GE420 Laboratory Assignment 9

Discrete Controller Design and Notch Filters

Goals for this Lab Assignment:

1. Introduce discrete notch filter design for reducing the effect of oscillatory plant poles.
2. Design and implement the notch filter along with an additional controller (P, PD/lead, PID, ??) to achieve the given design specifications.

SYS/BIOS Objects Used:

- Any of your choosing

Matlab Functions Used:

c2d, rltool

Prelab:

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

Control Design and Simulation

1. Show that the continuous PD controller $\frac{u(s)}{e(s)} = Kp + Kd * s$,(Kp=1, Kd=1) when emulated in the discrete domain, is simply a lead controller. Approximate s to be $s \approx \frac{20s}{s+20}$ and discretize the controller using the Tustin rule $s = \frac{2}{T}\frac{z-1}{z+1}$ with T equal to 0.05. Create a bode plot of the discrete controller transfer function and verify that phase lead is being added.

   Also remember from Lab 8 that many times when designing PD controllers you do not necessarily want to take the derivative of the error signal. This is especially true when you are applying step inputs to the system, because the derivative of the step is infinite. Instead the PD control is implemented by the equation $u_k = K_p e_k - K_d v_k$ where v_k is the derivative of the feedback signal. You should also remember from Lab 8 that both of these types of PD controllers produce the same closed loop poles. So for that reason you can perform a root locus design with the standard error path PD controller to get some intuition on what a PD control can do for you and then implement the same gains in the velocity feedback PD controller. The velocity feedback PD controller will normally have less overshoot because the derivative of the step has been removed.
2. Complete the following tutorial/example on notch filter design. Hand in a plot of the final response.

Notch Filter Design Pole/zero cancelation

Oscillatory poles can be difficult to deal with in control system design. The best method to deal with these unwanted poles is to redesign the physical system to add more damping. But many times this is not possible. One possible way to reduce the effects of these oscillations is to add a notch filter to your control design.

A notch filter has the discrete transfer function: $Nf(z) = K_f \frac{z^2 + 2\zeta_1\omega_1 z + \omega_1^2}{z^2 + 2\zeta_2\omega_2 z + \omega_2^2}$. The goal of notch filter design is to use the zeros of the notch filter transfer function to reduce the effect of unwanted plant poles. i.e. Place the

zeros very close to the problem poles so they are in a sense canceled. The placement of the poles of the notch filter transfer function is a bit more arbitrary and design specific. Normally they are fast poles placed a good distance away from the closed loop system poles that define the desired dynamics of the system output.

The best way to understand a notch filter design is by an example:

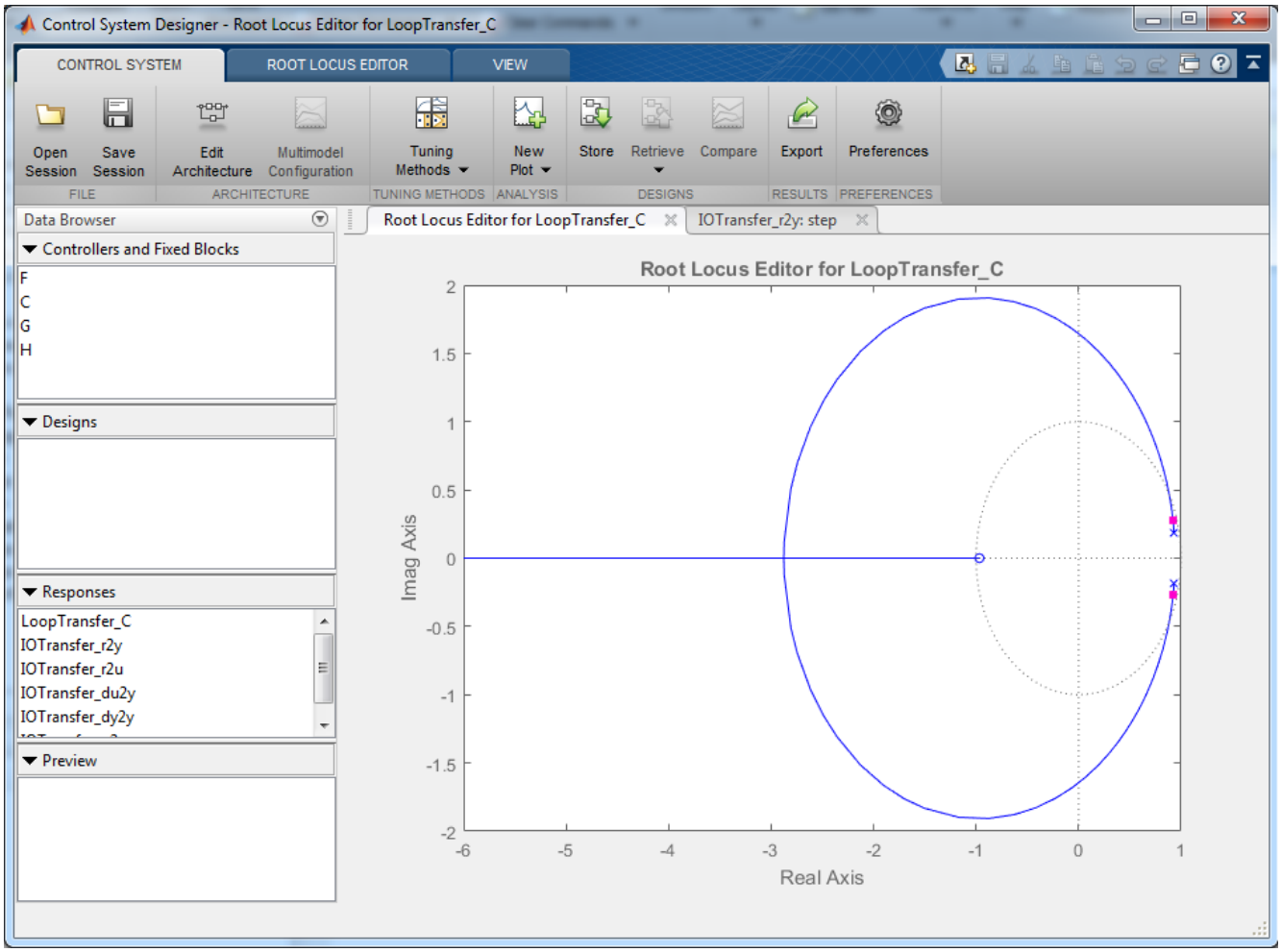
Say the system to be controlled (plant) has the continuous transfer function $G(s) = \frac{17}{s^2 + 2s + 17}$.

