SE420 Laboratory Assignment 8
PID Control of the Height of the End of a Cantilever Beam

Goals for this Lab Assignment:

1. Use what you learned from Lab 6 and Lab 7 to design a PI plus velocity feedback controller for the Cantilever Beam Experiment.
2. Use Root Locus techniques to design the PI plus velocity feedback controller.
3. Implement the PI plus velocity feedback controller for the real system.
4. Add a slow trajectory for the magnetic levitation experiment to follow instead of a step input.

Matlab Functions Used:

c2d, controlSystemDesigner, LTI System Simulink block.

Prelab:

Read entire lab assignment so you are prepared when you come to your lab session.

First Step:

Before you and your partner start the control design, take some time to get your lab 4 code running again. If you remember we saved a copy of your lab 4 code that we called, Lab8starter, or something like that. This code over sampled the ADCB input channels 16 times every 1 millisecond. Go back to that code and make changes to it so that it sends ADCB2's voltage value to Simulink for plotting. Then run this code and plot ADCB2's voltage value in Simulink. This voltage value is proportional to the beam's end height and will be used as our feedback signal for the remainder of this lab. If you flick the beam use should see its position oscillating in your Simulink plot.

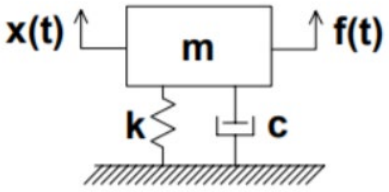
Control Design and Simulation:

For this lab I am not going to give as much explanation as I have in previous labs. What you learned especially in Lab's 6 and 7 should give you all the tools you need to design this controller for the cantilever beam experiment.

If you remember way back in Lab 1 you identified the continuous transfer function of the cantilever beam. You found values for its transfer function: Gain, ω_n and ζ.

Recall from Lab 1:

The MagLev (cantilever beam) experiment at your bench is a thin brass beam with two magnets at the end. Forty coils of printed circuit board traces are located below the magnets that produce a proportional force on the magnets when driven with current. Analog Hall Effect sensors are placed to the side of the magnets that sense the position of the magnets and output a proportional voltage. The below figure shows a free body diagram of the approximate model of the system. Your input (EPWM8A) to the MagLev beam is the force (f) in the figure. You will connect the feedback signal to the Oscilloscope for real-time viewing and this signal is also connected to analog to digital converter ADCB2 of your F28377s system. In this lab you will be using this feedback and force to control the position of the end of the beam.



The approximate equation of motion for this mass spring system is

$$m\ddot{x} + c\dot{x} + kx = f(t).$$

Which has the transfer function

$$\frac{X(s)}{F(s)} = \frac{1}{k} * \frac{{}^{k}/_{m}}{s^2 + {}^{c}/_{m}\, s + {}^{k}/_{m}}$$

We are not as interested in the values of k, m, c but instead the full transfer function. Thus, you can think of this transfer function as the standard second order transfer function

$$\frac{X(s)}{F(s)} = Gain * \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad poles = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2}, with\ \omega_d = \omega_n\sqrt{1-\zeta^2}$$

With this transfer we only had to identify the Gain, ω_n and ζ.

Using your identified values for Gain, ω_n and ζ, first enter the continuous transfer function of this system in Matlab. Then using the sample rate of 0.001s, find the ZOH discrete transfer function of this system.

Recalling the steps you performed in Lab 7 use controlSystemDesigner and its root locus plots to design a PI plus velocity feedback controller that controls the end of the beam with the following specifications:

Design Specifications to be met for a step response

1. Use 500s/(s+500) as derivative for velocity feedback.
2. Overshoot < 2%
3. Settling time < 150ms
4. 0% steady-state error

Hints: For PI controller in your root locus design use the single transfer function $K_{Loop} * \frac{z-zero_c}{z-1}$ that you derived in Lab 7. After you design your controller in controlSystemDesigner, solve for the values of Ki and Kp given K_{Loop} and $zero_c$. Also find your value for Kd in the velocity feedback. Then create a Simulink simulation of your discrete controller and discrete transfer function. Make sure to saturate u between -10 to 10. This Simulink simulation should use all discrete TFs. Your control effort should be u = Kp*e + Ki*Ik – Kd*vel. Do not use the transfer function $K_{Loop} * \frac{z-zero_c}{z-1}$ for the PI portion. Instead separate out Kp*e and Ki*Ik so in the real system you can watch for and fix integral windup.

Demonstrate your controller and Simulink simulation to your TA.

With the controller designed, implement the controller on the real system. Remember that the coil gets very hot if you leave the system on too long. So make sure to only turn on the coil's amplifier when you are controlling the system.

1. Power HBridge/Coil with Programmable Power Supply, 9Volts, 2Amps.
2. Overshoot < 2%
3. Settling time < 150ms
4. 0% steady-state error
5. Minimal oscillations at steady-state.

6. The reference input is a step from 1V to 2V and repeats back and forth every 4 seconds.
7. Implement actual controller u = Kp*e + Ki*Ik – Kd*vel. Do not let the integral windup by stopping the integral when u is saturated.

