Michael Mitsch and Clare Barnes
October 15, 2021

Lab 3 Report
PID Control Loop

Control Loop Frequency

Our control loop frequency is 200 Hz, or one sample every 5000 microseconds. We initially measured that our code inside the control loop takes about 1500 microseconds with no timing controls, but we realized this is not very consistent and can sometimes take over 3000 microseconds. To make our sampling happen at a uniform frequency, we added a variable sleep to the end of our code inside the control loop so the robot will always spend 5000 microseconds between the start of each iteration of the control loop. We chose 5000 microseconds because it is comfortably larger than the time it takes for the code to run.

$$T_{\text{Sleep}} = 5000\,\mu s - T_{\text{Code}}$$

There are 4 limiting factors to the time taken for each iteration of the control loop:

- Time taken to gather sensor data
- Time taken to calculate PID
- Time taken to set each motors' pwm
- Time taken to record data

These 4 factors are unavoidable because they are the key components of the process so we tried to make each of these efficient. The one that we have the most control over is the time taken to record the data. Instead of writing the data to a file each iteration, we append it to an array which is written to a file after the robot is finished.

Pseudo code

```
//First sweep over line to calibrate sensors
//wait for button

//open logging file
//in main loop:
        startTime = time.get_start_time()
        Position = read sensors
        Proportional = position - 2000 //position of line when line under center sensor is 0
        Derivative = Proportional - lastProportional //(change in position, so that P doesn't cause
value to change so dramatically)
        Integral += Proportional //(sum of positions, so that small offsets become larger)

        Power = Proportional/(some small number) + Integral/(some large number) +
        Derivative*some small number
//power difference for wheels decided by P, I, and D. Integral will get large quickly, so it needs to
be divided by a large number so it doesn't over power P and D. D is a small number and is used
to even out the other two, so it needs to be multiplied rather than divided.

        If power > absoluteValue(maximumSpeed)
```

Power = maximum

If power<0 //if power is less than 0, Proportional was negative, and therefore position was negative. So the line is on the left and the robot needs to turn right

Ab.right(maximum)

Ab.left(maximum + power)

Else

Ab.left(maximum)

Ab.right(maximum - power)

//every 2000 iterations:

Write values to file //don't want to have too much data

If button = pressed: //We need a way to break the loop so that we can close the file

Break

iteration+=1

endTime = time.get_end_time()

time.sleep((maxLoopTime-(endTime-startTime)) //so that each loop takes about the same amount of time

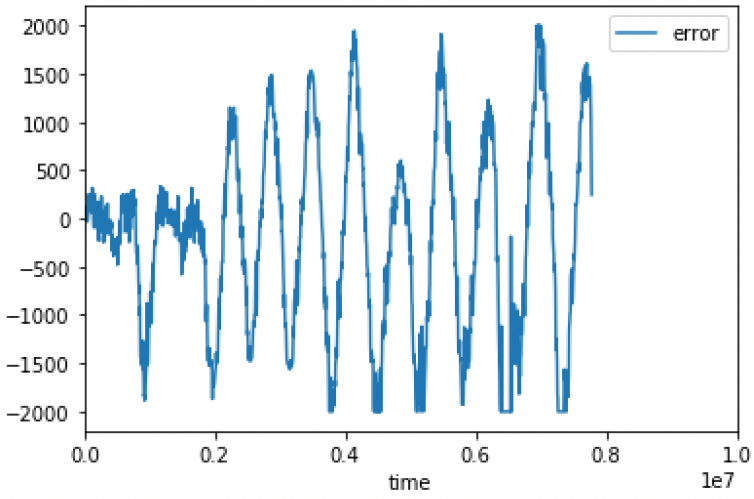
//out of loop, close file

Results

*The x-axis is measured in microseconds for each graph.