In Matlab find the Zero Order Hold mapping of this transfer function into the z-plane using the sample period of 50ms.

sysD = c2d(tf(17,[1 2 17]),.05)

Run rltool to design the notch filter.

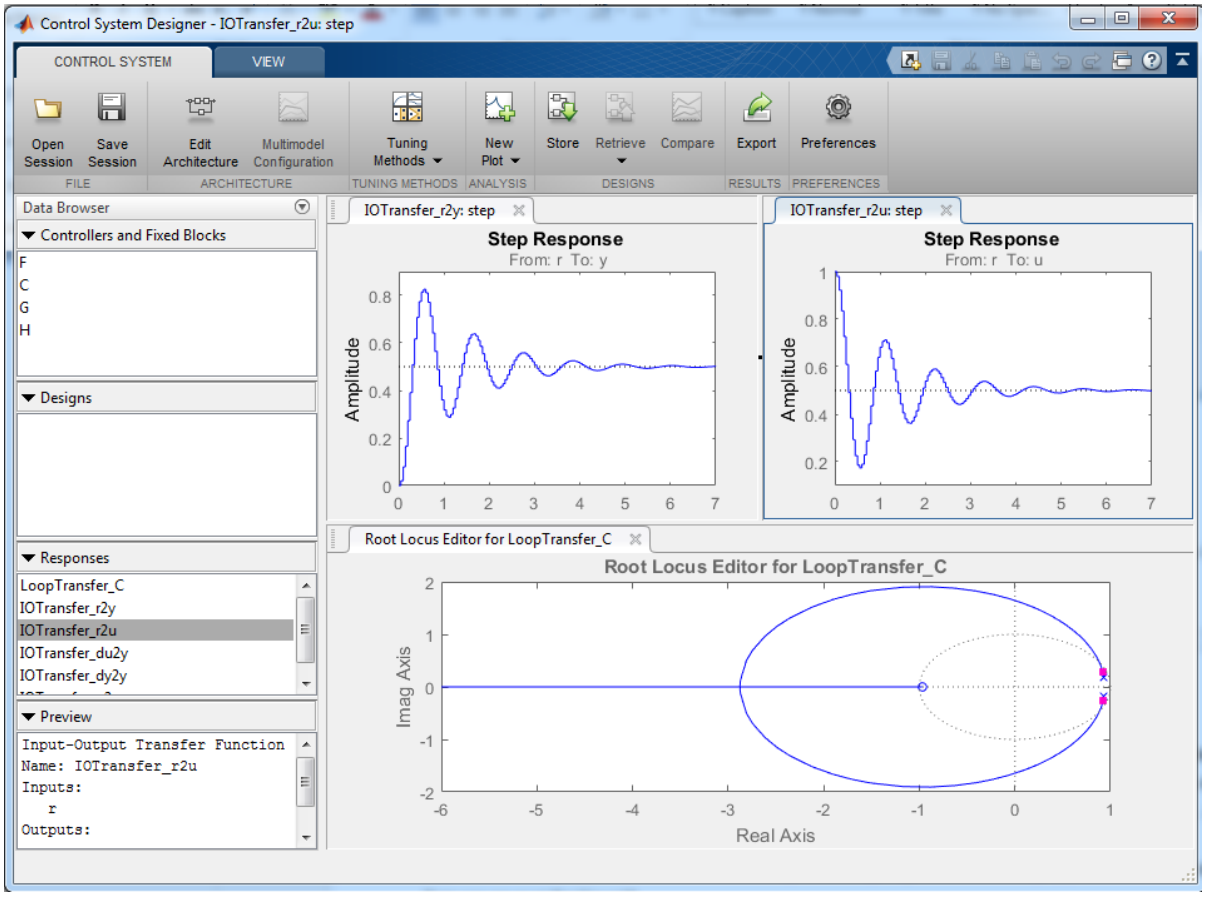
Rltool(sysD)



