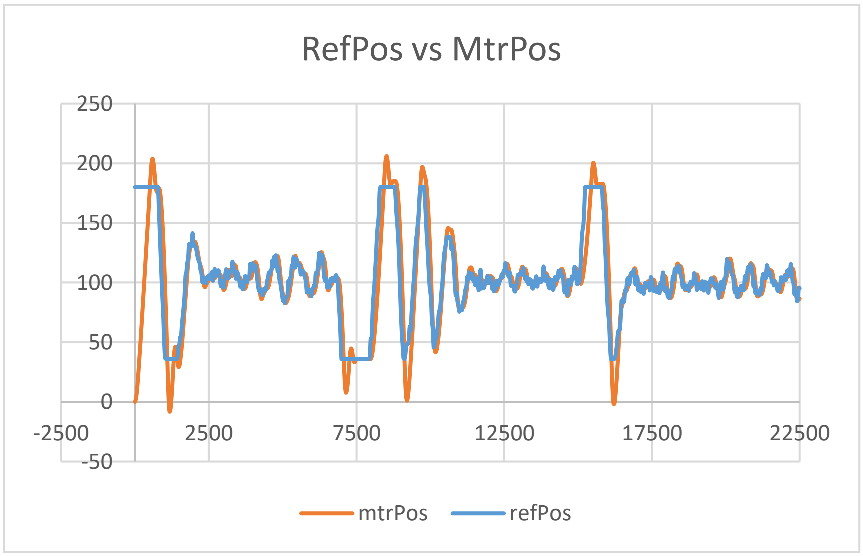
Jensen Li 5/10/2021

EEET 427.01 Professor Hochgraf

Control Systems Portfolio Project

I was able to complete my Ball on Beam Balance system and tune it through trial and error. During the process, I experimented with MatLab, reinforced my beam, and tried to learn why my system did not work at various points in the process. The final working Ball on Beam Balance system may be seen working in the embed link to my GitHub profile. However, during the process and even now, there are some parts of the system I do not understand and some function I would like to implement to improve it. I do think that I learned a lot through testing and building the whole system from no knowledge of control systems going into the class and it continues to prove to me that experiencing what I learn and getting repetition through hands-on experience is one of the ways that I learn best. I love that the whole project utilized every part of being an EMET from building the balance from foam core to wiring up the sensor, Arduino, and Maker drive to programming in C++ for the Arduino and in MatLab.

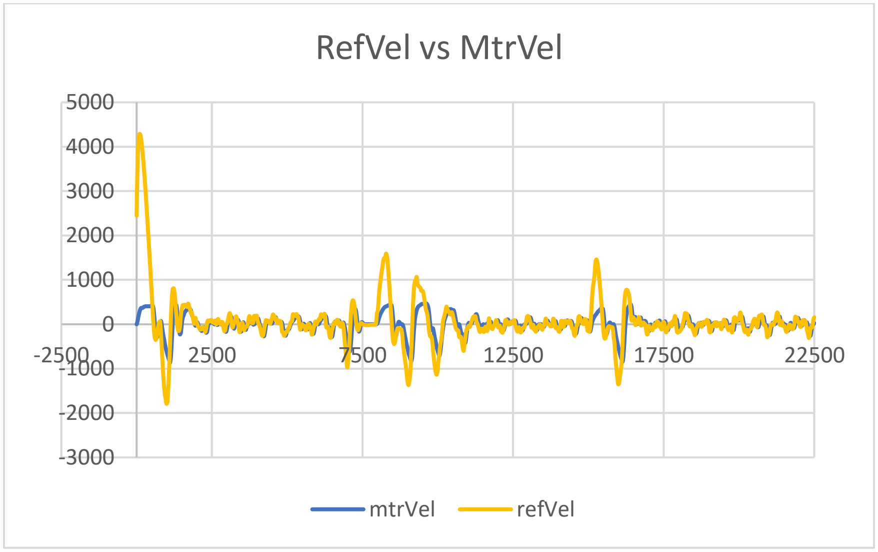
The final video showed the system getting the ball to the reference point on the beam, as indicated by the arrow in a relatively fast amount of time with almost no ringing to reach the point. However, when the ball was moved closer to the sensor at steady state, the system required a few oscillations to return to steady state. Moving the ball away from the sensor resulted in a scenario like the start of the system, where there was almost no ringing. The graph of the ball movement depicts this exact movement.



