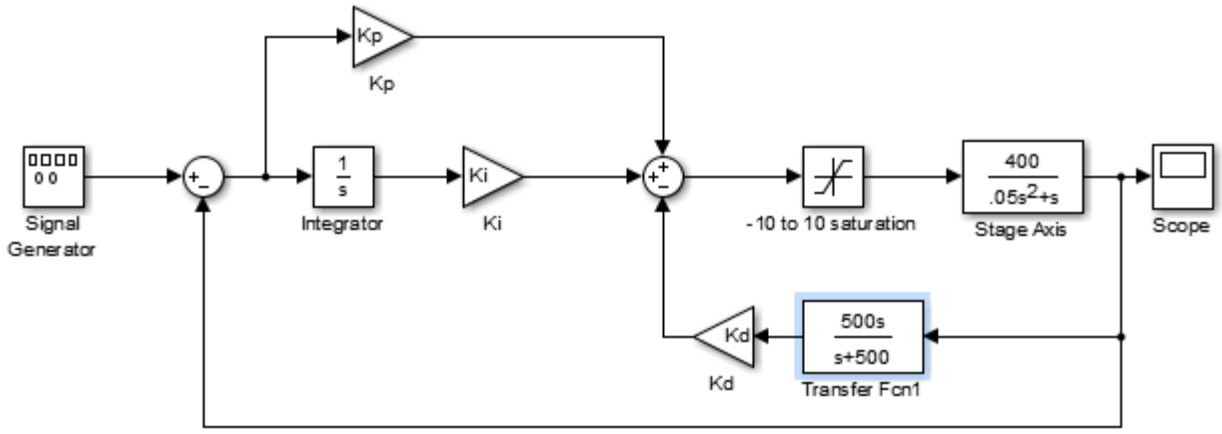
Lab 7: PID Control with Trajectory Following

Introduction

In Labs 6 you identified an approximate transfer function for both the X and Y linear drives of the XY stage. In this lab you are going to use the transfer functions you identified in Lab 6 to design PID controllers for each axis.
As you have probably figured out this semester, control texts focus quite a bit on "step responses" of systems both open loop and closed loop. The reason for this is step responses can tell us a lot about a system. Percent overshoot, damped oscillations and steady state value is enough to identify low order systems. Even in this lab, part of the design will be to command the X and Y axes with step inputs in order to fine tune the Kp, Kd and Ki gains. The final control you will implement will not use large step reference inputs. Instead you will be commanding the stage axes to follow specific trajectories. This way if a pen was fixed to the stage it could draw a circle or a pattern of an eight or any other path you would like to choose. This is, much of the time, how control systems are commanded outside of academics. Having a system repeatedly step between large position changes is probably not the best for the system when it comes to wear and tear on the system over time. Instead of commanding a system with a step, the system is commanded with a smoothly transitioning trajectory. Even if a system needs to repeatedly move back and forth between positions, a smooth trajectory can be created that, for example, ramps to the reference value instead of immediately jumping to the new reference value. In this lab you are going to try out a few trajectories and see how well the XY stage tracks the trajectory.