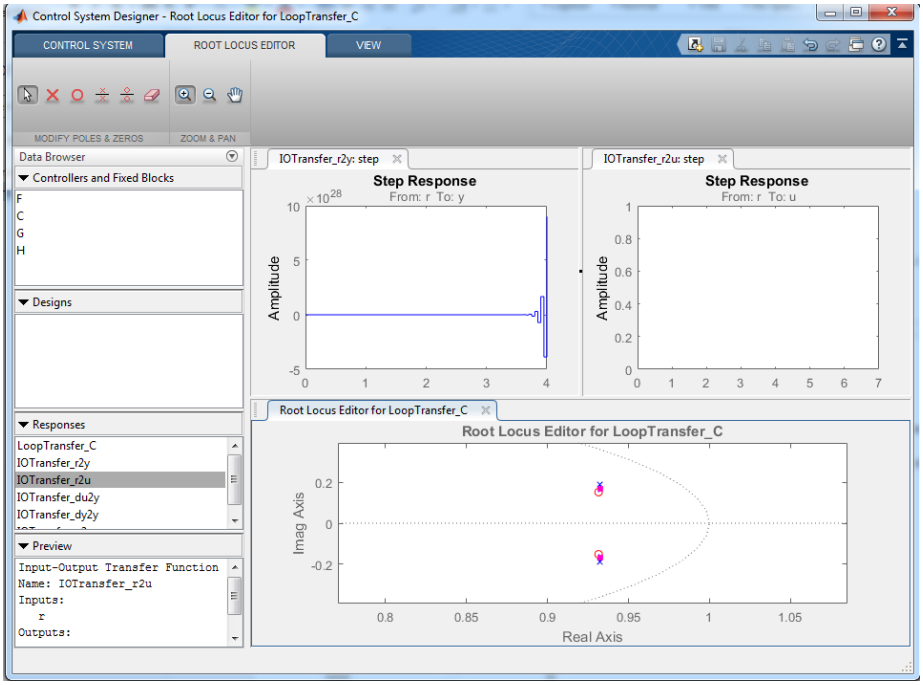
By default rltool displays the Y output Analysis Plot (IOTransfer_r2y) step response. Also setup rltool to produce the control effort (u) step response (IOTransfer_r2u) of the step response so you can monitor the control effort during your design. Arrange the three plots so that you can see each of them. See below.

Notice the oscillatory response of the system.

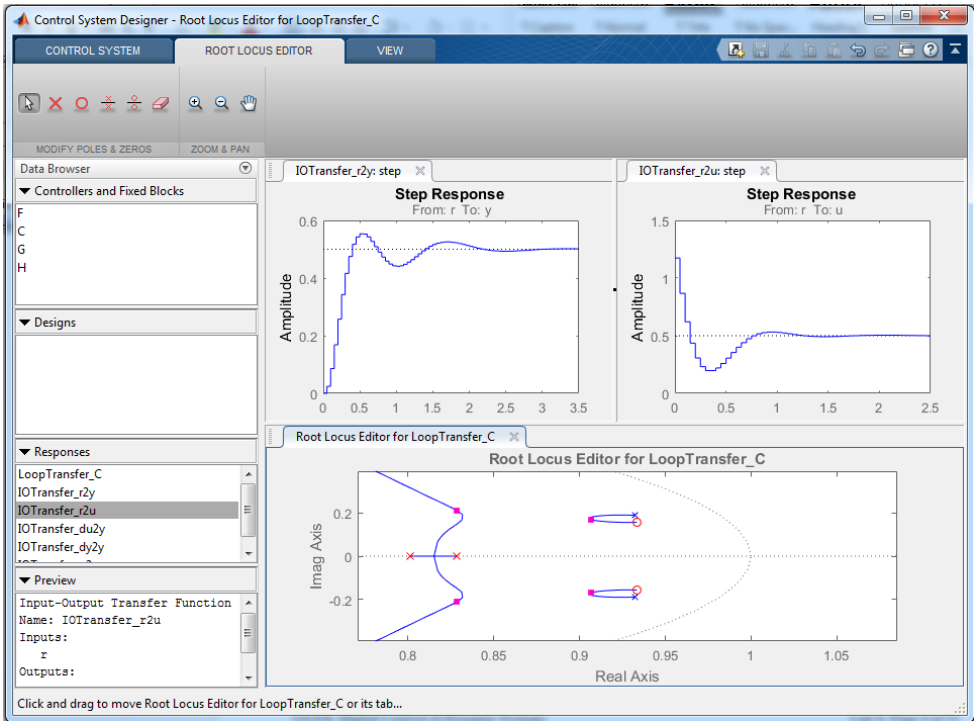
Also here in the "Preferences" gear icon, select the "Options" tab and under "Compensator Format" select "Zero/pole/gain." This will display your notch filter in the form you are most familiar with.



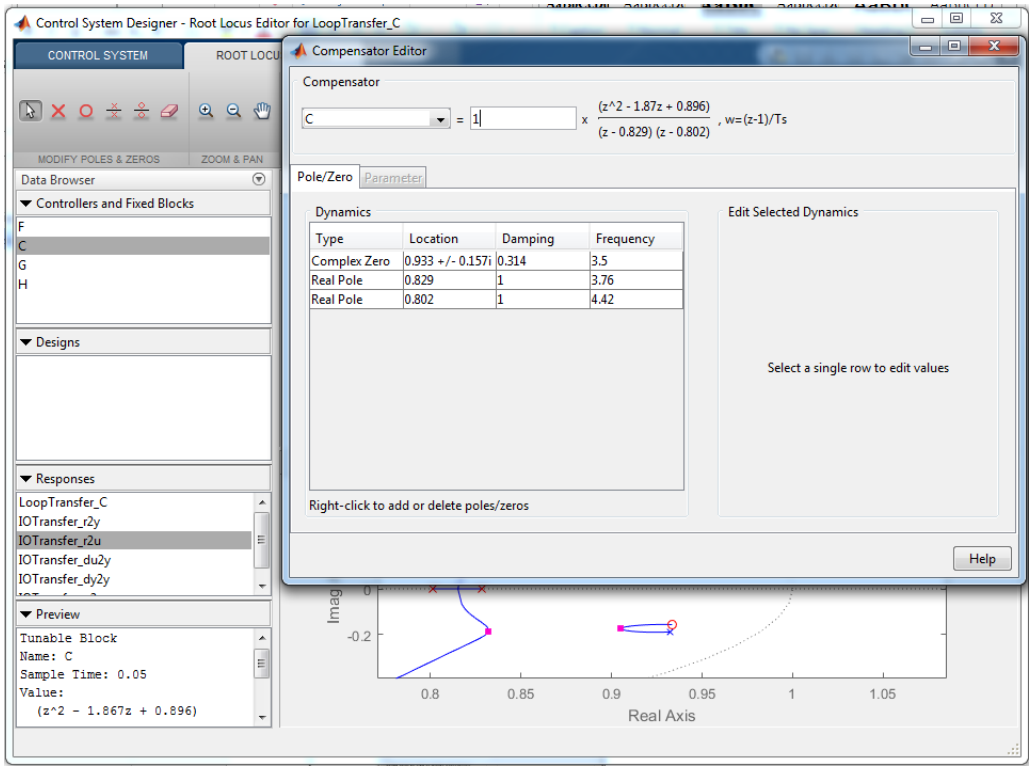
In the root-locus plot zoom in close to the oscillatory poles. Then click on the $\frac{o}{o}$ item and add a set of complex zeros just below the oscillatory poles. We place the zeros just below in order that the root locus branch to the left and then up to the pole instead of right (towards the unit circle) and up to the pole.