Graph 1: Position vs Time Graph of the Final Ball on Beam System

One thing I think I should look to fix is the amount of oscillation at steady state. Even in the position graph, at steady state, the ball appears to oscillate more and more over time. Otherwise, looking at the graph, I notice that the motor position and reference position have a little offset but are for the most part directly over the other. Getting this result was one of the most important parts of the trial and error to tune the system. When the motor and reference positions were offset by large degrees of margin, the system just oscillated with the ball hitting both ends of the beam.

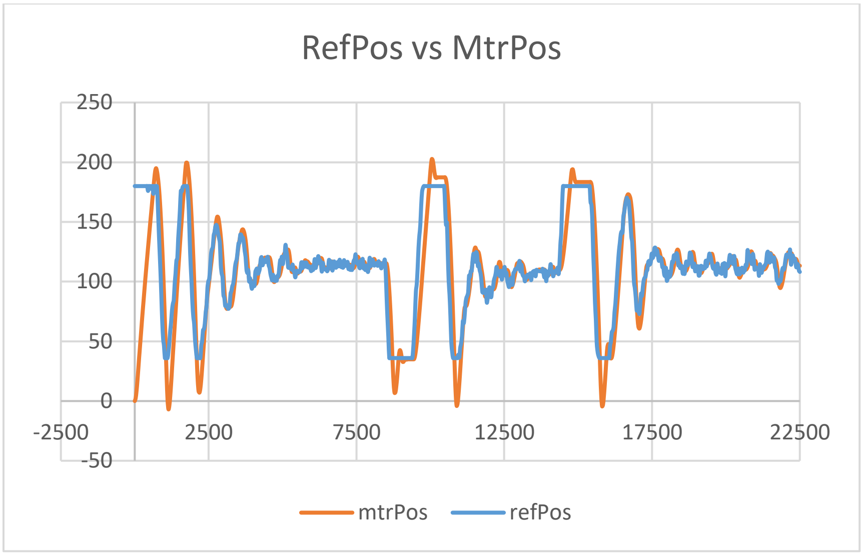
That oscillation may also have a similar reason as the final system when the ball was moved toward the sensor and oscillated a few times to reach steady state. I think the issue may be due to the motor not able to provide enough voltage for the motor velocity to reach the reference velocity the system wants, as shown below.