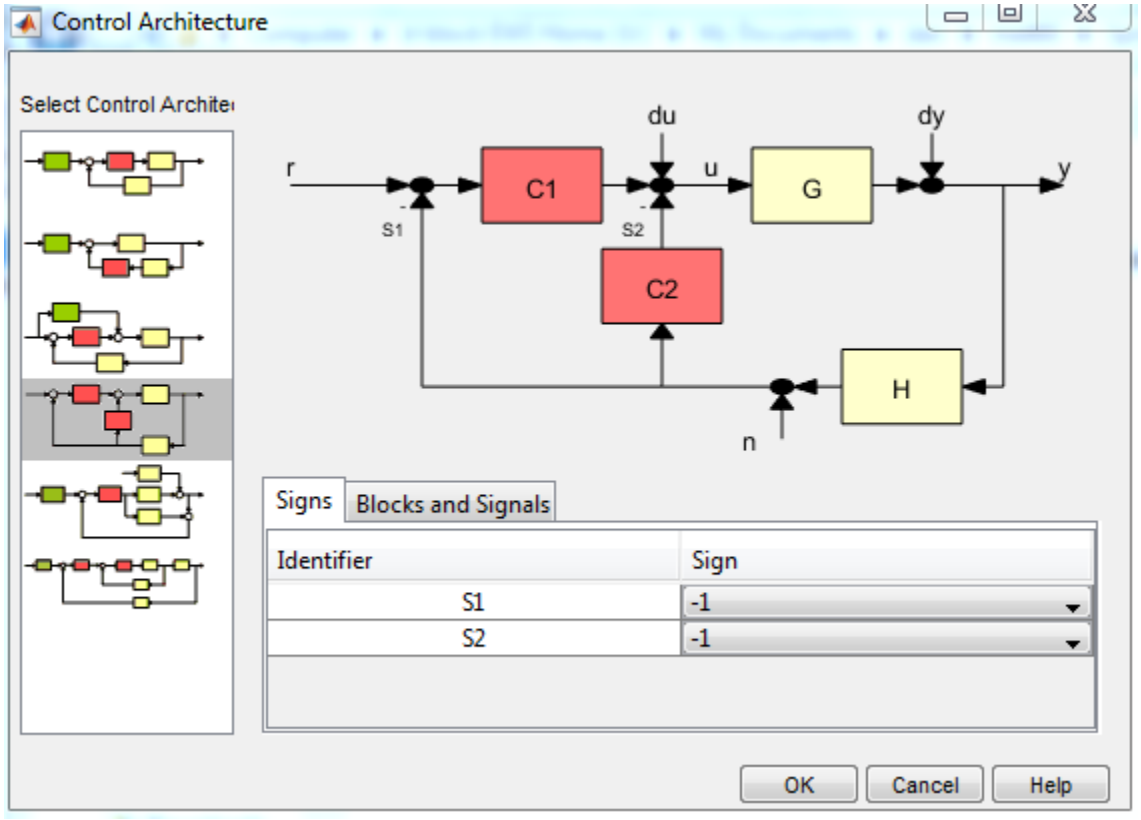
PID Controller Design
This being the last lab of the semester, the lab write up does not necessarily give you all the steps needed to design and implement the specified controller. The write up does guide you in how to do the design but feel free to add any techniques you have learned throughout the semester.
First step in this lab is to design the PID (proportional-plus-integral-plus-derivative) controllers for the X and Y stages. As in lab four, use root locus techniques and "rltool" in Matlab to perform your design. As stated in the introduction, initially for this design of the PID controller you will again monitor step responses of the system in order to easily tune the Kp, Ki and Kd gains of the controller. Once the controllers have been designed and tuned, you will switch to following trajectories.

1. Using the X stage's identified transfer function and "rltool", design a PID controller that gives a closed loop step response with around 20% overshoot and rise time approximately 0.025 seconds. The higher percent overshoot for this design will disappear when smooth trajectories are commanded to the system instead of step inputs. The type of PID controller you will be implementing is slightly different from the classical PID controller. A better way to name this type of controller would be PI with velocity feedback control. The below block diagram shows the implementation. The Kp and Ki gains act on the error signal where the Kd gain only is multiplied by the velocity of the stage. Think about and talk to your TA why this kind of controller makes sense especially when the system is driven with step inputs.

