ECE-520 Lab 4
Discrete-Time PI-D and I-PD Controllers and sisotool

Overview

In this lab you will be controlling the one degree of freedom systems you previously modeled using PI-D and I-PD controllers. Both one degree of freedom systems must be controlled, and if there are two people in your lab group each lab partner should do a different system.

You will need your Matlab model files for your systems and the files ***DT_PID_driver.m*** *and* ***DT_PID.mdl*** *from your homework or last weeks lab. You will need to copy and then modify* ***Model210_DT_PID.mdl*** *and* ***Model205_DT_PID.mdl*** *from last week's lab. You will also need the files from the basic files group (****ECPDSPReset.c, ECPDSPReset.dll, ECPDSPRest.c, ECPDSPDriver.c, ECPDSPDriver.dll, ECPDSPDriver.h****)*

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

Torsional Systems (Model 205)

- Settling time less than 1.5 seconds.
- Steady state error less than 2 degrees for a 15 degree step, and less than 1 degree for a 10 degree step (*the input to the Model 205 must be in radians!*)
- Percent Overshoot less than 25%

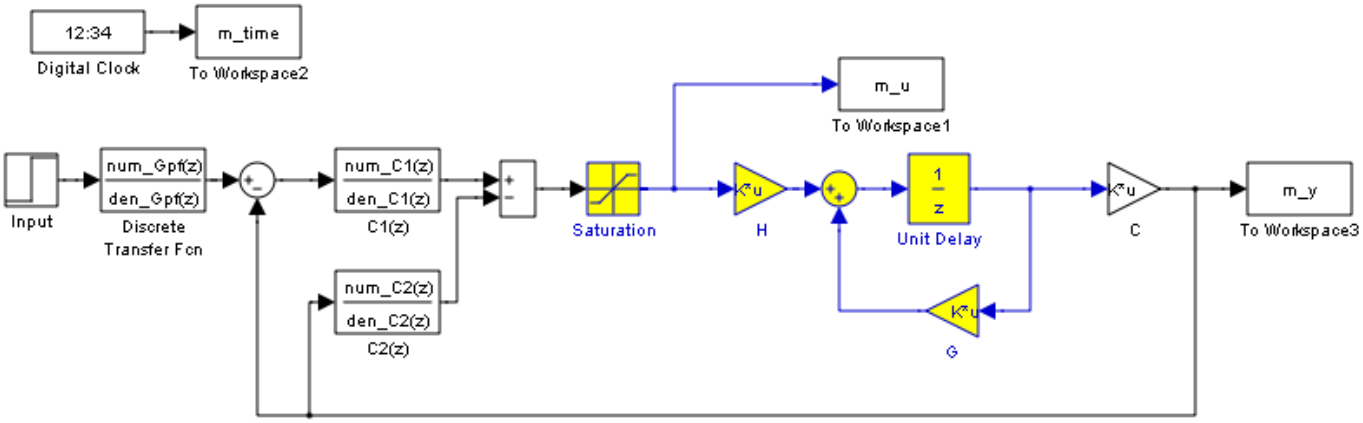
Rectilinear Systems (Model 210)

- Settling time less than 1.5 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%

Your memo should include four graphs for each of the 1 dof systems you used (one PI-D and one I-PD controller using two sample rates for each system.) 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.

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 $u(k)$ can be very large 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, which you should construct and name appropriately:



For the PI-D controller, we have

$$C_1(z) = K_p + \frac{K_i z}{z-1} = \frac{(K_p + K_i)z - K_p}{z-1}, \quad C_2(z) = K_d(1 - z^{-1}) = \frac{K_d(z-1)}{z}$$

while for the I-PD controller we have

$$C_1(z) = \frac{K_i}{1 - z^{-1}} = \frac{K_i z}{z-1} \quad , \; C_2(z) = K_p + K_d(1 - z^{-1}) = \frac{(K_p + K_d)z - K_d}{z}$$

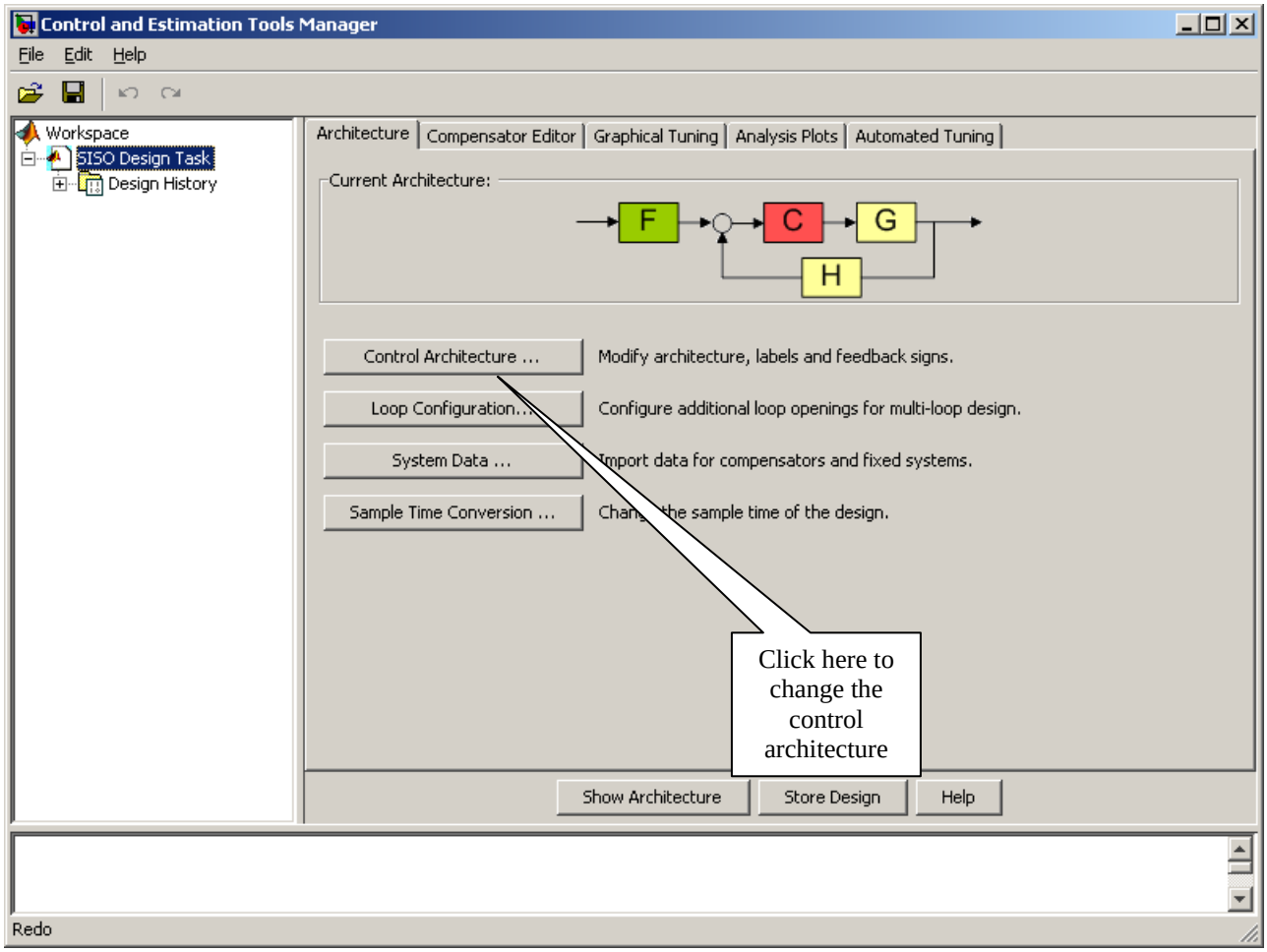
Note that for both controllers we might want to use a lowpass filter (such as a running averaging filer) in series with a differentiator.

