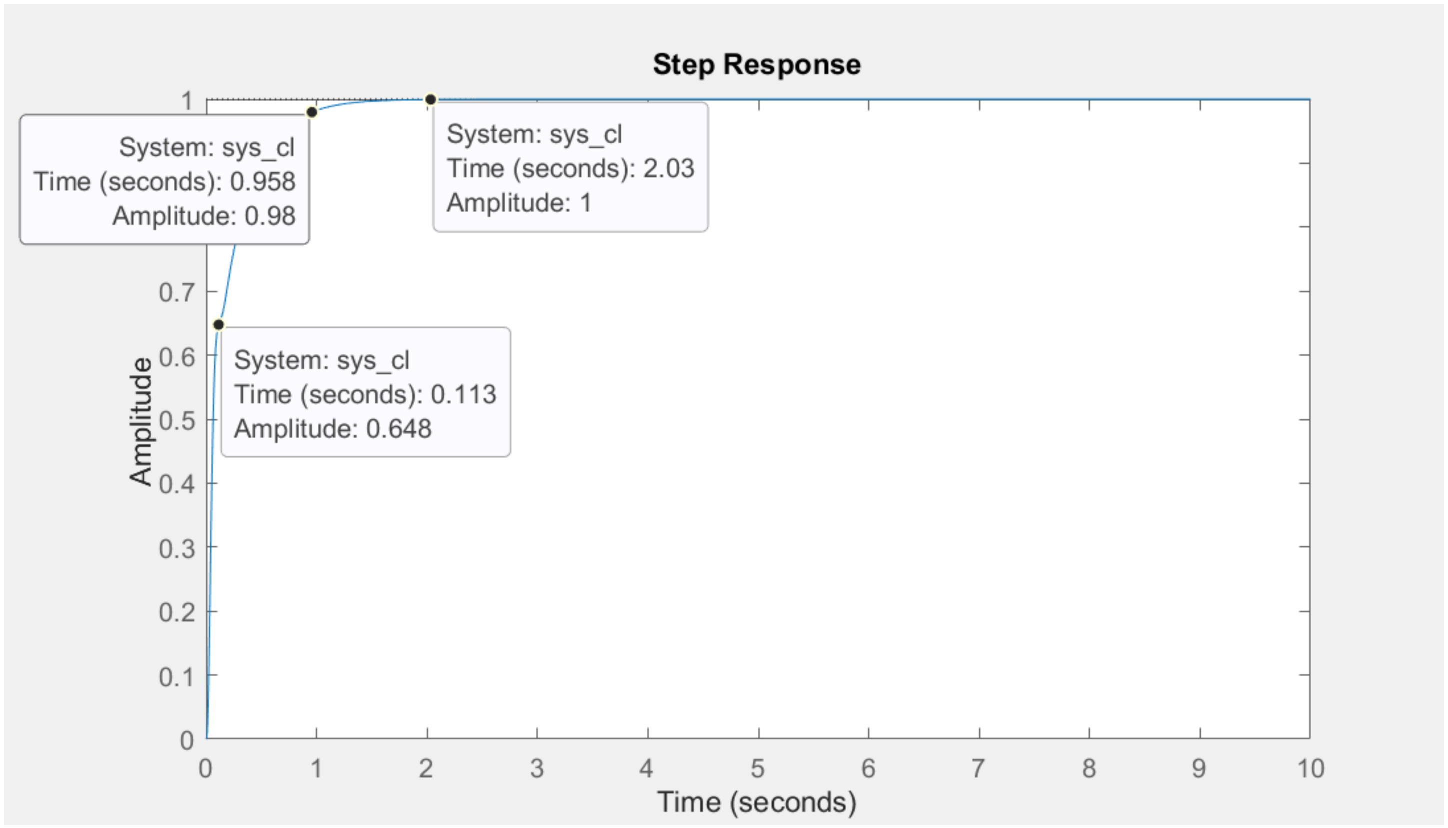


Fig 18: Step response with PI compensation and zero at -7 rad/s, and gain K = 0.796. Notice that $e_{step}(\infty) = 0, T_p = 113\ ms, T_s = 2.03\ s$ ***and Final Value is 1.***

This does not seem like a significant enough improvement from the previous compensation. Let us double the zero from -7 rad/s to -14 rad/s.

Table 6: Performance Comparison, P vs. PI Compensation at ζ = 0.6, K = 0.788 and PI Zero at -7 rad/s			
Parameter	**P Only**	**PI**	**%Δ**
T_P	90.20 ms	113 ms	22.44%
$Pole\ Location$	-29.5 $\pm$ j39.3	-28.1$\pm$ j37.5	-4.75%
T_s	131 ms	2.03 s	N/A
$\%OS$	6.07%	0%	N/A
FV	436 mV	1 V	☑

Table 6: Percent difference calculations between system before and after PI compensation with zero at -7 rad/s.

This does not seem like a significant enough improvement from the previous compensation. Let us double the zero from -7 rad/s to -14 rad/s.

The following MATLAB code was used to generate the root locus with the zero placed at -14 rad/s.

```
s = tf('s');
G = ((144375)*(s+14))/(s*(s+35)*(s+55)*(s+75));
zeta = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
axis([-100 100 -100 100])
```

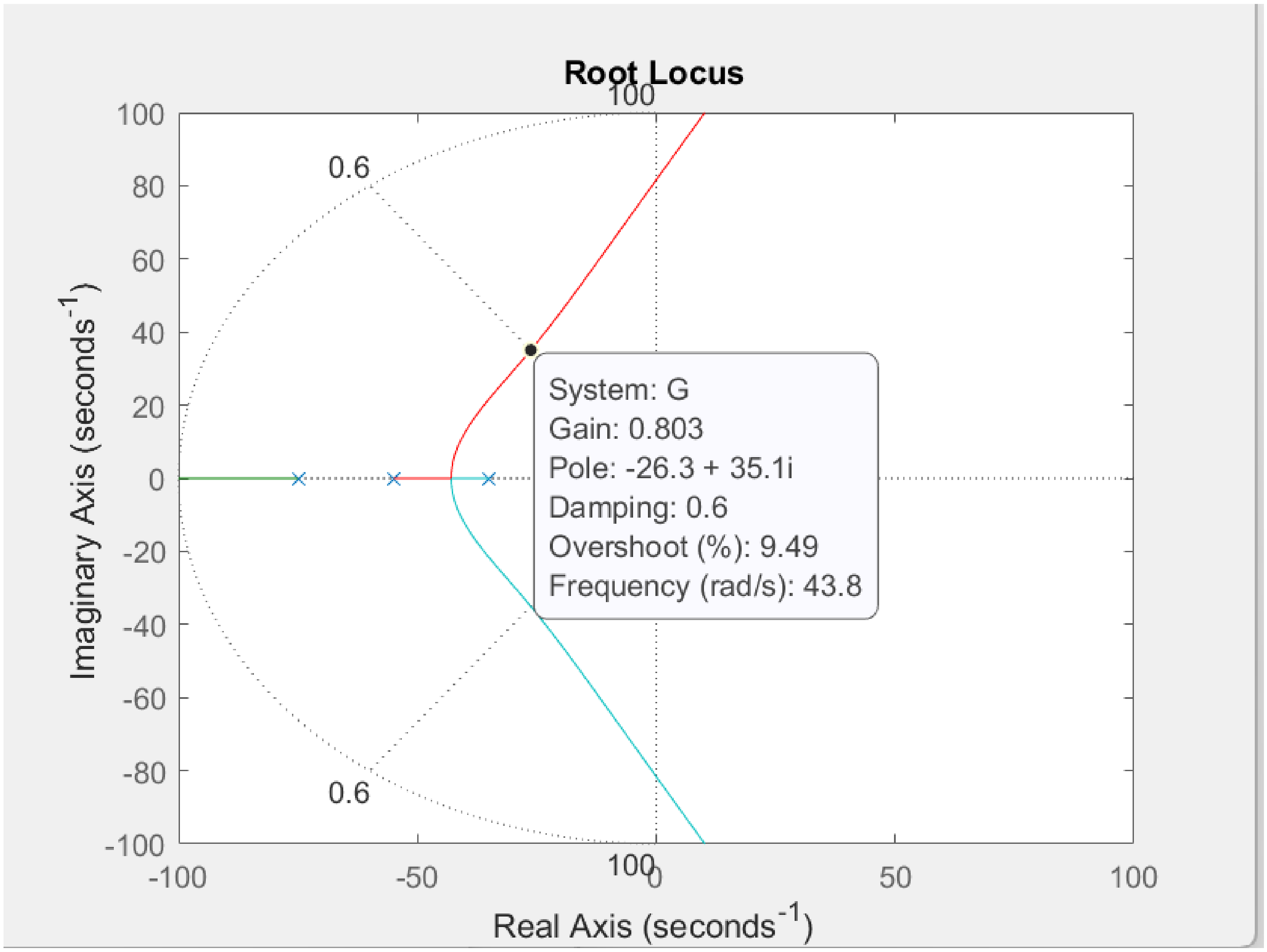
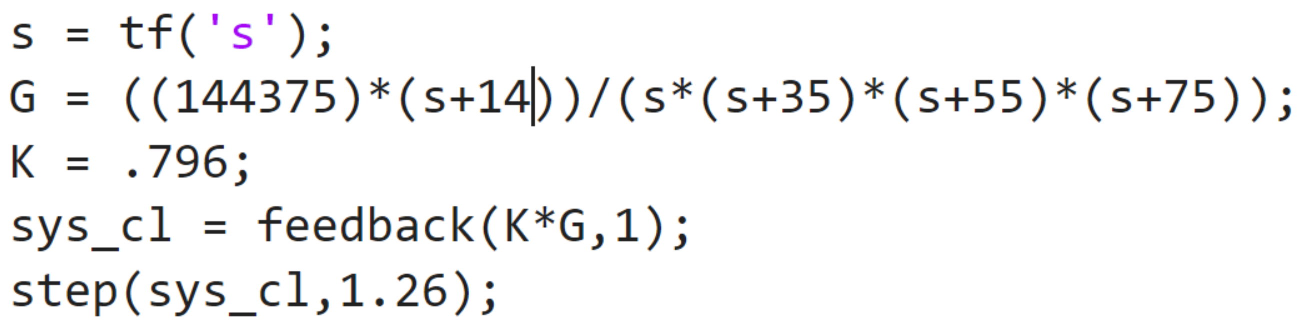


Figure 19: This figure depicts the root locus after the integral compensation was applied with zero at -14 rad/s. Notice that the root locus intersects the ζ line at -26.3 ± j35.1. Gain K is at 0.803.