*Graphs were made with the graph.py file using the CSVs from our best runs

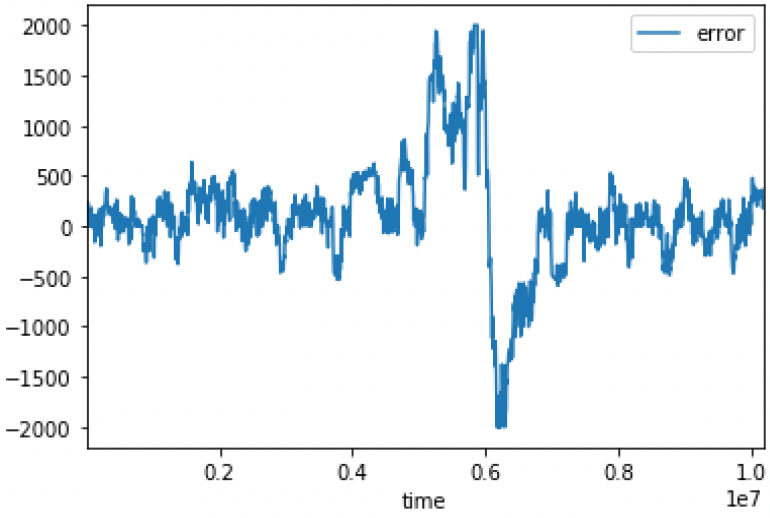
Only using P-term



The above graph shows the results when the P coefficient is set to 1/30 and the speed is set to 30% on a small oval track. The alpha bot makes 4 turns on its way around the track. The bot turned twice to start the run, then hit straight away from 2 seconds to 4 seconds, then made the last 2 turns at 5 seconds and 6 seconds, and finished on another straight away. As expected,

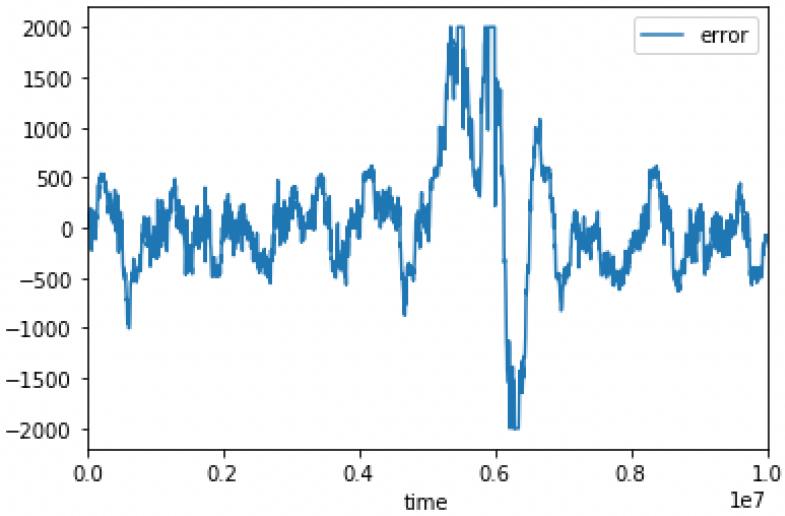
the amplitude of the graph is small at the beginning, then larger after coming off a turn. Because there is no D-term, the bot continually overcorrects itself on the straightaway, so the amplitude stays pretty constant. At 5 seconds the bot hits a right turn as it is about to oscillate to the right side of the line so the error is smaller, but error begins oscillating again when the bot gets back to a straightaway.

Using P-term and D-term



The graph above shows the trial when power was at 70%, the P term was 1/30, and the D term was 2.25. The robot was on the larger track and the first turn, shown above, was the more extreme one at the thinner end of the track. From time 0 to about 0.5, the robot was on the straight-way before it hit the turn. Then the line started to turn and it's position was to the right of the bot. The turn is rather extreme, so the position almost immediately jumps up to 1500 to 2000. And so the bot makes the turn and clears it at about time = 0.6, where it's P term over-corrects and the line position is on the left and drops to -2000. From time 0.6 to about 0.7 the bot finds the center of the line again, and by about 0.75 it finds the line, but oscillates on the left of it.

Using P-term, I-term, and D-term



The graph above shows the trial when power was at 70%, the P term was 1/30, the I term was 1/9000, and the D term was 2.5. The robot was on the larger track and the first turn, shown above, was the more extreme one at the thinner end of the track. The graph is almost identical to the one above, so it would be redundant to comment on what is happening, rather it is necessary to explain how the I term is changing the graph. The turn, at 0.5, has a greater error, cutting off at 2000, while the previous graph barely reached 2000. After the curve, the bot seems to oscillate more, most likely because of the build up of the I term, but the bot gets back to oscillating around 0, as it did on the beginning of the graph. This differs from the above graph, that stayed a little to the left of the line afterward. Although, the oscillations on this graph are more extreme because of the I term, which was probably made large by the extremity of the errors around this curve. I believe that they smooth out afterward, based on the video of this run, though.

Control Loop Recommendation

$$P = 1/30,\ I = 1/9000,\ D = 2.5$$

Videos of all our solutions are in the Lab3 folder. We were able to have the alphabot drive at 70% speed with the recommended PID coefficients.