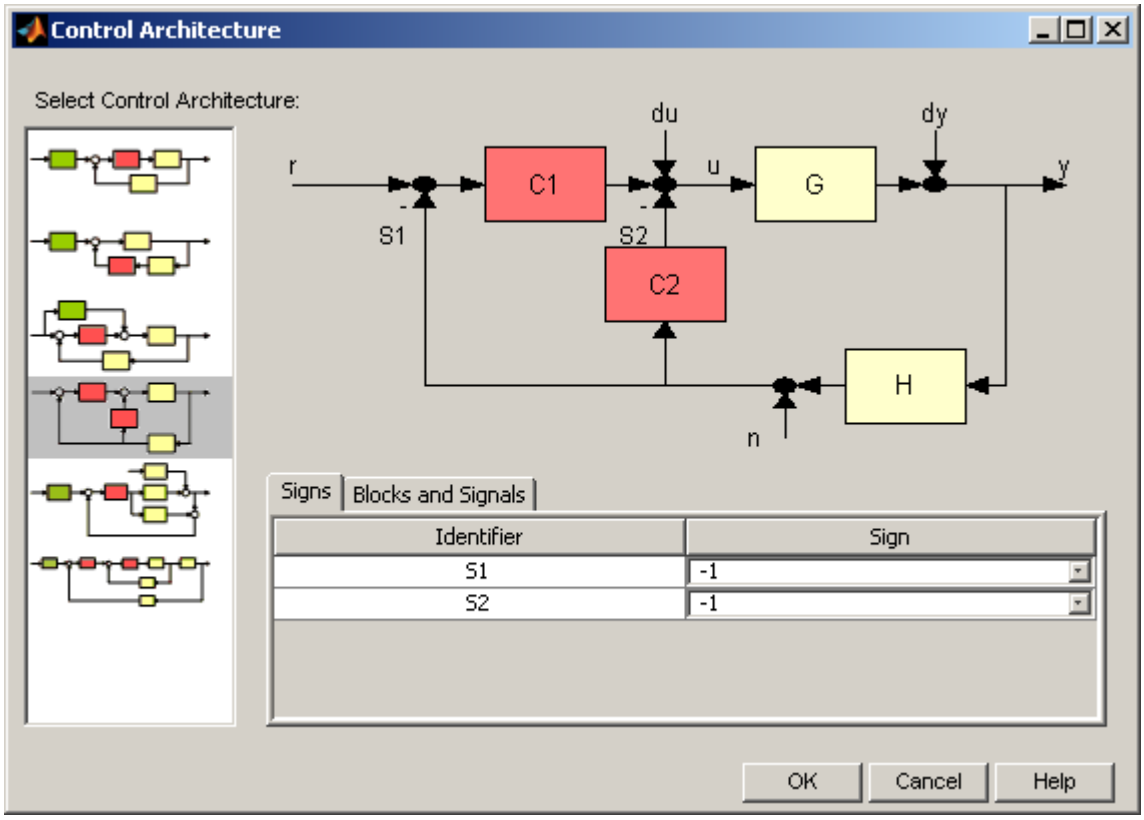
For both of these controllers, if we ignore the prefilter (assume it is unity), the transfer function from input to output is

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

In the Simulink models we have represented the plant $G_p(z)$ in state variable form. Next we need to use *sisotool* to help determine reasonable values for K_p , K_i, and K_d.

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.***


Control and Estimation Tools Manager
File Edit Help
Workspace
SISO Design Task
Design History
Architecture
Compensator Editor
Graphical Tuning
Analysis Plots
Automated Tuning
Current Architecture:
F
C
G
H
Control Architecture ...
Modify architecture, labels and feedback signs.
Loop Configuration...
Configure additional loop openings for multi-loop design.
System Data ...
Import data for compensators and fixed systems.
Sample Time Conversion ...
Chan the sample time of the design.
Click here to change the control architecture
Show Architecture
Store Design
Help
Redo

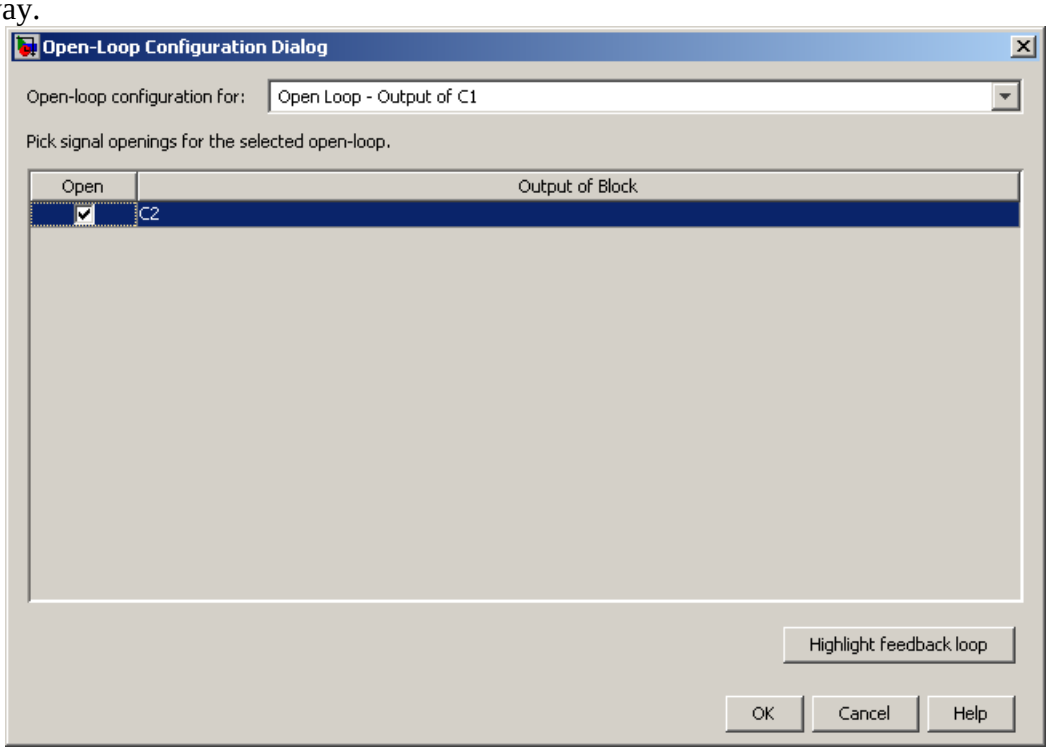

Control Architecture
Select Control Architecture:
r
du
dy
C1
u
G
y
S1
S2
C2
H
n
Signs
Blocks and Signals
Identifier
Sign
S1
-1
S2
-1
OK
Cancel
Help

