ECE-320 Lab 4: PI-D and I-PD Control with Dynamic Prefilters

Overview

In this lab you will be controlling the one degree of freedom systems you previously modeled using PI-D and I-PD controllers with and without dynamic prefilters.

You will need your mathematical models for your systems the ***closedloop_driver.m*** *and* ***closedloop.mdl*** *files for this lab.*

Design Specifications: *For each of your systems, you should try and adjust your parameters until you have achieved the following:*

- Settling time less than 1.0 seconds.
- Steady state error less than 0.1 cm for a 1 cm step, and less than 0.05 cm for a 0.5 cm step
- Percent Overshoot less than 25%

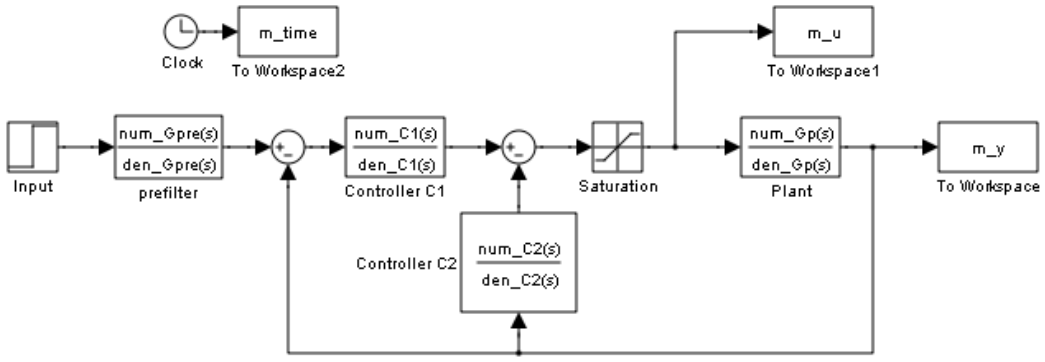
As a start, you should initially limit your gains as follows:

$$\begin{aligned} k_p &\leq 0.5 \\ k_i &\leq 5 \\ k_d &\leq 0.01 \end{aligned}$$

Your memo should include four graphs for each of the 1 dof systems you used (one PI-D and one I-PD controller with and without dynamic prefilters.) Be sure to include the values of k_p, k_i, *and* k_d *in the captions for each figure. Your memo should compare the difference between the predicted response (from the model) and the real response (from the real system) for each of the systems. How does the use of a dynamic prefilter change the response? Attach your Matlab driver file* ***closedloop_driver.m***

Background: While PID controllers are very versatile, they have a number of drawbacks. One of the major drawbacks is that for a unit step input, the control effort $u(t)$ can be infinite at the initial time. This is referred to as a *set-point kick*. There are two commonly used configurations of PID controls schemes that utilize a different structure, the PI-D and the I-PD controllers. These are a bit more difficult to model using Matlab's *sisotool*, but it can be done and we get to explore more of *sisotool*.

The PI-D controller avoids the set-point kick by putting the derivative in the feedback path, while the I-PD controller avoids the set-point kick by placing both the derivative and proportional terms in the feedback path. Both types of controllers can be implemented using the following Simulink model.



For the PI-D controller, we have

$$C_1(s) = k_p + \frac{k_i}{s}, \quad C_2(s) = \frac{k_d s}{\frac{1}{50}s + 1}$$

while for the I-PD controller we have

$$C_1(s) = \frac{k_i}{s} \quad , \; C_2(s) = k_p + \frac{k_d s}{\frac{1}{50}s + 1}$$