Graph 2: Velocity vs Time Graph of the Final Ball on Beam System

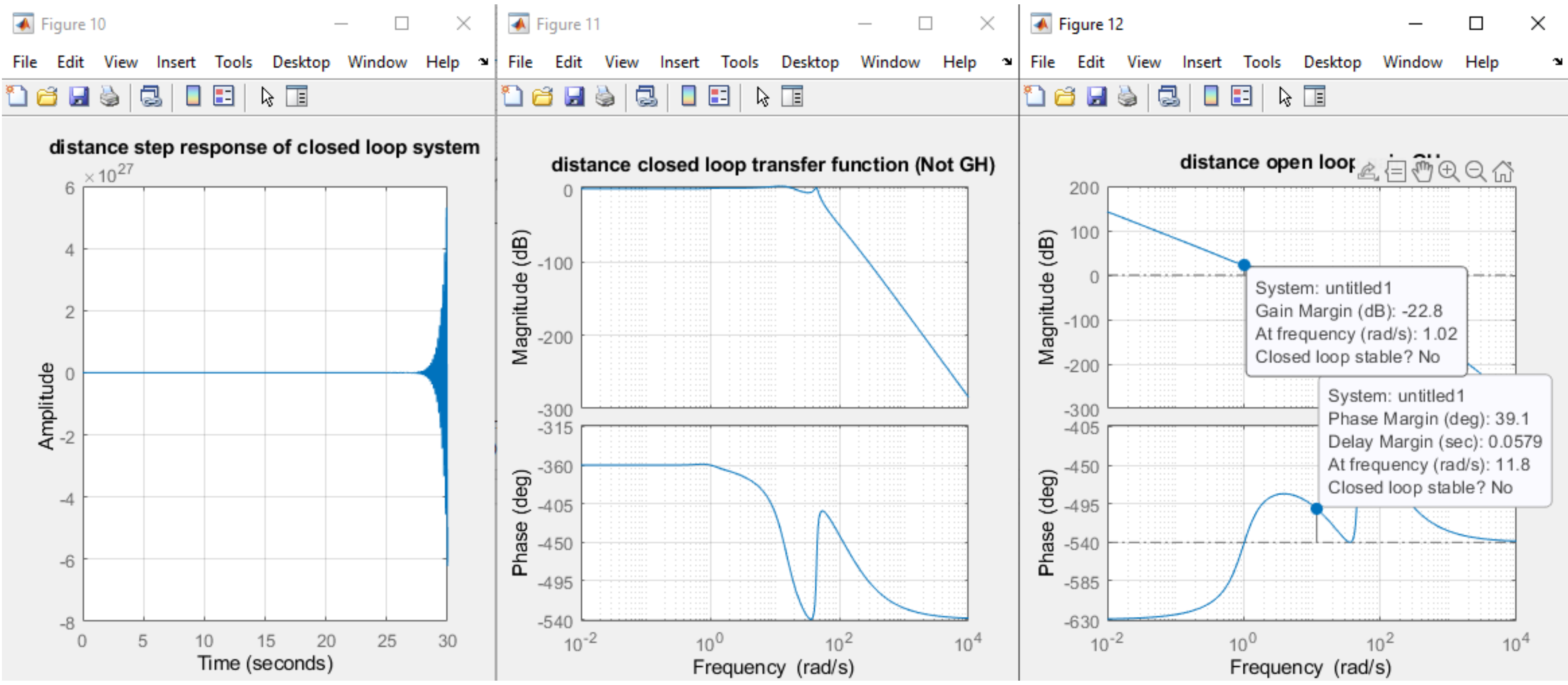
Looking at the motor velocity plotted against the reference velocity for the velocity vs. time graph, I can see that the motor velocity only reaches around 400 radians per second in either direction when the system is looking for over 1000 radians per second to get the ball to steady state. Otherwise, the motor velocity and reference velocity are basically directly over the other indicating that the system is working as intended.

However, one major thing I did notice was the issue of inconsistent motor data when testing immediately after uploading and on different days. It seemed whenever I uploaded the data, the first test run resulted in a lot more oscillation to reach steady state right after starting, as shown in Graph 3. The motor and reference position graphs also appear to have slightly more offset. However, after the initial ringing, the motor and system seem to be warmed up, so any movement to the ball from steady state or any following test runs result in graphs similar to Graph 1. I am uncertain why this does occur, but I think it may be due to the additional runs and ball changes from steady state having different feedforward and feedback values that the initial run did not have. So, for my final results, I decided to use the second run instead of the initial run from upload since those results were consistent to additional runs after it.



Graph 3: Position vs Time Graph of the Final Ball on Beam System (after upload)

After tuning the system, I wanted to check how stable it was from a more theoretical point of view. So, I tried to implement the professor's ball on beam system code in MatLab. I started by putting all my tested PID errors in with any necessary constants. The result was a big warning that my system was unstable with plenty of ringing as seen below.



System: untitled1
Gain Margin (dB): -22.8
At frequency (rad/s): 1.02
Closed loop stable? No

System: untitled1
Phase Margin (deg): 39.1
Delay Margin (sec): 0.0579
At frequency (rad/s): 11.8
Closed loop stable? No

Graph 5: MatLab graphs of Experimentally Tested Values

Since I wanted a clean copy of the code and wanted an idea of how to tune my system, I asked a classmate who had completed the MatLab configuration for the original design of a=1800 and b=6. I reviewed the class recording where the class tuned various parts of the code, such as the Kp, Ki, and Kd values for the different PIDs in the code to achieve the graphs.

So, I started changing the MatLab code to fit my system. I changed the initial first order model to my ball and beam system with a=2204.6 and b=9.09 then proceeded to tune the models for that first order system to get around a 60° phase margin with the frequency less than 40 rad/s after commenting out the other parts of the code. The results of the tuning are shown in the MatLab code and graphs below.

