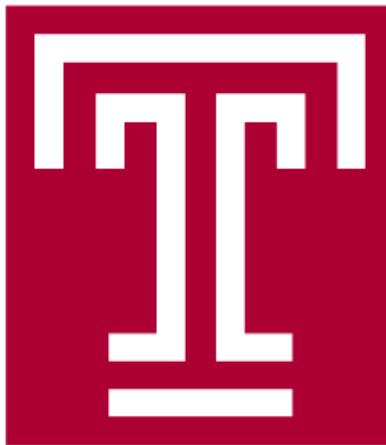


Temple University
College of Engineering
Department of Electrical and Computer Engineering (ECE)

Student Report Cover Page



Course Number: ECE 3412

Lab 7: Analytic approach to determining closedloop gain for a second order DC motor system

Student Name: Devin Trejo; Robert Irwin

TUid: 914924557; 914980083

Due Date: 3/31/2015

TA Name: Michael Korostelev

Grade: / 100

I. Introduction

Previously we have only experimented with implementing PID controllers into our systems, but sometimes we want to adjust the amount of correction in of PID controllers. A feedback system with proportional gain can easy adjustment to PID controllers without having to adjust the PID itself. Adjusting the proportional gain allows us to change the percent overshoot, and settling time of a system's response.

In this lab we will analyze the technique designers use to find to system responses. The lab confronts us using Routh-Hurwitz table to find the poles of a system. From the poles we apply our equations to find percent over shoot (%OS), and settling time (T_s). Also we incorporate Microelectronic circuit analysis to construct our PID controller using discrete analog components.

II. Procedure

The first step of the experiment required us to load raw data from the blackboard into the workspace. The data was very noisy, so we filtered it using a tenth order median filter. This filtered data was then passed to the System Identification Toolbox, where we estimated a second order transfer function. We then applied a proportionality gain controller to the system, and solved for the value of k that would result in a 10% overshoot. Once we found the new k , found the step response of the new transfer function (the transfer function with the controller applied). Once we had the step response, we validated that we did in fact get a 10% overshoot. We also plotted the root locus diagram and validated found the k value that would make the system critically damped.

Once this was completed, we repeated the process for a third order transfer function of the same filtered data. We redid all calculations, and analyzed an overshoot of 150%, and what happens when the proportionality constant grows too large.

III. Results

Below we see the filtered versus the unfiltered data produced by MatLab's median filter. The results are shown below.

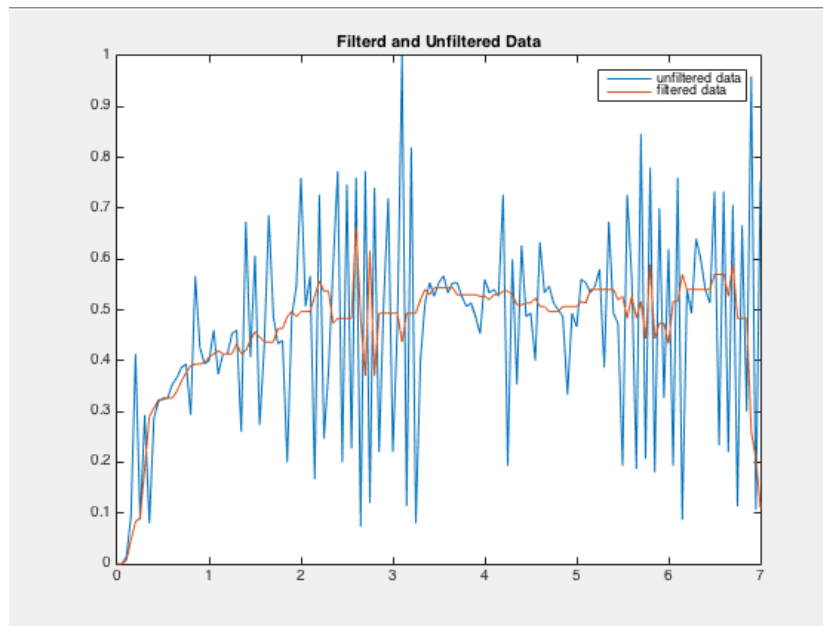


Figure 1: Filtered and Unfiltered Data

The blue curve represents the noisy, unfiltered data, and the orange curve represents the filtered data. We see that a tenth order median filter does a decent job of filtering out the noise in the data.

Second Order Analysis

Now we look at the transfer function the System Identification Toolbox produced.

```
tf1 =  
  
From input "u1" to output "y1":  
3986  
-----  
s^2 + 434.9 s + 3949
```

We see the second order transfer function is very close to standard form. This is useful because we can use our zeta and omega equations that describe a second order response. We can validate this transfer function by plotting it against the data.