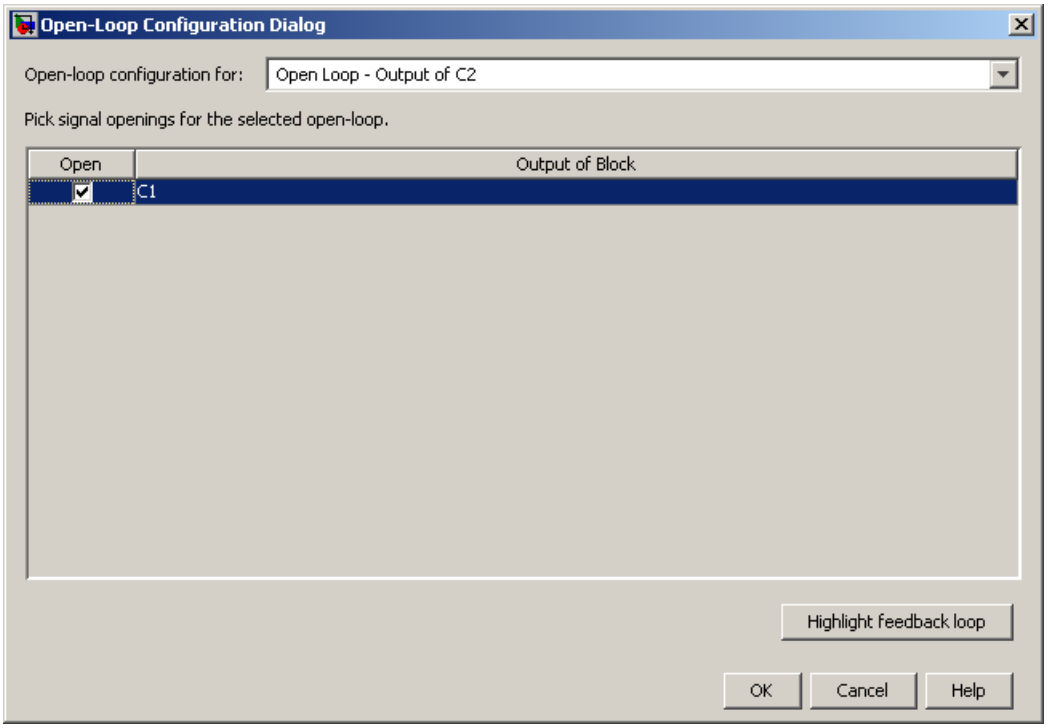
Click on OK and when the previous window appears, select **Loop Configuration.**

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

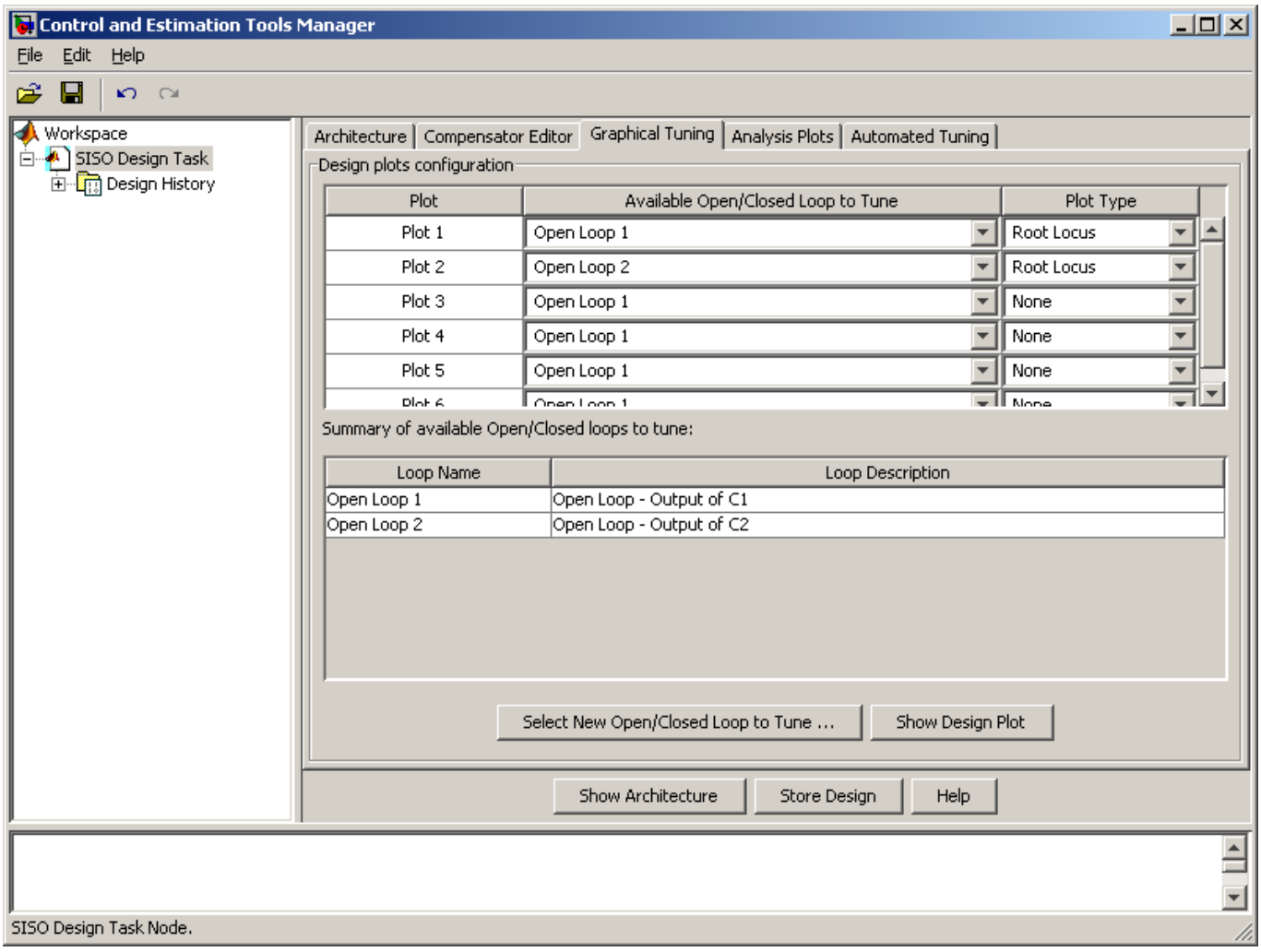
In the block Open-loop configuration for: select **Open Loop Output of C2** and select **Open** for **Output of Block C1.**

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:

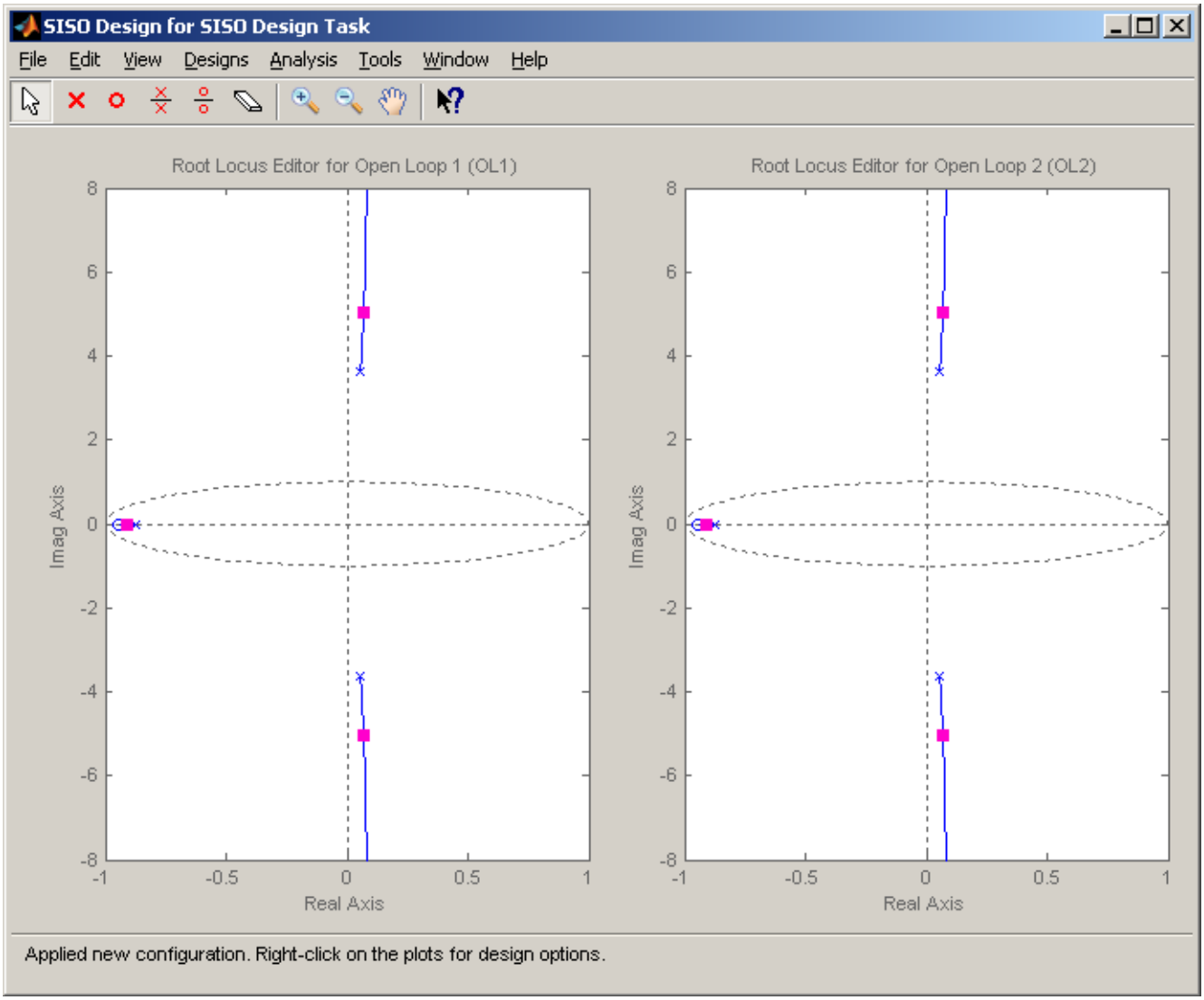


You should now have a design window with two (empty) root locus plots and we are ready to start.

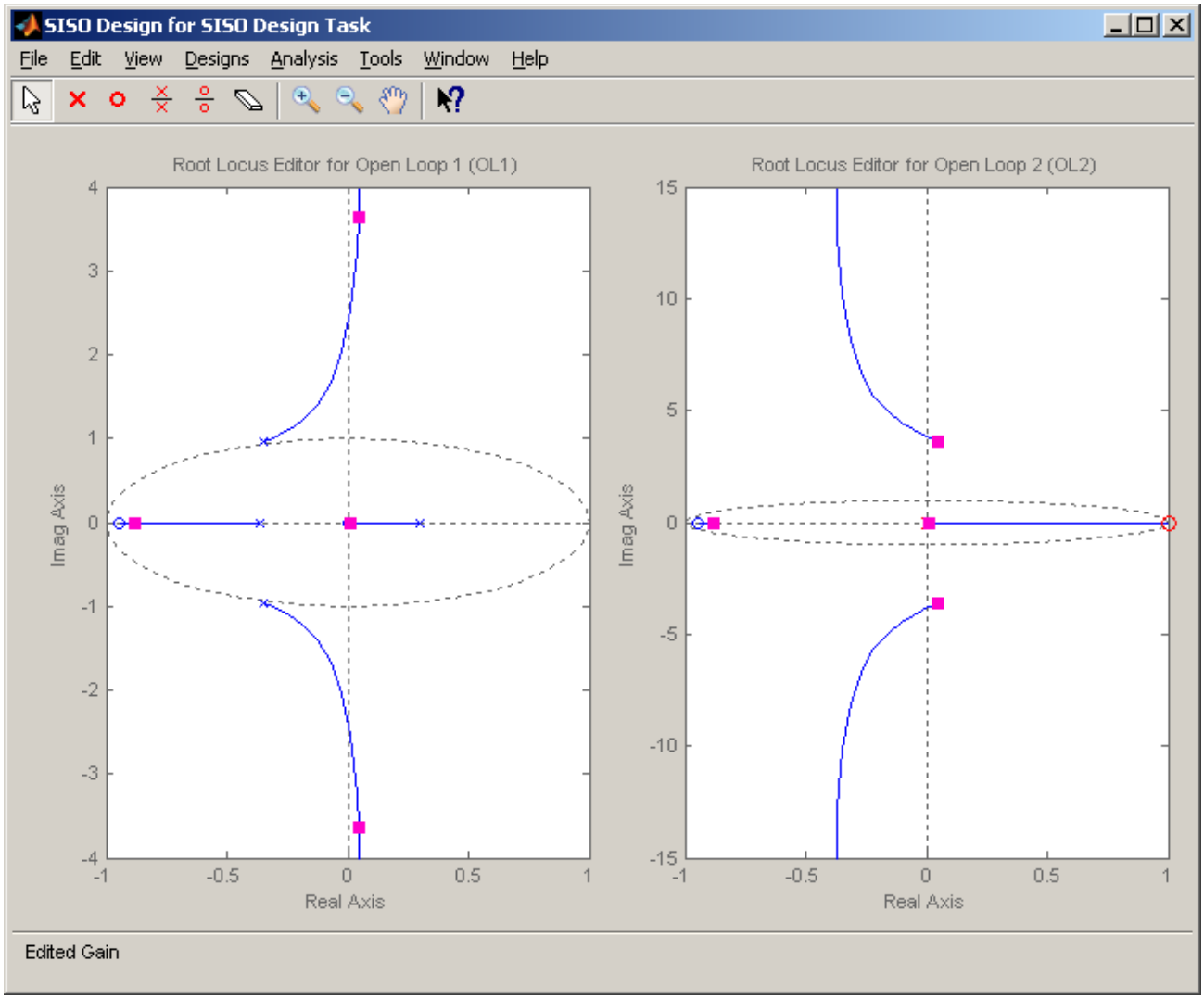
For practice, let's assume our plant is $G_p(z) = \dfrac{12.13z + 11.48}{z^3 + 0.7691z^2 + 0.8668z}$ where we have used a sampling interval of 0.1 seconds. To enter this into Matlab we need to be sure to include the sampling interval so Matlab knows this is a discrete-time system, so type

```
Gp = tf([12.13 11.48],[1 0.7691 0.8668 0], 0.1)
```

When we import the plant, we should get a design window that looks like the following figure. Each window shows the root locus for the plant and two identical proportional controllers.