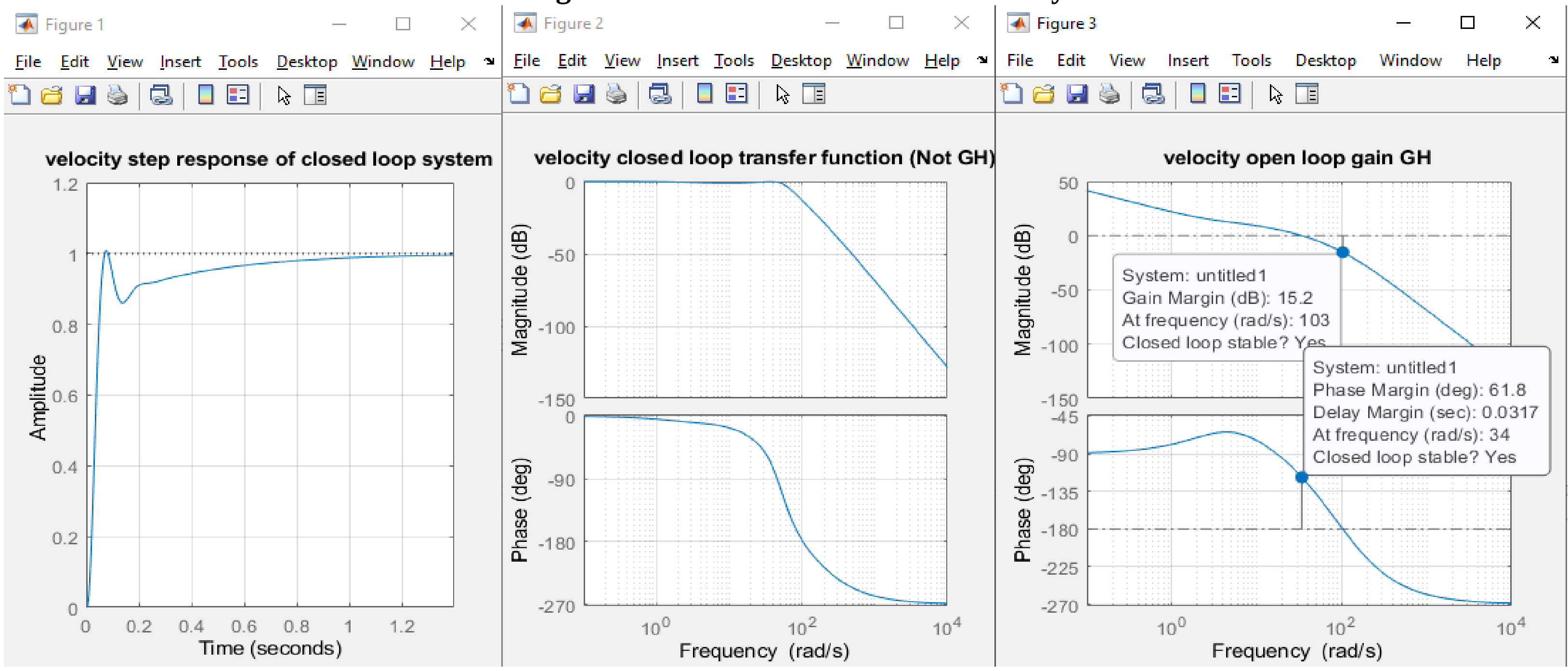
```
s=tf('s'); close all;
% start with motor as a velocity control loop
a=2204.6;
b=9.09;

GmtrVel=a/(s+b);
%Kp=0.015; Ki=0.05; Kd=0.0001; Derivative_cutoff_freq_rps=40;
Kp=0.017; Ki=0.05; Kd=0.0001; Derivative_cutoff_freq_rps=20;
GcVelPID=(Kp+(Ki/s)+(Kd*s*Derivative_cutoff_freq_rps/(s+Derivative_cutoff_freq_rps)));

% set sample interval to use in modeling the digital delay
TSAMP_MSEC=30;Td=TSAMP_MSEC/1000; % digital controller delay
Gdigital=((2/Td)/(s+(2/Td)))*((4/Td)/(s+(4/Td)));

% model effect of digital delay
GpVelCL=feedback(GcVelPID*Gdigital*GmtrVel,1);
figure(1); step(GpVelCL); grid on; title("velocity step response of closed loop system"); hold on;
figure(2); bode(GpVelCL); grid on; title("velocity closed loop transfer function (Not GH)"); hold on;
figure(3); bode(GcVelPID*Gdigital*GmtrVel); grid on; title("velocity open loop gain GH"); hold on;
[Gm,Pm,Wcg,Wcp] = margin(GcVelPID*Gdigital*GmtrVel);
Pm
%stop
```

Figure 1: MatLab code of First Order System



System: untitled1
Gain Margin (dB): 15.2
At frequency (rad/s): 103
Closed loop stable? Yes

System: untitled1
Phase Margin (deg): 61.8
Delay Margin (sec): 0.0317
At frequency (rad/s): 34
Closed loop stable? Yes

Graph 4: MatLab graphs of First Order System

After that, I continued to the next parts with tuning the PIDs. I started with the position PID controller, which was another set of trial and error with the graphs shown below. I learned that adjusting the Kp could be done with the log formulas.