The following MATLAB code was used to generate a step response with zero at -14 rad/s and gain K = 0.803



```
s = tf('s');
G = ((144375)*(s+14))/(s*(s+35)*(s+55)*(s+75));
K = .796;
sys_cl = feedback(K*G,1);
step(sys_cl,1.26);
```

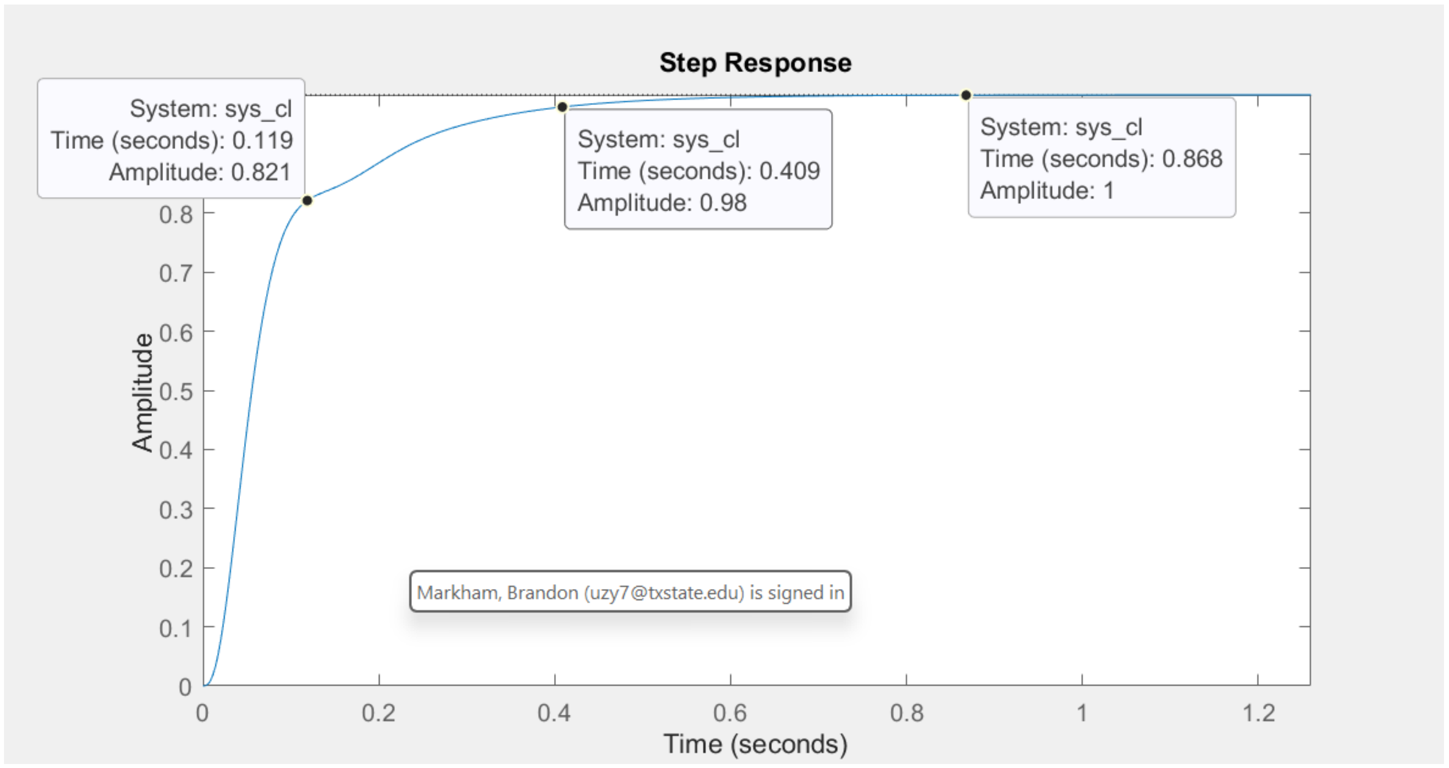


Fig 20: Step response with PI compensation and zero at -14 rad/s, and gain K = 0.803. Notice that $\boldsymbol{e_{step}(\infty) = 0, T_p = 119\ ms, T_s = 0.409\ s}$ ***and Final Value is 1.***

Table 7: Performance Comparison, P vs. PI Compensation at ζ = 0.6, K = 0.788 and PI Zero at -7 rad/s			
Parameter	**P Only**	**PI**	**%Δ**
T_P	90.20 ms	113 ms	22.44%
$Pole\ Location$	-29.5 $\pm$ j39.3	-26.3$\pm$ j35.1	-11.35%
T_s	131 ms	409 ms	N/A
$\%OS$	6.07%	0%	N/A
FV	.436	1.0	☑

Table 7: Percent difference calculations between system before and after PI compensation with zero at -14 rad/s.

Though the time to peak increased slightly the time to settle improved significantly. Seeing as the time to peak can be improved with the D channel this is good enough to proceed.

V.ii: Derivative Channel Design

The first step is to calculate the location of the new poles. Before PI compensation was added the dominant poles were at 29.5 ± j39.3 rad/s. Our target is to improve T_p by 20 %. This means the dominant pole pair must move farther away from the origin.

$$T_p = \frac{\pi}{|Im|} = \frac{\pi}{|39.3|} = 79.9\,ms \qquad [8]$$

Notice that this value is not what was measured in MATLAB since this is not a 2nd order system. This value will act as an approximation to serve as a starting point.

Reducing T_p of 79.9 ms by 20% yields 63.95 ms.

$$|Im| = \frac{\pi}{0.06395} = 49.12\,rad/s \qquad [9]$$

The real component can be found using the ratio in equation 10.

$$|Re| = \frac{49.12}{39.3} \times 29.5 = 36.87\,rad/s \qquad [10]$$

Thus, our desired pole location is -14.12 ± j21.49 rad/s.

To ensure that the desired pole location is on the root locus and an odd multiple of ± 180° we must calculate the existing angle at that point using all poles and zeros and find the angular difference required to get an odd multiple of ± 180° and then use this angle to find the frequency of the D channel compensating zero.

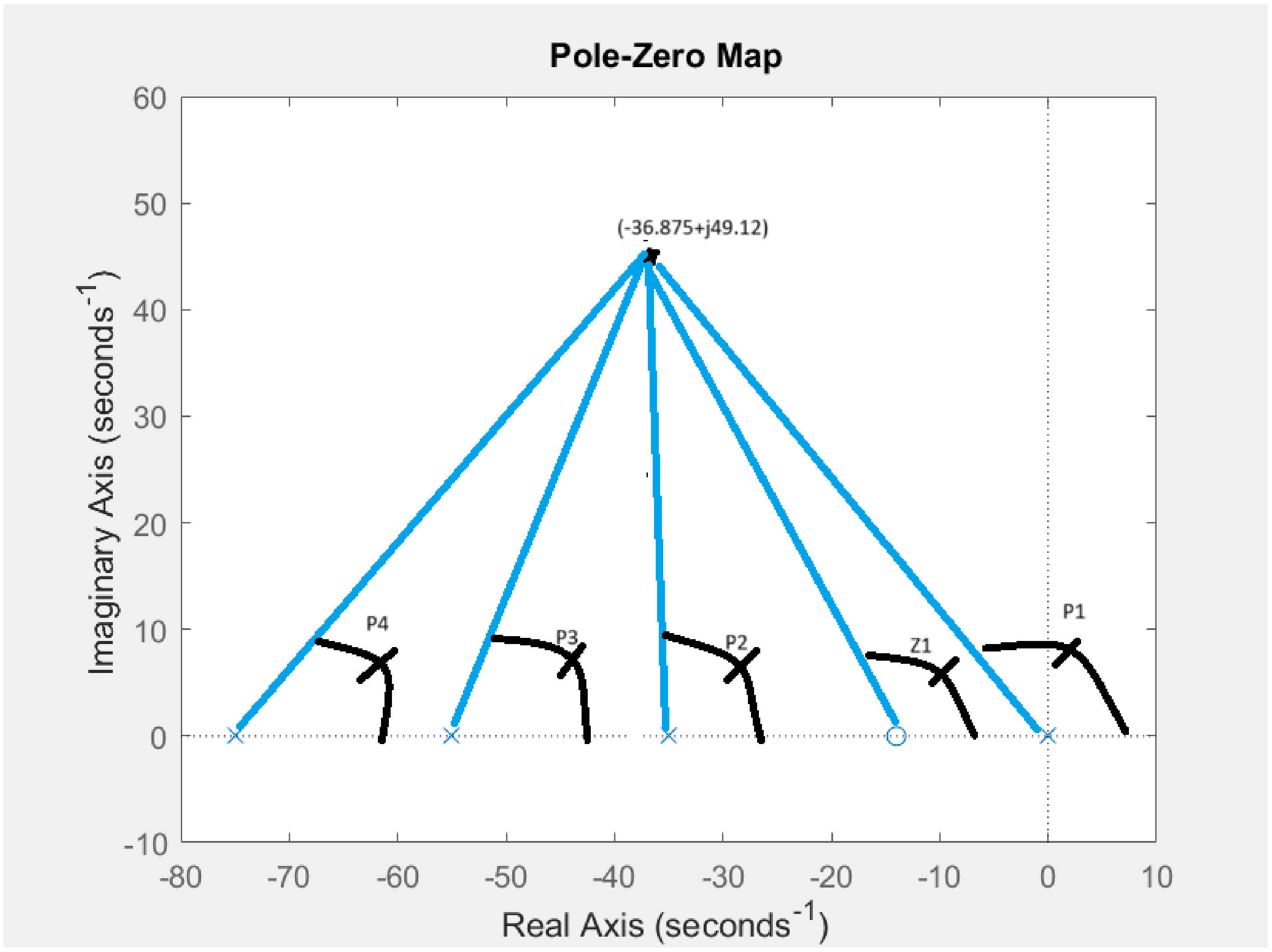


Figure 21: Pole zero map depicting the point of interest (-36.875+j49.12) and each pole and zero and their corresponding angles respectively.

The following formulas were used to calculate the angle to yield D channel frequency.

$$\theta_{p1} = 180° - \tan^{-1}\left[\frac{49.12}{36.875}\right] = 126.896° \qquad [11]$$

$$\theta_{p2} = 180° - \tan^{-1}\left[\frac{49.12}{36.875 - 35}\right] = 92.186°$$

$$\theta_{p3} = \tan^{-1}\left[\frac{49.12}{55 - 36.875}\right] = 69.75°$$

$$\theta_{p4} = \tan^{-1}\left[\frac{49.12}{75 - 36.875}\right] = 52.183°$$

$$\theta_{p1} + \theta_{p2} + \theta_{p3} + \theta_{p4} = 341.015°$$

$$\theta_{z1} = 180° - \tan^{-1}\left[\frac{49.12}{36.875 - 14}\right] = 114.971°$$

$$\theta_{z1} = 114.971°$$

$$\sum_i \theta_{zi} - \sum_j \theta_{pj} = 114.971° - 341.015° = -226.044 \qquad [12]$$

-226.044° +180° = -46.044°

Based off the above calculations we are looking to add a zero that will give us an angle of 46.044°

$$\tan[46.044°] = \frac{49.12}{\sigma} \rightarrow \sigma = \frac{49.12}{\tan[46.044°]} = 47.3618 \qquad [13]$$

$$-36.875 - 47.3618 = -84.2368\, rad/s \qquad [14]$$

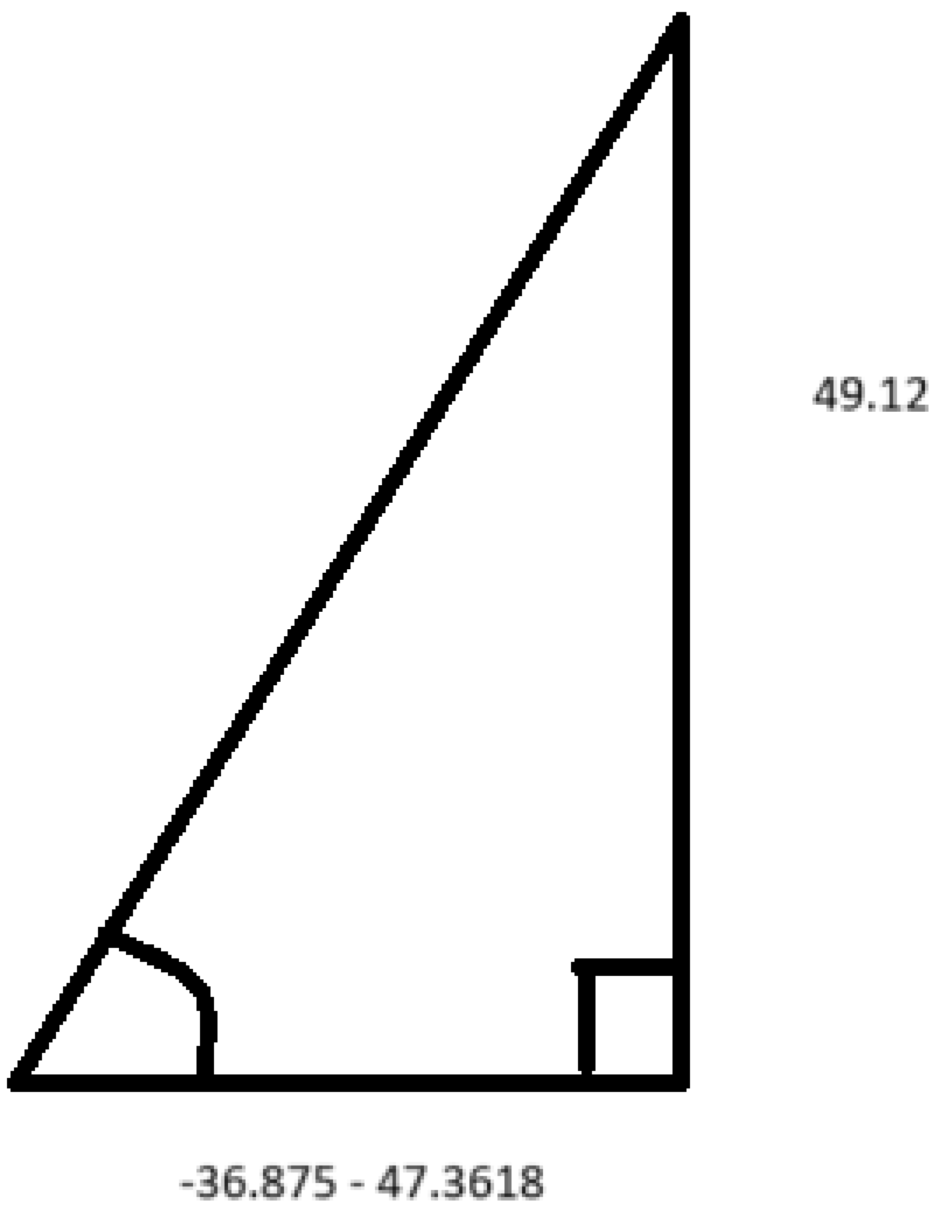


Figure 22: Right triangle for calculating location of compensating zero.

