ECE-320 Linear Control Systems
Laboratory 5

System Modelling, Controller Design, and the Real World

Preview In this Lab you will first obtain a second order model of your spring/mass/damper system, design a controller for it, implement the controller on the real system, and then compare the predicted response with the actual response. You will probably make at least two observations during this lab:

- models are not perfect, they are just guides
- real motors have real limits, which can make designing more difficult

For each of the controller types (with the exception of the P controller), you should attempt to

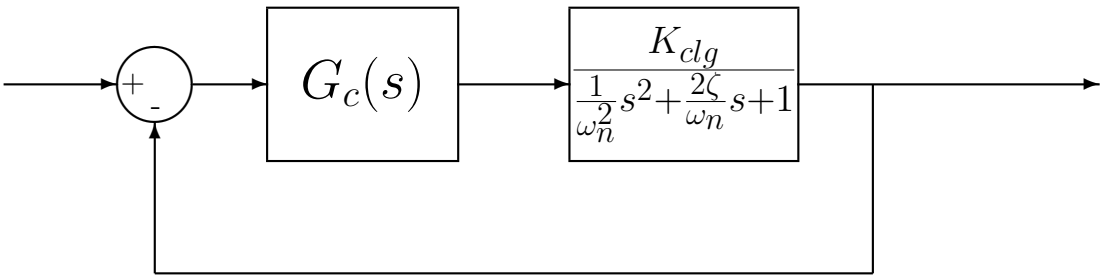
- produce a position error of less than 0.15
- reach steady state within 5 seconds (the faster the better)
- have as little overshoot as you can manage

I represent management, which means I expect things which may not be possible. If you cannot meet these vague criteria, do the best you can with each controller type to meet the criteria.

Pre-Lab

1) Print out this lab and **read** it.

In what follows, assume we have the unity feedback system shown below. $G_c(s)$ is the controller that you will be implementing, ω_n and ζ are estimates obtained using either time-domain or frequency domain methods, and k_{clg} is the system closed loop gain. Note that we are modelling the motor as contributing only a gain to the system, and we are lumping both the plant's gain and the motor gain together into one parameter.



The standard form for a PID controller is

$$G_c(s) = k_p + k_i\frac{1}{s} + k_d s$$

show that

a) for a PID controller in the form $G_c(s) = k(s+z_1)(s+z_2)/s,\ k_p = k(z_1+z_2),\ k_i = kz_1z_2,\ k_d = k$

b) for a PI controller in the form $G_c(s) = k(s+z)/s,\ k_p = k,\ k_i = kz,\ k_d = 0$

c) for a PD controller in the form $G_c(s) = k(s+z),\ k_p = kz,\ k_i = 0,\ k_d = k$

We generally need to keep the k_p, k_i and k_d less than about 0.1, though this really depends on your system. (Sometimes I have used a k_i of about 9) You need to use the above relationships to map from poles, zeros, and gains to k_p, k_i, and k_d.

Before we design any controller we need to first **identify** the system. Do not use a system you have used before!

1. Estimate an initial second order system model using time domain analysis (using either the **log_dec** or **fit** programs).
2. Measure the frequency response (make one measurement at 1Hz, 2Hz, ..., 7 Hz, and at least 4 points near the resonant peak)
3. Use the **process_data** and **fit_bode** commands to estimate the gain of the system.
4. Use the **opt_fit_bode** command to fine tune the system model.
5. Determine the closed loop system gain K_{clg}

Estimating the Closed Loop Gain K_{clg}

0. Set the units

Click **Setup** → **User Units** and set the units to **cm**.

1. Setting up the controller

Click **Setup** → **Control Algorithm**. Be sure the system is set for *Continuous Time*. Select **PID** under **Control Algorithm**. Click on **Setup Algorithm**. Be sure **Feedback** is from **Encoder 1**. Set k_p to a small number (less than or equal to 0.05) and be sure $k_d = 0$ and $k_i = 0$. Then click **OK**. Next Click **Implement Algorithm**. The click **OK**.

2. Setting up the closed loop trajectory

Click **Command** → **Trajectory**. Select **Step** and click on **Setup**. Select **Closed Loop Step** and set **Step Size** to 1 to 2 cm. Be sure to record this step size (we'll refer to the amplitude as *Amp* below). Set the **Dwell Time** to something like 2000 ms, this is the time the system will be recording data. Finally click **OK**, then **OK** and you should be back to the main menu.

3. Executing the closed loop step

Click **Command Execute**. A menu box will come up with a number of options, and a big green **Run** button. Click on the **Run** button. When the system has finished collecting data, a box will appear indicating the how many sample points of data have been collected. (If you have hit a stop, the system stops recording data. This usually means you're input amplitude was too large or k_p was too large.) Click on **OK** to get back to the main menu.

4. Determining the steady state value

Click **Plotting** → **Setup Plot**, or just **Plotting Data** → **Plot Data**. Look at the steady state value (y_{ss}). You may need to change the dwell time if your system has not reached steady state.

5. Estimating the closed loop gain

Estimate the closed loop gain K_{clg} using the formula: $K_{clg} = \frac{y_{ss}}{k_p} \frac{1}{Amp - y_{ss}}$. You need to go through this procedure at least three times for each configuration. You must use at least two different values of k_p and two different values of input amplitude *Amp*. If none of the steady state values is larger than 0.4 cm, increase either k_p or *Amp*. Average the three results to get your K_{clg} (they should be similar).

6. Estimating the transfer function

The final plant transfer function we will use is then

$$G_p(s) = \frac{K_{clg}}{\frac{1}{\omega_n^2}s^2 + \frac{2\zeta}{\omega_n}s + 1}$$

This is the transfer function to be used in both Mathlab's *sisotool* and in the **closed_loop_response** program.

P Controller First

1. Choose one of the values for k_p that you used to get the closed loop gain, K_{clg}. This type of controller will not work very well for this system, so we are just going to use it to go through all of the steps you will need for the other controllers.
2. Choose the amplitude of the input that goes with this value of k_p.
3. Implement this gain (k_p) and the amplitude and run the ECP system.
4. You may need to let the real system run for 5 seconds or more to reach steady state.
5. Export the step response of the real system and edit the data so Matlab can read it.
6. Use the program **closed_loop_response** to plot both the predicted closed loop response (based on the system model) and the true closed loop response (the real system). The arguments to this program are
   - Amp = the amplitude of the step (whatever you had the real system do, usually 0.5 or 1 cm)
   - Gp = the plant transfer function
   - Gc = the controller transfer function
   - final_time = the final time on the plot. Use this to try and make nice plots (zoom in on where things are happening). For example, if your system reaches steady state after 0.5 seconds, only plot out to about 1 second or so, do not plot out for 5 seconds.
   - filename = the name of the (edited) file that has the measured step response of the real system. This should be in single quotes.

You should get a figure with the predicted response of the system and the actual response of the system. You need to include this plot in your report and a similar plot for each of the controllers you implement.

I, PI, PD, and PID Controllers

For **each** of the following controller types, you should attempt to

- produce a position error of less than 0.15
- reach steady state within 5 seconds (the faster the better)
- have as little overshoot as you can manage

Go though the following steps to design, implement, and compare the predicted response with the measured response for an I, PI, PD and PID controller.

1. Design one controller using the model developed above and Matlab's *sisotool*.
    - Try and keep all k_p and k_d gains less than about 0.1 (it's OK if they are larger, just be prepared for the motor to buzz and the controller to not work) Look at the prelab for the relationships between the controller poles and zeros and k_p, k_i and k_d. If your controller makes the motor buzz and doesn't work, go back to *sisotool* and design another one. Do not implement a controller on the system unless you have simulated it.
    - Be aware that you enter the control parameters in the order k_p, k_d, then k_i
    - Be sure to zero out the gains you are not using (i.e., if you are using a PI controller, be sure k_d is set to zero)
2. You may need to reset the controller often, such as every time you want to implement a new controller. Click **Utility** → **Reset Controller**. Only do this **before** you have implemented a controller.
3. You may need to rephase the motor. Click **Utility** → **Rephase Motor**
4. Be sure to **Implement** the controller you have designed.
5. Try and track a step with an input amplitude of 1 cm.
6. You may need to let the real system run for 5 seconds or more to reach steady state.
7. Export the step response of the real system and edit the data so Matlab can read it.
8. Use the program **closed_loop_response** to plot both the predicted closed loop response (based on the system model) and the true closed loop response (the real system). The arguments to this program are
    - Amp = the amplitude of the step (whatever you had the real system do, usually 0.5 or 1 cm)
    - Gp = the plant transfer function
    - Gc = the controller transfer function

- final_time = the final time on the plot. Use this to try and make nice plots (zoom in on where things are happening). For example, if your system reaches steady state after 0.5 seconds, only plot out to about 1 second or so, do not plot out for 5 seconds.
- filename = the name of the (edited) file that has the measured step response of the real system. This should be in single quotes.

Memo

Your memo should compare (briefly) the response of the model and the response of the real system for the different types of controllers. You should also indicate if any of the controllers could not be made to work, and why. Finally you should have some description of the configuration of the system you were trying to control.

You should include the following items as attachments. Most of these are figures which should have reasonable captions.

- The time domain estimate of ζ and ω_n (from log-decrement)
- The initial frequency response of the system (the best you can do without changing ζ or ω_n).
- The optimized frequency response of the system.
- The data used to determine the closed loop gain, K_{clg}.
- The predicted and actual response of the system to each of the different controllers.