Then add the two poles of the notch filter. Click on the x. These poles can be placed anywhere that benefits your controller. Placing them on the real axes is a good start. Make sure to monitor how much control effort is being applied given the pole locations you choose. Fast poles create higher control effort. Poles close to the added zeros can affect your desired response. Drag the poles around a bit and notice the changes in the response and control effort.



The entire time you are designing this notch filter you should keep the gain of the controller "C" around 1. This way when you export your notch filter to Matlab's workspace in the next step the transfer function will have approximately unity gain. Of course your final controller for this system (if you were controlling it) does not have to have unity gain. This is a preferred method to separate the gain of the controller from the gain of the notch filter transfer function. Check that the gain of your notch filter is 1 by double clicking on "C" in the "Controllers and Fixed Blocks" section.



When you are satisfied with your notch design export it to Matlab's workspace to be used in Simulink or to list its coefficients for your DSP C program. In the "Control Systems" tab select "Export." Check the box next to "C" and select "Export."

Now you have a variable in Matlab call "C". Rltool exports C in a pole/zero/gain transfer function form. Convert C to a normal transfer function form by typing in Matlab: notchD=tf(C).

Now you can use this notch in your Simulink simulations or implement it on the DSP. To implement on the DSP you will need to copy the coefficients of the transfer function to your C-file. Use the M-file "arraytoCformat" to do this. If you wanted to use this notch filter in a C program you would:

1. Change Matlab to display many digits of precision and in scientific form "format long e".
2. Then type "arraytoCformat(notchD.num{1}'). Then for example if you were going to implement this notch filter in your C file (you will not be using this notch since this is just an example), copy and paste the output into your C-file and assign the array a name associated with the numerator of the notch filter. Note the '{' to access the numerator elements. "num" is a "cell" type in the "tf" structure.
3. Then type "arraytoCformat(notchD.den{1}'). Again if this was your actual notch filter you would copy the output into your C-file and assign the array a name associated with the denominator of the notch filter.

Laboratory Exercise

This lab is a bit more open ended than the previous labs in that you have a few more choices in deciding what controller to implement. You will need to design a notch filter to cancel the most under-damped poles of the system. But does it also pay to add a second notch to cancel the other set of oscillatory poles? Once the notch filter has been designed, what controller will you use to meet the given design specifications? We have used a number of different controllers in previous labs that you can implement here to accomplish that goal. Figure 1 shows a general block diagram of your controller design task. It lists a number a different choices in controllers that we have learned in earlier labs that can be used in your control implementation. Notice how the notch filter (or notch filters) is just one of the blocks in the diagram. The notch filter does not perform all the needed tasks of the controller. Its only job is to help reduce the effect of oscillatory poles. Additional control is then needed to adjust the closed loop dynamics to the desired specifications.

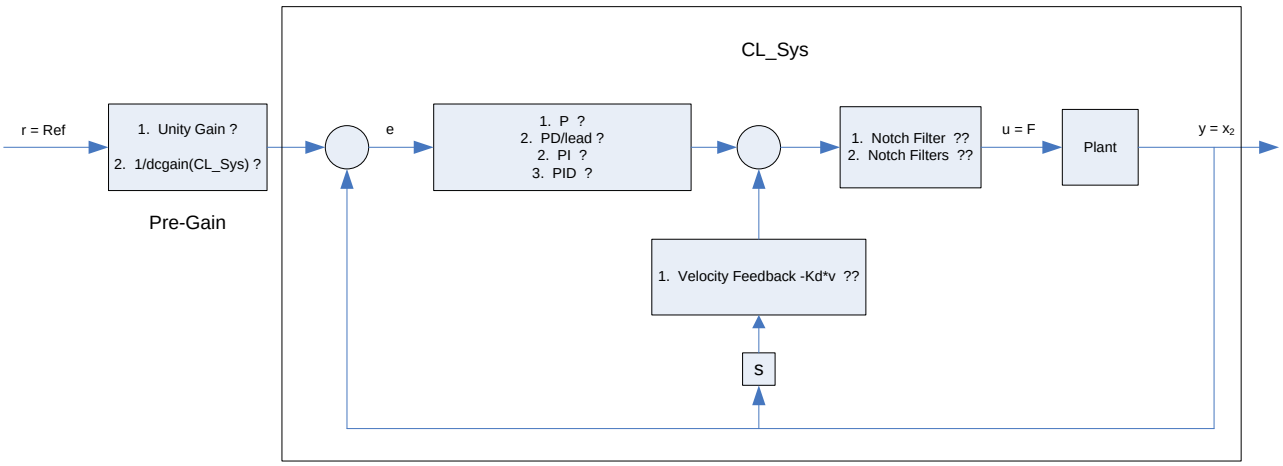


Figure 1.