The following MATLAB code was used to generate the root locus with the implementation of the D channel with a zero at -84.2368 rad/s

```
s = tf('s');
G = ((144375)*(s+14)*(s+84.2368))/(s*(s+35)*(s+55)*(s+75));
zeta = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
axis([-100 100 -100 100])
```

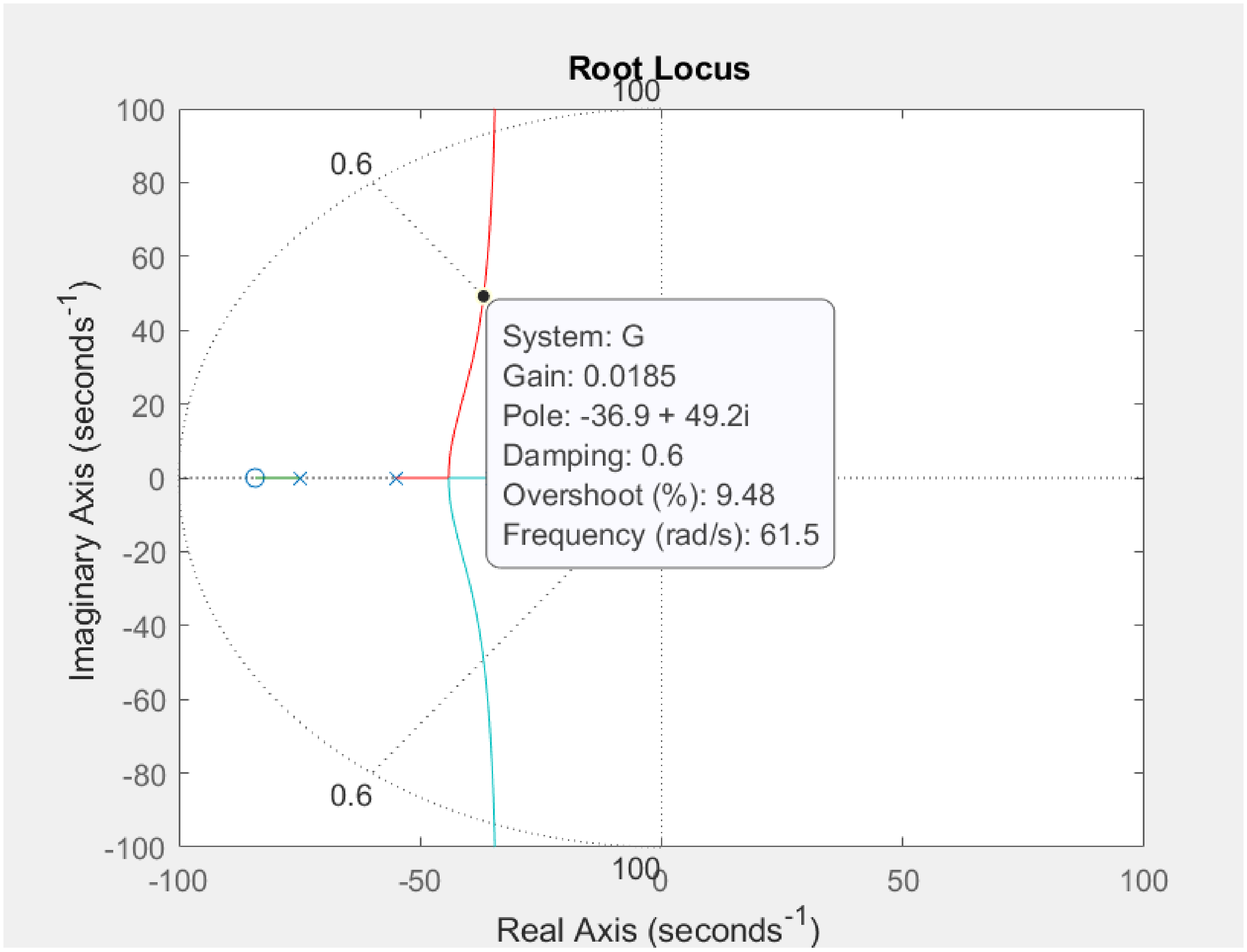


Figure 23: This figure depicts the root locus after the differential compensation was applied with zero at -84.2368 rad/s. Notice that the root locus intersects the ζ line at -36.9 ± j49.2. Gain K is at 0.0.0185.

The following MATLAB code was used to generate a step response with zero at -84.2368 rad/s and gain K = 0.0185.

```
s = tf('s');
G = ((144375)*(s+14)*(s+84.2368))/(s*(s+35)*(s+55)*(s+75));
K = 0.0185;
sys_cl = feedback(K*G,1);
step(sys_cl,1.26);
```

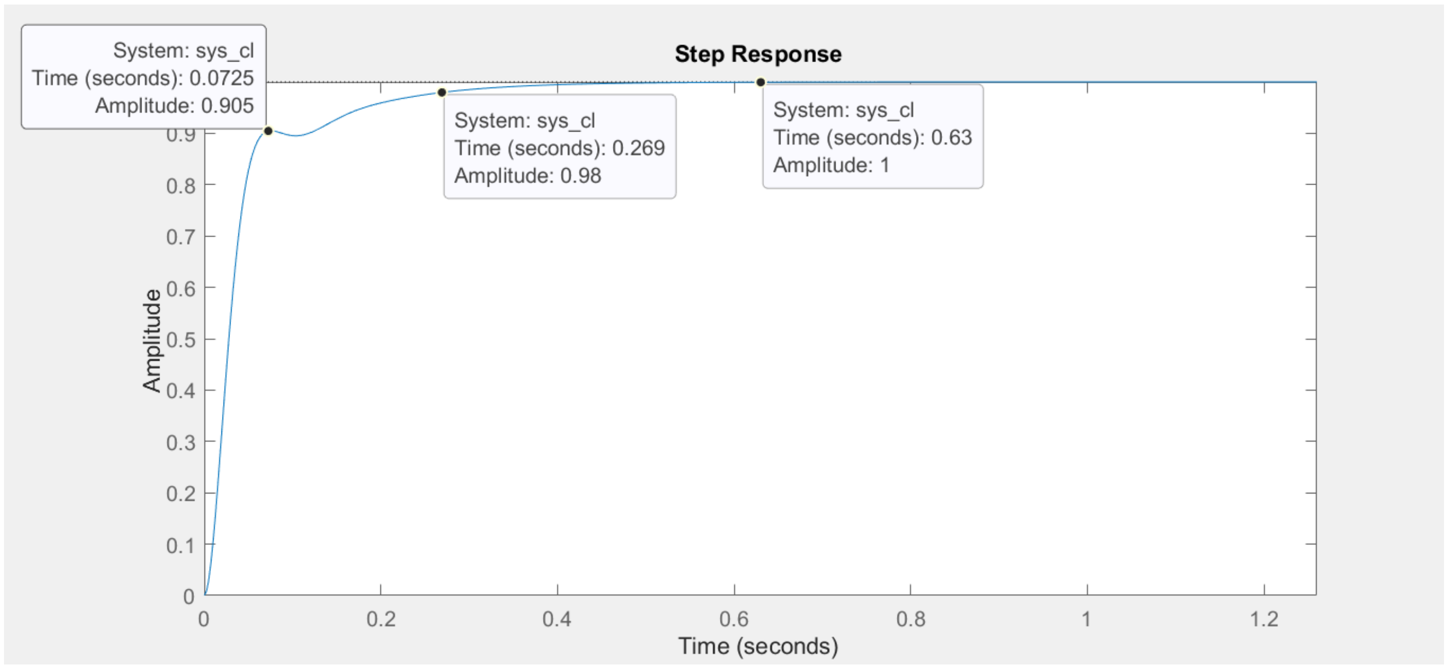


Fig 24: Step response with differential compensation and zero at -84.2368 rad/s, and gain K = 0.0185. Notice that $e_{step}(\infty) = 0, T_p = 72.5\ ms, T_s = 269ms$ ***and Final Value is 1.***

Table 8: PID performance Comparison			
Parameter	**Target**	**MATLAB result**	**%Δ**
Pole Location	-36.875± j49.12	-36.9± j49.2	0.13%
T_P	113 ms	72.5 ms	-43.66 %
T_s	-	269 ms	N/A
$\%OS$	9.48 %	0 %	-9.48 %
FV	1.0	1.0	☑

Table 8: Percent difference calculations between system before and after differential compensation with zero at -84.2368 rad/s.

Our target for the increase in time to peak was 72.16 ms, which was a 20% improvement on 90.20 ms. However, we are just short of our goal. We can adjust gain K to 0.02 and see if that yields a value closer to our target.

The following MATLAB code was used to generate a step response with zero at -84.2368 rad/s and gain K = 0.02.

```
s = tf('s');
G = ((144375)*(s+14)*(s+84.2368))/(s*(s+35)*(s+55)*(s+75));
K = 0.02;
sys_cl = feedback(K*G,1);
step(sys_cl,1.26);
```

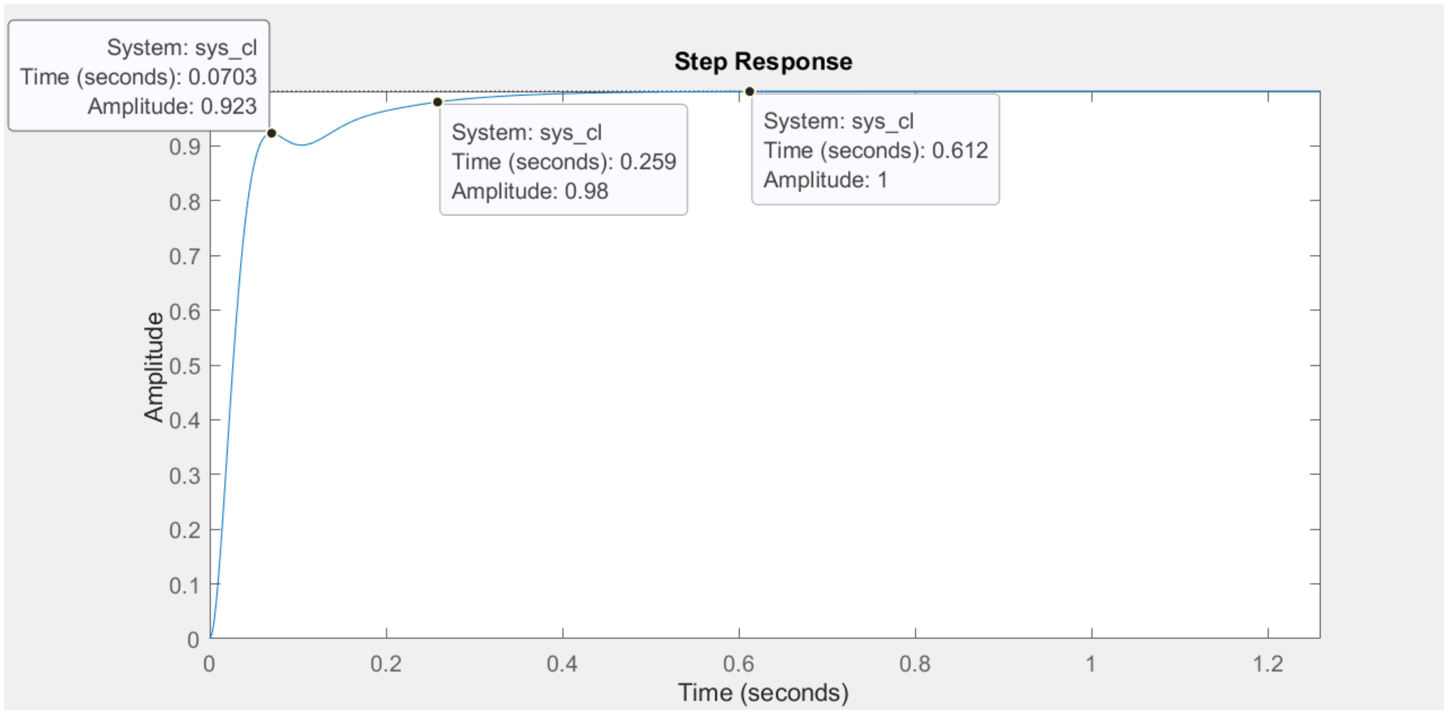


Fig 25: Step response with differential compensation and zero at -84.2368 rad/s, and gain K = 0.02. Notice that $e_{step}(\infty) = 0, T_p = 70.3\ ms, T_s = 259ms$ and Final Value is 1.