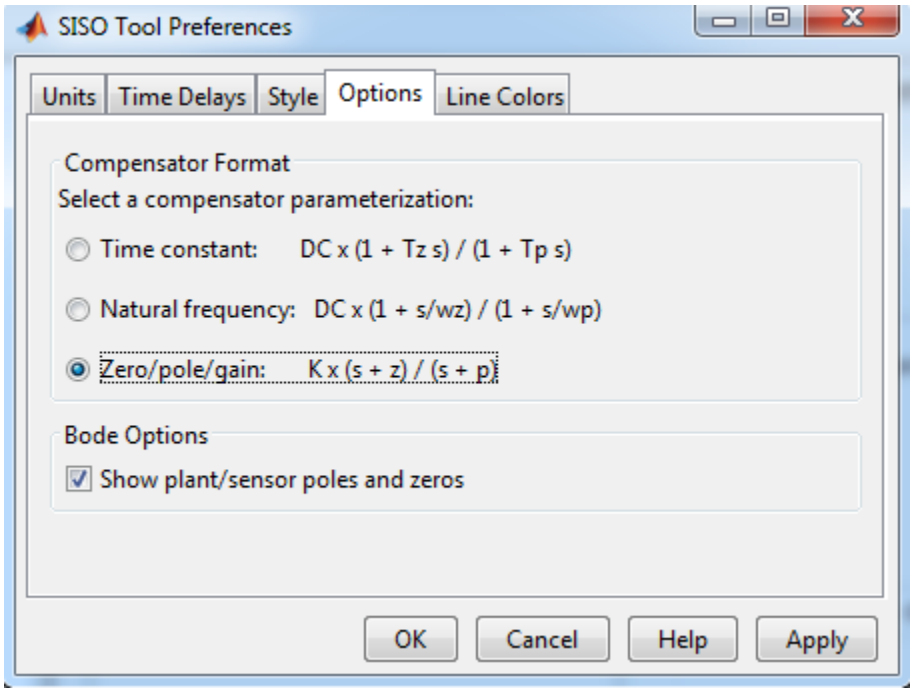


The $\frac{500s}{s+500}$ transfer function is estimating the derivative. (The true derivative "s" is impossible to implement.) The 500 is a tunable gain causing the derivative approximation to perform differently. You should not have to change this 500 value because it has already been tested to work well with the XY stage sampling the feedback every 1 millisecond. You will however want to tune the Kd gain.

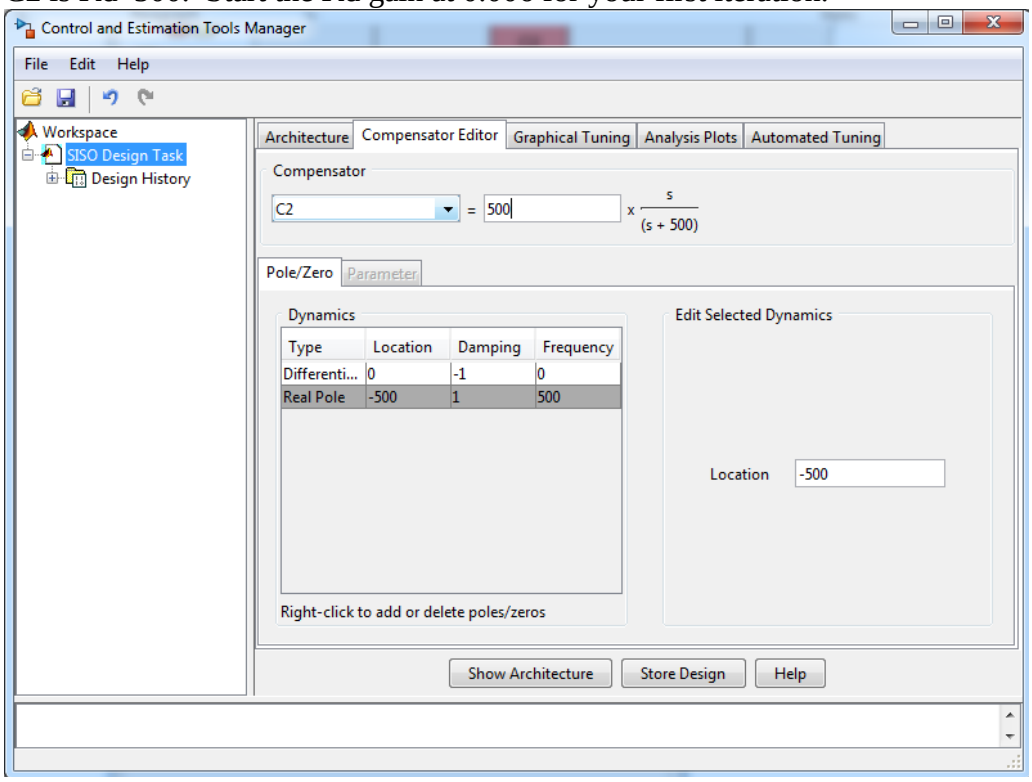
To design this controller in rltool you will need to use the control architecture pictured below. C1 is the error path part of your controller $Kp + \frac{Ki}{s}$. C2 is in the feedback path and its transfer function is $\frac{Kd*500s}{s+500}$.