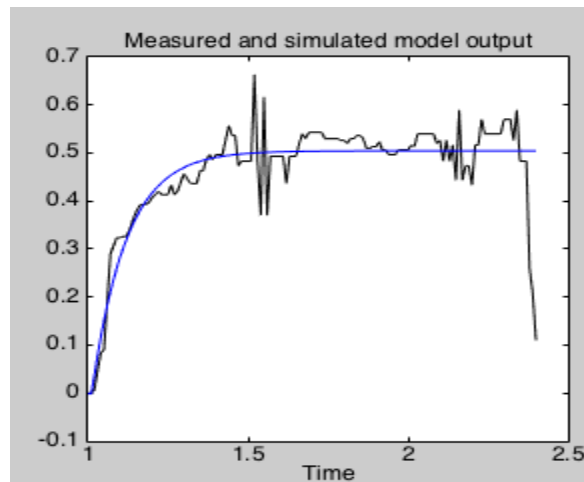


Figure 2: Validate Transfer Function

From the plot above we can see that the transfer function is a good estimate of the data.

Now that we know that our transfer function accurately describes the system, we can plot the root locus diagram.

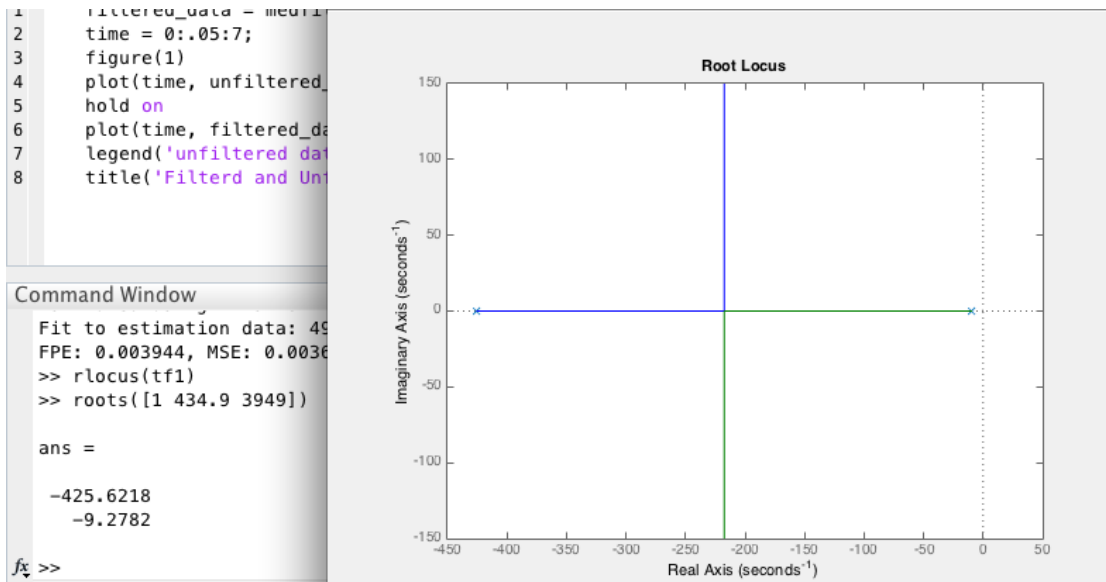


Figure 3: Root Locus for Second Order

We see that the original poles are at -425.6218 and -9.2782. If we compare these values to the original pole locations on the root locus, everything is in agreement.

We will now solve for the k that results in a 10% overshoot and a .05sec settling time. To find k we must first find the value of ζ .

$$\%OS = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} * 100\% = .01$$

$$\ln(.01) = -\frac{\pi\zeta}{\sqrt{1-\zeta^2}}$$

$$\left(\frac{\ln(.01)}{\pi}\right)^2 = \frac{\zeta^2}{1-\zeta^2}$$

$$\zeta = .5912$$

We now solve for ω_n using the following equation.

$$2\zeta\omega_n = 434.9$$

$$\omega_n = 367.8112$$

Now we can solve our natural frequency constraint equation to find the value of k .

$$\omega_n = 367.8112 = \sqrt{(3949 + k3986)} \rightarrow k = 32.9493$$

We can plug the value k into our controller transfer function model (seen below) and observe the response to see if we created a system with the desired %OS.

