Lab3 Report

AJ Funari, Blake Dunaway

To obtain the minimum loop period that we can run our control loop consistently, we first started by figuring out the runtime of the control loop at maximum speed. This is performed by calling time.perf_counter() at the beginning and end of the control loop and taking the difference between the two times. We found that the average of this period is about 0.00163 seconds (1630 microseconds), which results in a frequency of 613.49Hz. To control the loop period, we are going to use the time.sleep() function. We know the duration of the control loop with no sleep function is about 1630 microseconds. We started by creating a variable and giving it a value of 2500 microseconds. This would be our target time for the total duration of our control loop. The value that is passed to the time.sleep() function is the difference between our set target value and the duration of the control loop, about 1630 microseconds. This means that the total duration of one pass through the control loop should be equal to the duration of the control loop plus the amount of time the program is asleep. What we found was our initial target time of 2500 microseconds was too small. The minimum loop period we were able to obtain was 6000 microseconds. To be considered a consistent period, the time should stay within +/- 2%. We used a function call to validate that our period is consistent.

((Target Time) - (Real Time) / (Target Time)) * 100

```
def freq(self, a, b):
    target_freq = 1/a  # frequency from time.sleep() function
    real_freq = 1/b  # real frequency of loop
    out = ((target_freq - real_freq) / target_freq) * 100
    return(out)
```

The target time is the set time we allow our function to run which is 6000 microseconds. The real time is the time duration of the control loop plus the amount of time the program is asleep. With the equation above, we are about to determine each time through the control loop whether or not the real time of the control loop is within +/- 2% of the target time.

The limiting factors driving this period were the real duration of the control loop, the amount of time to sleep, the maximum set speed of wheels, and the PID terms. We mitigated the factors by watching how the robot performed whilst following the line, and

changing the terms to make the robot more consistent. We also analyzed the graphs from each performance, and made adjustments based on that.

Pseudo Code:

- Import Alphabot, TRSensors, PIDclass, and time
- Initialize 3 variables to each function grabbed from the imports
- Run the calibration function from TRSensors.
- Initialize variable Maximum = 80
- Initialize variables Integral, Last_porportional, count = 0
- Call function perf_counter in variable start_time
- Start While Loop
    - Call function perf_counter in variable time_stamp
    - Call function readLine from TRSensors that returns 2 outputs: position and a 5 length array of values from sensors. Store in variables positions and sensor
    - Start if statements(if all 4 sensors are on black line)
        - Call function alphabot and stop right and left wheels
        - Initialize variable loop time = perf_counter - start_time
        - Print the looptime
    - Begin else statement
        - Initialize variable error = the position of bot - 2000
        - Initialize variable derivative = error - last_proportional
        - Initialize variable integral = integral + error
        - Initialize variable last_proportional = error
        - Initialize variable pterm = error multiplied by .04
        - Initialize variable iterm = integral multiplied by 1/10000
        - Initialize variable dterm = derivative multiplied by 6
        - Initialize variable = sum of pterm, dterm, iterm
        - Begin if statement(if output is greater than maximum)
            - Reset max to output
        - Begin if statement(if output is less than - maximum)
            - Reset output to -max
        - Begin if statement(if output is less than 0)
            - Turn right
        - Begin else statement(
            - Turn left
        - Initialize variable runtime = duration of control loop
        - Initialize variable target = 166.66Hz
        - Initialize variable x = target - runtime
        - Sleep for x time

- Initialize variable total_time = total time the loop took
- Call function freq from PID to calculate percentage total time is compared to target
- Begin if statement(if count is 20 iterations in the loop)
    - Call loginfo() from PID and log all our terms
- Increment count

Source Code: Control Loop

```
output = pterm + iterm + dterm

if (output > maximum):
    output = maximum
if (output < - maximum):
    output = - maximum
# print(position,output)
if (output < 0):
    left = maximum + output
    right = maximum
    Ab.setPWMA(left)
    Ab.setPWMB(right)
else:
    left = maximum
    right = maximum - output
    Ab.setPWMA(left)
    Ab.setPWMB(right)
```

Once we obtained our minimum consistent period of 6000 microseconds, we were able to start tuning the PID loop. We first started tuning the Alphabot with just the pterm. We started with an arbitrary value of Kp, ran the control loop and evaluated the response of the robot. If we noticed the robot overshoot the line, that means our Kp value was too strong. If the robot did not react when moving away from the line, the Kp value was too low. When the robot was consistently following the line and able to move around the track, we moved on to start tuning the Kd term. The Kd term acts as a dampener for the Kp term. We adjusted the value of the Kd term until the robot consistently followed the

line and repositioned itself accurately relative to the line. Ki is the accumulation of the error position divided by a large number; it helps to make small adjustments over a long period of time. For the line following project, we found that Ki did not impact the performance of the robot when following the line.