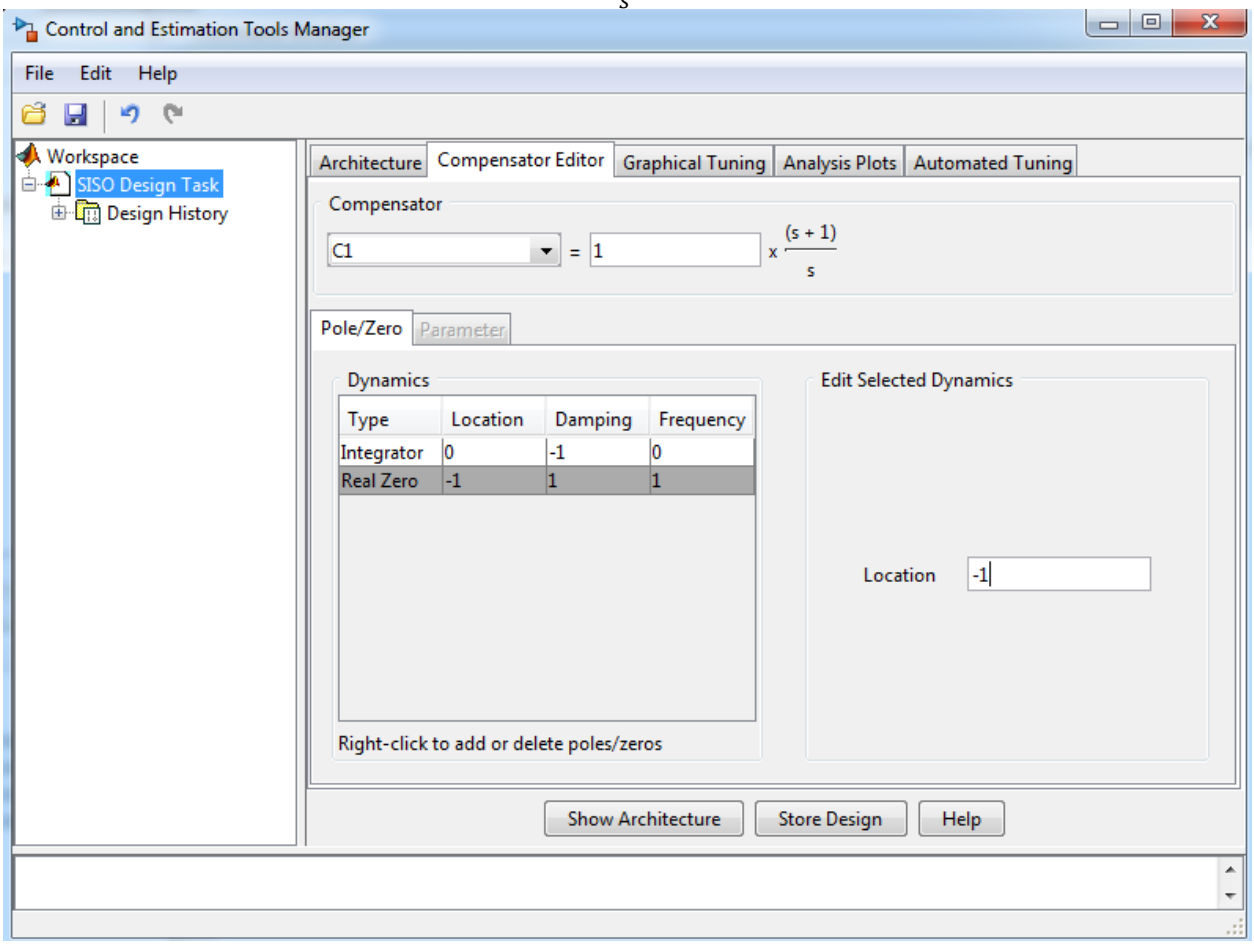
Also change the preferences of rltool to display the controllers in Zero/pole/gain format.



C2 is straight forward to setup by assigning it a real pole at -500 and real zero at zero. Then the gain of C2 is Kd*500. Start the Kd gain at 0.006 for your first iteration.



