E105 "Mini Lab": PID Control via Arduino

ENGR 105: Feedback Control Design
Winter Quarter 2013
Due no later than 4:00 pm on Friday, Mar. 15, 2013
Submit in class or in the box outside the door to area of Room 107, Building 550

This mini lab is to be completed in groups of two. Only one submission is needed for the group, and both partners will receive the same score. Each group will need to bring one laptop with a USB port and MATLAB installed to the laboratory. It should be a Windows or Mac OS X computer, since we have not tested on Linux. We expect that this assignment will take 30-45 minutes in the lab and ~30 minutes after.

As soon as possible, sign up for a lab slot on http://doodle.com/ubcibhk29fu3hvuz. If you cannot make any of the available slots, please contact Allison via piazza (by Wednesday, March 6) with a list of the times you and your partner are available (use tag #labtime).

Labs will be conducted in Room 128, Bldg. 660 (the Mechanical Engineering Research Lab [MERL] Building), under supervision of one of the instructors. Up to 5 groups can work in the lab at the same time. You should use the entrance to MERL on Panama Mall, as the entrance on the other side of the building is locked at all times. Since the Panama Mall entrance is also locked after 5 pm, so if your lab time is scheduled for after that, the TA will come to the Panama Mall entrance to let you in at the lab start time (so please arrive on time).

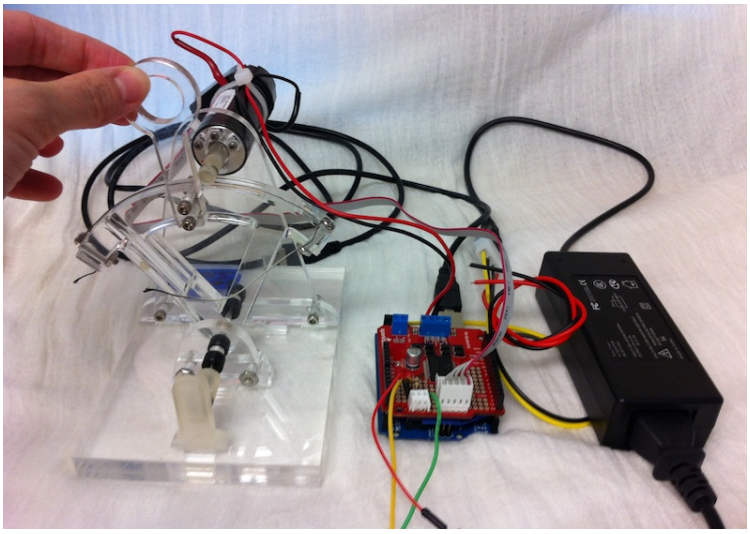
Before you go into the lab

Install the Arduino programming environment on your laptop before going into the lab. Go to http://arduino.cc/en/Main/Software and download and install the appropriate version of Arduino 1.0.3 for your operating system (Windows, Mac OS X, or Linux).

Download the file E105_minilab.zip from the "Homework" section on the course resources web page: https://piazza.com/stanford/winter2013/engr105/resources.

Initial setup and testing of your laboratory equipment

In the lab, you will find a setup like the one shown below.



The equipment consists of:

- Arduino Uno (an inexpensive, open-source microcontroller board, colored blue)
- Ardumoto shield (motor amplifier circuit that is attached to the Arduino, colored red)
- An A-B type USB cable to connect the Arduino to your laptop
- The "haptic paddle", a one-degree-of-freedom back-drivable robotic device equipped with an RE25 Maxon motor and HEDS-5540 optical encoder
- Power supply (12V, 2A) for the motor/Ardumoto

Perform the following steps for initial setup and testing:

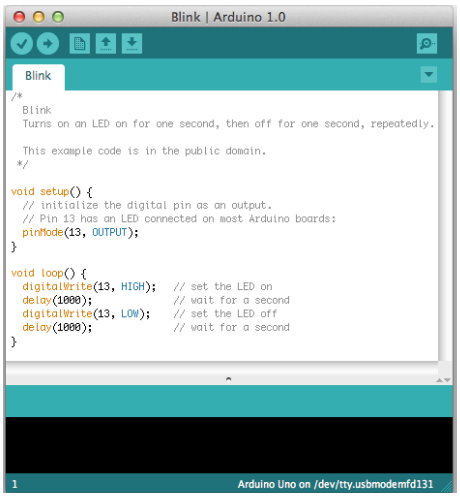
- ❑ Connect the Arduino board to your computer using the USB cable. The green power LED should go on. (You can see it by peeking underneath the Ardumoto Board.)

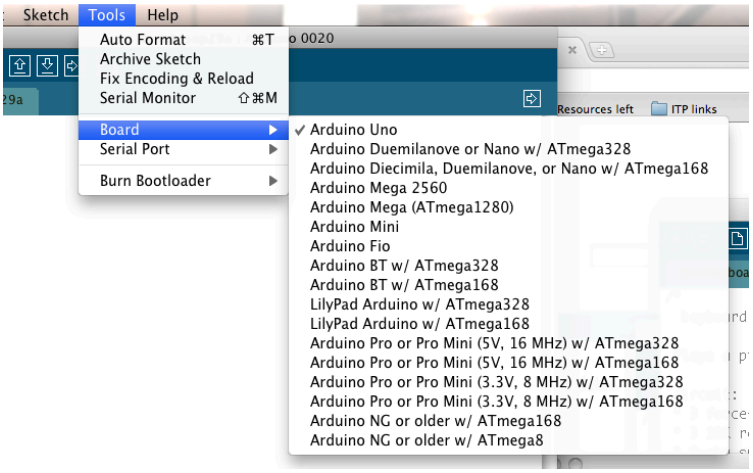
- ❑ If you have a Windows 7, Vista, or XP operating system, you need to install the drivers for the Arduino Uno as follows. (These instructions are from http://arduino.cc/en/Guide/Windows. You may need to do things slightly differently for your computer. If you have a Mac OS X operating system, you do not need to do this step.)
    - Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts.
    - Click on the Start Menu, and open up the Control Panel.
    - While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
    - Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)". [In some cases, we have seen jus "Unknown device" or similar.]
    - Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.
    - Next, choose the "Browse my computer for Driver software" option.
    - Finally, navigate to and select the Uno's driver file, named **ArduinoUNO.inf**, located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
    - Windows will finish up the driver installation from there.

- ❑ Double-click the Arduino application you downloaded earlier. The icon looks like this: 

- ❑ Open the LED blink example sketch: File > Examples > 1.Basics > Blink. It should look something like the image on the left below.

- ❑ Then select your board. You'll need to select the entry in the Tools > Board menu that says "Arduino Uno". It should look something like the image on the right below.





- Select the serial port being used by the Arduino board from the Tools > Serial Port menu. For a Windows computer, this is likely to be COM3 or higher. If multiple ports are shown and you are not sure which port it is, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port. For a Mac OS X computer, this should be something with "/dev/tty.usbmodem" in it.

- Upload the Blink program to the Arduino by clocking the button that looks like this:  . If the upload is successful, the message "Done uploading" will appear in the status bar. A few seconds after the upload finishes, you should see an LED on the board start to blink (in orange, close to the USB plug). Again, you need to peek underneath the Ardumoto to see it. (Please show it to the lab instructor to verify.) If there are no errors, then you are ready to do the lab assignment!

Lab steps

- Unzip (if needed) the **E105_minilab.zip** file and open the resulting E105_minilab folder.

- You will now need to install a few libraries to (1) allow the Arduino to communicate effectively with the optical encoder, which is the position sensor on the motor, (2) allow changes in direction of motor torque using the Ardumoto board, and (3) make the control loop run at a specific frequency. Open a new operating system window and navigate to (and create, if necessary) the Arduino libraries folder. For Windows, this is My Documents\Arduino\libraries. For Mac OS X, this is ~/Documents/Arduino/libraries/.

- Copy the folders called **Encoder**, **pinchangeint**, and **TimerOne** from the E105_minilab folder to the Arduino libraries folder created in the previous step. Quit the Arduino program. Then double-click on the **e105_pid.ino** file in the E105_minilab/e105_pid folder to restart the Arduino program. To check that your libraries are in the correct place, you can select Sketch > Import Library in the Arduino program and see that "Encoder", "pinchangeint", and "TimerOne" are listed.

- Scroll through the **e105_pid.ino** file and look at its structure. You will only need to edit code in item 4, the loop() function, but if you are curious, you can look at all the components of this Arduino program:
  1. The file begins with some preliminary settings, which include library functions and variable definitions. (Don't edit anything here.)
  2. A **setup()** function, which enables communication and activates settings. (Don't edit anything here.)
  3. A **loop()** function, which prints out useful data to the Serial Monitor. The Serial Monitor displays data coming through the serial (USB) port. In addition, it waits for an "Enter" or "Return" to be hit, after which it applies a step input reference command of `xr = 0.05` (in meters).
  4. A **ControlLoop()** function, which is the control loop where you will make your edits. The function does the following:
     - Reads the encoder position and computes the handle position in meters. Also computes position error, its derivative, and its integral.
     - Computes an output force, based on a proportional controller (with a gain `kp`, initially set to what is considered a relatively low value [on most systems] of 100 N/m) to control the device to stay at its current position. The controller is written in the line
       `f = kp*e;`
       where is `e` the error between xr (reference position) and x (measured position) in meters.

- Converts the output force to a desired output torque and generates the appropriate output signal to the Ardumoto board.

5. A **setPwmFrequency()** function, which uses a pulse width modulation of the digital output of the Ardumoto to generate analog signals for motor voltage. (Don't edit anything here.)

- ❑ Now you are ready to test the default proportional controller. Place the handle in the vertical position. While the handle remains vertical, upload the e105_pid program to the Arduino. Once the program is uploaded, you should be able to move the handle around and feel the effects of the (soft) proportional controller. If something feels amiss, alert the instructor and we'll help you debug.

- ❑ Try recording some data using the Serial Monitor and then plotting it in MATLAB[1]. As soon as your Arduino program is running, open the Serial Monitor by selecting Tools > Serial Monitor. Deflect the handle away from its equilibrium position and release it. When you have taken the data you want, uncheck the "Autoscroll" checkbox (to freeze the scrolling data), copy the relevant data from the Serial Monitor, and paste it into a text file (easiest to do in a MATLAB editing window), which you can call, for example, `data1.dat`. In MATLAB, you can use the command `load data.dat` and then plot the second and third columns (reference position and output position) versus the first column (time) using the command `plot(data1(:,1),data1(:,2),data1(:,1),data1(:,3))`.

Now that you have finished basic testing, we get to the main goal of the lab: Create a PID controller and record the system response to a step input with the controller in place.

1. Proportional control

Ideally, the system would respond to a step input with fast rise time, low overshoot, small settling time, and no steady-state error.

Increase the `kp` gain from the nominal "low" value to a higher value and feel the effect of this – it should feel like a stiffer spring. You can deflect the handle and see how it responds to initial conditions. Raise `kp` high enough that you see a marked improvement in the response, i.e. it returns to the initial state quickly but without much oscillation.

Now apply a step (already set to a value of `xr = 0.05`) input by pressing "Enter" or "Return" when your cursor is in the Serial Monitor's command send line. To remove the step, you can press the small red "reset" button on your Arduino (next to the USB plug) and move the handle to vertical before the Arduino finishes resetting. Or you can center the handle while uploading the program. (The program sets the current position read from the encoder [a relative sensor] to zero at start.)

As you did earlier, you can copy data from the Serial Monitor to plot in MATLAB. Once you have a data set and plot that you are happy with, write down your value of `kp` and save the data file you want to keep so you can refer to them when you answer the questions below. Note that you should only submit one plot. You can try a few different values of `kp` if you are curious, but there's no need to do an exhaustive search for the optimal gain. The purpose of this lab is just to give you a quick hands-on controls experience, not a complete PID tuning experience.

---

[1] I was originally hoping to control the device through Simulink so that we would not have to do this switch between Arduino environment and MATLAB, but to get a sufficiently fast control loop requires the Real-Time Windows Target (http://www.mathworks.com/products/rtwt/) in MATLAB. This is a $29 add-on to the Student version of MATLAB, and it only works for Windows (not Mac OS). This is just FYI in case you want to try this on your own robot some day!

Questions to answer (can be done after you leave the lab):

a. What do you think is the form of the plant model for the device you are controlling? I.e., what is the general form of the transfer function?
b. Is the plant inherently stable or not? Explain.
c. What is the value of `kp` that you selected? (There is not a right answer here; every device is a little different.)
d. Annotate on your plot (just with arrows, no numbers needed) the rise time, overshoot, settling time, and steady-state error.

2. Proportional-derivative (PD) control

Add derivative control to create a PD controller by modifying the line `f = kp*e;` to include another term. Note that we have already defined a variable `kd` for you, and the code also calculates `dedt`, a filtered version of the velocity error. Increase the `kd` gain to try to improve the step response from pure proportional control. A rule of thumb for systems like this is that the appropriate `kd` gain will be one or two orders of magnitude less than the `kp` gain.

Apply a step and save data as you did earlier. Make sure to save your new data file under a different name. Save your value of `kd`, as well as the data file you want to keep, so you can refer to them when you answer the questions below. In addition, write down the control law you implemented (i.e., copy down the modified line of code you wrote).

Questions to answer (can be done after you leave the lab):

a. What is the value of `kd` that you selected?
b. Annotate on your plot (just with arrows, no numbers needed) the rise time, overshoot, settling time, and steady-state error.
c. What performance metrics did adding derivative control improve? Did anything degrade?

3. Proportional-integral-derivative (PID) control

Add integral control to the "`f =`" line create a PID controller. Note that we have already defined a variable `ki` for you, and we also calculate `eint`, the integrated error. Increase the `ki` gain to try to further improve the step response. A rule of thumb for systems like this is that the appropriate `ki` gain will be one or two orders of magnitude less than the `kd` gain.

Apply a step and save data as you did earlier. Save your value of `ki`, and the data file you want to keep, and your new control law so you can refer to them when you answer the questions below.

Questions to answer (can be done after you leave the lab):

a. What is the value of `ki` that you selected?
b. Annotate on your plot (just with arrows, no numbers needed) the rise time, overshoot, settling time, and steady-state error.
d. What performance metrics did adding integral control improve? Did anything degrade?

You are done! Unplug the USB cable from your computer and leave the lab setup as it was. If you think anything might be broken, please alert the instructor.

Submission: Your lab submission should include **three plots** and the **answers to the questions** for (1) P control, (2) PD control, and (3) PID control. While this assignment is not due until March 15, we strongly recommend that you complete it and submit as soon as possible after your lab session.