Demo you working controller to your TA.

Following a trajectory

Create and implement a trajectory that makes the magnets move from one height to another height in approximately 0.5 to 0.75 seconds, so much slower. For this trajectory, I would like you to implement a filter that filters your step input to achieve a slow transient response. For example, you could filter your step input with the discretized transfer function of the continuous filter $\frac{Out(s)}{In(s)} = \frac{20^6}{(s+20)^6}$ to get a nice slow response like the figure below. If you use this filter (and probably other slow filters) you will need to use "long double" variables to implement its discrete filter due to numerical precision and numerical errors. Test if your discrete filter will work with floats or needs to use doubles in Simulink. On the TMS320F28377S processor a 64 bit floating point number is called a "long double." The below Matlab commands will help you check if your filter works with 32bit floats or needs to use 64bit long doubles.

```
doublenum = filterforStep.num{1};
doubleden = filterforStep.den{1};
singlenum = single(doublenum);
singleden = single(doubleden);
singleFilterD = tf(singlenum,singleden,0.001)
```

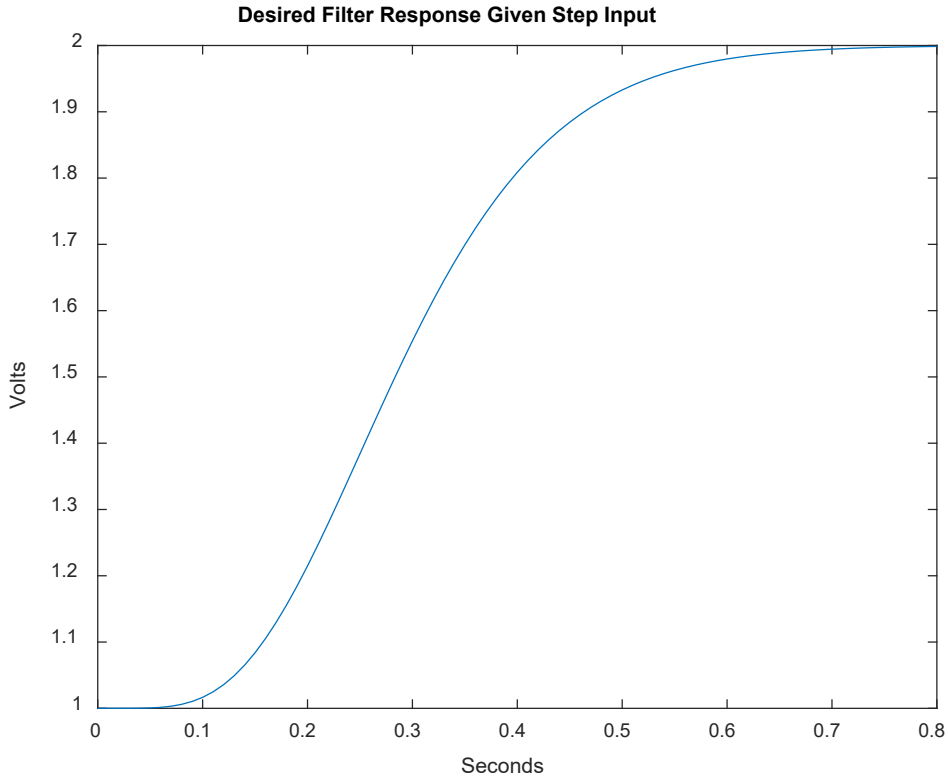
Then when exporting your filter coefficients to your C code, perform the following in Matlab:

```
format long e
arraytoCformat_double(filterforStep.num{1}')
arraytoCformat_double(filterforStep.den{1}')
```

Copy the output of the filter numerator and denominator to your C code and then make sure to make the type of the arrays either float or long double which ever you found to work.

Now in your C code filter your step input with the discrete filter and use the output of this filter as your reference input to your controller.

Demo your working system following the slow trajectory to your TA.



Lab Check Off:

1. Demo your final root locus design.
2. Demo your Simulink controller
3. Demo your implemented controller performing a step response of the cantilever beam experiment.
4. Demo your implemented controller performing the slow trajectory of the cantilever beam experiment.
5. Submit a report for this lab with the items listed below.

Report

For this lab you will need to put together a report on the work that you did. **This is an individual report**. Partners can of course use the same data and control implementation but the report needs to be original work. This report will have:

1. The steps you used in Matlab / controlSystemDesigner to design your controller.
2. Any issues you had with the system, non-linearities, noise, numerical computation issues.
3. If you hand tuned parts of the controller, how you chose the final gains.
4. Must include plots of final controlled responses for both step input and slow trajectory input both simulation and actual system.
5. Other relevant plots that help describe your control design and results.
6. Root-locus plots that show how you designed your controller. Make sure your controller's poles, zeros, gains are shown in the plots. If hard to see, show some zoomed plots.
7. How well did simulation match the actual controlled system responses?
8. Screen shot of your Simulink block diagram. Make sure all details of the block diagram are easily seen.
9. Your commented C-code that implemented the controller.
10. Discuss implementing the trajectory.