$$T(s) = \frac{kN(s)}{D(s) + kN(s)}$$

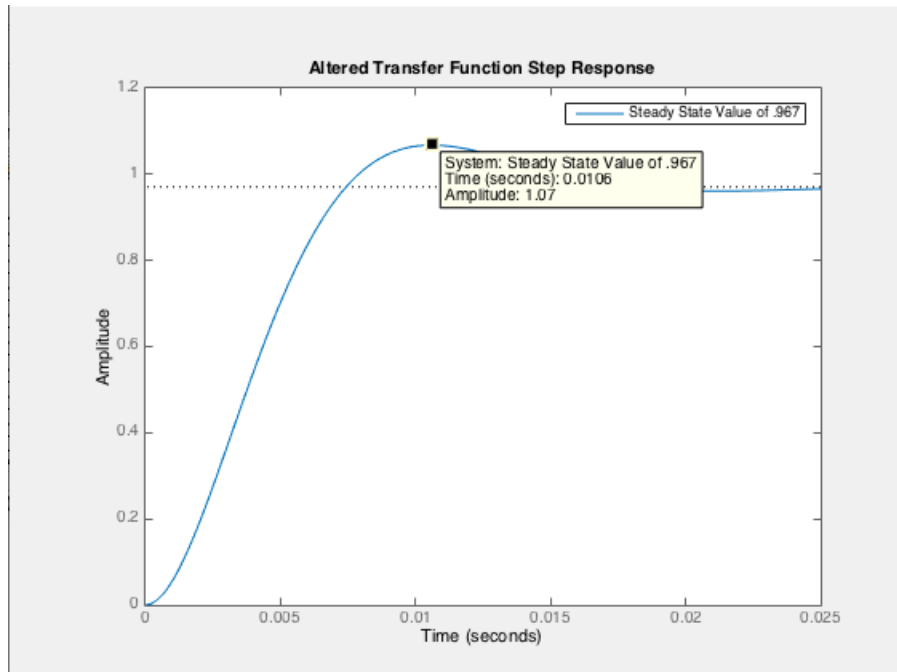


Figure 4: 10% Overshoot

In the figure above we see that the k value of 32.9493 yields a 10 percent overshoot. We are not able to achieve a 10 percent overshoot and a .05 second gain with a simple proportional controller however. This is because we can only change the value of ω_n with a proportional controller. As seen in the derivation above, in order to change ζ and ω_n simultaneously, we would need an integral gain so we could change the s coefficient and the constant of the transfer function. We also notice that there is almost no steady state error.

We are also concerned with finding the k value that makes the system become critically damped. We know that the imaginary part of our roots will determine the system type. We can perform the following analysis.

$$T(s) = \frac{kN(s)}{D(s) + kN(s)} = 1 + \frac{k(3986)}{s^2 + 434.9s + 3949 + k(3986)}$$

Roots:

$$s^2 + 434.9s + 3949 + k3986 = 0$$

$$s_{1,2} = -434.9 \pm \frac{\sqrt{(434.9)^2 - 4(1)(3949 + k3986)}}{2(1)}$$

Focusing on the imaginary portion:

$$(434.9)^2 - 4(1)(3949 + k3986) = 0 \rightarrow k = 10.87$$

Therefore we have our critical point. If our k value is less than 10.87 then our system is purely real thus we have the over damped case and if k is greater than 10.87 we have underdamped. Validating this calculation on the root locus diagram give us confidence our results are correct. We plug in the critical point and observe if our roots are repeated real.

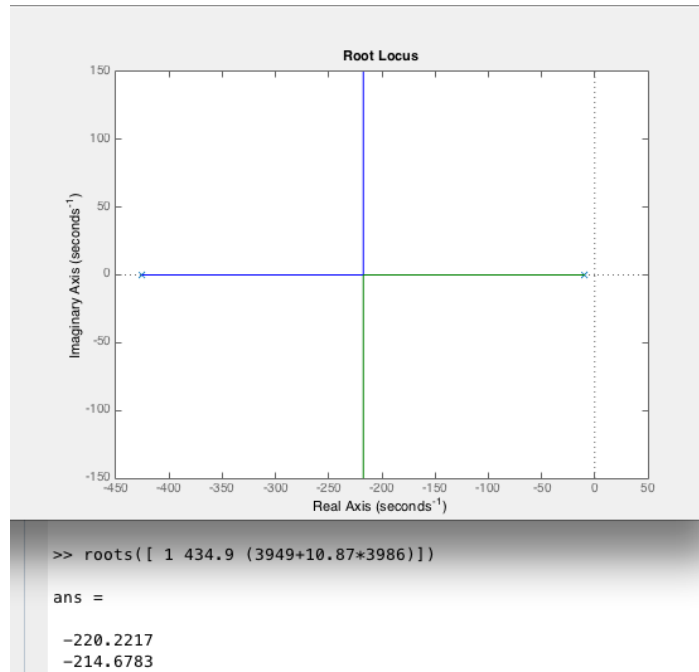


Figure 5: Critical K Check

We can see that at a k value of 10.87, the poles are extremely close to each other. There is some error simply due to rounding, but the poles agree with the root locus diagram shown above.