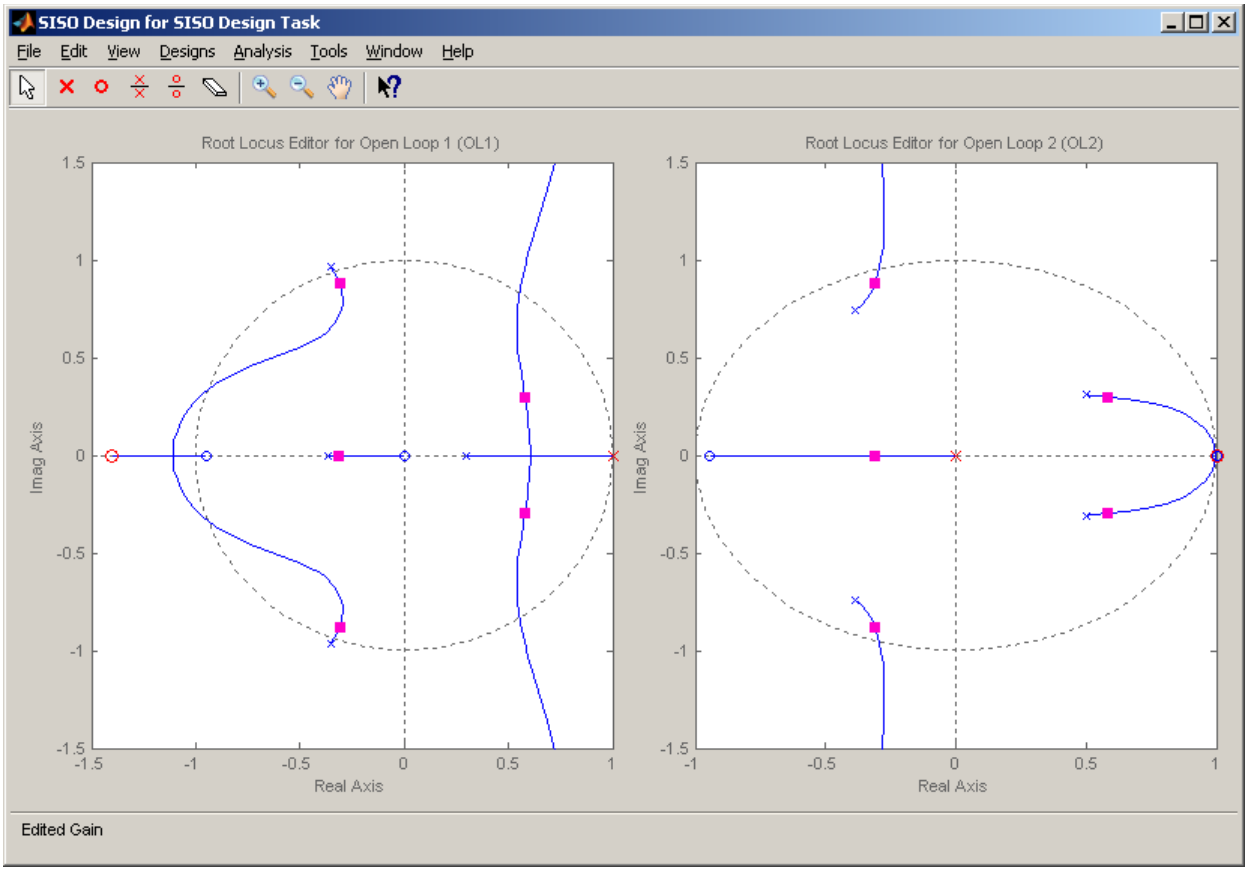
PI-D Controller: Assume we want to use a PI-D controller. In this case, for simulation purposes, we have $C_2(z) = K_d(1 - z^{-1}) = \dfrac{K_d(z-1)}{z}$. We will start off assuming $K_d = 0.01$, but this is just a guess! After entering this into sisotool (and leaving $C_1(z) = 1$, the default), we get the following root locus plot:



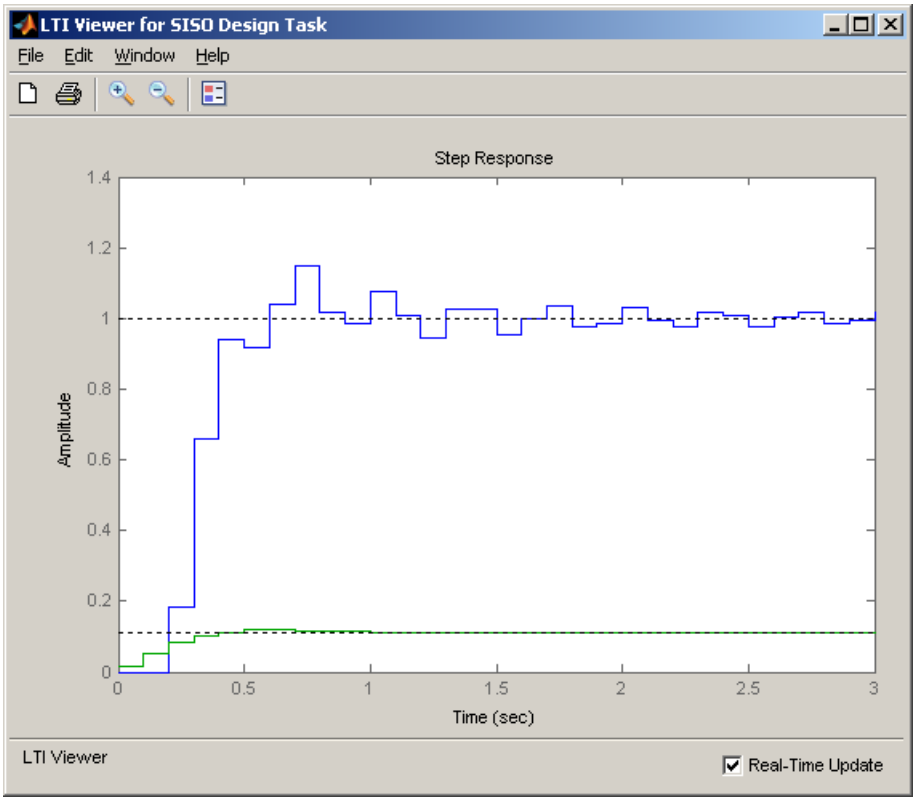
Next we'll enter the PI part of the controller into. As a starting point, let's assume

$$C_1(z) = \frac{0.0152(z+1.4)}{z-1}$$

We will get the following root locus plot (after scaling both axes. To do this, **right click** in each window and select **Properties**)



At this point we can drag the zero around and also drag the red squares around (which change the gain value), as well as the poles and zeros of the controllers (note that we can really only vary the zero, otherwise we will not have a PI-D controller). The step response for this controller configuration is shown below:



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 $C_1(z)$ or $C_2(z)$ to see if we get acceptable performance.