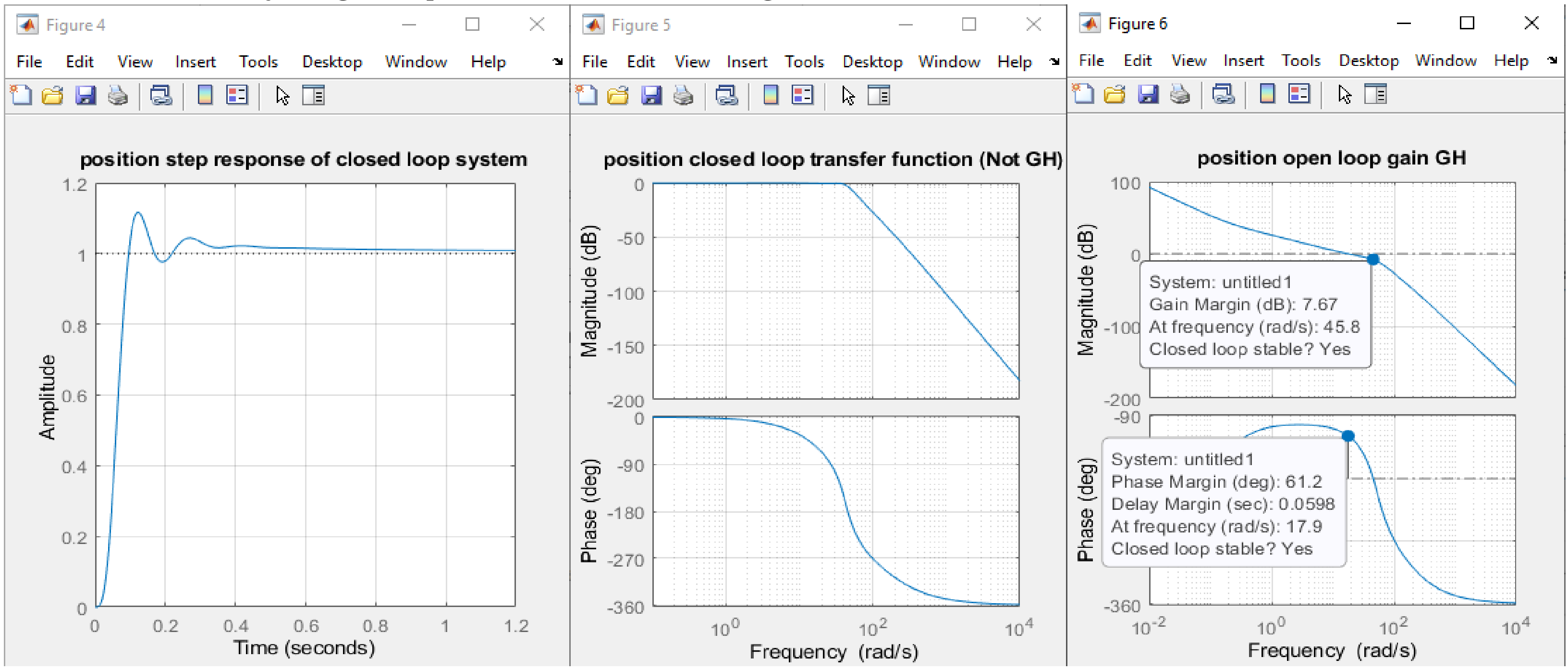


Figure 4
File Edit View Insert Tools Desktop Window Help
position step response of closed loop system
Amplitude
Time (seconds)
Figure 5
position closed loop transfer function (Not GH)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
Figure 6
position open loop gain GH
Magnitude (dB)
Phase (deg)
Frequency (rad/s)

System: untitled1
Gain Margin (dB): 7.67
At frequency (rad/s): 45.8
Closed loop stable? Yes

System: untitled1
Phase Margin (deg): 61.2
Delay Margin (sec): 0.0598
At frequency (rad/s): 17.9
Closed loop stable? Yes

Graph 5: MatLab graphs of Position PID

Then, I moved onto the Pole-Zero Compensator. I reset the values of C and D back to one so I could test and learn the process myself. I understood that the pole-zero compensator would create a phase bump which would increase the phase margin. The point was to get the phase margin to 60° again. I did not figure out how to get the 0db gain to be at the peak of the phase bump, so I continued since I did manage to get the phase margin to around 61.2°.