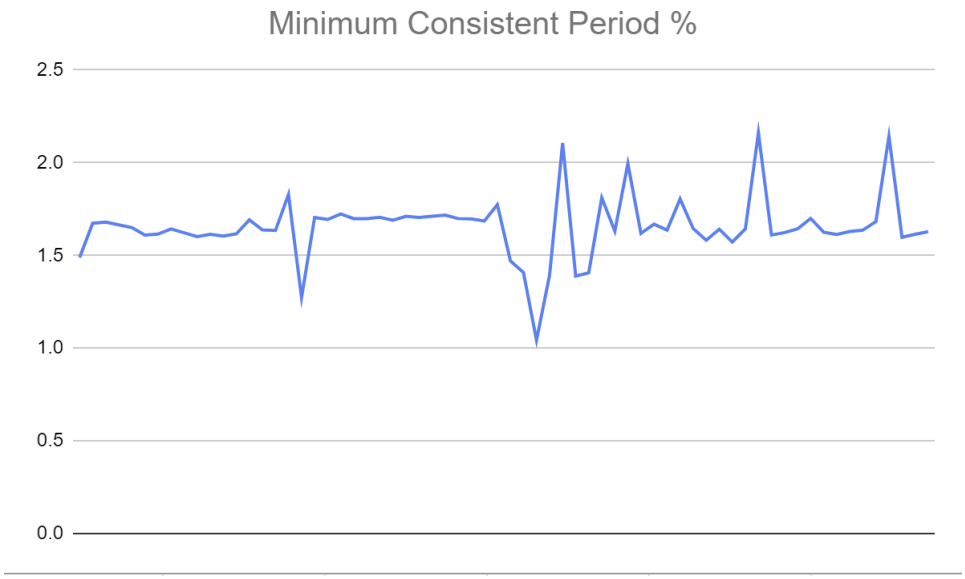
To log our data, we created a function. The function takes 8 variables (timestamp, dev, error, pterm, iterm, dterm, left, right) and appends each variable to a list which is then written to a csv file.

```
def loginfo(self, timestamp, dev, error, pterm, iterm, dterm, left, right):
    self.data.append(timestamp)
    self.data.append(dev)
    self.data.append(error)
    self.data.append(pterm)
    self.data.append(iterm)
    self.data.append(dterm)
    self.data.append(left)
    self.data.append(right)
    self.f.write(str(self.data) + '\n')
    del self.data[:]
```

Kp: small track

For adjusting the Kp, we found that by setting the maximum to 50 and Kp to 0.1, the robot was able to consistently follow the line on the small track. When we increased Kp, the robot would overshoot the line. When implementing only Kp in the control loop, we found that the robot is not able to settle back on the line, but only zig zags back and forth. The fastest time we recorded around the small track using only Kp was 8.06084 seconds.

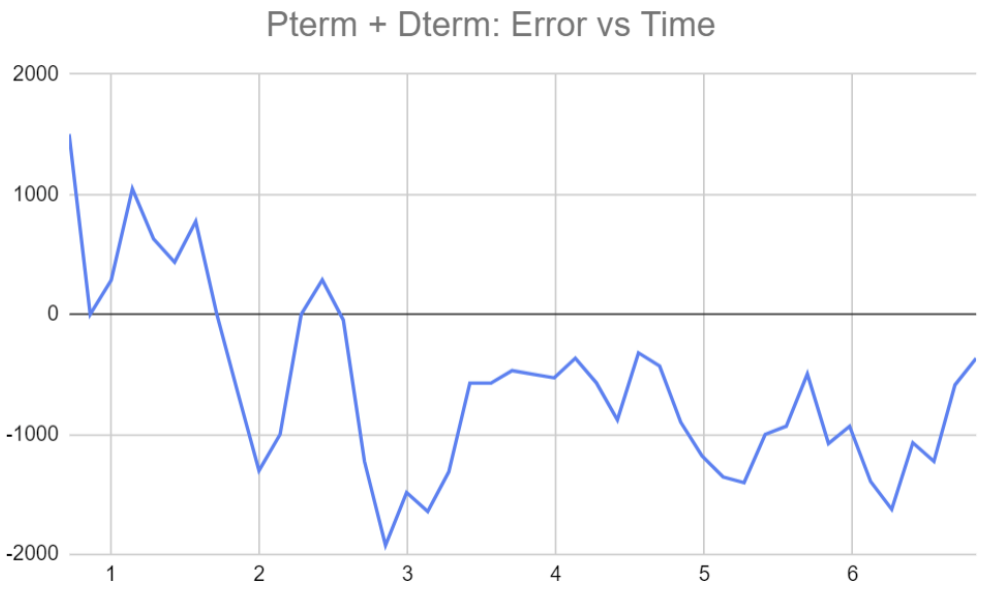
https://drive.google.com/file/d/15z8AC42YyJVGkcQLXgixejWlZhWrXc0o/view?usp=sharing