For the error path what are the poles and zeros of C1 that has the transfer function $Kp + \frac{Ki}{s}$. Show that this transfer function can be rewritten as $Kp * \frac{s+\frac{Ki}{Kp}}{s}$. So in C1 add a real pole at zero and a real zero.



With C1 and C2's form set you can now pull up an analysis plot for r to y and tune your controller. The three "knobs" you can tune with this controller are C1's gain Kp and zero Ki/Kp and C2's Kd gain (make sure to keep C1's real pole at zero otherwise you will no longer have integral control). The Kd gain cannot be changed in the root locus plot like C1's gan and zero. Instead you will have to iteratively tune Kd changing it in the compensator editor window. So for example start out with a Kd gain of 0.006. Then go to the rltool design window and adjust C1's gain and real zero and see if you can find a controller that meets the specifications. If the specifications are difficult to achieve, go back to the compensator editor window and change the value of your Kd gain. Then repeat the tuning of C1's gain and real zero attempting to achieve the specifications. Repeat this until you are able to meet the specifications and also not exceed these limits on Kp, Kd and Ki. Kp<5, Kd<0.02 and Ki < 200.

Repeat these steps to find Kp, Kd and Ki gains for both the X stage and Y stage.

PID Controller Implementation in Simulink

With your Kp, Kd and Ki gains tuned, implement the PI plus velocity feedback controllers for both the X and Y stages. Again use the starter file "XYstage_start.mdl" found at N:\Hydraulicslab\ME460. Make sure to save this file with a new name in a folder located on the C:\ drive and change Matlab's current directory to the directory where this new Simulink file is saved.

For the reference for both X and Y, start out with a step back and forth between two positions every .5 seconds or so. Do this first to make sure your controller is performing well. If the actual response is not very close to the design specifications tune, your Kp, Kd and Ki gains to achieve the appropriate response.

Now that you have controllers that meet the specifications, change the step inputs to a circle trajectory:

$$x = rsin(\omega t)$$
$$y = rcos(\omega t)$$

A simple way to create these trajectories is to use the "Clock" source block to generate time in seconds and pass time to two "Fcn" (function) blocks, found in Simulink's "User Defined Functions" library. Use these two "Fcn" blocks to generate both the x and y trajectory. If you have never used the "Fcn" block before ask your TA for guidance.

Run your controller and see how well the X Y stage traces out a circle. Using scope blocks and their "Save Data to Workspace" feature save both the desired trajectories x and y points along with the actual x and y positions the stage traversed. In a single Matlab plot plot both the circle trajectory and the actual movement of the stage. How well did the stage follow the circle? If you increase the speed of the trajectory by changing ω, is the X Y stage able to follow well? Of course there will be a limit where the XY stage can no longer keep up.

Finally come up with two more trajectories for the stage to follow. You can come up with your own or below are the trajectory of a straight line and a figure eight. For each produce an X Y plot of the actual response on top of the desired trajectory.

Figure Eight:
http://gamedev.stackexchange.com/questions/43691/how-can-i-move-an-object-in-an-infinity-or-figure-8-trajectory

$$scale = \frac{2 * r}{3 - \cos(2\omega t)}$$
$$x = scale * \cos(\omega t)$$
$$y = scale * \frac{\sin(2\omega t)}{2}$$

Straight Line:

$$Straight\ Line\ from\ (x_a, y_a)\ \ to\ \ (x_b, y_b)$$

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \begin{bmatrix}x_b - x_a\\ y_b - y_a\end{bmatrix}, vector\ direction\ to\ travel$$

$$\mathrm{t_{total}} = \frac{distance\ between\ a\ and\ b}{desired\ speed\ along\ the\ line}$$

$$\begin{bmatrix}x^d\\ y^d\end{bmatrix} = \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} * \frac{t}{t_{total}} + \begin{bmatrix}x_a\\ y_a\end{bmatrix}.$$

To make this work nicely you need to make time count up to t_{total} and back down to zero repeatedly. To do this find the Matlab Function block in the User Defined Functions Simulink library. Then double click the block and add the follow m-code:

```
function y = fcn(u)

repeattime = 10;
if mod(floor(u/repeattime),2) == 0
    y = mod(u,repeattime);
else
    y = -1*mod(u,repeattime)+repeattime;
end
```

Then use the block as pictured: