ECE320 Lab 3: PID and PD Controllers

Overview

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

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 six graphs for each of the 1 dof systems you used (two different PID controllers and one PD controllers with and without dynamic prefilters.) Be sure to include the values of k_p, k_i, and k_d and whether the PID controller had real zeros or complex conjugate zeros 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?

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

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

Step 2: Modify **closedloop_driver.m** to read in the correct model file. You may have to copy this model file to the current folder.

Step 3: Modify **closedloop_driver.m** to use the correct *saturation_level* for the system you are using.

Step 4: Modify closedloop_driver.m for a PID controller. To do this, comment out all of the other controllers, and add the lines

```
kp = 0.2;      % just a dummy value
ki = 0.02;     % and even dummer value
kd = 0.002;  % way stupid value

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

Note that we have modified the derivative controller so that it is in series with a one pole lowpass filter with pole at 50 rad/sec (about 8 Hz). This will help smooth out the derivatives.

Step 5: PID Control (complex conjugate zeros)

- Design a PID controller with complex conjugate zeros using *sisotool* to meet the design requirements. Do not try to include the lowpass filter, just use a pole at the origin and two complex conjugate zeros. Use a **constant prefilter** (i.e., a number, most likely the number 1)

- Implement your controller in **closedloop.mdl**

- Simulate the system for 1.5 seconds. Your step response should be similar to that you obtained using *sisotool*. 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 ECP system.

- 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 in **closedloop.mdl** to cancel the zeros of the closed loop system and still have a steady state error of zero. Since the PID controller makes the system a type 1 system, we don't need a prefilter to have a steady state error of zero. However, sometimes we can use the prefilter to make the transient response a bit nicer, or reduce the control effort. However, this is done at the expense of a block outside the control loop, which may be bad. Never the less, we continue anyway... We also $G_{pf}(0) = 1$. Hence we have

$$G_{pf}(s) = \frac{D_o(0)}{N_o(s)}$$

  For us, we set the prefilter Gpf(s) to **den_Go(end)/num_Go**. (This should all be done in Matlab! Comment out your old code and add this new code.) This will cancel out the zeros of the closed loop system. Your numerator polynomial for Go(s), which is denoted as $N_o(s)$, should be second order. If it is not, be sure you have not removed the lines

```
num_Gp = (abs(num_Gp) > tol*ones(1,length(num_Gp))).*num_Gp;
den_Gp = (abs(den_Gp) > tol*ones(1,length(den_Gp))).*den_Gp;
```

- 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 5: PID Control (real zeros)

- Design a PID controller with real zeros using sisotool to meet the design specs (you may have already done this in the homework). Use a **constant prefilter** (i.e., a number, most likely the

number 1)

- Implement the correct gains into **closedloop.mdl**

- 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 system.

- 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 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.*

- 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: PD Control

- Design a PD controller with real zeros using sisotool to meet the design specs. (It may be difficult to meet the PO constraint, do the best you can.) Use a **constant prefilter** (i.e., a number). However, the number will probably not be 1! You will probably need to use *sisotool* to get reasonably close, but then use the prefilter to get the steady state error correct.

- Implement the correct gains into **closedloop.mdl**

- 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. The results for the torsional systems must be displayed in degrees. 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.*