Table 9: PID Performance Comparison After Gain Adjustment				
Parameter	**Target**	**K = 0.0185**	**K = 0.02**	**% Δ**
T_P	72.16 ms	72.5 ms	70.3 ms	-2.61 %
T_s	-	269 ms	259 ms	-3.29%
$\%OS$	≤ 9.84 %	0 %	0%	0%
FV	-	1.0	1.0	☑

Table 9: Percent difference calculations between system before and after gain adjustment.

V.iii: G_{PID} and PID Gains

$$G_{PID}(s) = \frac{0.02(s+14)(s+84.2368)}{s} = \frac{0.02s^2 + 1.96474s + 23.5863}{s} \quad [15]$$

$$G_{PID}(s) = \frac{0.02s^2}{s} + \frac{1.96474s}{s} + \frac{23.5863}{s} \quad [16]$$

$$G_{PID}(s) = 0.02s + 1.96474 + \frac{23.5863}{s} \quad [17]$$

$$PID\ Gains \rightarrow K_P = 1.96474, K_I = 23.5863, K_D = 0.02$$

Table 9.1: PID Performance Parameters

Parameter	**Target**	**K = 0.02**	**% Δ**
T_P	72.16 ms	70.3 ms	-2.61 %
T_S	-	259 ms	N/A
$\%OS$	≤ 9.84 %	0%	N/A
FV	-	1.0	☑

Table 9.1: Percent difference calculations between target goals and performance parameters.

Section VI: Circuit Simulation of PID Controller

After designing the PID controller to meet our target values in MATLAB, component values were calculated to yield the proper performance and the controller was implemented using OpAmps in LTSPICE.

VI.i: Calculation of PID Component Values

We calculated the three gain values in the previous section and will use those to calculate component values.

They were:

$$G_{PID}(s) = 0.02s + 1.96474 + \frac{23.5863}{s} \qquad [18]$$

The proportional channel gain needs to be (-) 1.96474.

The Integral channel needs to be (-) 23.5863.

The Derivative channel needs to be (-) 0.02.

The Proportional gain needs to be (-) 1.96474. The gain is calculated as (-)Rf/Rs

Staying with 10kΩ as our basis resistor value, Rf would need to be

$$1.96474 * 1E4 = 19647.4\Omega \qquad [19]$$

The Integral gain needs to be (-) 23.5863. The gain is calculated as (-) $\frac{1}{R_I * C_I}$.

Staying with 10kΩ as our basis resistor value C_I would have to be

$$C_I = \frac{1}{(1E4)(23.5863)} = 4.24\mu F \qquad [20]$$

The Derivative gain needs to be (-) 0.02. The gain is calculated as (-) $R_D C_D$.

Staying with 10kΩ as our basis resistor value C_D would have to be.

$$C_D = \frac{0.02}{1E4} = 2\mu F \qquad [21]$$

With all the calculations complete each part of the compensator was implemented into the circuit simulator and is depicted in the next section.

VI.ii: Complete System

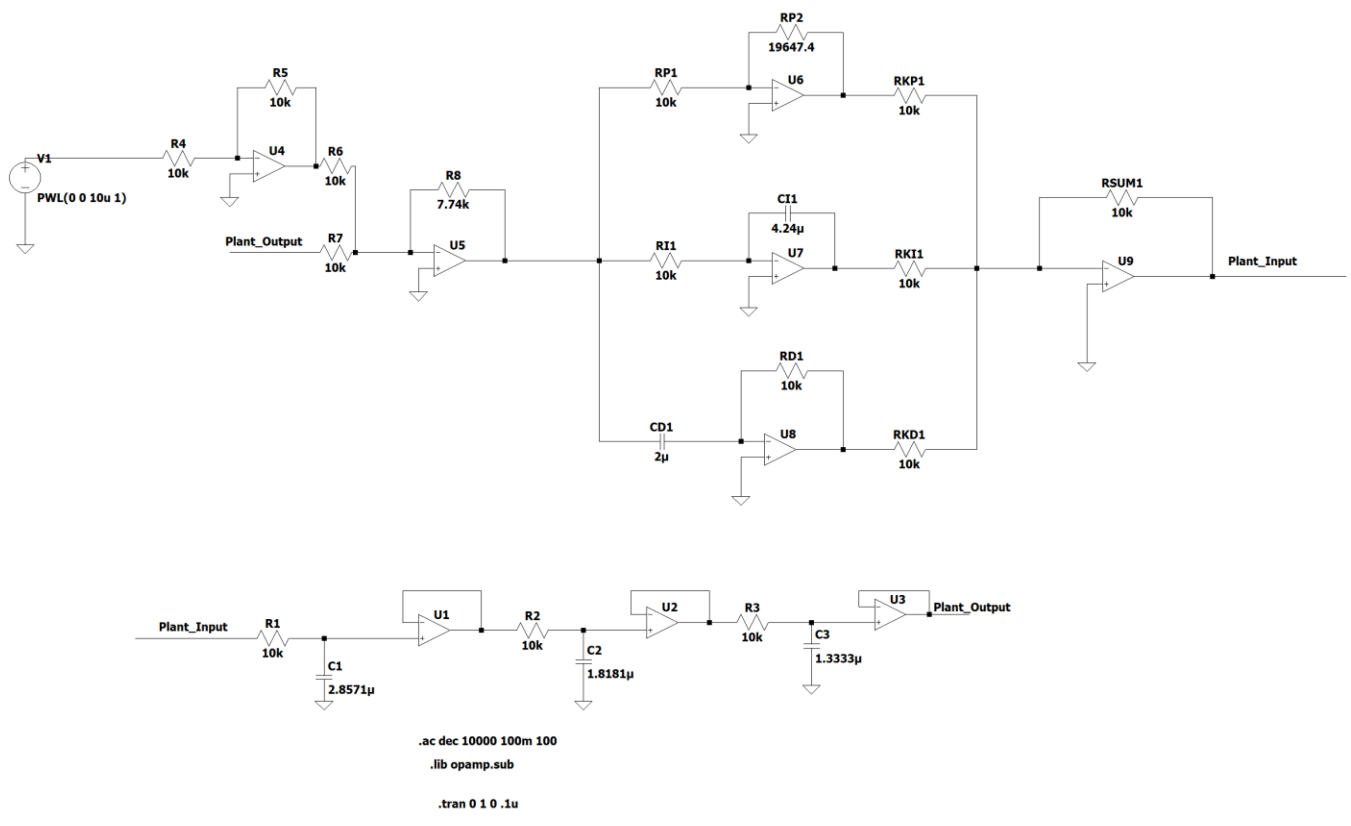
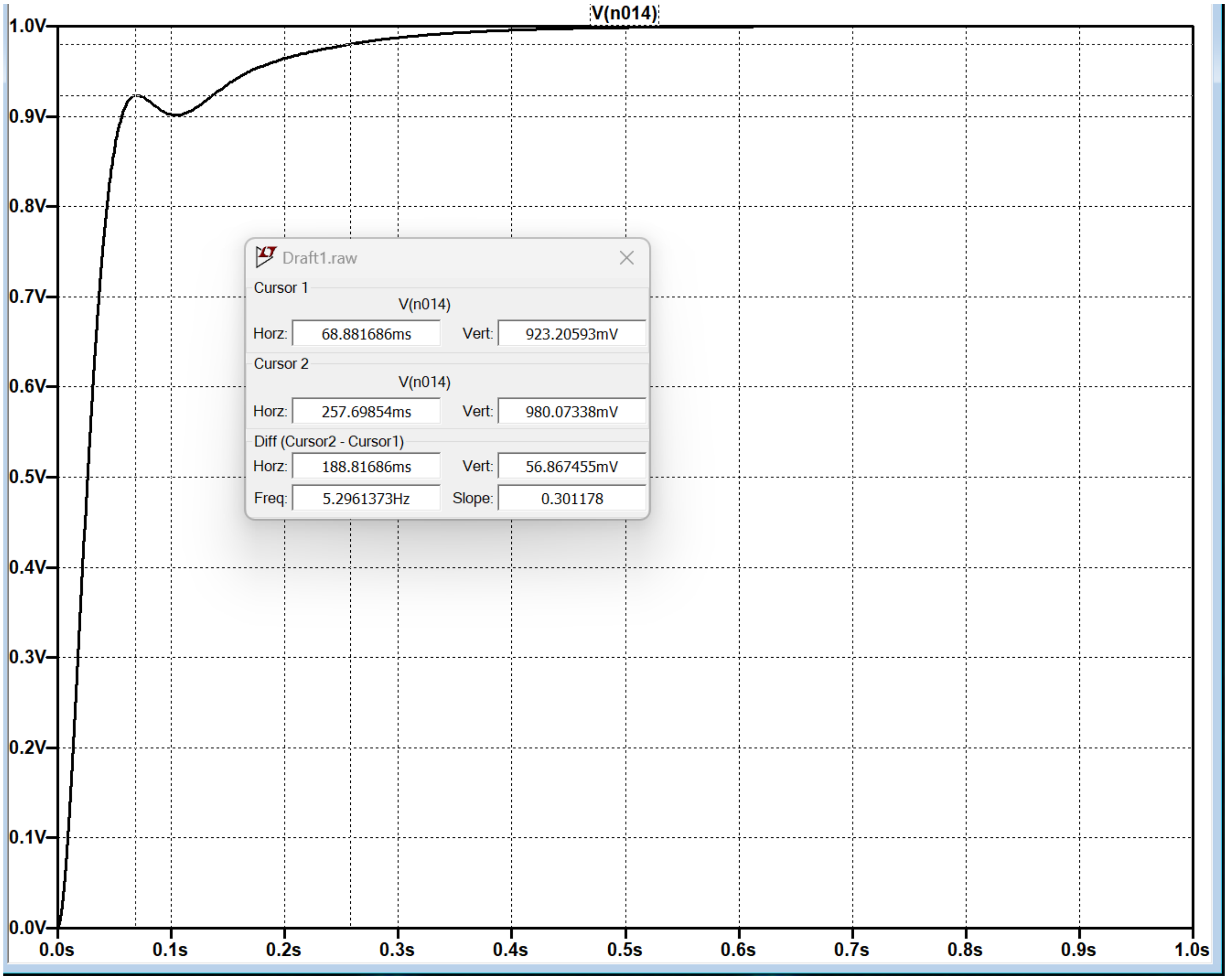


Figure 26: PID Controller, Plant, and summing junction as captured in LTSPICE circuit simulator.

VI.iii: Circuit Simulator Step Response.



```
Cursor 1
                V(n014)
Horz: 68.881686ms      Vert: 923.20593mV
Cursor 2
                V(n014)
Horz: 257.69854ms      Vert: 980.07338mV
Diff (Cursor2 - Cursor1)
Horz: 188.81686ms      Vert: 56.867455mV
Freq: 5.2961373Hz      Slope: 0.301178
```

Figure 27: Complete system step response in LTSPICE circuit simulator with Tp marked by cursor 1 occurring at approximately 68.88 ms with a magnitude of 923.2 mV, and Ts marked by cursor 2 occurring at approximately 257.7 ms with a magnitude of 980 mV.

VI.iv: Previous MATLAB Step Response

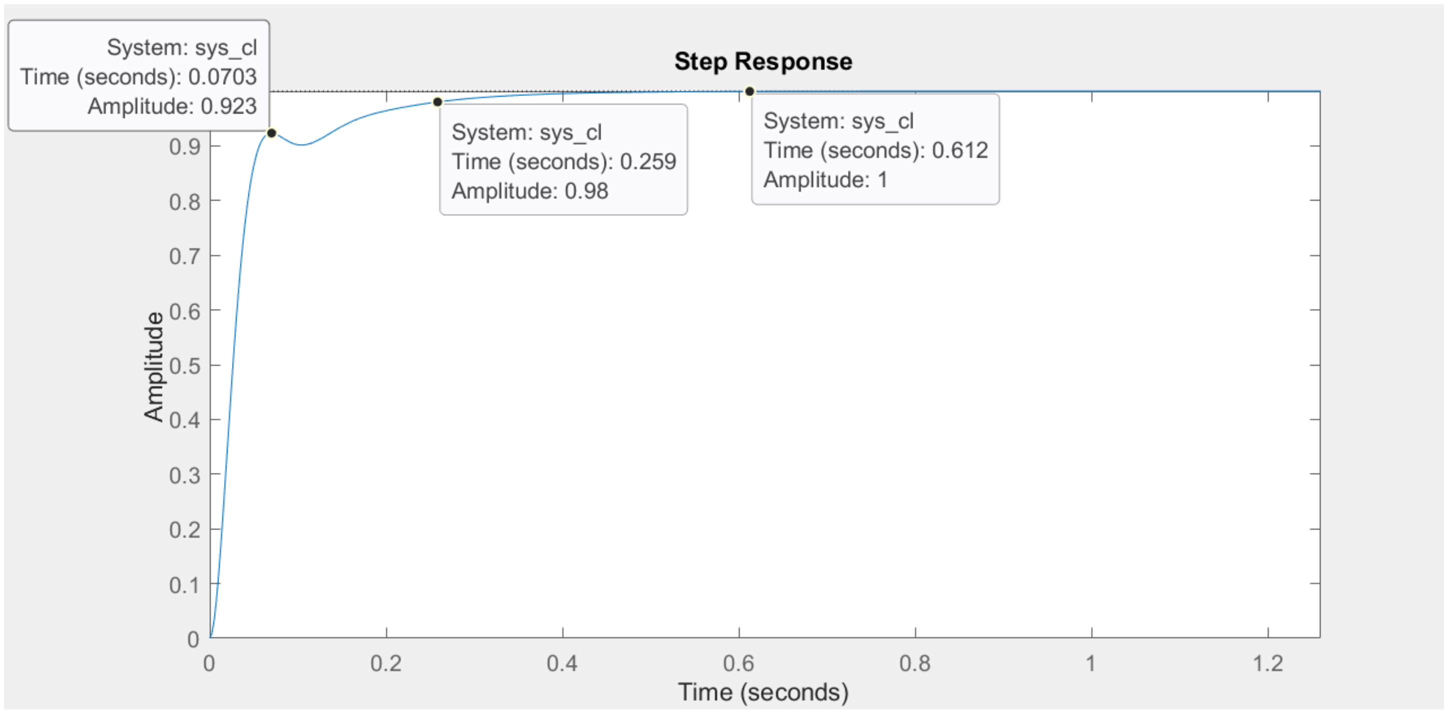


Figure 28: MATLAB step response with differential compensation and zero at -84.2368 rad/s, and gain K = 0.02. Notice that $\boldsymbol{e_{step}(\infty) = 0, T_p = 70.3\ ms, T_s = 259ms}$ ***and Final Value is 1.***

VI.v: Performance Comparison

Table 10: PID Performance Comparison MATLAB vs. LTSPICE Circuit Simulator				
Parameter	**Target**	**MATLAB**	**LTSPICE**	**%Δ**
Tp	72.16ms	70.3 ms	68.88 ms	2.04 %
Ts		259 ms	257.7 ms	0.5 %
%OS	≤ 9.48%	0%	0%	0 %
FV		1.0	1.0	☑

Table 10: Target goals alongside MATLAB and LTSPICE results showing a percentage difference between MATLAB and LTSPICE simulations.

After comparing the values between the MATLAB and LTSPICE simulations there appears to be a 2.04% difference between the values for time to peak and a 0.5% difference for the values of time to settle. This is within an expected margin of error. Both systems still show that there is 0% OS and $\boldsymbol{e_{step}}(\infty) = \mathbf{0}$ for the compensated system.

Section VII: PID Tuning

For this section the system was tuned using the Ziegler – Nichols method which sets the derivative and integral gain to zero leaving only the proportional gain active. Once the proportional gain is active it is gradually increased to put the system in to a state of marginal stability. Once the system is marginally stable the period between the oscillations is measured to find the critical period. Once the critical gain and period are found the table below can be populated to find the formula for the PID controller.

Table 11: Gain Constants or Ziegler - Nichols Tuning

Type of Controller	K_p	T_i	T_d
P	0.5 $*K_{critical}$	∞	0
PI	0.45 * $K_{critical}$	0.833 $*$ $P_{critical}$	0
PID	0.6 * $K_{critical}$	0.5 * $P_{critical}$	0.125 * $P_{critical}$

Table of gain constants for Ziegler – Nichols Tuning.

$$GPID(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \qquad [22]$$

Once the initial formula for GPID is found it can be fine-tuned to achieve the target goals for the project.

V.II.i: Separating PID from Plant

The equation for the plant

$$G_{plant}(\mathrm{s}) = \frac{144375}{(\mathrm{s}+35)(\mathrm{s}+55)(\mathrm{s}+75)}$$

The equation for the PID

$$G_{PID}(s) = 0.02s + 1.96474 + \frac{23.5863}{s}$$

Once the plant and the PID were separated the value of gain k was set to 1 and the step response was administered and captured in MATLAB using the code below to verify that the results to the step response were unchanged indicating that the separation had been performed correctly and that the system was responding as expected.

```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = (0.02*s)+1.96474+23.5863/s;
G = Gplant*GPID;
K = 1;
sys_cl = feedback(K*G,1);
step(sys_cl,1);
axis([0 1.2 0 1.2]);
```

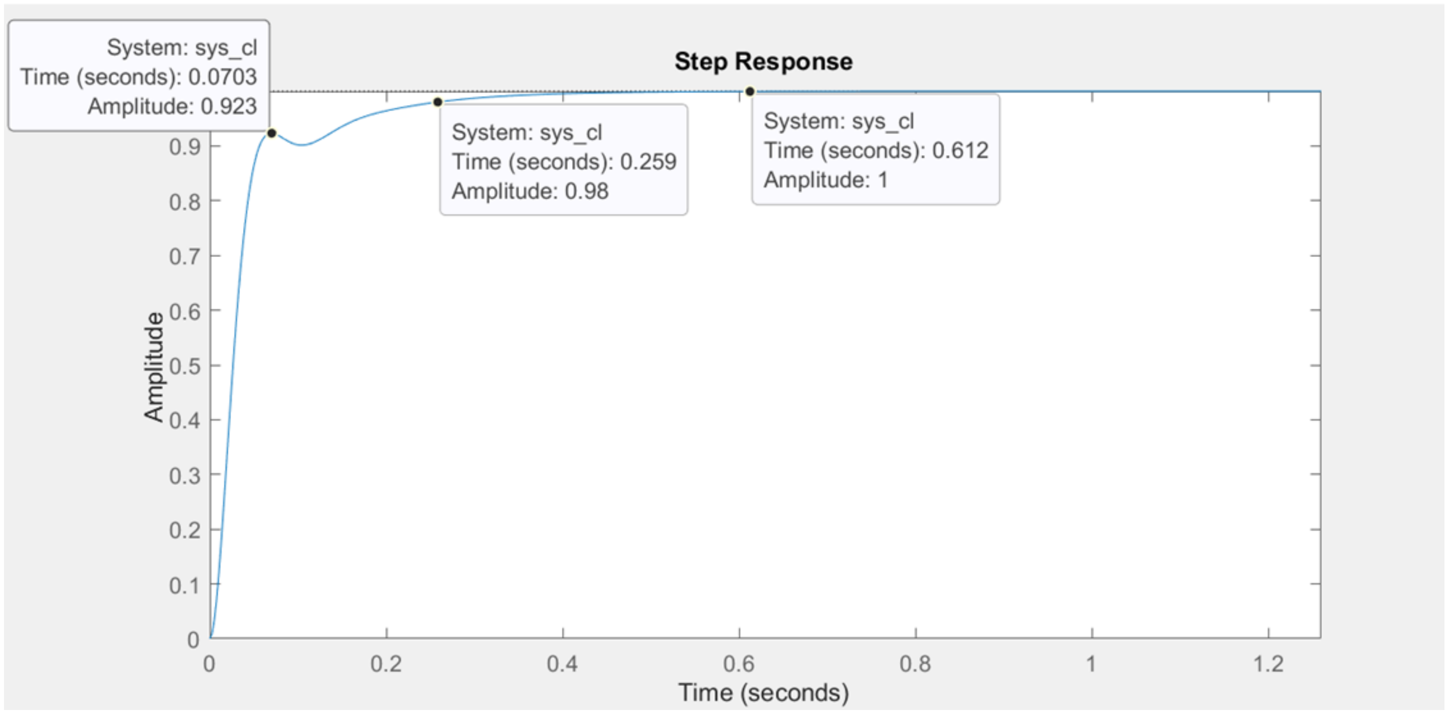
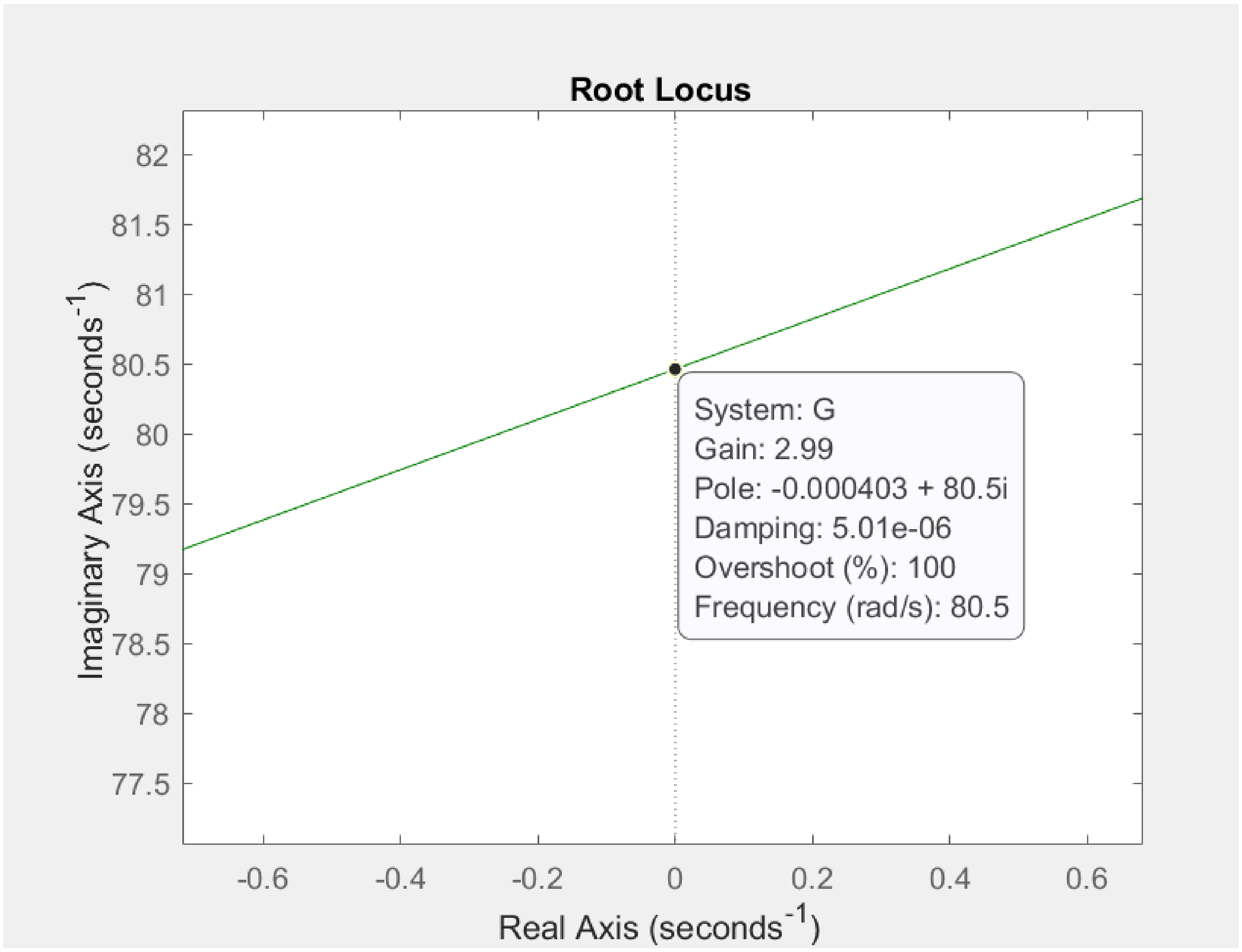


Figure 29: MATLAB step response with plant separated from PID controller based on differential compensation and zero at -84.2368 rad/s, and gain K = 0.02. Notice that $e_{step}(\infty) = 0, T_p = 70.3\ ms, T_s = 259ms$ ***and Final Value is 1.***

VII.ii: Ziegler – Nichols

To perform the Ziegler – Nichols tuning the other gain constants must be forced to zero leaving the proportional gain K_p as the only gain constant that is in the controller. Performing this operation allows us to use the root locus to determine the maximum gain before the system is marginally stable. The following MATLAB code was used to run the root locus and find the desired value of gain to proceed with the tuning.

```
s = tf('s');
Gplant = 144375 / ((s+35) * (s+35) * (s+75));
GPID = (0.0*s + 1.96474 + 0/s);
G = Gplant * GPID;
zeta  = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
```



System: G
Gain: 2.99
Pole: -0.000403 + 80.5i
Damping: 5.01e-06
Overshoot (%): 100
Frequency (rad/s): 80.5

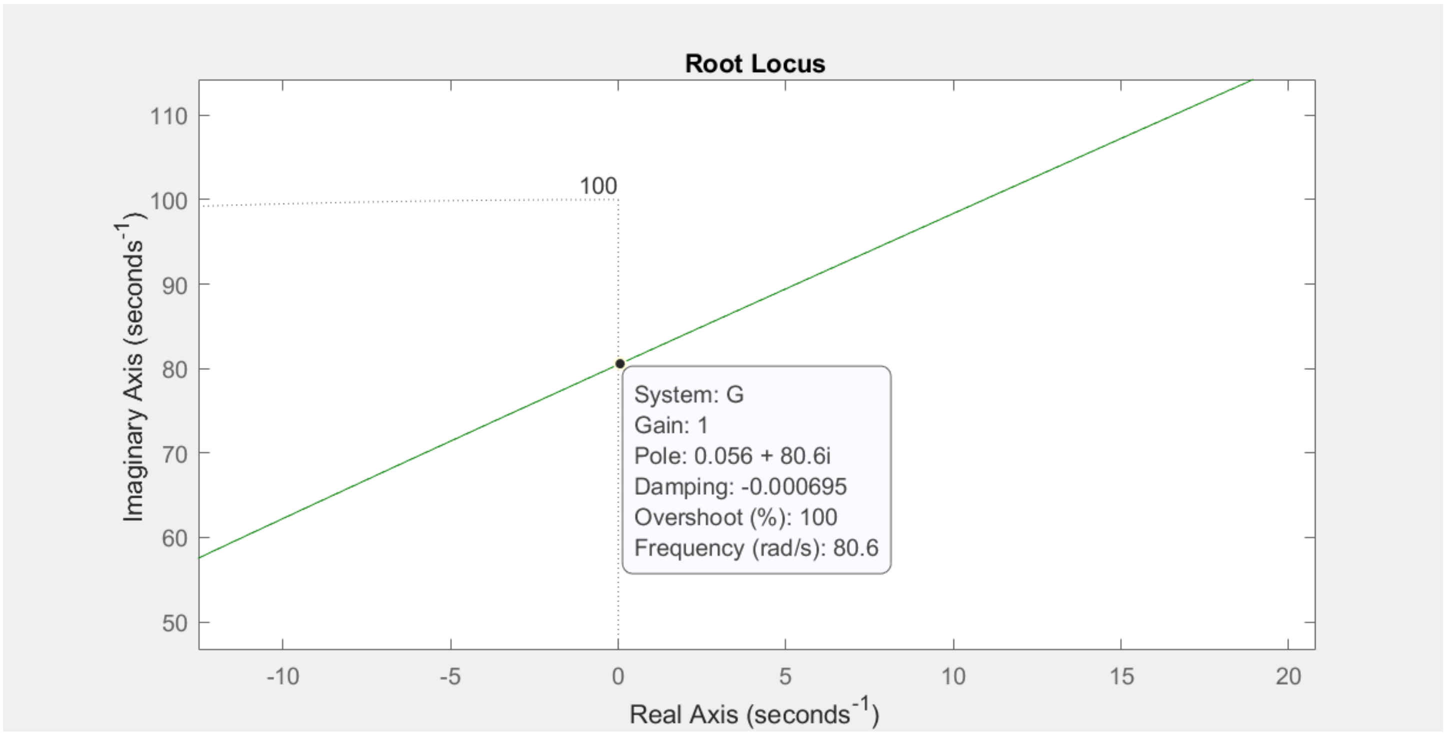
Figure 30: MATLAB root locus plot showing system instability when gain is set to 2.99 and a frequency of 80.5 rad/s.

In this manner if the gain is increased by a factor of 3 the critical gain value will be achieved.

$$K_{critical} = K_p * 3 = 1.96474 * 3 = 5.89422$$

The following MATLAB code was ran to verify

```
s = tf('s');
Gplant = 144375 / ((s+35) * (s+35) * (s+75));
GPID = (0.0*s + 5.89 + 0/s);
G = Gplant * GPID;
zeta  = 0.6;
wn = 100;
rlocus(G);
sgrid(zeta,wn);
```



System: G
Gain: 1
Pole: 0.056 + 80.6i
Damping: -0.000695
Overshoot (%): 100
Frequency (rad/s): 80.6

Figure 31: MATLAB root locus plot validating the new value for $K_{critical}$ at 5.89 showing that there is a gain value of 1 when the system is marginally stable.

With the value for $K_{critical}$ verified the following MATLAB code was used to run the step response with the new value to examine the results and see if the system is marginally stable.

```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = 0.0*s + 5.89 + 0/s;
G = Gplant*GPID;
sys_cl = feedback(G,1);
step(sys_cl,1);
```

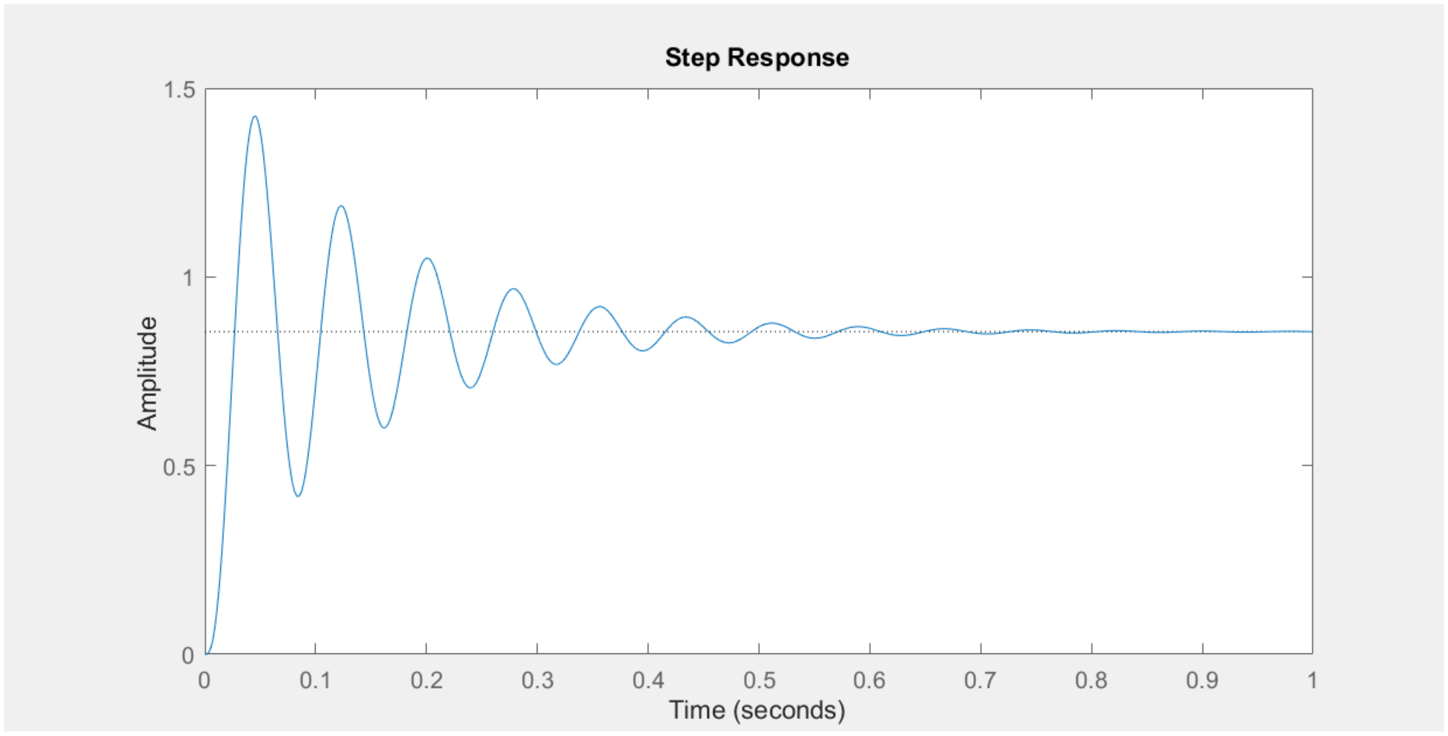


Figure 32: MATLAB step response with value for $K_{critical}$ at 5.89. Due to the lack of infinite oscillations we can tell that this system is not marginally stable.

It is evident that the system is not marginally stable at the initial $\mathrm{K_{critical}}$ alue, as indicated by the absence of sustained oscillations. To accurately determine the gain value that would lead to marginal stability, the Routh-Hurwitz stability criterion was employed. Through this analysis, the critical gain $\mathrm{K_{critical}}$ was found to be 8.9, the value at which the system exhibits marginal stability. The updated gain was then used in MATLAB to generate the step response plot shown in Figure 28. From this plot, the critical period $P_{critical}$ was measured using MATLAB's data cursor tool and calculated to be 70.4 ms.

```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = 0.0*s + 8.9 + 0/s;
G = Gplant*GPID;
sys_cl = feedback(G,1);
step(sys_cl,1);
```

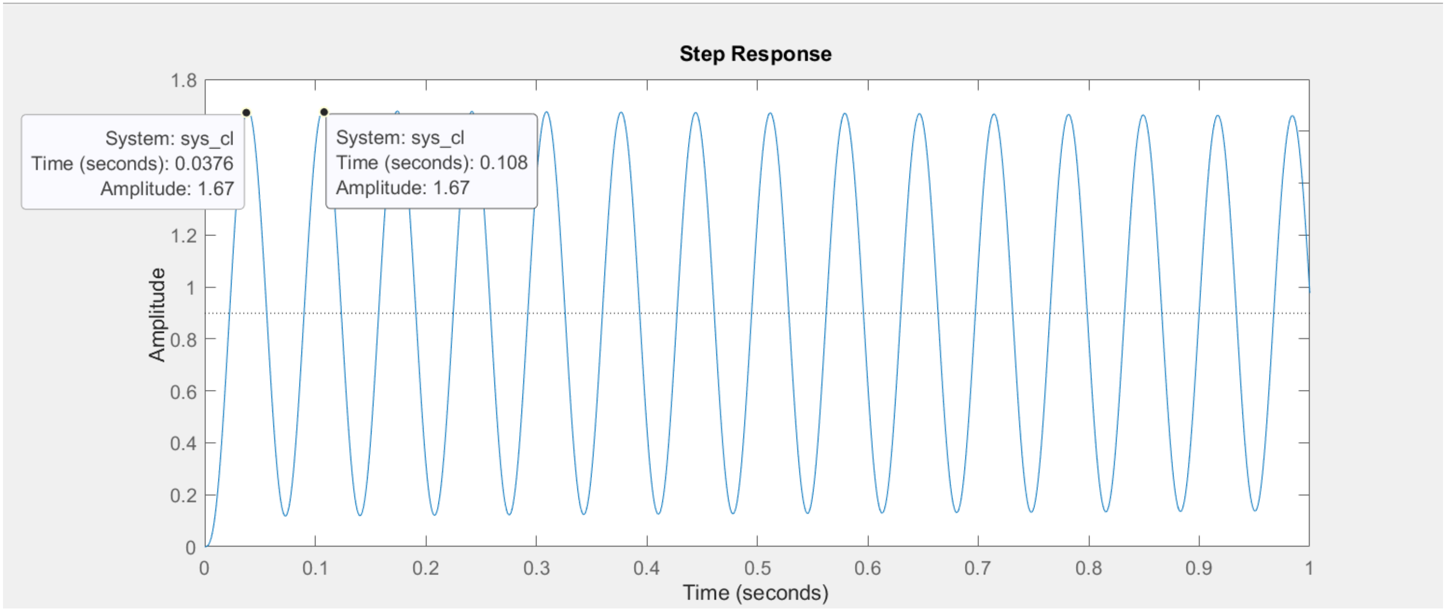


Figure 33: MATLAB step response with value for $K_{critical}$ at 8.9. We can tell from the infinite oscillations that this system is marginally stable and that the critical period can be measured using the peaks of the sinusoid and taking the difference between them.

$$P_{critical} = 0.108 - 0.0376 = 0.0704 = 70.4\,ms \qquad [23]$$

Now that we have the values for $P_{critical}$ and $K_{critical}$ we can use these values to populate our table below and design our PID controller

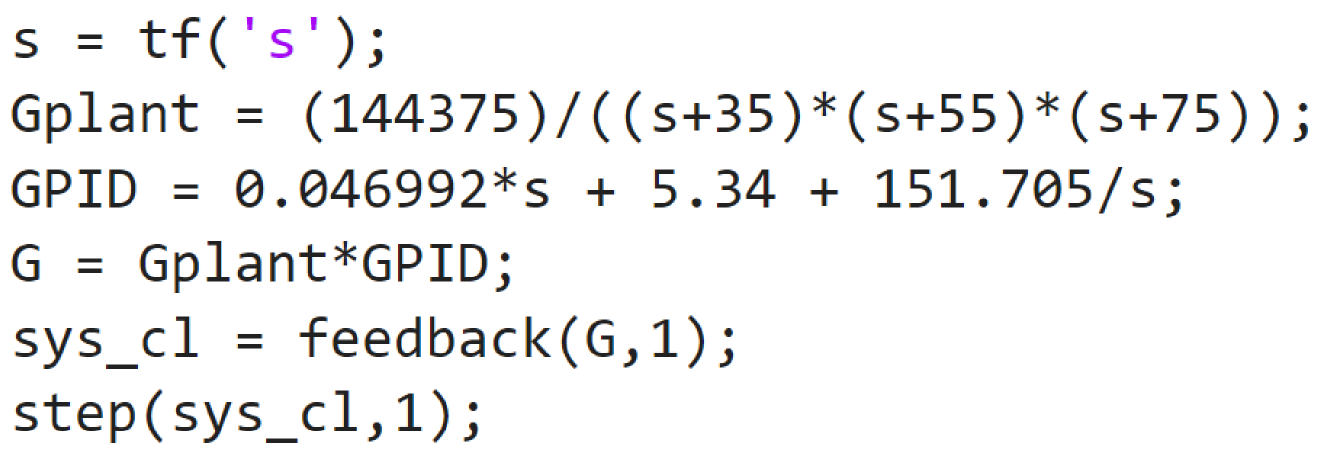
Plugging the values into our table gives us a new equation for our GPID(s)

$$GPID(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \qquad [22]$$

$$= 5.34\left(1 + \frac{1}{0.0352s} + 0.0088s\right)$$

$$= 5.34 + \frac{151.705}{s} + 0.046992s$$

This equation was entered into MATLAB and the step response was run using the following code to generate the plot depicted in Figure 34.



```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = 0.046992*s + 5.34 + 151.705/s;
G = Gplant*GPID;
sys_cl = feedback(G,1);
step(sys_cl,1);
```

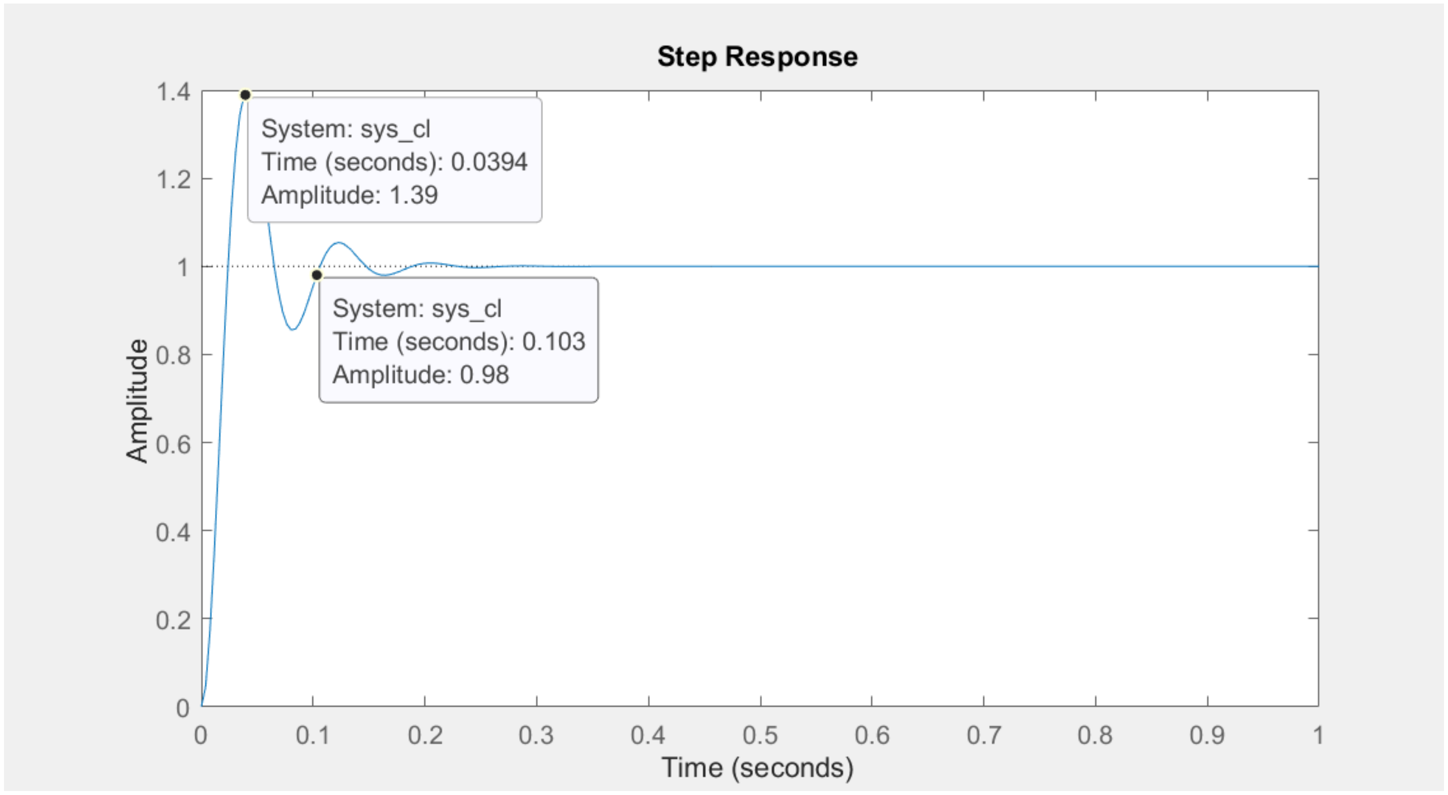


Figure 34: MATLAB step response with new PID formula showing a T_p to be 39.4 ms and a T_s to be 103 ms. The percent overshoot for this system is 41.84%.

This system has an overshoot that is well above our target of 9.48%. We can try to adjust this by keeping our proportional gain the same, reducing our integral gain and increasing our derivative gain. We will start by decreasing the integral gain by 50% and double the derivative gain.

$$GPID(s) = 5.34 + \frac{75.85}{s} + 0.094s$$

The following MATLAB code was ran using the new PID values.

```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = 0.094*s + 5.34 + 75.85/s;
G = Gplant*GPID;
sys_cl = feedback(G,1);
step(sys_cl,1);
```

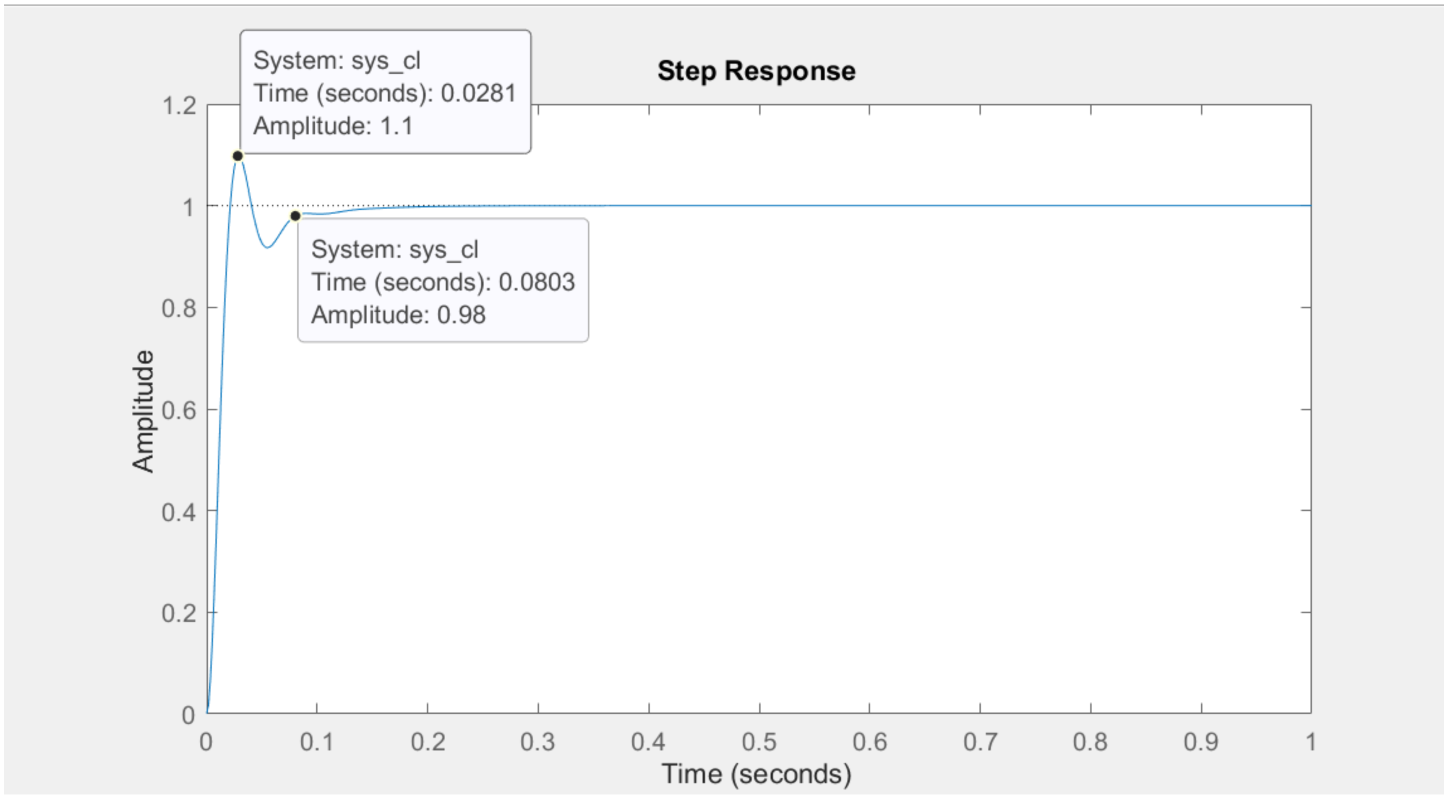


Figure 35: MATLAB step response with new PID formula showing a T_p to be 28.1 ms and a T_s to be 80.3 ms. The percent overshoot for this system is 12.25%.

This has imroved our percent overshoot significantly but we are still not close to where we would like to be let us reduce the integral gain by 20% and increase the derivative gain by 20%.

$$GPID(s) = 5.34 + \frac{60.68}{s} + 0.1128s$$

The following was run using the new PID values. The corresponding step response plot can be seen in figure 36.

```
s = tf('s');
Gplant = (144375)/((s+35)*(s+55)*(s+75));
GPID = 0.1128*s + 5.34 + 60.68/s;
G = Gplant*GPID;
sys_cl = feedback(G,1);
step(sys_cl,1);
```

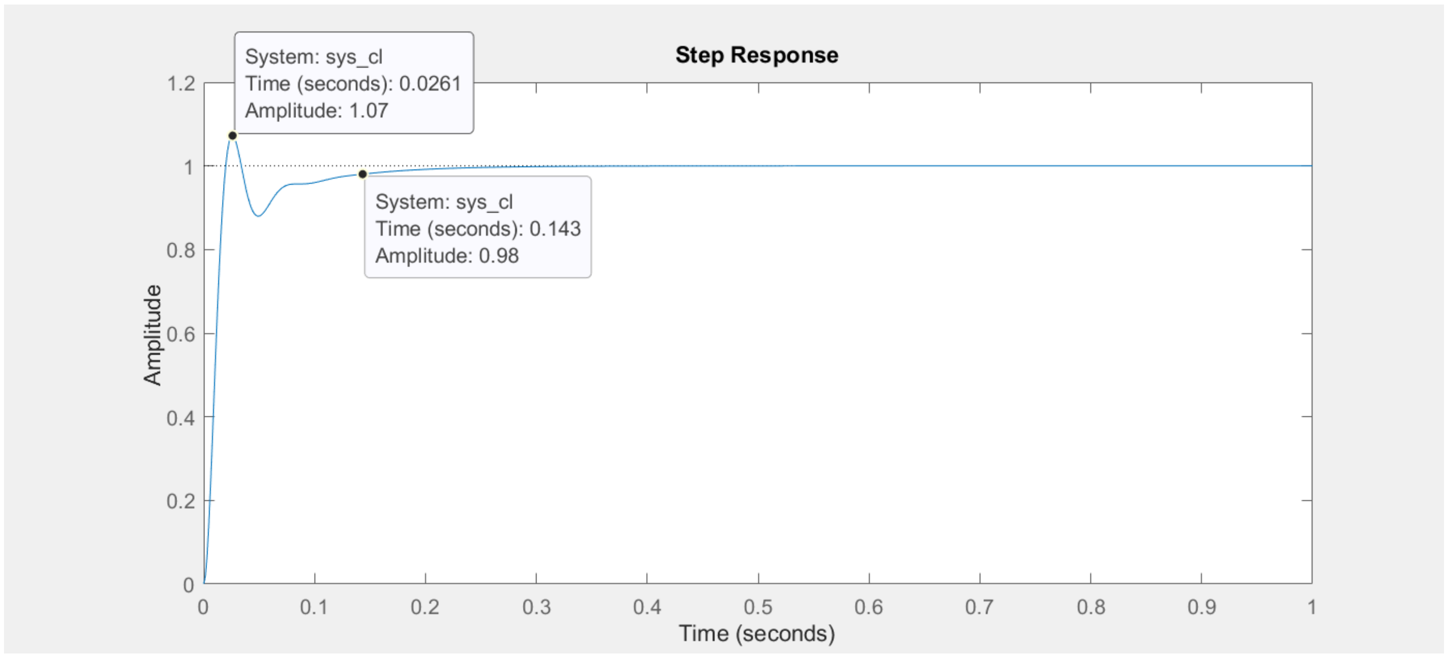


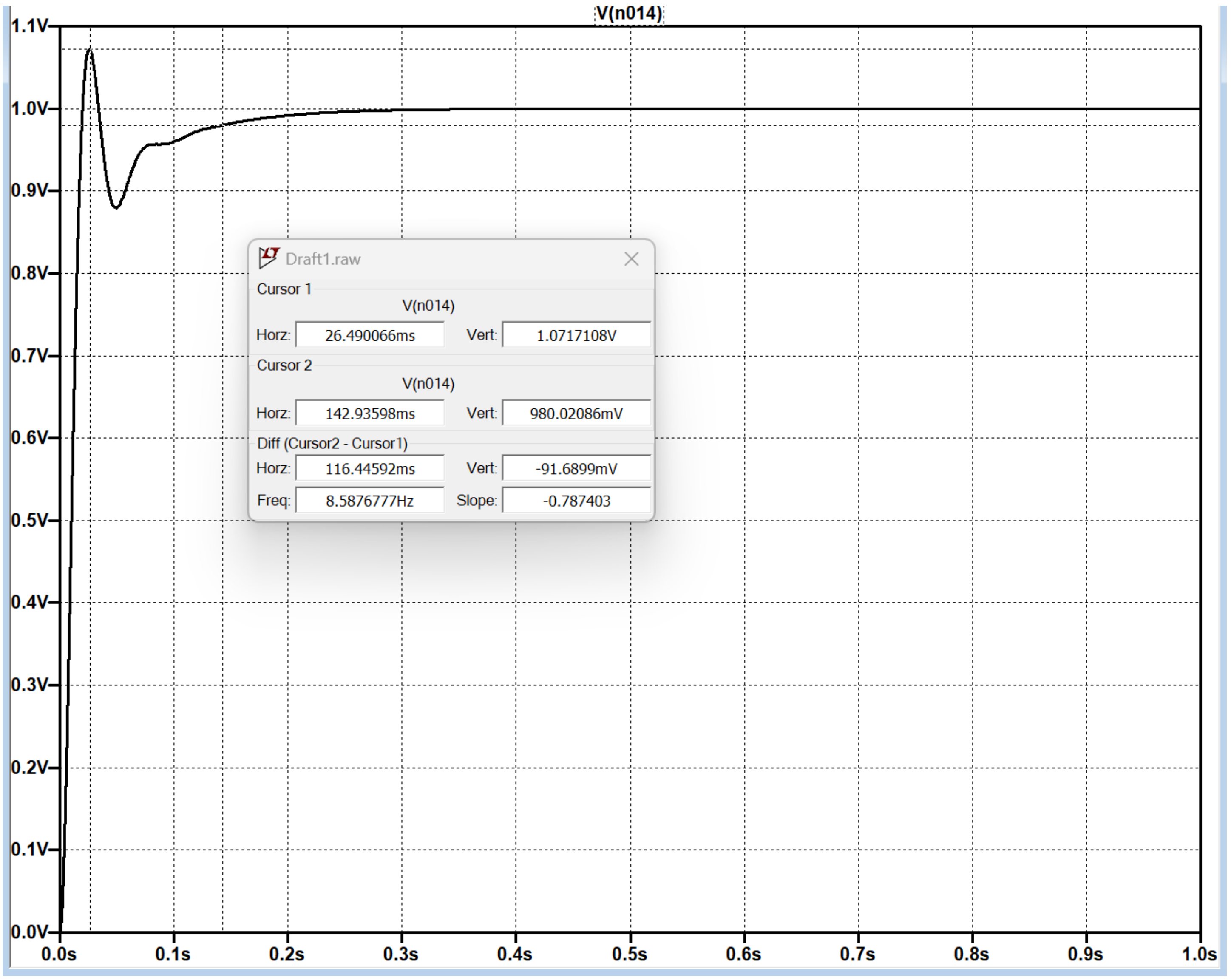
Figure 36: MATLAB step response with new PID formula showing a T_p to be 26.1 ms and a T_s to be 143 ms. The percent overshoot for this system is 9.18%.

Using the new Ziegler – Nichols method for tuning the PID controller we were able to achieve our target goals for maximum overshoot of 9.48% and reduce our time to peak by 20%.

Table 12: PID Performance After Ziegler – Nichols Tuning			
Parameter	**Target**	**Achieved**	**% Δ**
T_P	72.16 ms	26.1 ms	-93.75%
T_s	-	143 ms	-
$\%OS$	$\leq$ 9.84 %	9.18%	-6.94%
FV	-	1.0	☑

VII.iii: LTSPICE Verification

After the system was tuned in MATLAB the same techniques used in Section VI were implemented to design the circuit and run the step response in LTSPICE. The results can be seen below in Figure 37.



Cursor 1
V(n014)
Horz: 26.490066ms Vert: 1.0717108V
Cursor 2
V(n014)
Horz: 142.93598ms Vert: 980.02086mV
Diff (Cursor2 - Cursor1)
Horz: 116.44592ms Vert: -91.6899mV
Freq: 8.5876777Hz Slope: -0.787403

Figure 37: MATLAB step response with new PID formula showing a T_p to be 26.49 ms and a T_s to be 142.93 ms. The percent overshoot for this system is 9.18%.

Table 13: PID Performance Comparison Ziegler - Nichols MATLAB vs. LTSPICE Circuit Simulator				
Parameter	**Target**	**MATLAB**	**LTSPICE**	**%Δ**
Tp	72.16ms	26.1 ms	26.49 ms	1.48%
Ts		143 ms	142.93 ms	-0.05%
%OS	≤ 9.48%	9.18%	9.18%	0%
FV		1.0	1.0	☑

Table 13: Target goals alongside MATLAB and LTSPICE results showing a percentage difference between MATLAB and LTSPICE simulations.

Section VIII: Conclusion and Discussion

VIII.i

PID controllers offer several advantages, including straightforward implementation using basic components and the ability to deliver desirable response characteristics, such as eliminating steady-state error when properly tuned. Their versatility allows them to be applied across a wide range of systems and implemented in both analog and digital formats. However, PID controllers face challenges when dealing with systems that exhibit significant time delays or high levels of nonlinearity. Additionally, the derivative term can amplify system noise. Tuning a PID controller can also be time-consuming, often requiring multiple iterations to fine-tune the control parameters for optimal performance.

VIII.ii

For this project, my primary research on PID tuning came from several YouTube tutorials, particularly those explaining the Ziegler–Nichols method for controller tuning. I also utilized MATLAB's official YouTube channel, which provides in-depth coverage of PID controllers. These resources were instrumental in helping me adjust the integral and derivative gain constants to minimize system overshoot. Through this process, I successfully achieved my target overshoot and exceeded my time-to-peak improvement goal by more than 90%.

VIII.iii

By analyzing Table 13 in the previous section, it is evident that MATLAB and LTspice produced very similar results in the step response analysis. The largest percentage difference occurred in the time-to-peak value at +1.48%, while the time-to-settle differed by only -0.05%. Both simulations yielded an identical percent overshoot of 9.18%, resulting in a 0% difference. These slight discrepancies can be attributed to the differing modeling approaches: MATLAB relies on purely mathematical models, whereas LTspice attempts to simulate the behavior of physical components. As a result, minor variations in the step response plots are expected. However, these differences fall within acceptable tolerances when compared to the real-world components used in the Texas State University laboratories. Given that MATLAB operates under idealized conditions, I believe the LTspice simulation results are more representative of what would be observed in a physical, analog implementation of the circuit.

VIII.iv

A digital implementation of a PID controller differs significantly from the analog version presented in this report, primarily in how it processes signals and performs calculations. The analog design uses continuous signals manipulated through op-amps, resistors, and capacitors. In contrast, a digital PID controller samples signals discretely via analog-to-digital converters and processes them numerically using software algorithms. This shift from continuous to discrete processing fundamentally changes the system's operation and offers different advantages and challenges.

VIII.v

Throughout this PID controller design project, one of the most valuable learning experiences was gaining an appreciation for the discrepancies between theoretical design and practical implementation. MATLAB simulations provided a solid foundation in the mathematical principles of PID control, but transitioning to LTspice circuit simulations highlighted the real-world challenges that arise when modeling physical components. Observing how circuit-level implementation affects system performance was both insightful and instructive.

The process of tuning the controller using the Ziegler-Nichols method reinforced the delicate balance inherent in control system design. Optimizing one performance metric often comes at the expense of another, making it a continual process of trade-offs and fine adjustments. Additionally, this project deepened my understanding of the practical applications of second-order system approximations. The ability to simplify complex systems while still achieving accurate and useful predictions is a powerful analytical tool, particularly when designing and analyzing control systems.

Overall, this project has significantly enhanced my confidence in designing PID controllers and, more importantly, in anticipating and addressing the practical challenges that emerge during real-world implementation.