Your task is to control the system wired for you on the analog computer. First you will design your controller in "rltool", tune the controller by hand or use other design techniques that you choose. Then you will create a Simulink simulation of your closed loop system. Make sure not to forget a saturation block of +/-10 before the input to the plant in your simulation. When you are pleased with the performance of your controller, implement the control on the actual system using the DSP and its ADC and DAC channels. Below you will find the specifications your final system response should obtain. Also you are asked to produce a report that you will be graded on describing your design process.

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 show:

1. An explanation of the work you performed.
2. The steps you used in Matlab to design the controller.
3. Any issues you had with non-linearities in the system.
4. If you hand tuned parts of the controller, how you chose the final gains.
5. Any control attempts that failed.
6. Relevant plots that help describe your control design and results.
7. Root-locus plots that show how you handled the oscillatory poles.
8. Did simulation match the actual controlled system?

9. Were the simulation results effected by saturation?
10. Your Simulink block diagram.
11. Your C-code that implemented the controller.
12. You will also need to demonstrate your controller working to your TA.

The System that you will be Controlling

The analog computer at your bench is wired as the plant you are controlling for this lab. You can think of this system as a standard mass-spring-damper system as shown in Figure 2. We are interested in controlling the position of the x_2 state. In control literature this is often called the "non-collocated" state of this system. This is due to the fact that x_2 is the position of the mass that is not directly connected to the force input to the system. Non-collocated systems can be challenging to control because the only feedback for the system is located away from the input and attached through a spring.

Your single input u (DAC1) to the analog computer can be thought of as the force (F) to this mass-spring-damper system. Your single feedback signal from the analog computer (sampled by ADC1)is the x_2 state.

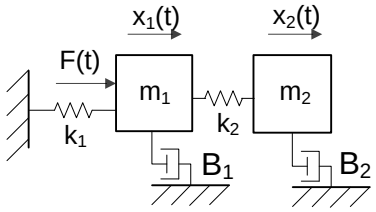


Figure 2

The transfer function of $\frac{X_2(s)}{F(s)}$ is the non-collocated system

$$\frac{X_2(s)}{F(s)} = \frac{\frac{k_2}{m_1 m_2}}{s^4 + (\frac{B_1}{m_1} + \frac{B_2}{m_2})s^3 + (\frac{B_1 B_2}{m_1 m_2} + \frac{k_1 + k_2}{m_1} + \frac{k_2}{m_2})s^2 + \frac{(B_1 k_2 + (k_1 + k_2)B_2}{m_1 m_2}s + \frac{k_1 k_2}{m_1 m_2}}$$

Performing an identification of the system at a 50 millisecond sample period the discrete transfer function is found to be.

$$\frac{X_2(z)}{F(z)} = \frac{3.692\text{x}10^{-7} z^3 + 4.037\text{x}10^{-6} z^2 + 4.015\text{x}10^{-6} z + 3.632\text{x}10^{-7}}{z^4 - 3.964z^3 + 5.901z^2 - 3.910z + 0.973}$$

The full precision of these numbers are given here in a few lines of M-code that you can cut and paste into Matlab to generate this transfer function.

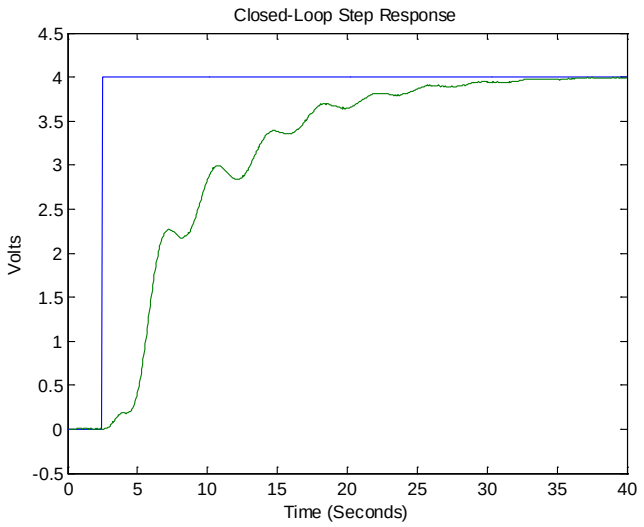
```
num=[    0.0e+000, 3.6922269594e-007, 4.0373228906e-006, 4.0152439996e-006, 3.6319804247e-007];
den=[    1.0e+000, -3.9640076149e+000, 5.9011375399e+000, -3.9100716675e+000,9.7295272376e-001];
myDtf = tf(num,den,.05);
```

Notice that this system is “type 0”. This means that it cannot follow a step input with zero steady state error. To achieve zero steady state error given a step input, your controller will either need to add an integral term or you will need to multiply the reference “r” by a “pre-gain”. This “pre-gain” value is simply the inverse of the DC-gain of the closed loop system. See Figure 1.

Design Specifications to be met

1. 2% overshoot
2. Settling time < 25 seconds
3. 0% steady-state error
4. Minimal oscillations at steady-state.
5. The reference input is a step from 0 to 4.

The plot below shows a response of a controller that would meet these specifications.



Matlab Help for checking if Single (float 32bit) precision will work for your notch design.