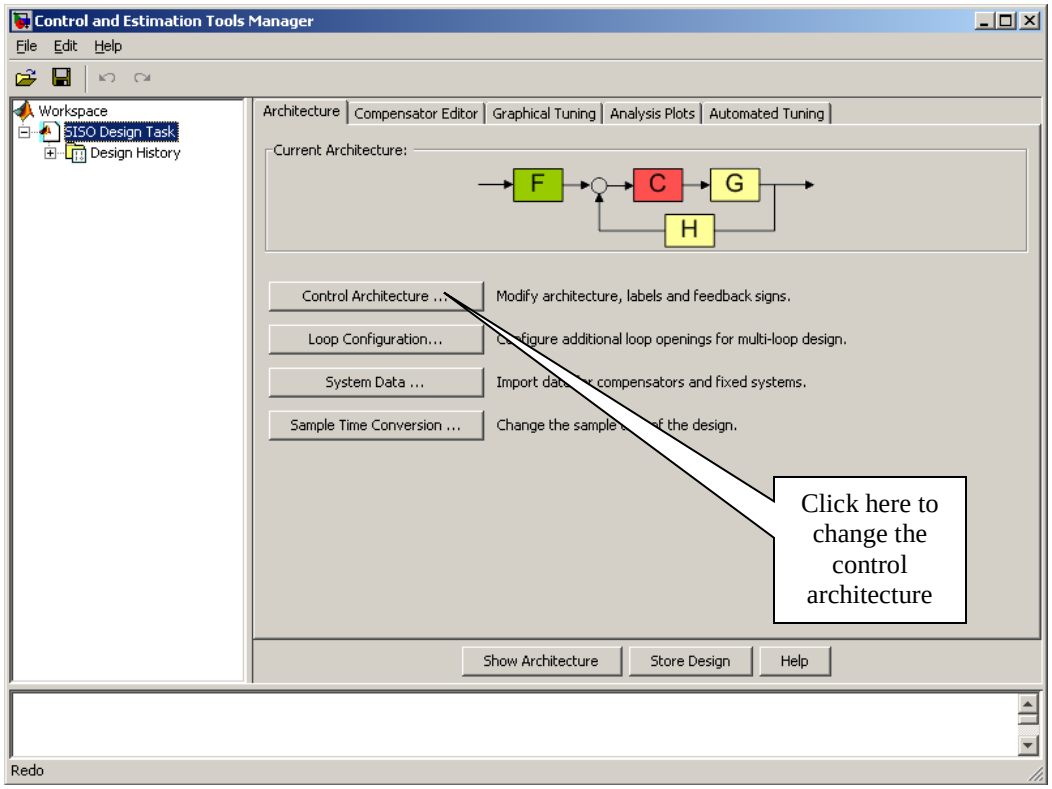
Note that for both controllers we continue to use a lowpass filter (with a cutoff of 50 rad/sec) in series with a differentiator.

For both of these controllers, if we ignore the prefilter (assume it is unity), the transfer function from input to output is

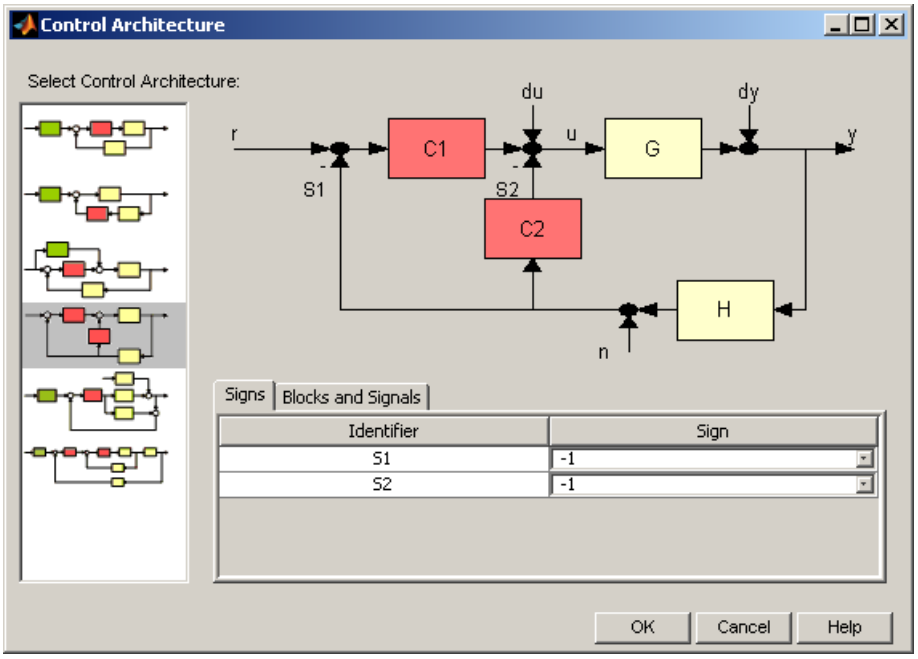
$$\frac{Y(s)}{R(s)} = \frac{C_1(s)G_p(s)}{1 + C_2(s)G_p(s) + C_1(s)G_p(s)}$$

Next we need to use *sisotool* to help determine reasonable values for k_p, k_i, and k_d. You should go through the following example before you try to design a controller for your own system.

When you start sisotool, you need to click on Control Architecture to get the proper configuration. Be sure that sisotool uses negative feedback for both loops.



Next we need to select the correct architecture. Select the following architecture and be sure that sisotool uses negative feedback for both loops.