Third Order Analysis

Now we present the root locus diagram for the third order transfer function shown below. This transfer function was estimated from the system identification toolbox.

$$\frac{114.4}{s^3 + 9.588 s^2 + 91.31 s + 112.6}$$

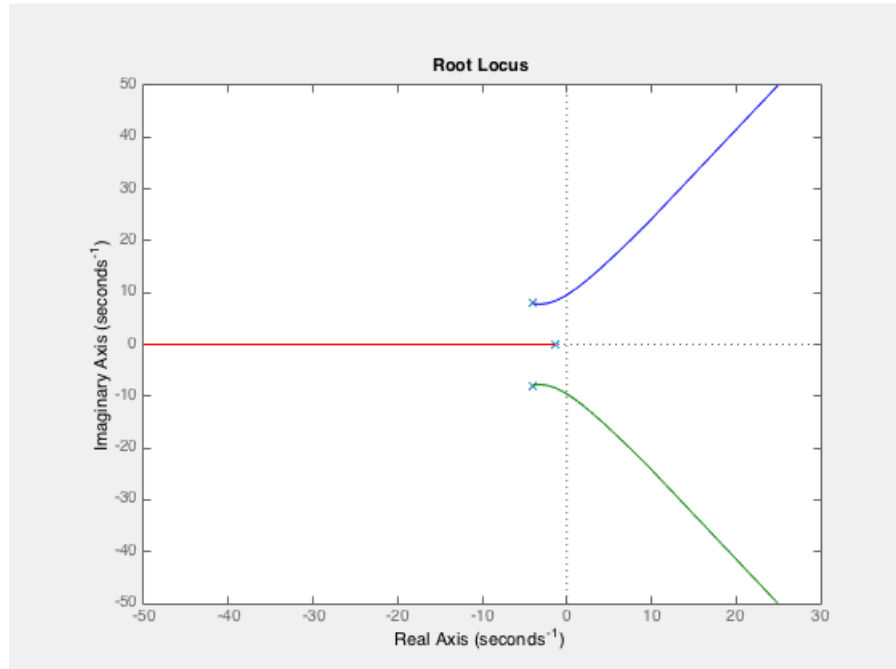


Figure 6: Root Locus 3rd Order

As we expect, there are three poles on this transfer function. We also expect this system to behave as a first order circuit because the real pole is 4 times closer to the imaginary axis than the complex conjugate poles. The step response of the third order system is shown below.

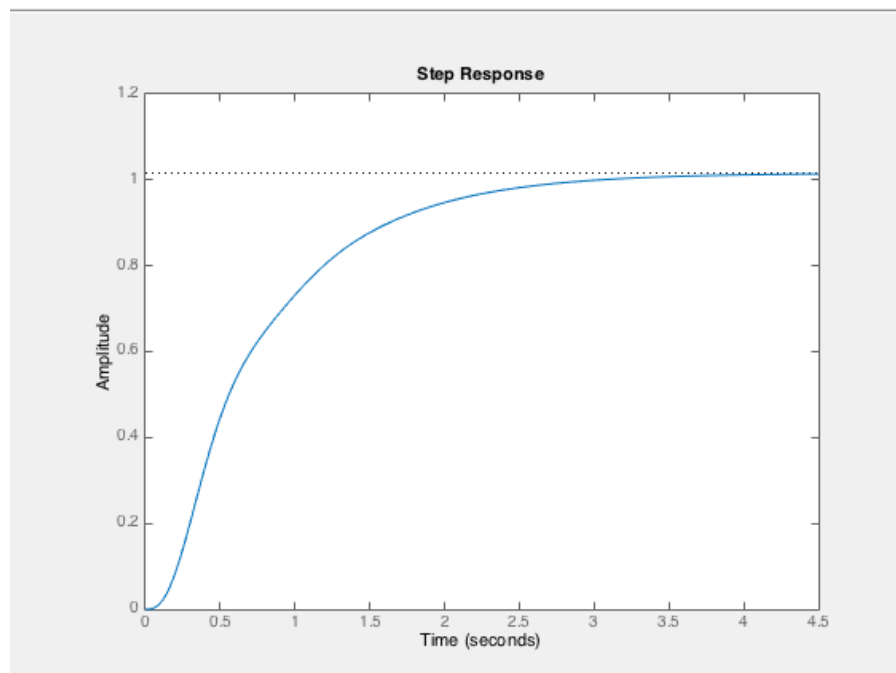


Figure 7: 3rd Order Step Response $K = 0$

As we expect, the response resembles a first order response.

We are now interested in the k value that makes the system unstable. In the Routh Hurwitz table, if we obtain a row of 0's we know that the poles lie on the $j\omega$ axis. Any increase in k from this point

will cause the system to become unstable. It is also important to note that our system will never be critically damped, because both poles will never be on the real axis. We now solve for k in the Routh Hurwitz table.

Table 1: Routh Hurwitz Table for 3rd Order Response

s^3	1	91.31	0
s^2	9.588	$112.6+114.4k$	0
s	$(875.48-112.6-114.4k)/9.588$	0	0
1	0	0	0

From the table we can see that if we solve the equation containing k for 0, we will obtain a row of 0. Therefore we get $k = 6.66$. This will now be checked on the root locus graph.

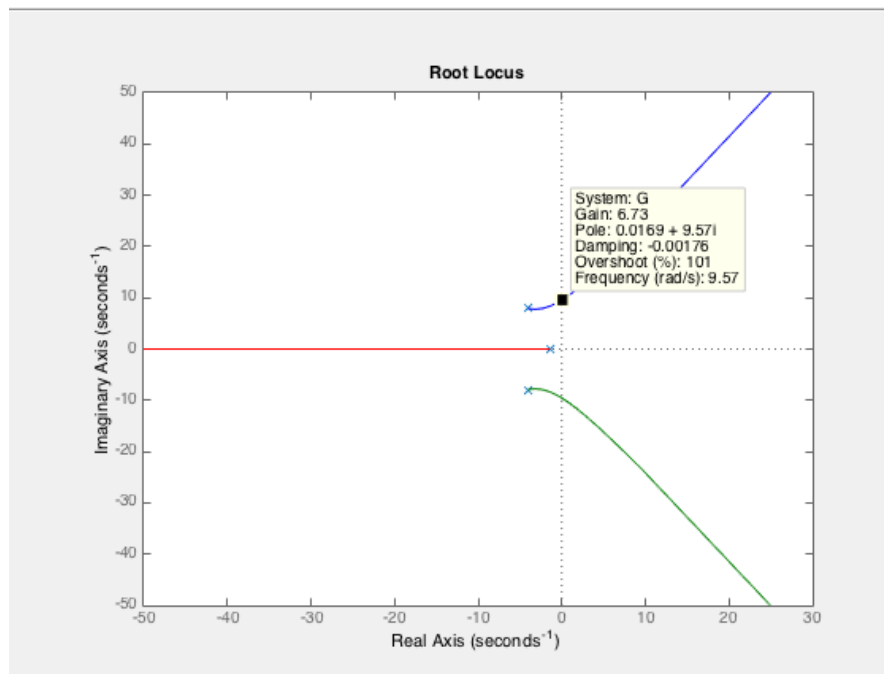


Figure 8: K Verification

Our root locus says the critical k value should be 6.73, so our calculation is very close. Again, this slight error can be attributed to rounding issues. At this k , we expect our system to be very close to unstable. The step response is shown below.

