PID Controller Breakout Session 1:

1. Using the PID\_v1.h library create a PD controller to keep a fixed distance away from an object in front of the robot. (We did this in class). Remember to add your scaling factor in raw\_motor\_control() to make sure the robot drives relatively straight.

2. Write down your controllers parameters:

Setpoint = ?

Output limits = [?, ?]

SampleTime = ?

3. For this controller choose a kp gain that causes the system to be UNSTABLE. Write down the gains used for this setup.

Kp = ?

Ki = 0

Kd = 0

4. Decrease your Kp gain until you get a system that is RELATIVE STABLE. This means that in a sufficient amount of time (you analyze if you feel it’s sufficient) the robot holds its position a fixed distance from the object. Make sure this distance is approximately equivalent to your setpoint. +/- 3cm should be a good start.

Kp = ?

Ki = 0

Kd = 0

Settling Time = ?s (use the reset button and either count in your head or use a phone timer)

5. Now that we have a semi-stable system with only a Kp gain, let’s slowly adjust our Kd value until we have an optimal response. Optimal response means very little if any overshoot, a very quick response time, and low steady state error. (Steady state error means when the robot holds position, it has some displacement from the desired setpoint. But since the controller doesn’t provide enough power to the motors to move the robot it will stay in this position) To make steady state error easier to judge, change the setpoint to 10cm instead of 80, that way you can use your hands to estimate the distance. Take a video of the robot controlling itself from a big error to near zero error.

Kp = ?

Kd = ?

Ki = 0

Settling Time = ?

Overshoot amount = ? (use your eyes to estimate this)

Stead Stay error = ? (use your hand to estimate this)

6. Okay, now we should have an optimal response to error in the system. The last thing we will try to fix is the steady state error. The way we do this is by using the Ki value. Remember the Ki value keeps a “history” of errors in the past and will provide an output based on this history. So if we have steady state error, as time goes on the Integrator of the PID controller will continue to increase until the robot will start to move towards zero error. When tuned correctly this will eliminate any steady state error.

Kp = ?

Ki = ?

Kd = ?

Does it seem like the steady state error disappeared? Does it seem worse than when Ki=0? What do you think could be some reasons for the worse response, or the fact that steady state error hasn’t disappeared? (Think about the motor, and how much power is required to move).

EXTRA CHALLENGE: Now that you have some experience creating a PID controller try to make a controller that will control the acceleration of the robot in the x-axis (forward facing axis). Remember that the x-axis provides us acceleration values in the unit ‘g’s. So 1g will be 9.81m/s^2. Can we use this sensor variable as feedback to control the acceleration of the robot? Try to keep the desired acceleration relatively low (~0.2). The robot most likely does not have enough power to provide 1g of acceleration so asking for 1g of acceleration will just max out the motors.

Kp = ?

Ki = ?

Kd = ?

From your perspective, does it seem the robot is accelerating consistently once you have tuned the PID gains? At some point the robot will max out it’s velocity, so add a switch after some time that will negate the desired acceleration (maybe after 1000 milliseconds). (Hint: use a variable “switch\_time = millis()”, and then if millis()>switch\_time flip the sign of the setpoint variable to make the PID controller compute acceleration in the opposite direction. Don’t forget to reset the “switch\_time” variable!