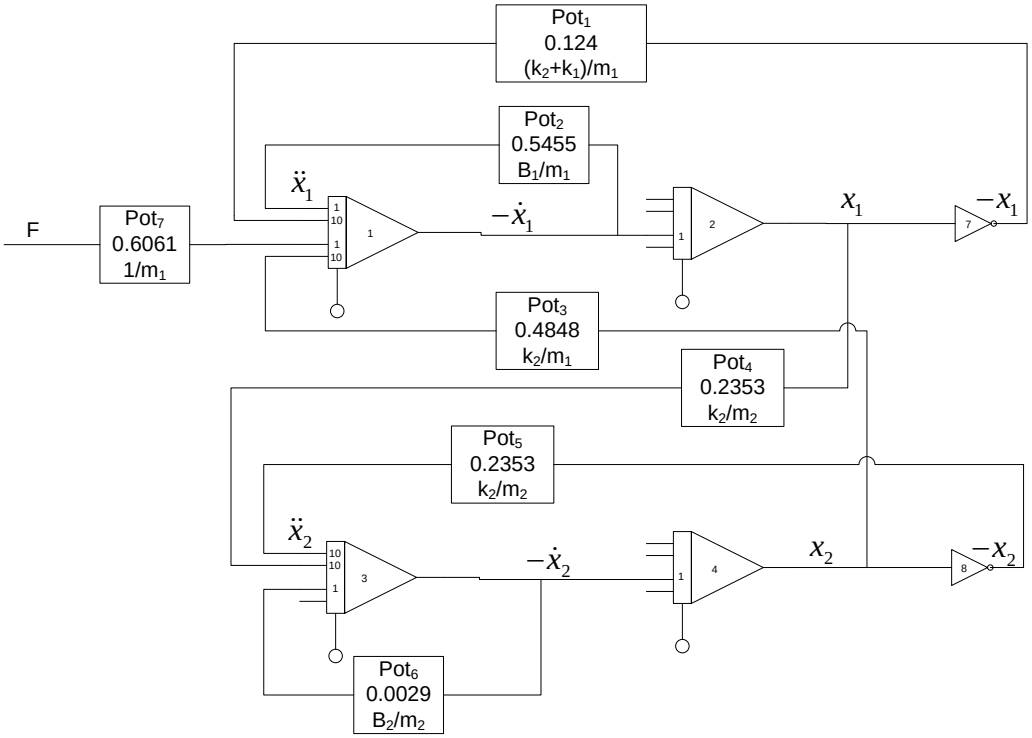
```
doublenum = notchD.num{1};
doubleden = notchD.den{1};
singlenum = single(doublenum);
singleden = single(doubleden);
singlenotchD = tf(singlenum,singleden,0.05)
```

Then use “singlenotchD” in your Simulink simulation and see if the notch filter still works.

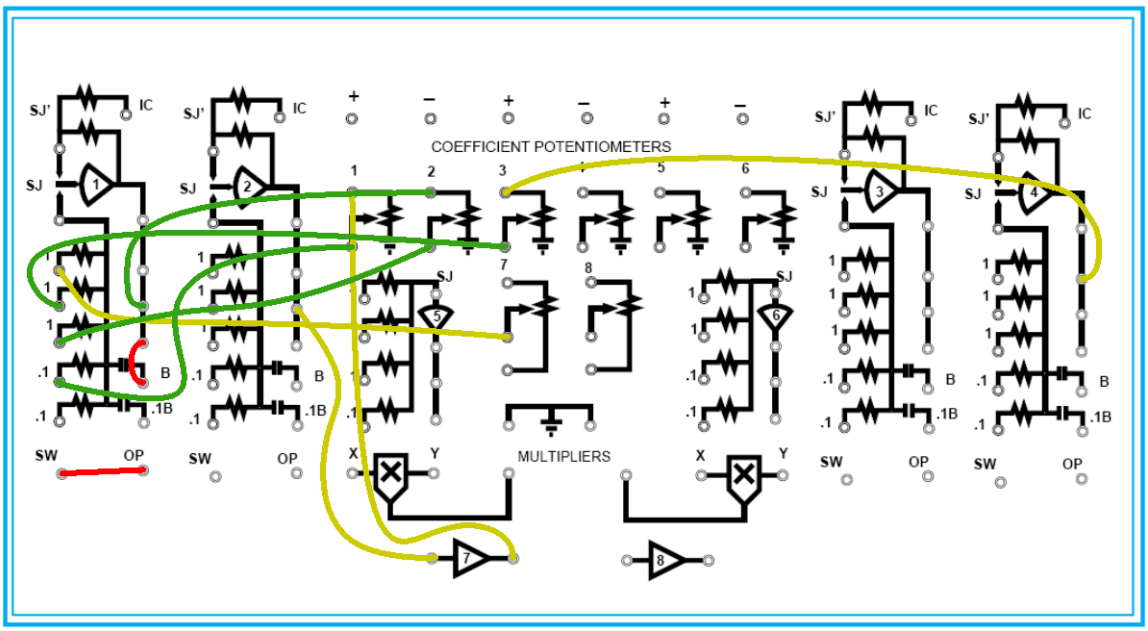
Lab Check Off:

1. Demo your working controller to your TA
2. Hand-in your individual report explaining the work that you did in this lab. See bullet points given above.

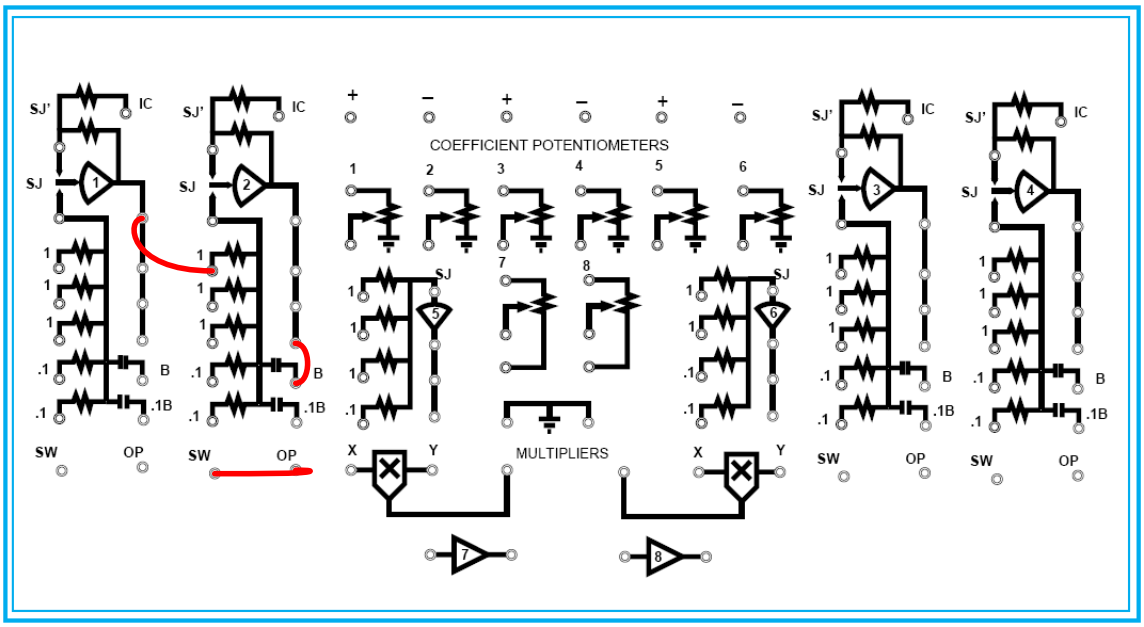
Appendix A: TA instructions for wiring the analog computer.