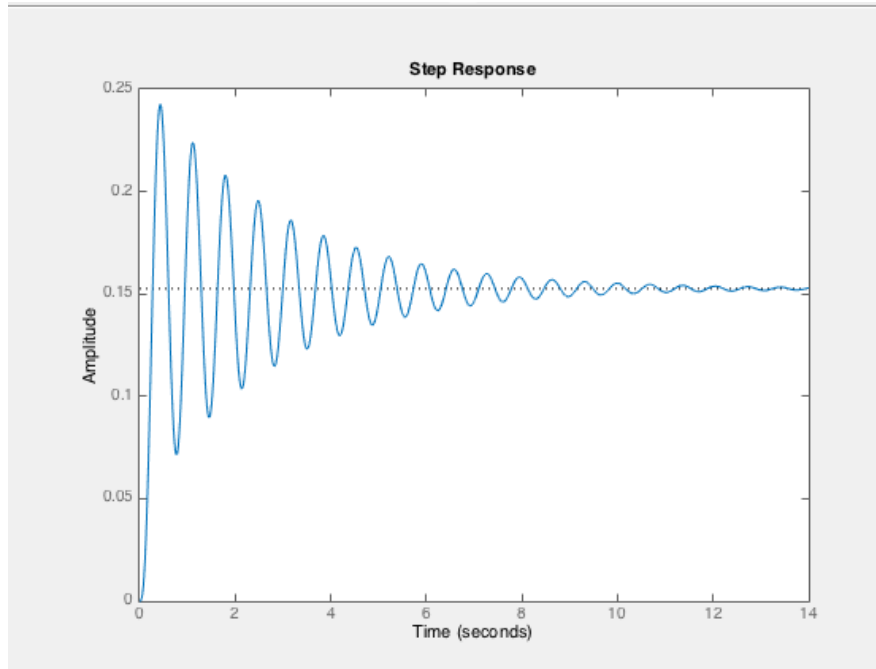


Figure 9: Step Response for Critical K Value

Through trial and error, we found the k that produces a 10% overshoot in the system response. For the same reasons as described above, with this controller, a settling time of .05s and a 10% overshoot is not realizable. When $k = 3$ however, we achieve a 10% overshoot. The step response is shown below.

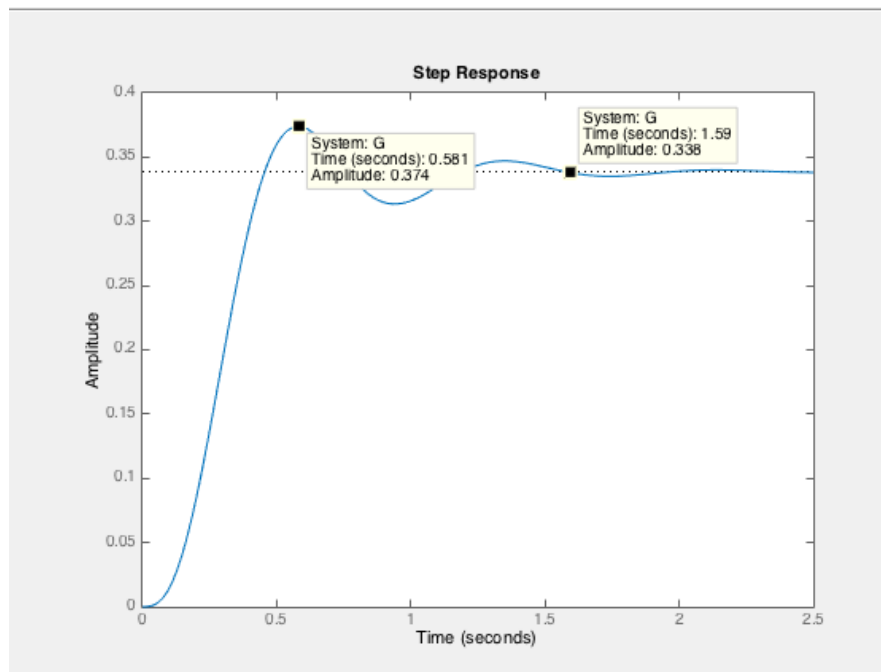


Figure 10: $K = 3$ 10% Overshoot

We see that playing with k can drastically change the system response. If k becomes too large however, our system becomes unstable. For example, if $k = 7$, which is larger than the critical gain specified earlier, the step response is unstable. This plot is shown below.