If we assume these controllers are OK, then for our systems we will enter

$$C_2(z) = \frac{0.01(z-1)}{z}, \quad C_1(z) = -0.02128 + \frac{0.03648z}{z-1}$$

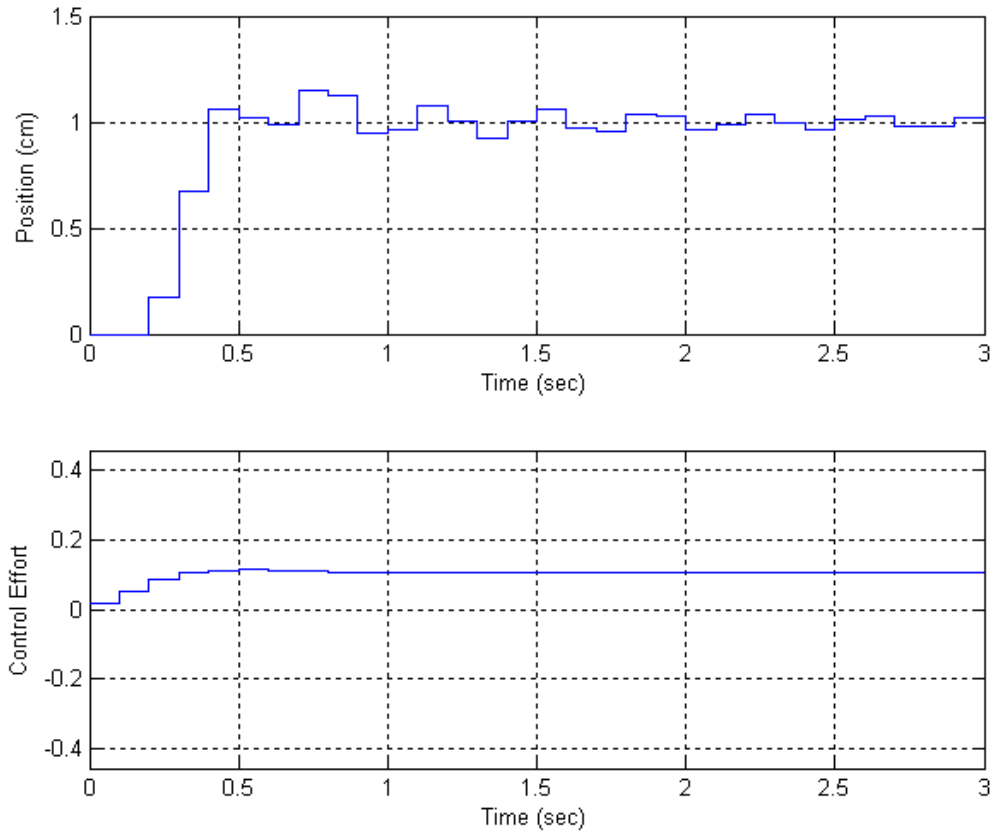
where we have determined

$$K_p = -0.02128 \quad K_i = 0.03648 \quad K_d = 0.01$$

At this point we have the choice of either converting to a PI and D controller, or using $C_1(z)$ and $C_2(z)$ directly. The second choice is easer to do from *sisotool*, but is not as used in practice. To use these transfer functions in Matlab you would type something like the following:

```
C2 = 0.01*tf([1 -1],[1  0],Ts);
C1 = 0.0152*tf([1 1.4],[1 -1],Ts);
[num_C2,den_C2] = tfdata(C2,'v');  % extract the numerator and denominator of C2
[num_C1,den_C1] = tfdata(C1,'v');  % extract the numerator and denominator of C1
```

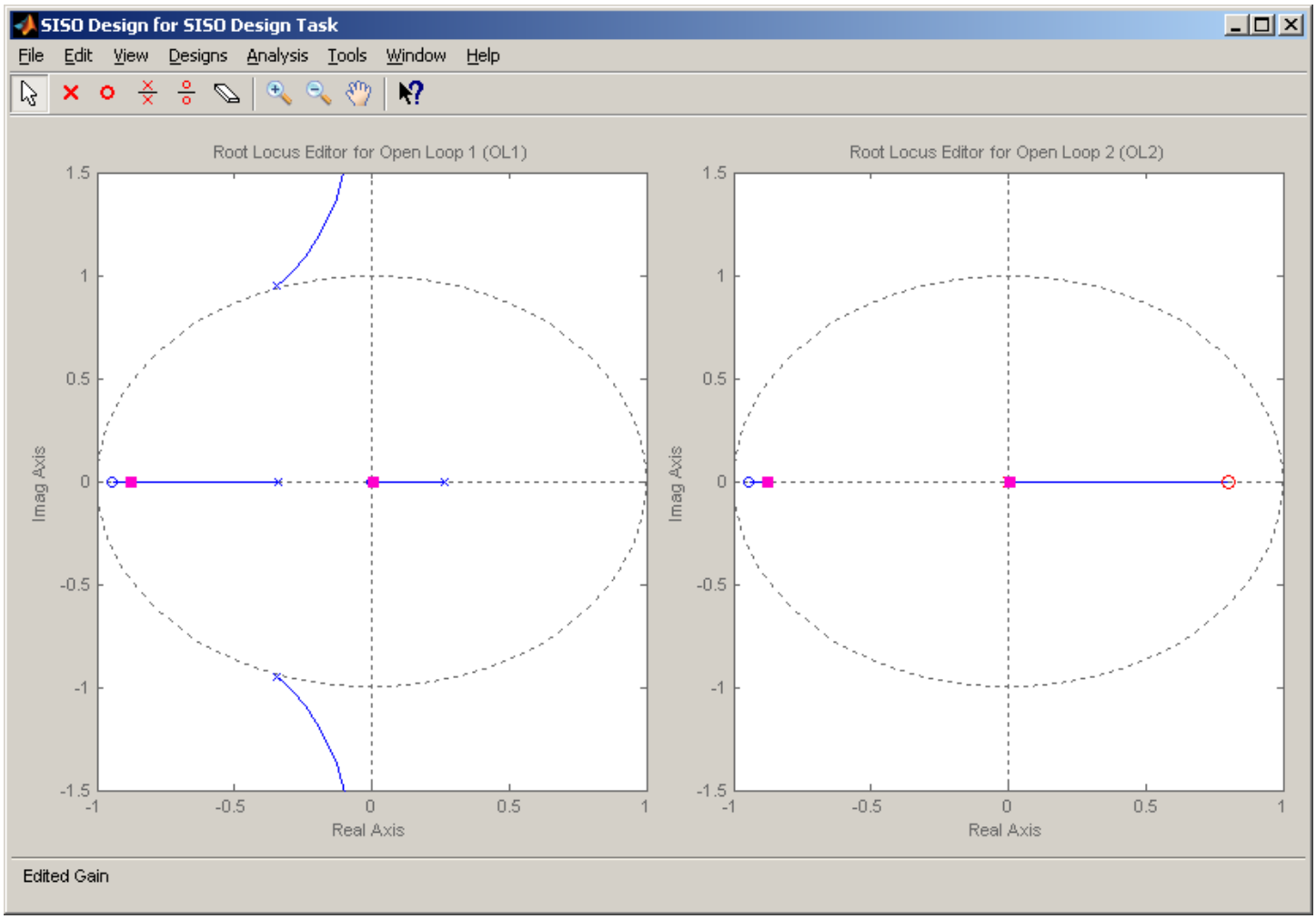
Finally, we should simulate the controller in Matlab/Simulink to be sure it matches the results from *sistool* reasonably well before we go on to implement the controller on the ECP system. The Matlab/Simulink results are shown below:



I-PD Controller: 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. As a start, we'll try the following PD controller