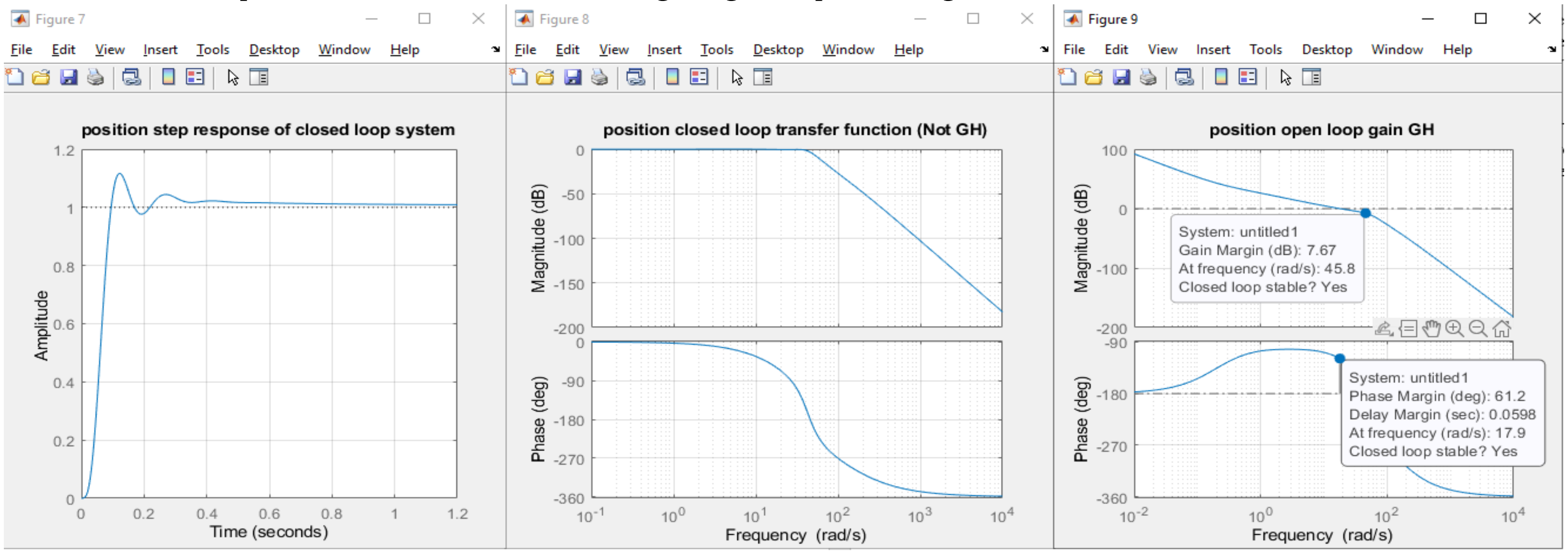


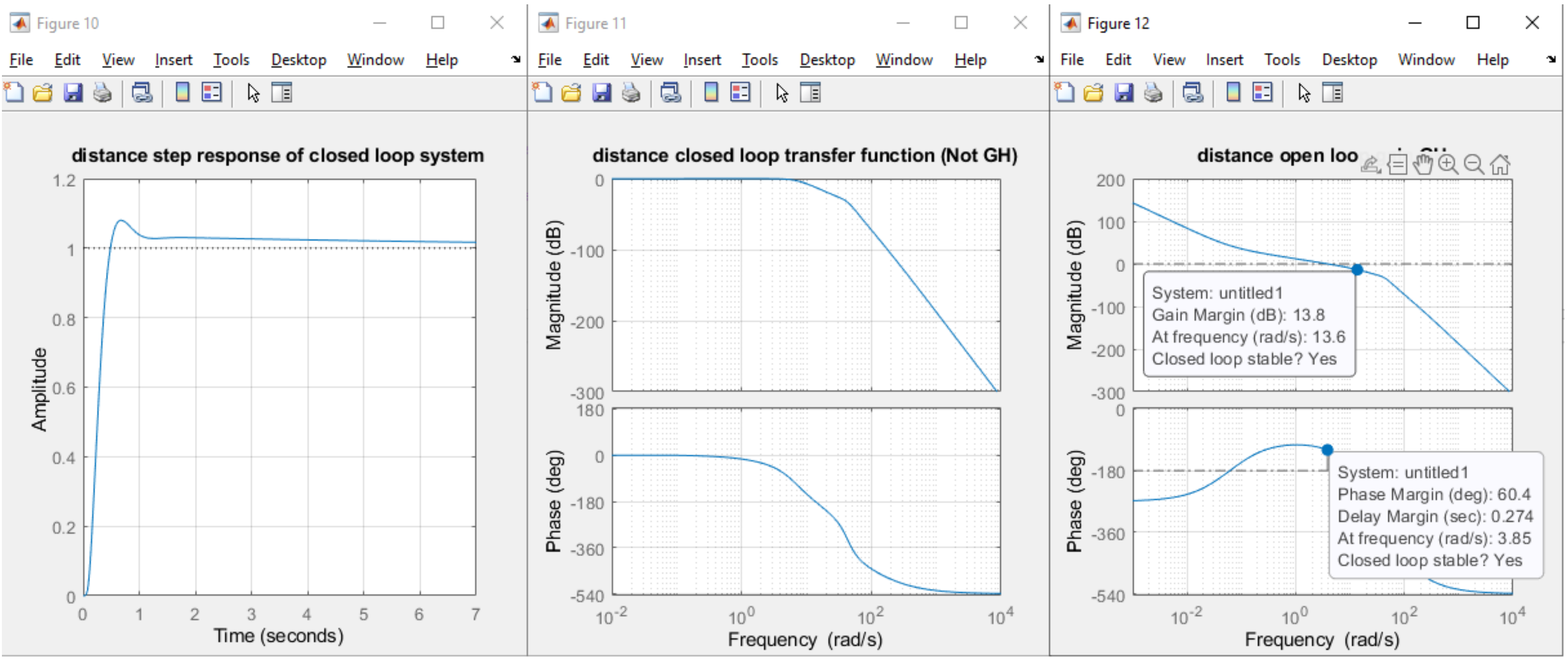
Figure 7
File Edit View Insert Tools Desktop Window Help
position step response of closed loop system
Amplitude
Time (seconds)
Figure 8
position closed loop transfer function (Not GH)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
Figure 9
position open loop gain GH
Magnitude (dB)
Phase (deg)
Frequency (rad/s)

System: untitled1
Gain Margin (dB): 7.67
At frequency (rad/s): 45.8
Closed loop stable? Yes

System: untitled1
Phase Margin (deg): 61.2
Delay Margin (sec): 0.0598
At frequency (rad/s): 17.9
Closed loop stable? Yes

Graph 6: MatLab graphs of Pole-Zero Compensator

After that, I finished tuning the remainder of the PIDs through trial and error to get the following final graphs for my specific system.



Graph 7: MatLab graphs of Distance Controller

I thought having completed the tuning in MatLab would allow me to transfer over everything to the Arduino code and see the balance beam work better than what I got through trial and error. Since I had programming background and understood how classes worked from previous experience coding in C++ and Java, I added a pole zero compensator from the PID class to the Distance PID and change all my PID values to reflect my MatLab findings. Turn out that the Ball and Beam Balance system did not work anymore. The system would not try to get the ball to balance, and no changes would happen even when the ball was moved by hand as shown in Graph 8. I think I was able to apply what I learned about Control Systems through the MatLab code, but I need to change the Arduino code to fully simulate each step in the MatLab process, or the actual transfer function in Figure 2.

So, although my balance system works after experimental tesing, I want to redo the Arduino code to correctly test the MatLab simulation findings and see how differently the system responds. I hope that the system will be able to oscillate less and reach steady state faster when the ball is move away from the balance point after implementing the MatLab findings.