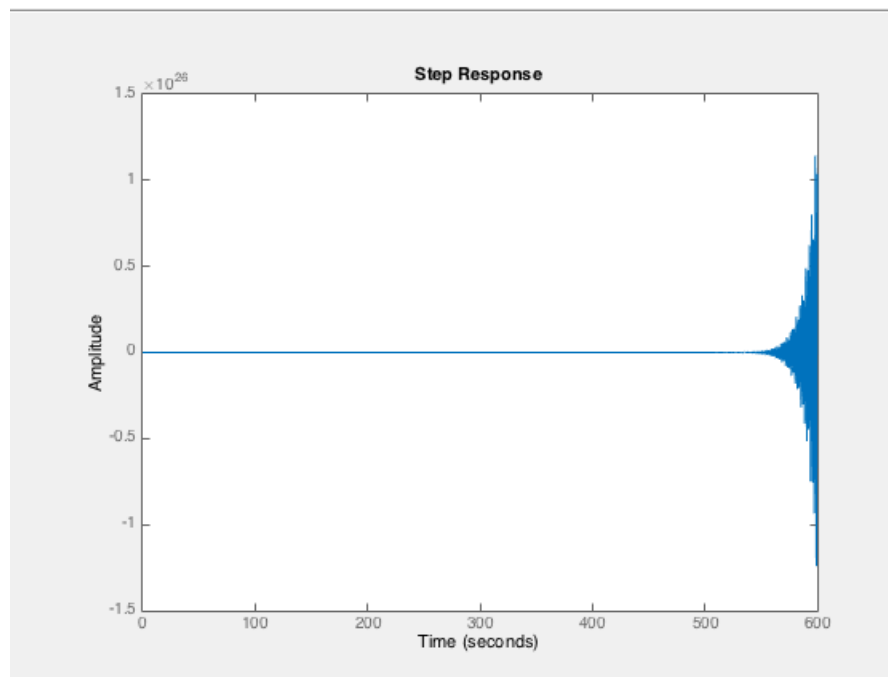


Figure 11: Unstable Step Response

We now use the root locus to determine if 150% overshoot is possible. We see in the figure below that it is, at a gain of about 13. The root locus is shown below.

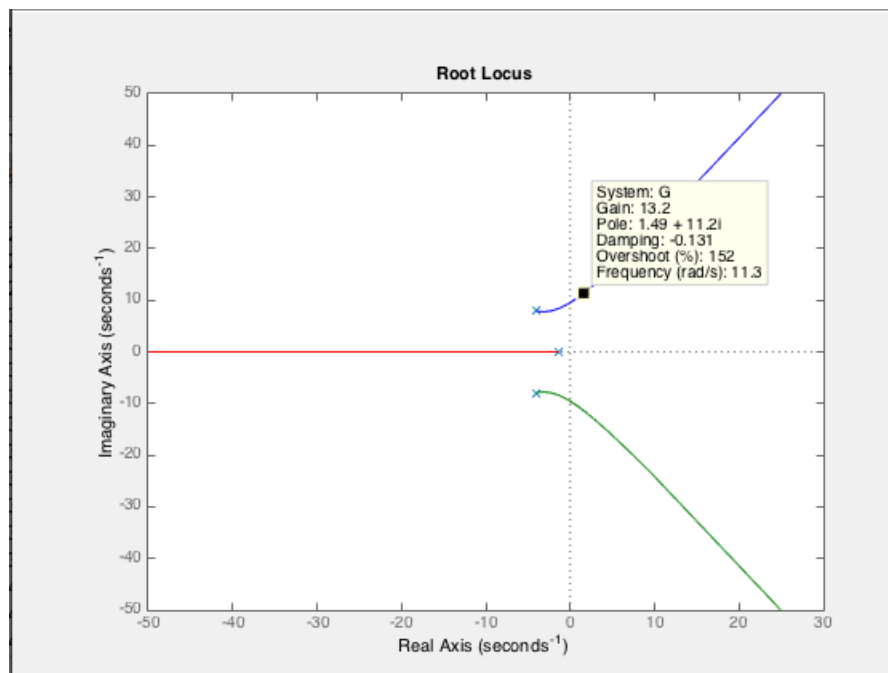


Figure 12: K for 150% Overshoot

If we compare the second order system to the third order system, we notice that the third order system is a lot more sensitive to the gain than the second order system was. This is because the poles start closer to the $j\omega$ axis in the third order response. Therefore, a smaller value of k puts the poles closer to the $j\omega$ axis, which results in a higher frequency of oscillation. Also, the complex poles quickly become dominant in the third order system, whereas the two original poles of the second order system were purely real.

We can incorporate microelectronics to create a circuit of analog components. For ease we have only used the Arduino to handle our controller process until now. First we must create the difference node in our schematic.

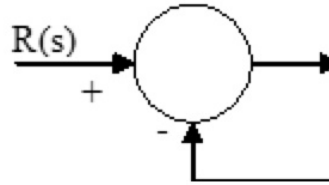


Figure 13: Difference Node

The difference node can be constructed using a difference amplifier.

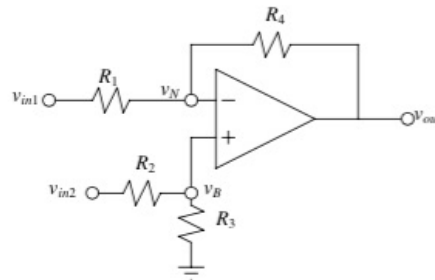


Figure 14: Difference Amplifier

The difference amplifier allows us to input two voltages and find the difference between them. The output of the amplifier is as follows:

$$V_{out} = V_{in2} - V_{in1}$$

To match up with our difference node we would set our feedback of our output $c(t)$ equal to the first input voltage and $r(t)$ equal the second input voltage. The output is the error of our output which we would feed into our PID controller. The PID controller can also be realized using analog components. The following circuit provides the basis for our proportional (P), integral (I) and derivative (D) gain.

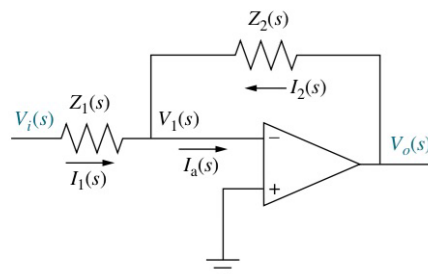


Figure 15: Op-Amp for transfer function realization

The above circuit can be changed to act as an integrator or derivator. Keeping Z_1 and Z_2 resistors allows us to construct a proportional gain. The gain value will rely on what ratio values we pick for the resistors. To construct the integrator we replace Z_2 with a capacitor. To construct the derivator we replace Z_1 with a capacitor. Again the gains for each respective controller is dependent on the values we pick for the resistor and capacitance. We can combine multiple aspects of PID controllers in the following fashion to obtain a PID controller constructed from a single Op-Amp.








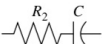
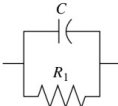

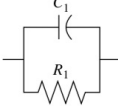

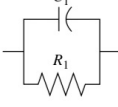
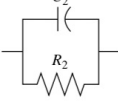
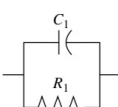
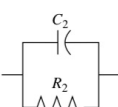
Function	$Z_1(s)$	$Z_2(s)$	$G_c(s) = -\frac{Z_2(s)}{Z_1(s)}$
Gain			$-\frac{R_2}{R_1}$
Integration			$-\frac{1}{RCs}$
Differentiation			$-RCs$
PI controller			$-\frac{R_2}{R_1} \left(s + \frac{1}{R_2 C} \right)$
PD controller			$-R_2 C \left(s + \frac{1}{R_1 C} \right)$
PID controller			$-\left[\left(\frac{R_2}{R_1} + \frac{C_1}{C_2} \right) + R_2 C_1 s + \frac{1}{R_1 C_2} \right]$
Lag compensation			$-\frac{C_1}{C_2} \left(\frac{s + \frac{1}{R_1 C_1}}{s + \frac{1}{R_2 C_2}} \right)$ where $R_2 C_2 > R_1 C_1$
Lead compensation			$-\frac{C_1}{C_2} \left(\frac{s + \frac{1}{R_1 C_1}}{s + \frac{1}{R_2 C_2}} \right)$ where $R_1 C_1 > R_2 C_2$

Figure 16: Ways to configure Op-Amp Realization Circuit

Source: (Korostelev)

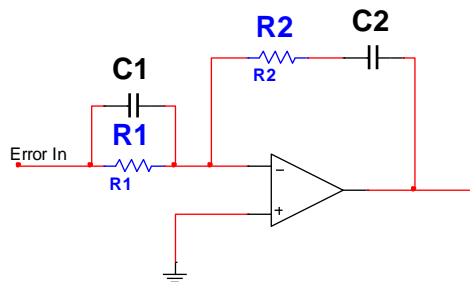


Figure 17: PID controller

If our PID controller and difference node are analog now we need a way to measure our motor speed. The motor speed the feedback network that we calculate our error from. The problem we encounter is the way we measure motor speed is by using a digital encoder. The digital encoder outputs a pulse train whose frequency corresponds to the speed of the motor. We therefore need a

circuit to extract the speed of the motor from the frequency of our pulse train. The process for converting frequency to voltage is further explained in lab 7.

Finally we have our output of our control system that needs to be fed into the motor. The output of the PID controller cannot be used directly to control the speed of the motor. We still require assistance of the Arduino board to provide power to the motor. Therefore our analog voltage output from our analog controller is fed into an analog input of our Arduino board. The motor requires additional power to run which the motor shield handles. The output of our PID controller should only be used as a reference to whether the motor should speed up/slow down and in what direction.

IV. Discussion

We have shown how to design controllers to transform a system response into something we desire. In this lab we were given raw data of the motor's system response. After analyzing the data we used the system identification toolbox to get a transfer function that describes the motor. Now we get to manipulate the proportional gain of our system. The proportional gain allows us to change the systems response. We demonstrated how to get a desired percent overshoot or settling time. In some instances it is impossible to get both the desired settling and percent overshoot. A compromise needs to be made in order to achieve the best desired response. Also we brought in Microelectronic knowledge to construct the entire PID control system in analog components. The results are less noisy since we have no problems with sampling the signal through the Arduino.

The lab is practical in demonstrating the limitation of controllers. For example, in the perfect world we would want an elevator with no overshoot, no settling time, with a comfortable rise time. The problem is that there is a compromise that must be made consistently between all your parameters.