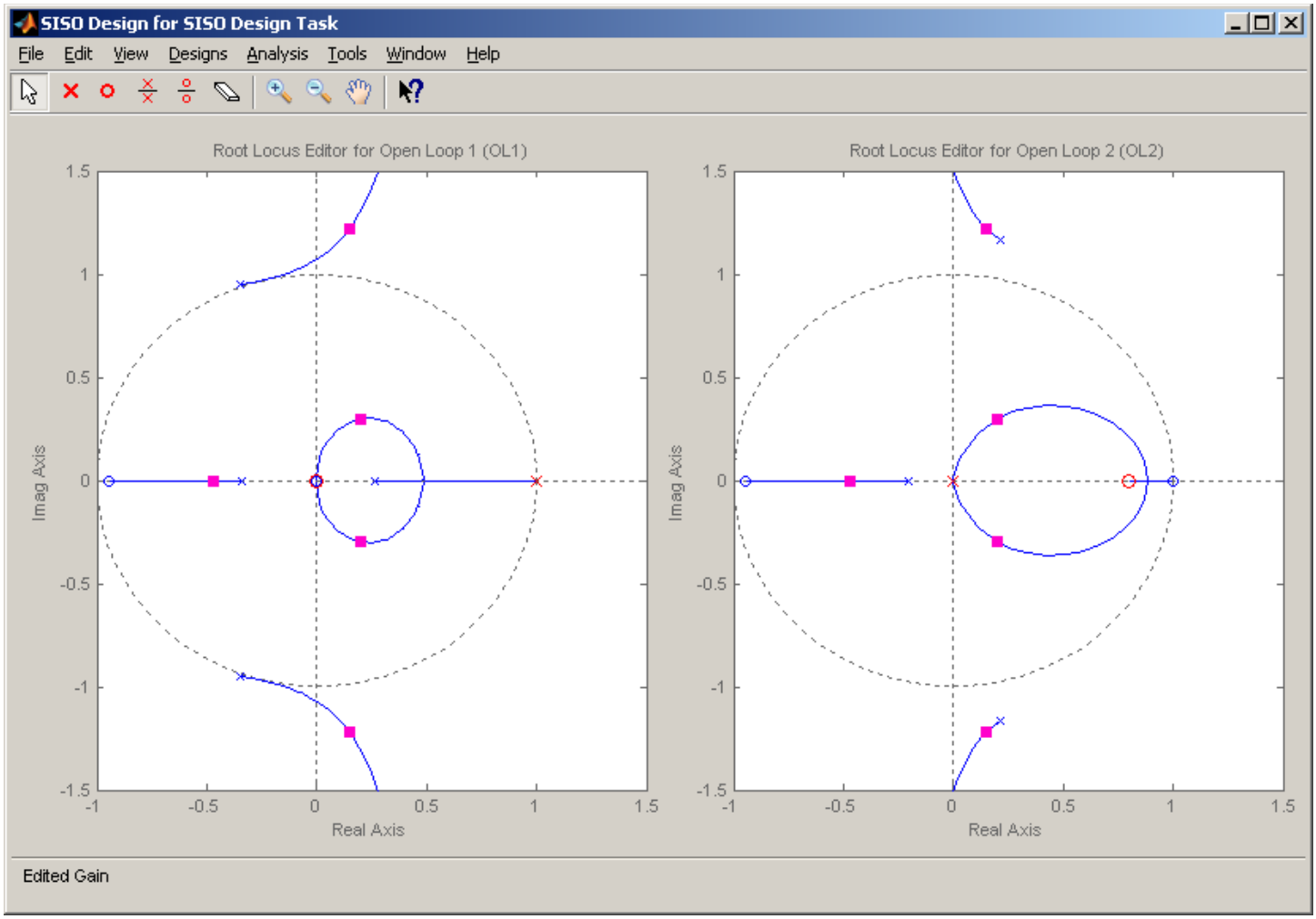
$$C_2(z) = K_p + K_d(1 - z^{-1}) = \frac{(K_p + K_d)z - K_d}{z} = \frac{0.01(z - 0.8)}{z}$$

The root locus plot at this point ($C_1(z) = 1$) is shown below



Next we have to try an I controller for $C(z)$. Let's assume $C_1(z) = \frac{K_i}{1 - z^{-1}} = \frac{K_i z}{z - 1} = \frac{0.1z}{z - 1}$.

Note that the only thing we can change here is the gain, the pole at 1 and zero at the origin are fixed! After entering this into sisotool, we have the following root locus plot.



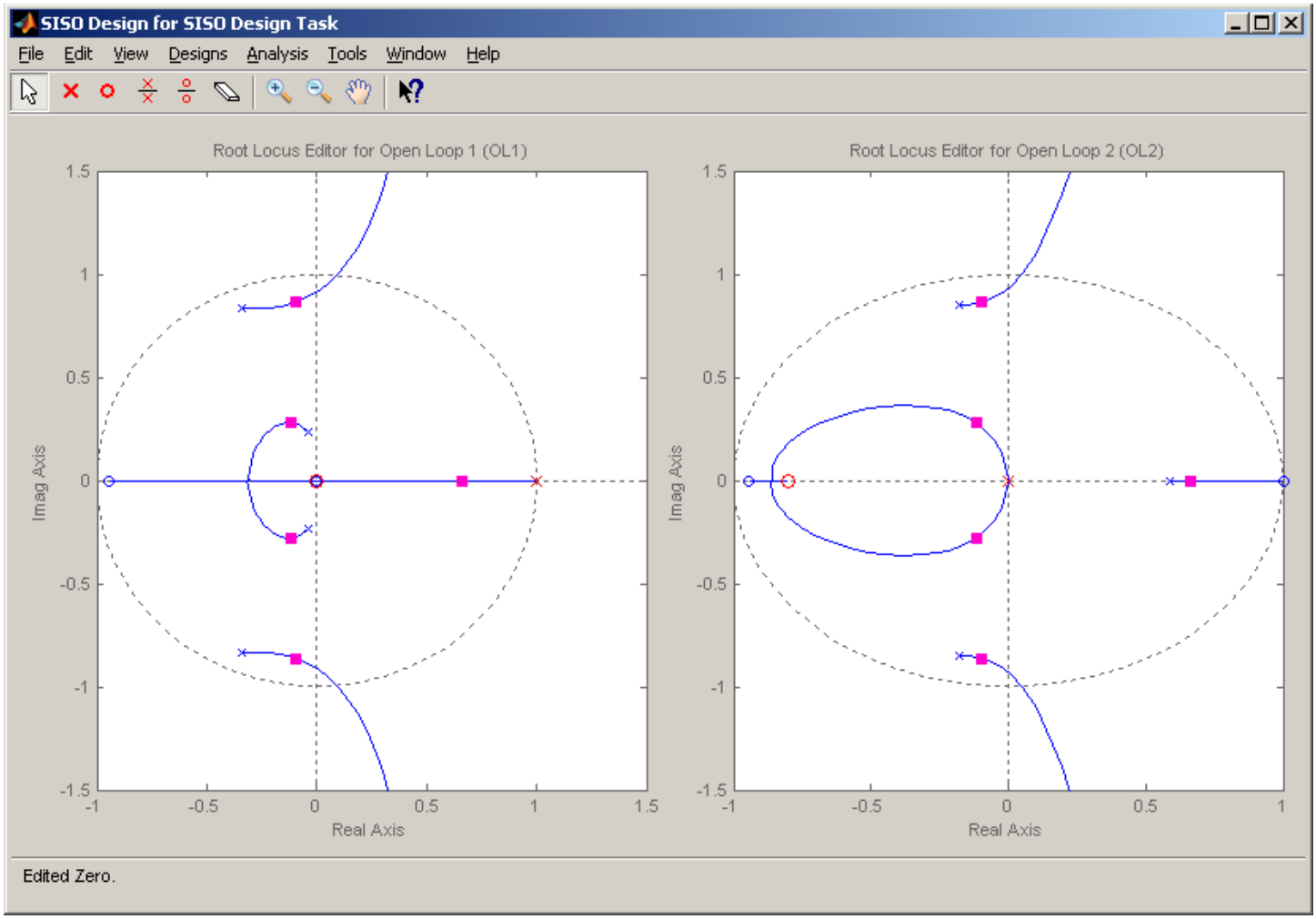
This is clearly an unstable system, so we need to change something! With the $C_1(z)$ we can only change the gain, which will not help us here (we have two poles outside the unit circle with no way to get in). Clearly we must change $C_2(z)$. After some playing around, I ended up with