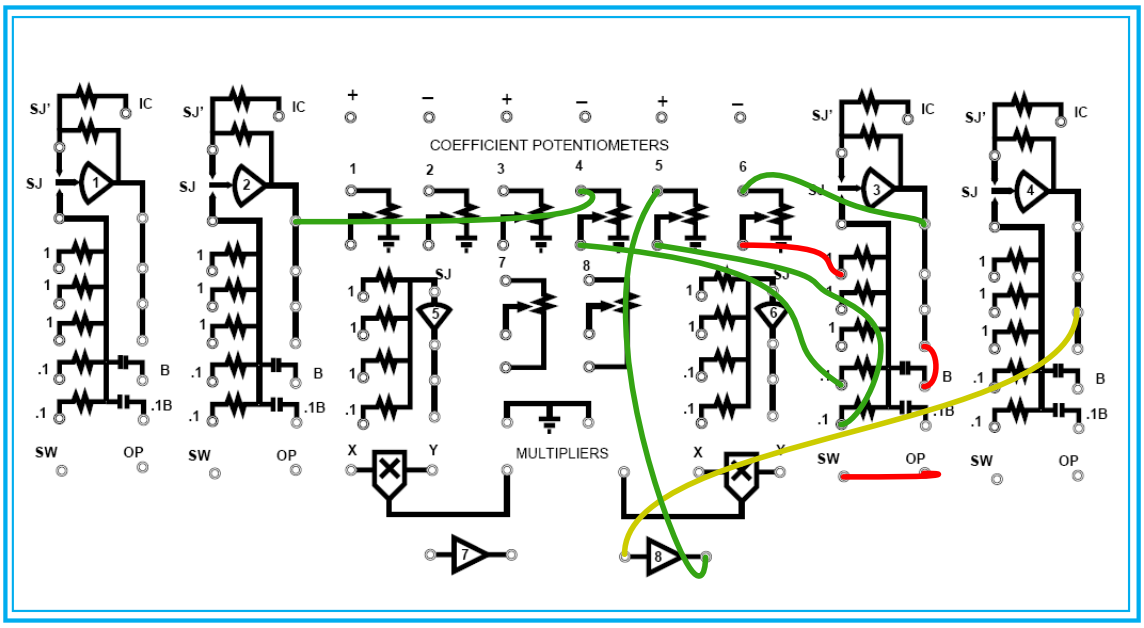
Wiring a four integrator circuit



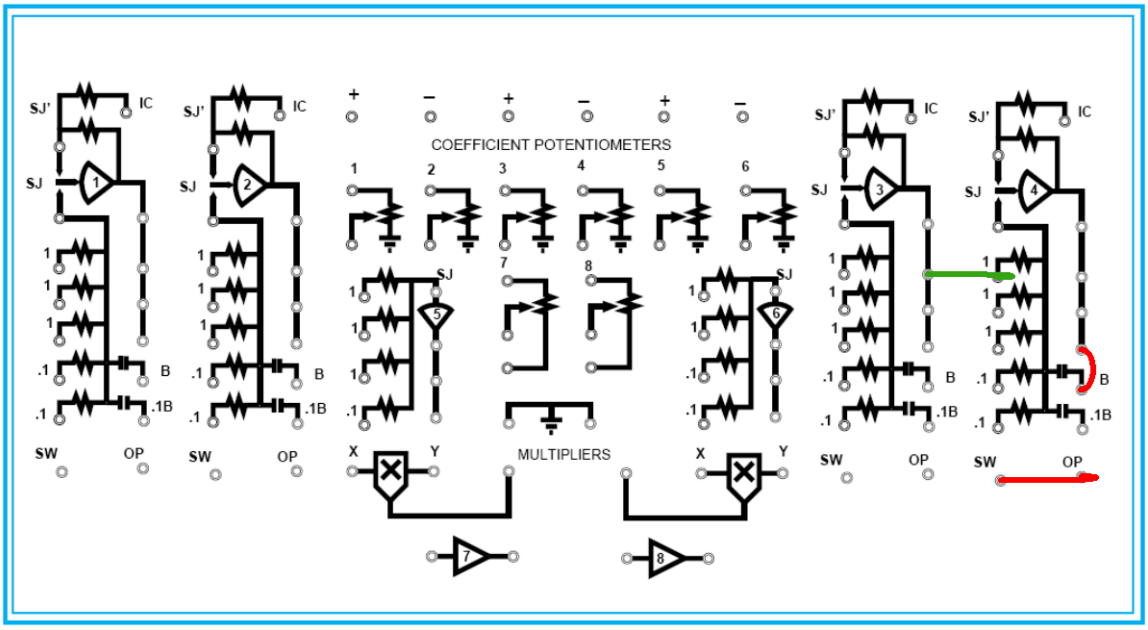
Integrator 1



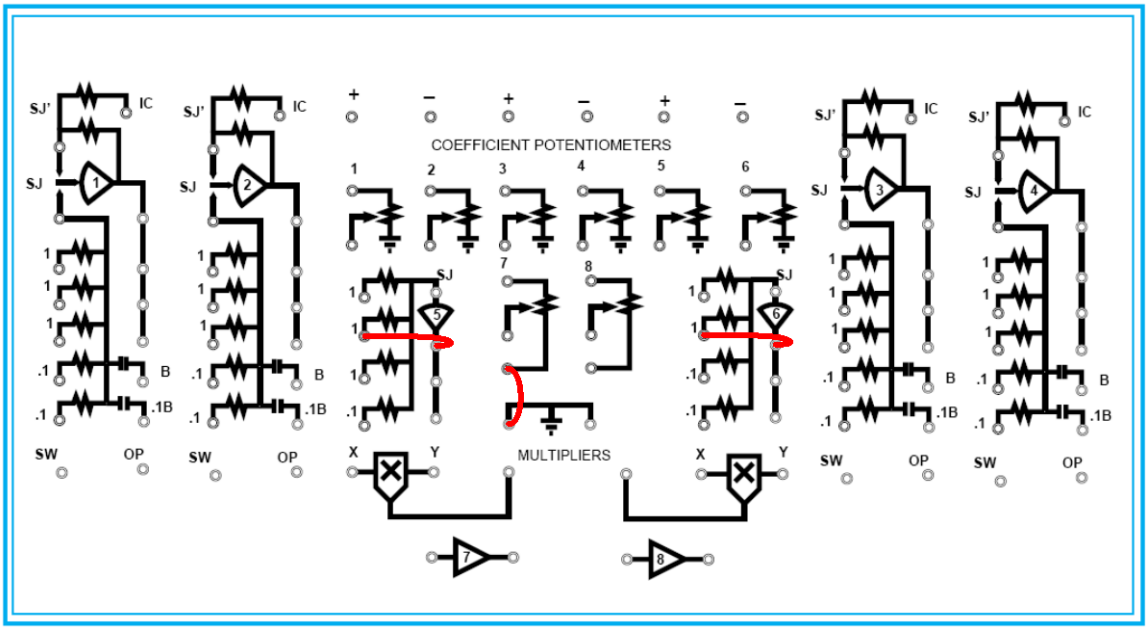
Integrator 2



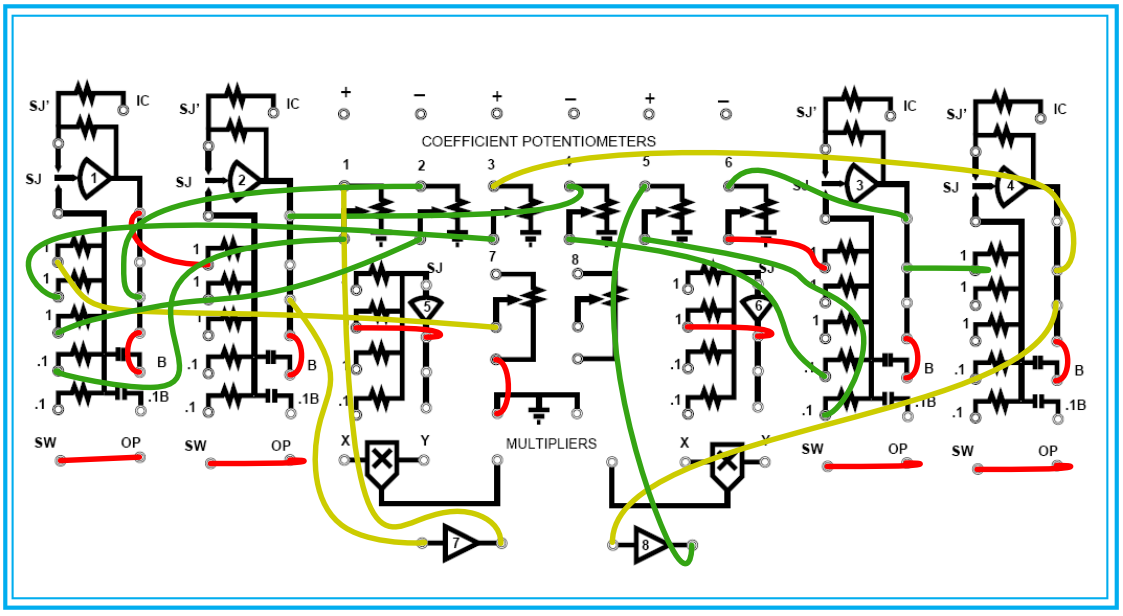
Integrator 3



Integrator 4



Final Connections



Full layout