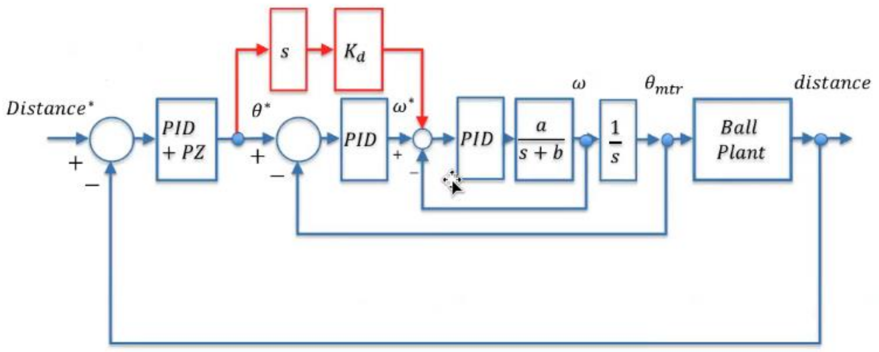
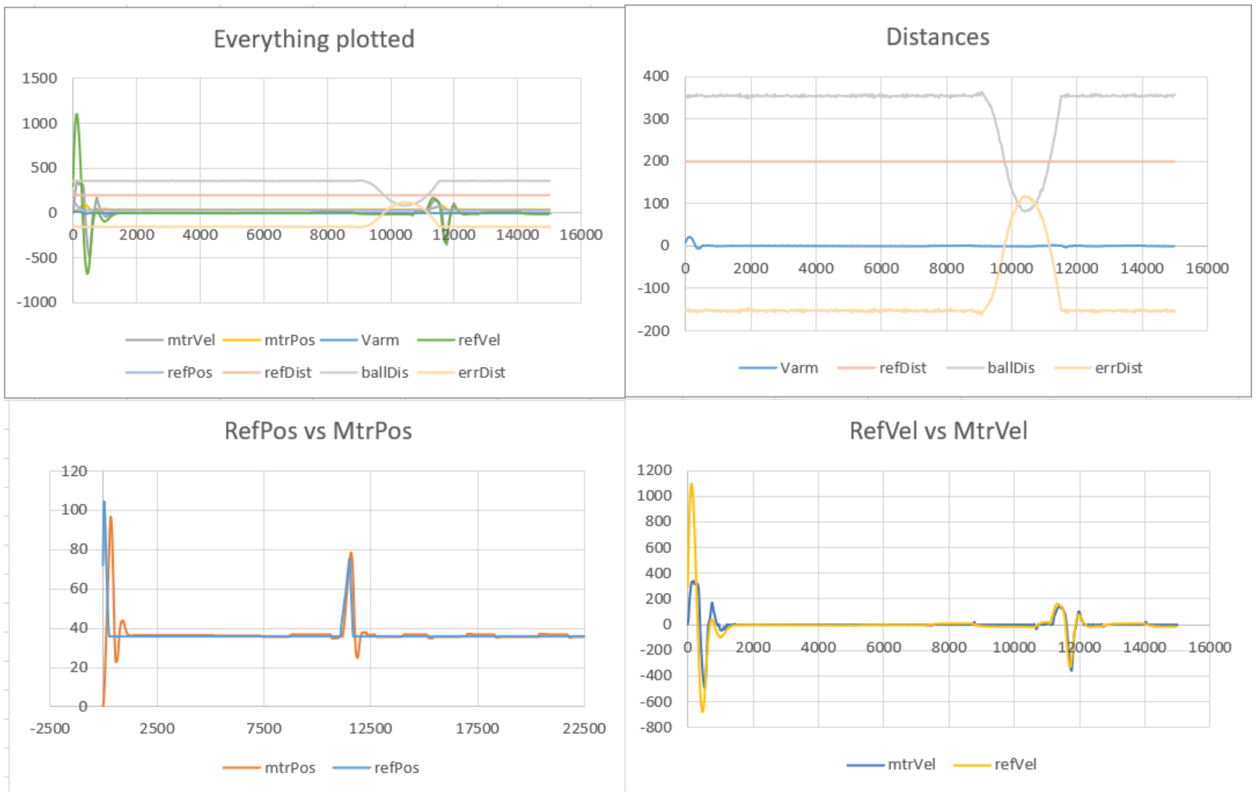


Figure 1: Transfer Function used in MatLab Simulation



Graph 8: Graphs of attempting to implement MatLab findings