$$C_2(z) = \frac{0.01(z+0.8)}{z} \quad , \; C_1(z) = \frac{0.0375z}{z-1}$$

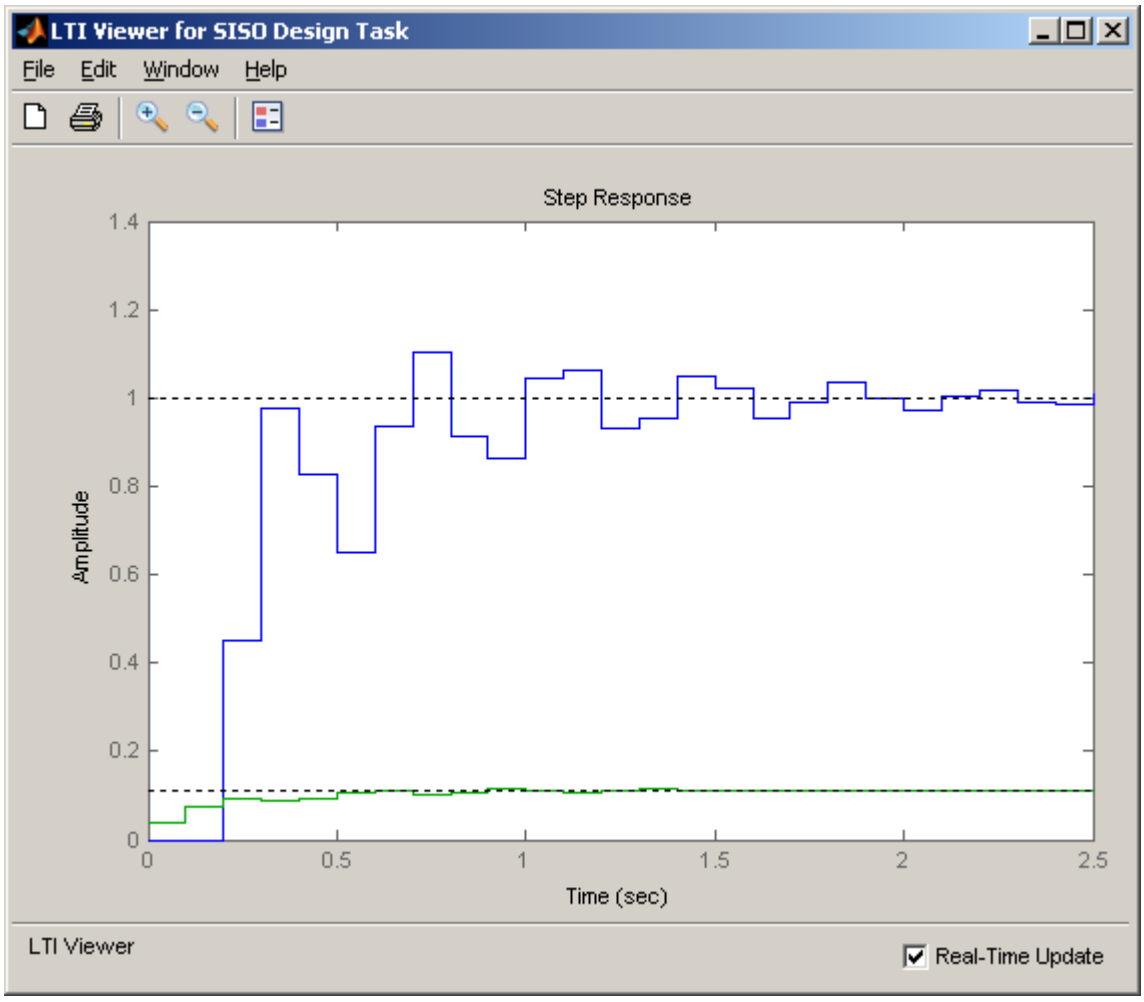
Here we have

$$K_p = 0.009 \quad K_i = 0.0375 \quad K_d = -0.008$$

The corresponding root locus plot is as follows:


SISO Design for SISO Design Task
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open Loop 1 (OL1)
Root Locus Editor for Open Loop 2 (OL2)
Imag Axis
Real Axis
1.5
1
0.5
0
-0.5
-1
-1.5
Edited Zero.

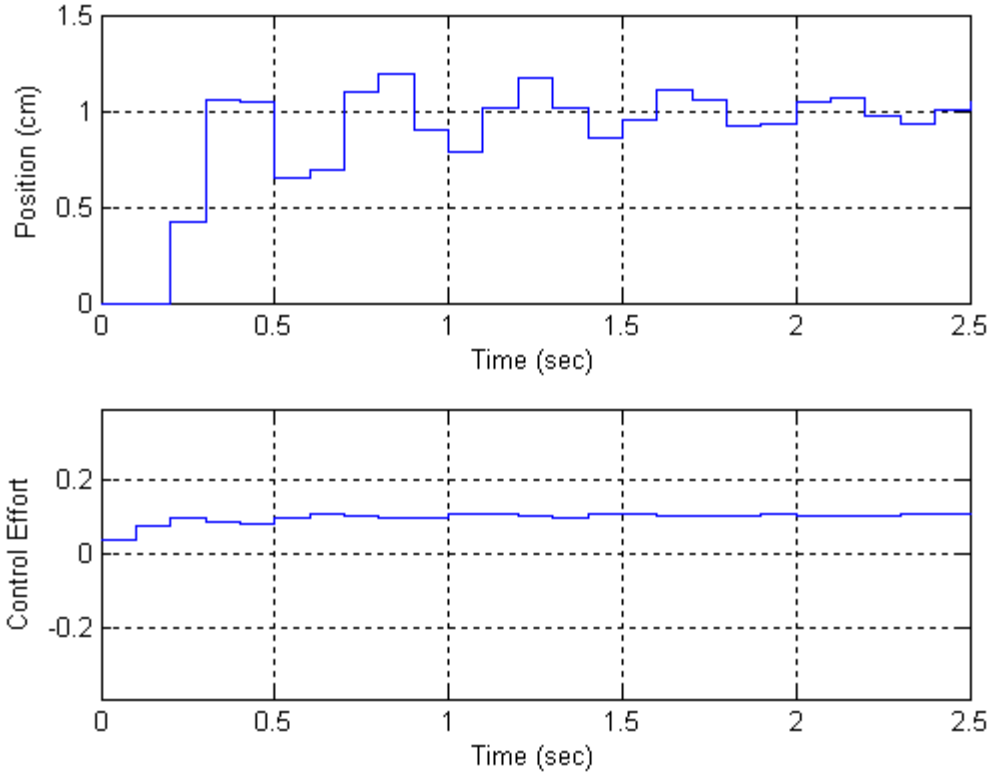
The step response for this system is then:



If we assume these controllers are OK, then for our systems we will enter

$$C_2(z) = \frac{0.01(z+0.8)}{z} \quad , \; C_1(z) = \frac{0.0375z}{z-1}$$

into our Matlab/Simulink model. Finally, we should simulate the controller in Matlab/Simulink to be sure it matches the results from *sistool* reasonably well before we go on to implement the controller on the ECP system. The Matlab/Simulink model produces the results on the following page:



These results are close enough to continue.

Step 1: Copy, rename, and modify **DT_PID_Driver.m** to read in the correct model file and implement the new structure. In particular, it must now determine $C_1(z)$ and $C_2(z)$.

Step 2: Copy, rename, and modify **DT_PID.mdl** to implement the new structure in Simulink.

For each of your two 1 dof systems, you will need to go through the following steps:

Step 3: Set up the 1 dof system exactly the way it was when you determined its model parameters.

Step 4: Modify your …**driver.m** file to read in the correct model file. You may have to copy this model file to the current folder.

Step 5: Modify your **…driver.m** to use the correct *saturation_level* for the system you are using.

Step 6: *Set the sampling interval to 0.1 seconds (the first time) and then 0.05 seconds*

Step 7: Modify the ECP driver files and rename appropriately. It is best to copy your previous files, then rename them, and then edit.

Step 8: 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.

- Simulate the system for 3.0 seconds. *Be sure to use radians for the Model 205 system!* 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 system.

- Use the **Compare_DT1.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. The results for the torsional systems must be displayed in degrees. You need to include this graph in your memo.

Step 9: 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). Be sure to observe the limits on the other gains.

- Simulate the system for 3.0 seconds. *Be sure to use radians for the Model 205 system!* 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 system.

- Use the **Compare_DT1.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. The results for the torsional systems must be displayed in degrees. You need to include this graph in your memo.

Your memo should include 8 graphs (2 systems, 2 PI-D controllers/system, 2 I-PD controllers/system – remember you need to run the systems using two different sampling intervals …)