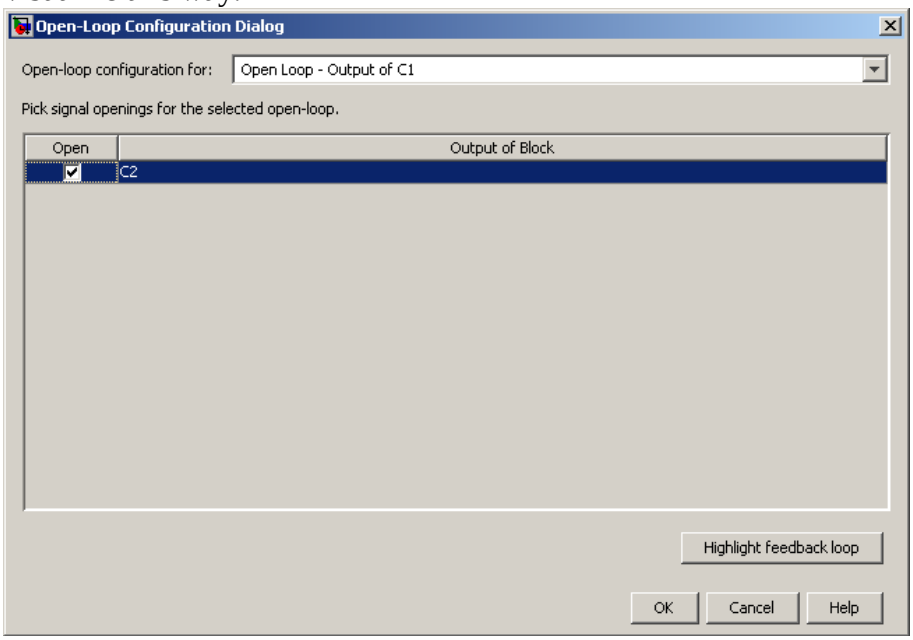


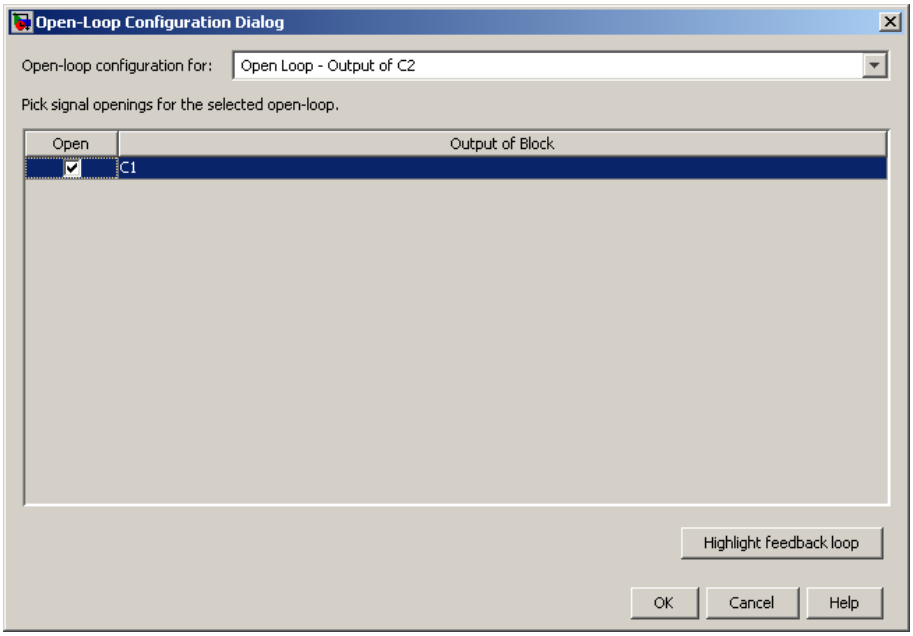
Click on OK and when the previous window appears, select **Loop Configuration.**

In the block Open-loop configuration for: select **Open Loop Output of C1** and select **Open** for **Output of Block C2.**

