Lab 5 - PID Control

ME 451 - Introduction to Instrumentation and Measurement Systems, Spring 2019

## Lab Objectives

* Make a closed loop control system.
* Implement PID in a motor scenario.
* Get experience with PID tuning.

**Sensors for Report:** PID (tell us what it is and how it is late and wrong, and how the P and D constants contribute to that). *This is a 3 day lab.*

*Note:* Groups will use motors that belong to the class for this lab. These motors will be kept in the lab room at all times. If a lab group breaks a motor, they will have to borrow motors from other groups.

# Section 1: Setting up PID

1. At the end of Lab 4 we made a system that will try to settle on a specific position. We call that specific position the setpoint (in Lab 4 it was 1000 ticks).
   1. In Lab 4, we used a bang-bang strategy to settle at the setpoint. In Lab 5, we will use a more complex control strategy: PID control.
2. First, import the Arduino [PID library](https://playground.arduino.cc/Code/PIDLibrary/) as a .zip file into the Arduino IDE like in Lab 2 (found here, use the import zip option).
   1. Choose the ‘Latest Version of Github’ link.
   2. Select the ‘Clone or Download’ button, and download the zip file for the library.
   3. Use import zip option.
3. Using someone else’s code always makes things more complicated. Take some time to read through the functions that the library has to offer, specifically:
   1. Main page of the library (specifically, the basics section): [Main Page](https://playground.arduino.cc/Code/PIDLibrary/)
   2. The setup function for PID: [PID()](https://playground.arduino.cc/Code/PIDLibraryConstructor/)
   3. The computation function: [Compute()](https://playground.arduino.cc/Code/PIDLibraryCompute/)
   4. Set output limits function: [SetOutputLimits()](https://playground.arduino.cc/Code/PIDLibrarySetOutputLimits/)
4. We will use the [PID\_basic](https://playground.arduino.cc/Code/PIDLibaryBasicExample/) example as a base for our code. If you installed the library correctly, it should show up in your Arduino examples.
   1. We also provide a supplementary document for setting up the PID that provides further clarification: [link](https://docs.google.com/document/d/1eMPFfsfoNRrOqSZQFUmBKMVGaC-QCoIcTvVgwtgiUec/edit?usp=sharing)
   2. Let’s go through the example code together:
      1. First, find the setup function for PID in the PID\_basic example. There are three variables that have an ‘&’ sign before them. Keep the variables the way that they are - in basic terms the ‘&’ sign indicates that you don’t have total control over the variable. In our case you will have to share it with whatever happens backstage in the PID library code.
      2. As stated before, there are three variables in the setup function. Their roles are as follows:
         1. *Input:* A place to store your encoder ticks for the PID library to use.
         2. *Setpoint:* The target position that we want to settle on. For Lab 5, use 1500 ticks.
         3. *Output:* The power that we send to the motor.
            1. *Note:* PID is completely arbitrary in what it outputs. We need to make it output something useful to our scenario. In our case, something useful means that the motor should run at different speeds and directions.

You will need to make sure that the PID can drive the motor in both directions to reach the setpoint. You will have to use the SetOutputLimits() function to do this.

* + 1. Now let’s look at the loop function. Consider the general flow of the loop function:
       1. *First line inside loop:* We use the Input variable to store the state of our sensor.
       2. *Second line:* PID.Compute is used to update the value of Output.
       3. *Third line:* we do analogwrite with the Output variable.
    2. We will keep this general flow in our PID code using the motor, but we have to change what happens:
       1. *First:* we want to set Input to be our current motor ticks. We don’t want to analogRead anything.
       2. *Second:* we use the PID.Compute function.
       3. *Third:* we use output to set the motor direction and speed.
       4. We will also add a delay of 1 millisecond at the end of our code.
    3. *Note:* Treat the PID as a black box. It should be the thing that actually determines how fast and in what direction to drive the motor. Your task is to write code that will listen to what the PID wants and controls the motor accordingly. Don’t second guess the PID!

1. **Signoff 1:** You got PID working!
   1. Demonstrate to your TA that your PID is correctly configured.
   2. You will need to describe your code and demonstrate that you understand what is going on in the code.

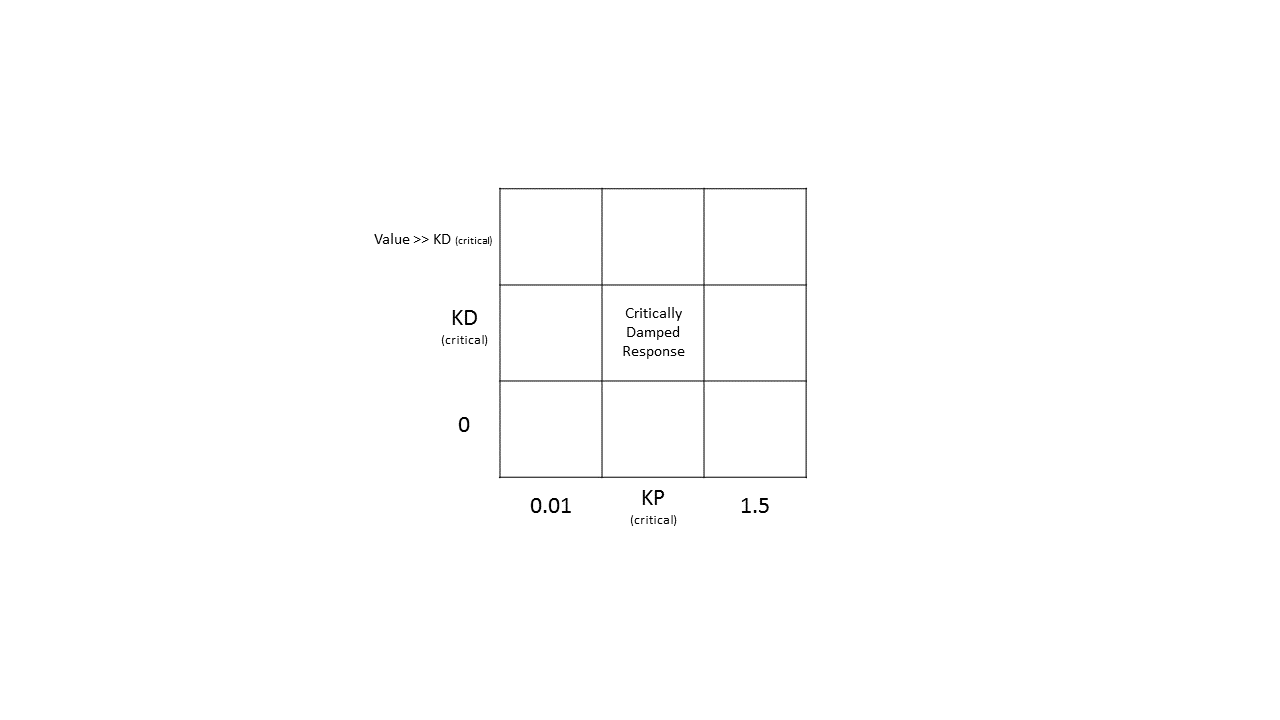
## Section 1 Discussion Questions

**Discussion Question 1:** PID is just a bunch of math that we apply to our system. Describe how we applied PID to our motor system, specifically describing how we conditioned the output value to be useful to our system and how the library works in conjunction with your code.

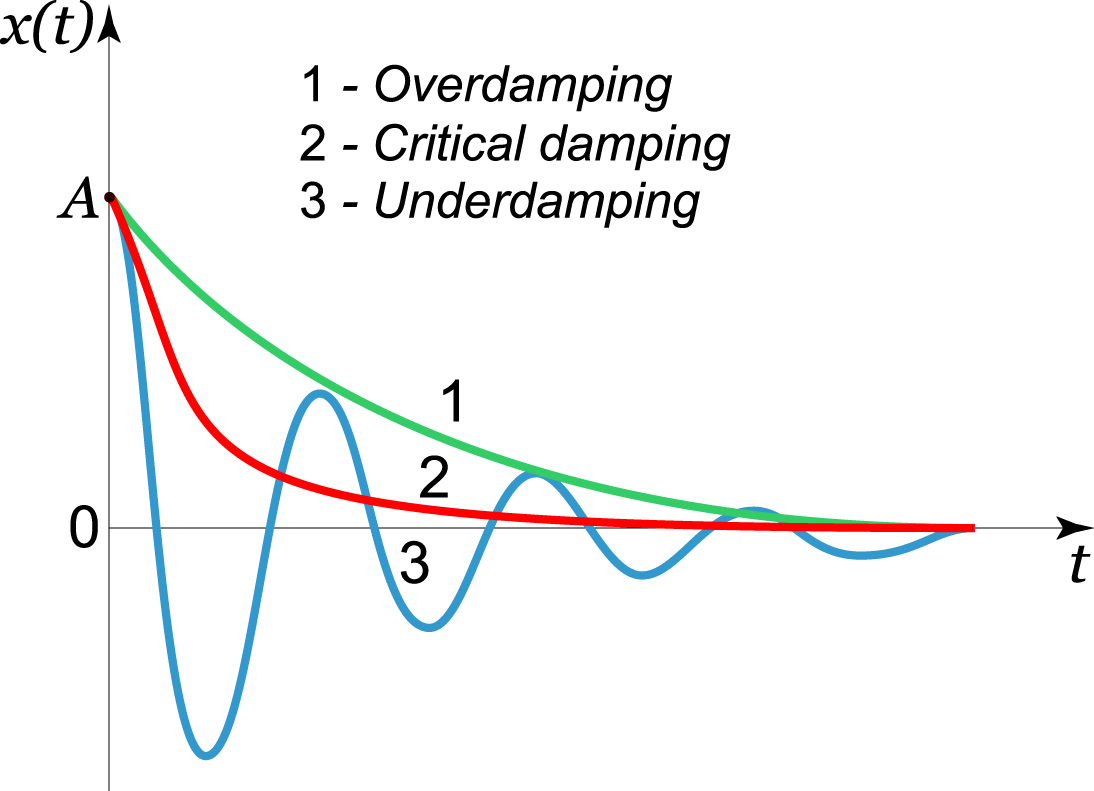
**DQ 2:** What does the ‘&’ sign do, in your own words (use an outside source)? What does the setMode function do in the PID library? List all of the display functions built into the PID library. In general, what do all of the display functions do?

# Section 2: PID with a static setpoint

1. Now it’s time to tune the PID. We tune PID to adjust its behavior and make it respond better for our purposes.
   1. We will only use the P and D constants for this lab to make things simpler. Set the K\_i constant to zero.
   2. *Note:* As you tune, check out the Serial Plotter tool. It will plot your data in real time so you can get a visual sense of how your system is responding (this is really helpful). However, the tradeoff is that you will not be able to collect data at the same time.
   3. **EXTRA CREDIT:** (If you really want to put yourself through the extra pain...) If you implement a PID controller (i.e., using a non-zero K\_i term), you will get double points per signoff that you use it.
2. First, generate a tuning map with nine representative plots (encoder ticks vs time) showing the effects of both KP and KD .
   1. Show KP and KD values above and below the critically-damped condition making sure to vary the values enough to see a change in response.
   2. Arrange the nine plots in a subplot grid, as seen below, with KP varying vertically in the grid and KD varying horizontally. Use the KP and KD constants indicated unless otherwise specified.



1. In the center of the tuning map is our critically damped response (for reference, see image on next page). Tweak the KP and KD constants of your control system so that it consistently brings you within around +/- 30 ticks in about 1 second.
   1. *Note:* You will always have to manually tune the system. If you are stuck and don’t know what to change to improve your system, consider this:
      1. Analyze what you need more of to improve the system. Do you need more power to get to the setpoint, or do you need to regulate the speed more?
      2. Which constant provides more power and which constant provides more speed control (in a general sense)? Act accordingly.
      3. Also consult the practical tuning resource from your prelab. The tuning methods are not the best, so mileage may vary.
   2. *Note:* For the top row, your KD value should be above at least 0.75.



1. **Signoff 2:** Critically damped system.
   1. Show your TA your critically damped system.
2. The sample rate of your PID loop matters greatly. We will dictate how fast it updates by changing the value of the delay that you added to the end of loop.
   1. Given your critically damped system, use time delays of 1ms, 10ms, 100ms, 1000ms, and 5000ms at the end of your loop.
   2. **Record** the data for each trial showing the response.
   3. For the next section, set your delay back to 1ms when you are done.

## Section 2 Discussion Questions

**DQ 3:** Show your tuning chart. What effect does the ‘P’ and ‘D’ constant, separately, have on the system response? Which of the constants do you think is more important?

**DQ 4:** Plot the responses for each type of delay (plot them on one plot, on top of each other). What effect do each of the delays have on a stable system? Describe why each delay behaves in the way it does. What is the longest delay you are most comfortable with having for our motor scenario? Why?

# Section 3: PID with a moving setpoint

1. We have implemented PID on a scenario with a static setpoint. Now we will implement both bang-bang and PID on a moving setpoint.
   1. *Note:* The Serial Plotter is especially helpful in this section.
2. We will use a setpoint that is a sine wave with a frequency of 0.1 Hz and an amplitude of 750 ticks.
   1. *Note:* This setpoint will need to be recalculated every time loop run. Put it at the top of your loop function. Use millis() (but in seconds) as the time variable for your sine wave.
3. First, edit your bang-bang controller from Lab 4 to follow our setpoint sine wave.
   1. *Note:* Use your bang-bang code from Lab 4 without the tweaks you made to improve the bang-bang’s response.
   2. **Record** the bang-bang response. Also print the setpoint to your serial monitor, so you can compare the bang-bang to your setpoint.
   3. Make edits to your bang-bang to make it follow the setpoint smoother, with at most an estimated phase lag of 15 degrees and an estimate signal attenuation of 10%.
      1. **Record** the bang-bang response and setpoint.
4. Next, modify your PID controller to follow the same sine wave.
   1. Tune the controller to follow the sine wave closely, with at most an estimated phase lag of 15 degrees and an estimated signal attenuation of 10%.
   2. **Record** the PID controller’s response and setpoint.
5. **Signoff 3:** Sine wave following bang-bang and PID.
   1. Show both your bang-bang (the tweaked version of your bang-bang) and PID implementations following the sine wave.
6. You are done! Make sure to get some time away from any motor sounds.

## Section 3 Discussion Questions

**DQ 5:** Plot your sine wave-following bang-bang and PID controllers on separate plots. On each plot, include the setpoint sine wave for reference. Make FFT plots of your setpoint sine wave, bang-bang response, and PID response and compare how well they match up. Which response has the most noise?

# Post Lab Questions

**Post Lab Question 1:** What does the ‘I’ constant do? What problem is it designed to solve for? Describe how the ‘I’ constant could lead to an unstable system (use outside sources).

**PLQ 2:** What is a PI and PD controller? What kind of controller did we use in lab? What is each controller best suited for? In industry, what is more common PD or PI? Why? Provide links to your sources.

**PLQ 3:** Apply PID to the following scenario:

You are told to set up an air conditioning system with both heating and cooling capabilities. This air conditioner controls the temperature of a single room, which is measured by a digital thermometer in the room. We want the temperature of the room to be room temperature, in Farenheit. Our air conditioner has different heating and cooling capabilities. The range of values that can be accepted by the air conditioner to control it are: 0-750 (when heating), and 0-500 (when cooling).

1. What will we use for feedback in the PID system? What value will we use as our setpoint?
2. Using your previous code as a reference, write what the code would look like if you used an Arduino to control the air conditioner.
   1. Use the function RoomTemperature() to get the current temperature of the room.
   2. Use the function CoolRoom() and HeatRoom() to cool and heat the room, respectively. Put the magnitude of the action inside the parentheses.

Group Names:

(in pen)

# Lab 5 Signoffs

1. \_\_\_\_\_\_\_ Demonstrated that your PID works.
2. \_\_\_\_\_\_\_ Presented critically damped PID controller.
3. \_\_\_\_\_\_\_ Presented bang-bang and PID controller with a sine wave setpoint.

**TA Signature**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Date**:\_\_\_\_\_\_\_\_\_\_\_