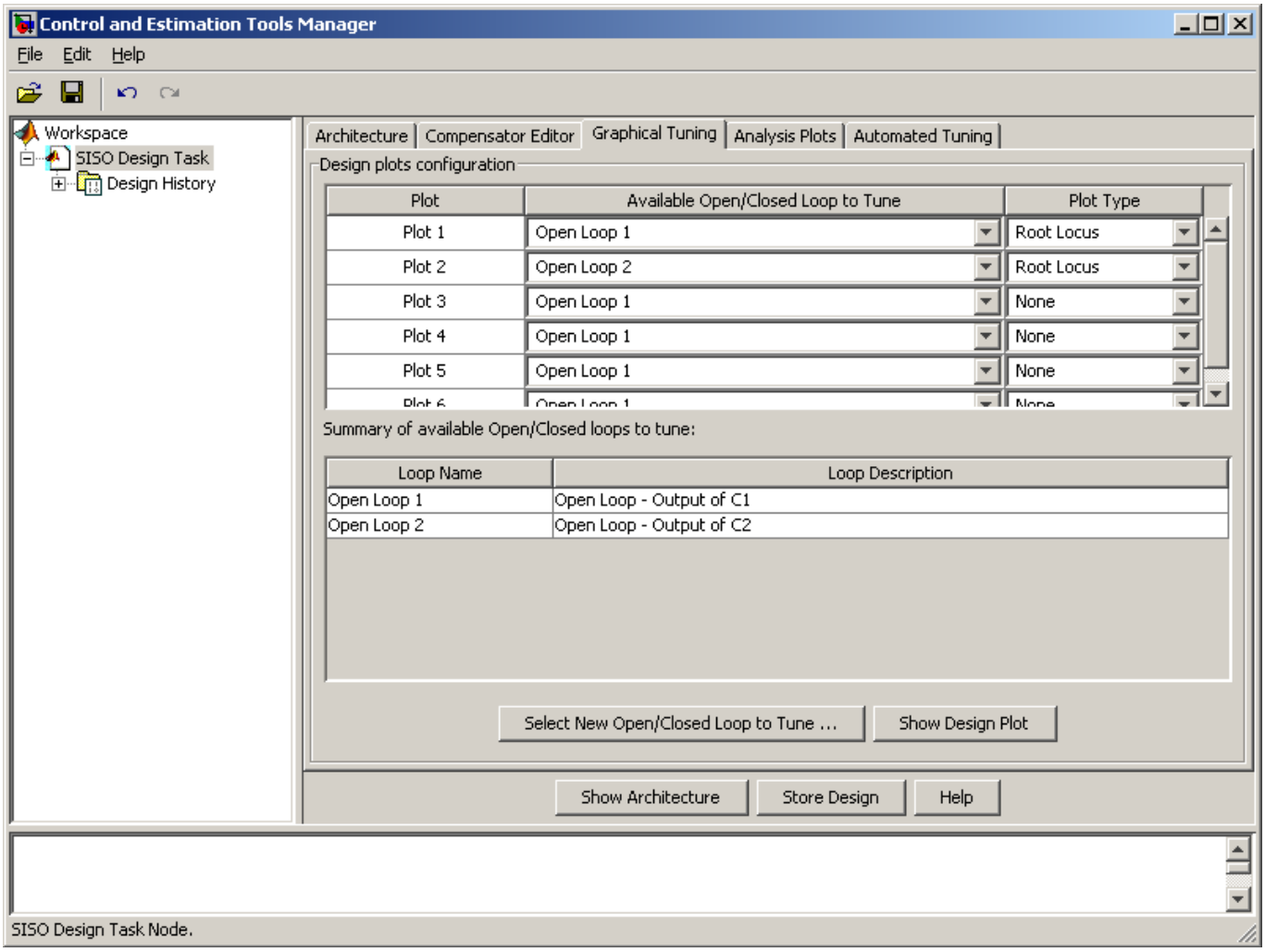
In the block Open-loop configuration for: select **Open Loop Output of C2** and select **Open** for **Output of Block C1.**

The following figures demonstrate what you are to do. Note that this step is not really necessary, but I find it a bit less confusing. The controllers in each window may not be exactly what you are expecting, but it is easier to visualize this way.



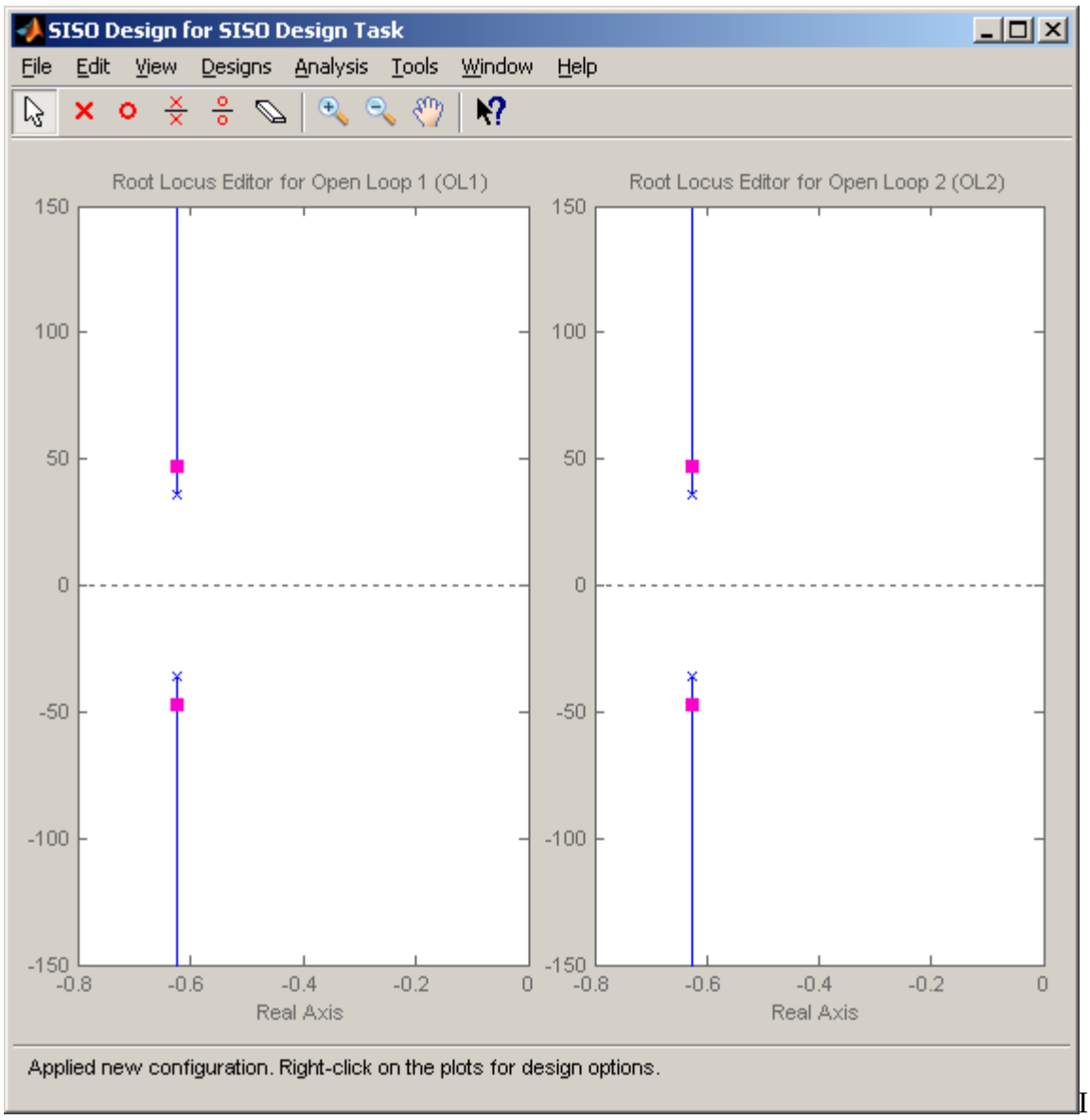


Next, go back to our design window, and select **View**, then **Design Plots Configuration.** Then choose to plot the **Root Locus** plot for both **Open Loop 1**and **Open Loop 2**, as the following window shows:



You should now have a design window with two (empty) root locus plots and we are ready to start.

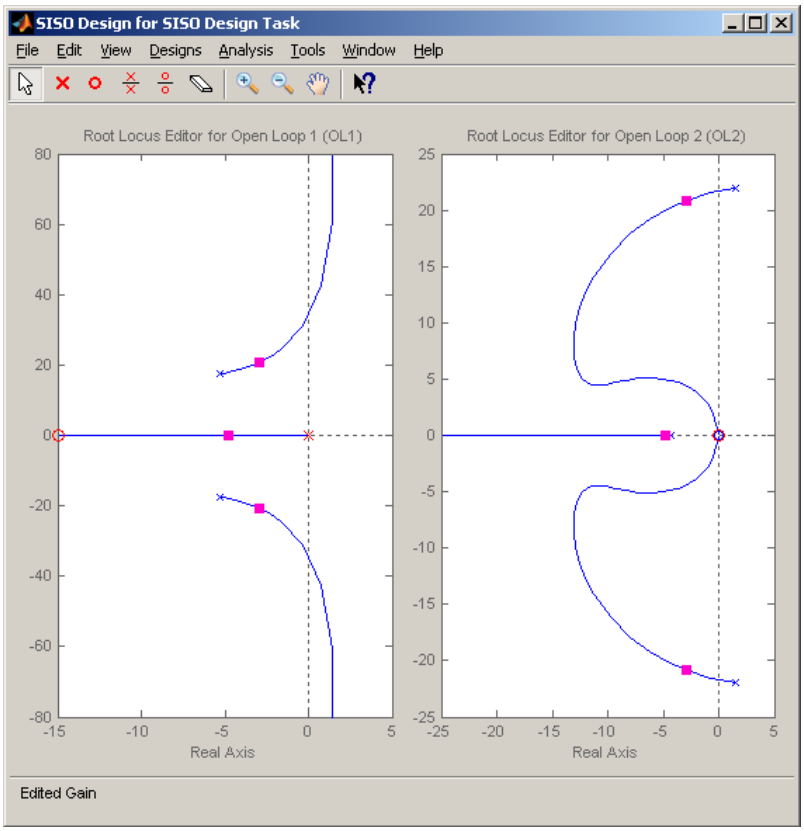
For practice, let's assume our plant is $G_p(s) = \dfrac{938.4}{s^2 + 1.25s + 329.8}$ .When we import the plant, we should get a design window that looks like the figure on the following page. Each window shows the root locus for the plant and two identical proportional controllers.



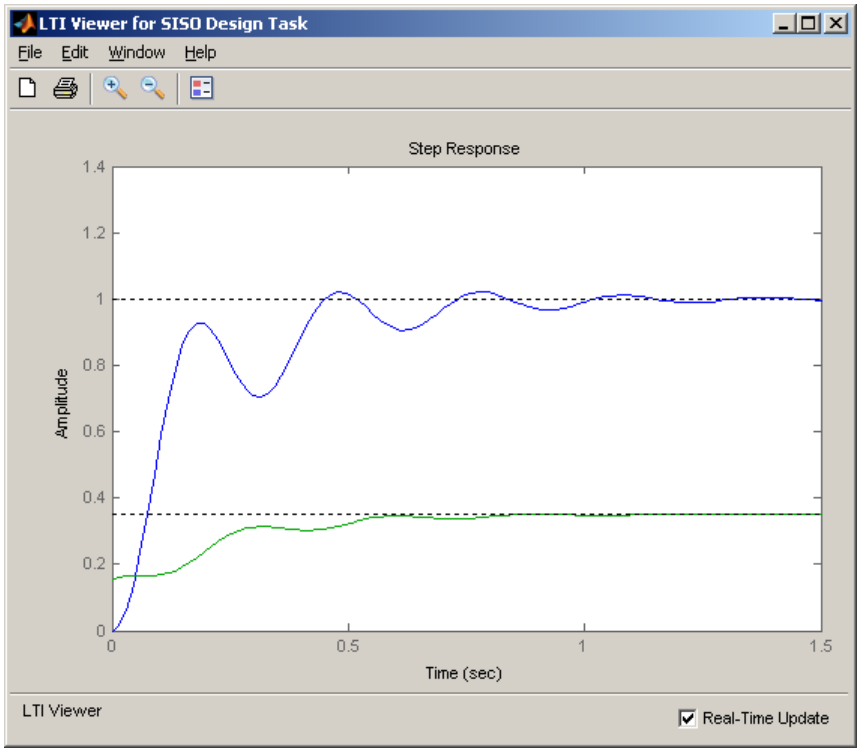
Let's initially assume we want to use a PI-D controller. In this case, for simulation purposes, we have $C_2(s) = k_d s$ (we won't worry about the lowpass filter here). We will start off asuming $k_d = 0.01$, but this is just a guess! To assign this to controller C2, in the design window we select **Designs**, the **Edit Compensator**, then select **C2** in the compensator window, and enter the controller.

Next we'll enter the PI part of the controller into $C_1(s)$.

As a starting point, let's assume $C_1(s) = \dfrac{0.15(s+15)}{s}$ and enter this into **C1.** If you have done this correctly, you should get the following root locus plots.



The step response for this controller configuration is:

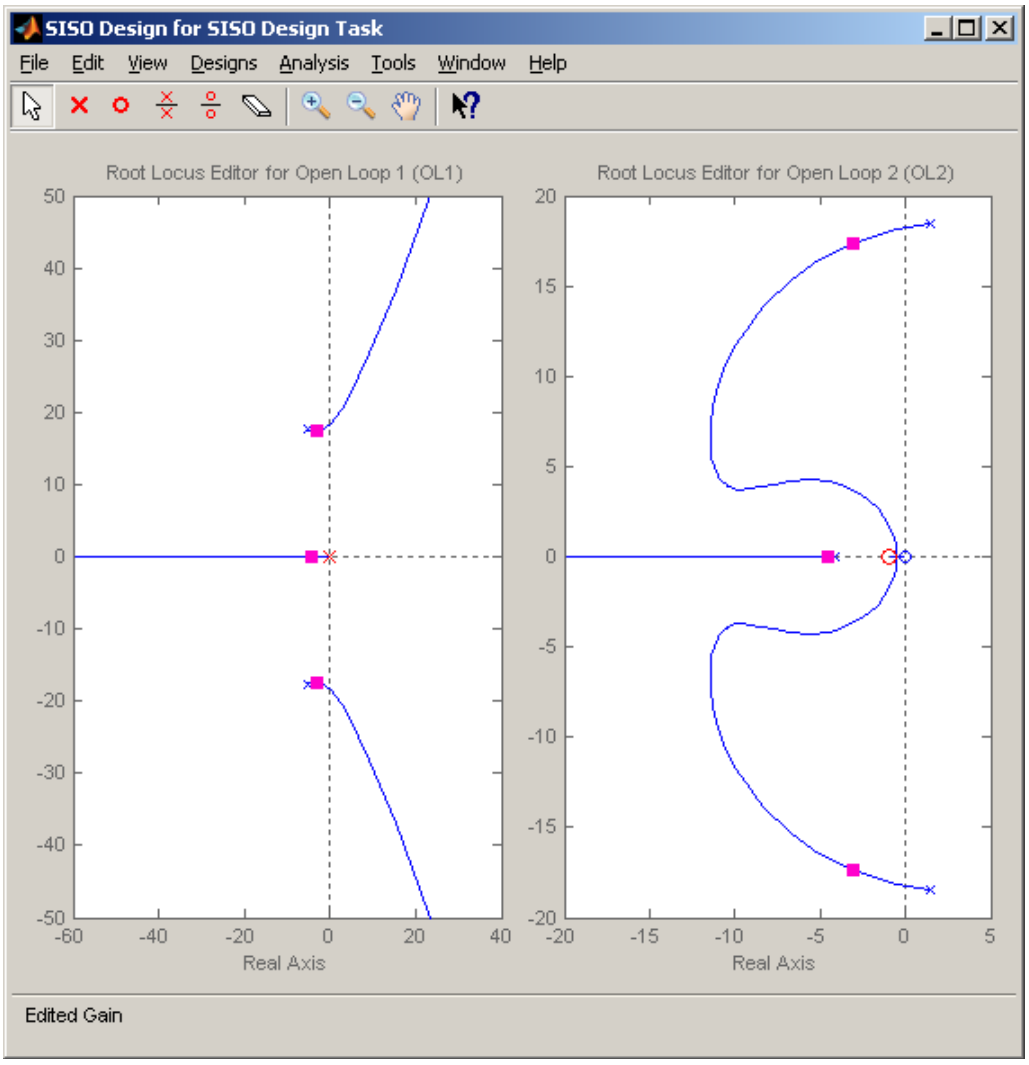


Note that, compared to a normal PID controller, the control effort is not infinite at 0, and actually builds as time goes on (like an integral controller). At this point we might want to go back modify our controllers C1 and C2 to see if we get acceptable performance. Note that we can modify the two controllers (and gains) independently.

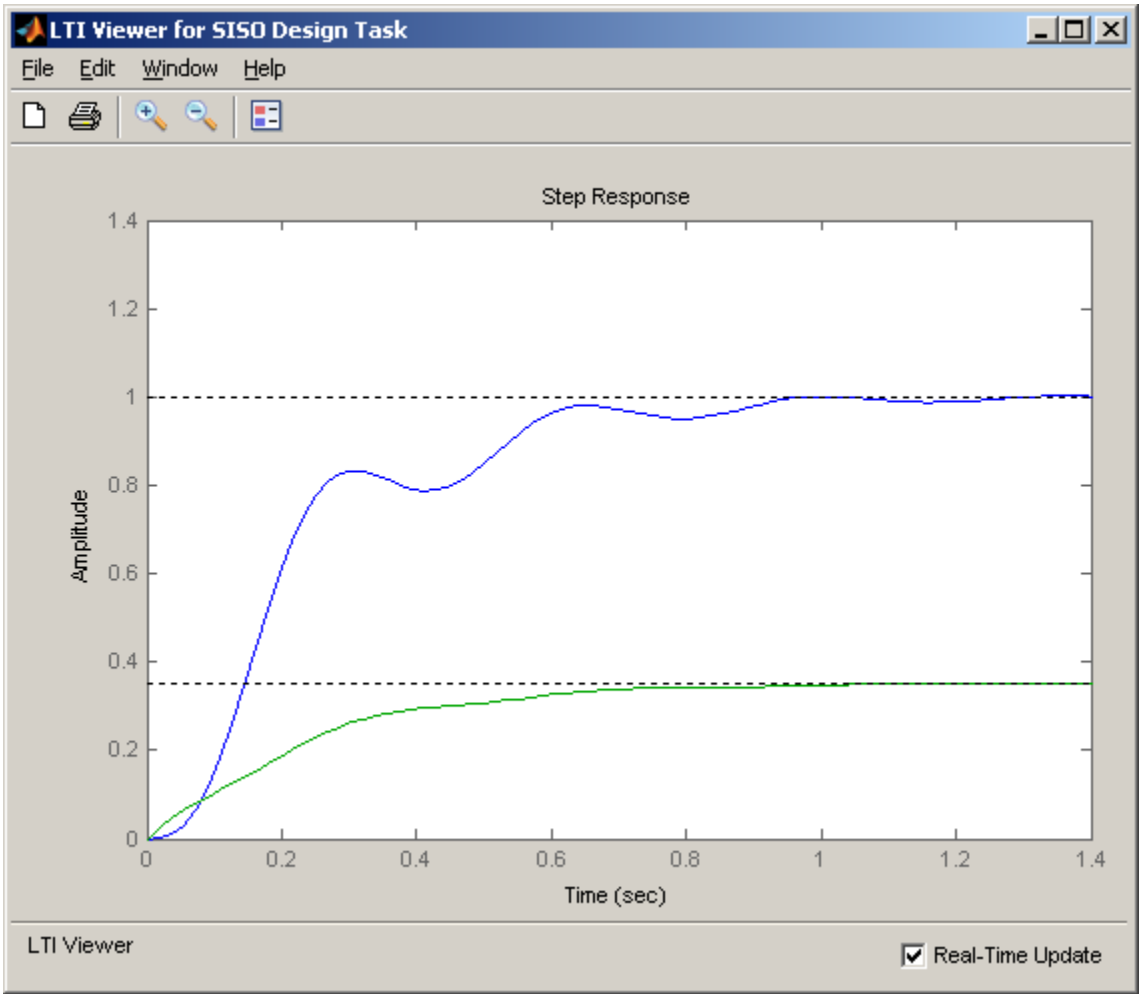
Now we'll assume we want to control the same plant, but this time use an I-PD controller. We first have to guess a PD controller. For the systems I tired, I found that the zero should be fairly small, between -1 and -5. As a start, we'll try the following PD and I controllers

$$C_1(s) = \frac{1.5}{s}, C_2(s) = 0.01(s+1)$$

If you enter these controllers, you should get the following root locus plots:

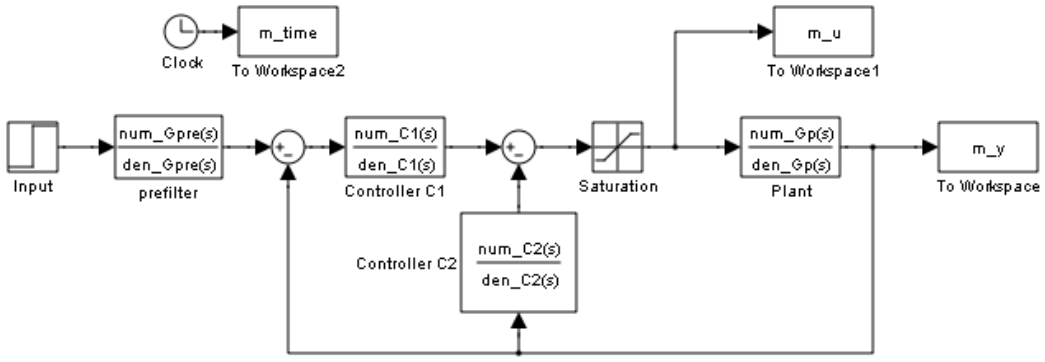


The step response for this system is plotted on the following page. Again, you can change the controllers independently to try to obtain an acceptable response.



Now we want to design and implement these types of controllers for your systems. You should go through the following steps for each of your systems.

Step 1: Save the file **closedloop.mdl** as **closedloop2.mdl**, then change the Simulink model in **closedloop2.mdl** to match the new control structure shown below:



Step 2: Modify **closedloop_driver.m** to read in the correct model file for your system and implement the new control structure by *running* **closedloop2.mdl**. In particular, you must now determine C1(s) and C2(s). The best way to implement the controller, for the PI-D controller (for example) is

C1 = tf(kp,1)+tf(ki,[1 0]); C2 = tf([kd 0],[1/50 1]);

You then need to determine the numerator and the denominator for C1 and C2 (since the Simulink model file **closedloop2** expects them). You should look to see how this was done for Gc(s) for the regular PID controller.

Finally, to determine the closed loop transfer function (for the dynamic profiler) type

Go = mineral(C1*Gp/(1+C1*GP+C2*Gp),tol);

and then get the numerator and denominator as before.

Step 3: Set up the 1 dof systems exactly the way they were when you determined their model parameters.

Step 4: Save the ECP driver file Model210_Closedloop.mdl as **Model210_Closedloop2.mdl** and modify it to implement the new control structure, just as you did in **closedloop2.mdl**.

.

Step 4: PI-D Control

- Design a PI-D controller to meet the design specs. Use a **constant prefilter** (i.e., a number, most likely the number 1). Be sure to observe the limits on the other gains.
- Implement the correct gains into **closedloop_driver.m**
- Simulate the system for 1.5 seconds. If the design constrains are not met, or the control effort hits a limit, redesign your controller (you might also try a lower input signal)

- Compile the correct closed loop ECP Simulink driver, connect to the system, and run the simulation.

- Use the **compare1.m** file (or a modification of it) to plot the results of both the simulation and the real system on one nice, neatly labeled graph. You need to include this graph in your memo. *Be sure to include the values of* k_p, k_i, *and* k_d *in your memo.*

- Change the prefilter to cancel the zeros of the closed loop system and still have a steady state error of zero. Rerun the simulation, recompile the ECP system, run the ECP system, and compare the predicted with the measured response. You also need to include this graph in your memo. *Be sure to include the values of* k_p, k_i, *and* k_d *in your memo.*

Step 6: I-PD Control

- Design an I-PD controller to meet the design specs. Use a **constant prefilter** (i.e., a number, most likely the number 1). Keep the zero of the PD controller small, between -1 and -5, and be sure to observe the limits on the other gains.

- Implement the correct gains into **closedloop_driver.m**

- Simulate the system for 1.5 seconds.If the design constrains are not met, or the control effort hits a limit, redesign your controller (you might also try a lower input signal)

- Compile the correct closed loop ECP Simulink driver, connect to the system, and run the simulation.

- Use the **compare1.m** file (or a modification of it) to plot the results of both the simulation and the real system on one nice, neatly labeled graph. You need to include this graph in your memo. *Be sure to include the values of* k_p, k_i, *and* k_d *in your memo.*

- Change the prefilter to cancel the zeros of the closed loop system and still have a steady state error of zero. Rerun the simulation, recompile the ECP system, run the ECP system, and compare the predicted with the measured response. You also need to include this graph in your memo. *Be sure to include the values of* k_p, k_i, *and* k_d *in your memo.*