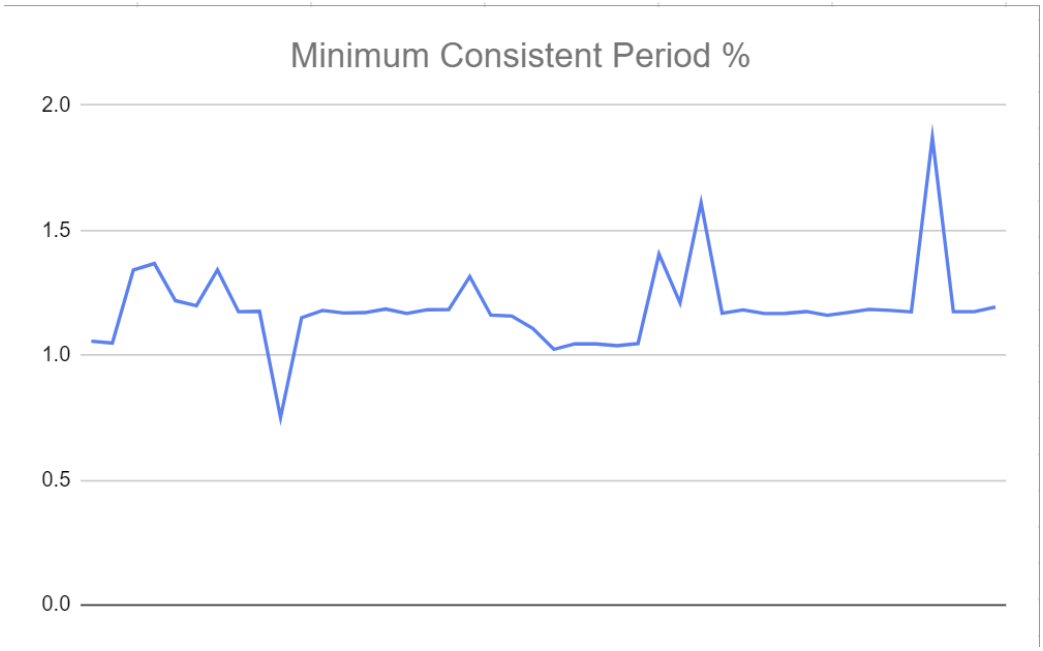


Kp + Kd: small track

When building the solution for the control loop using Kp and Kd, we were able to increase the maximum speed. We started with Kp = 0.1 and Kd = 2. Using these values we recorded a time of 6.09177 seconds. We tuned the maximum speed, Kp, and Kd until we reached our fastest solution around the track. When maximum speed = 60, Kp = 0.15, and Kd = 4, the robot recorded a time of 6.21453 seconds. From analyzing the Error vs Time graph, we can estimate it takes the robot about 0.5 seconds to settle back on the time.

https://drive.google.com/file/d/1epNb2dn7iSa1QfXoCqsTR4Fu0f-7LLaA/view?usp=sharing



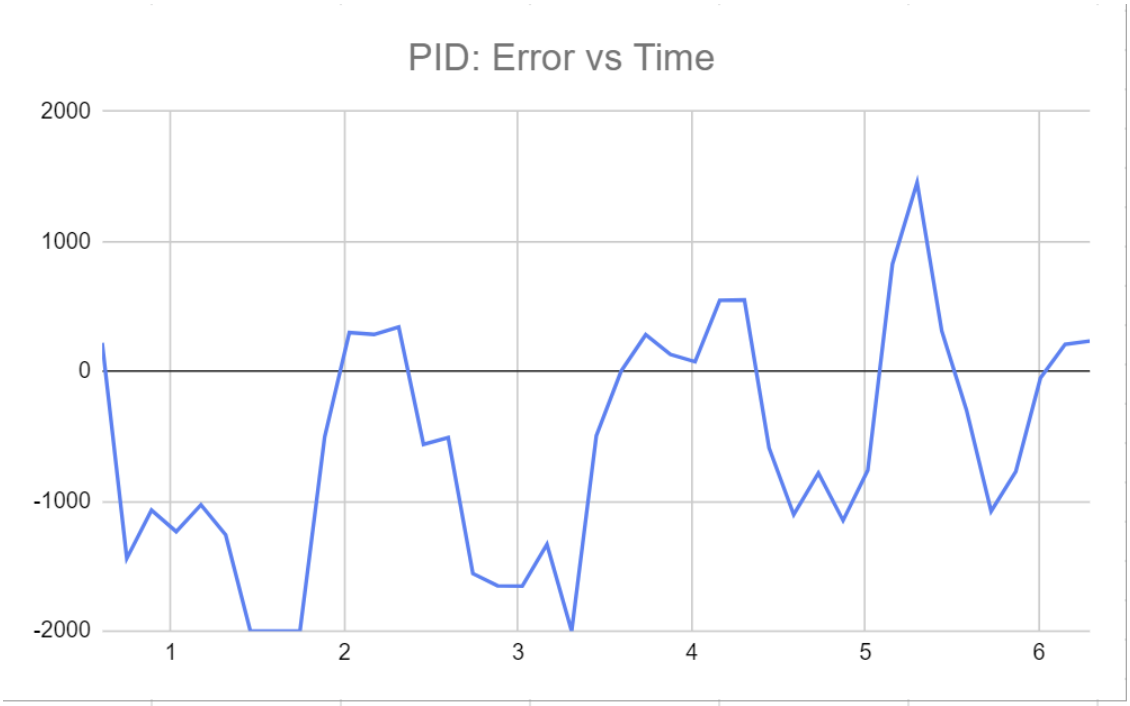


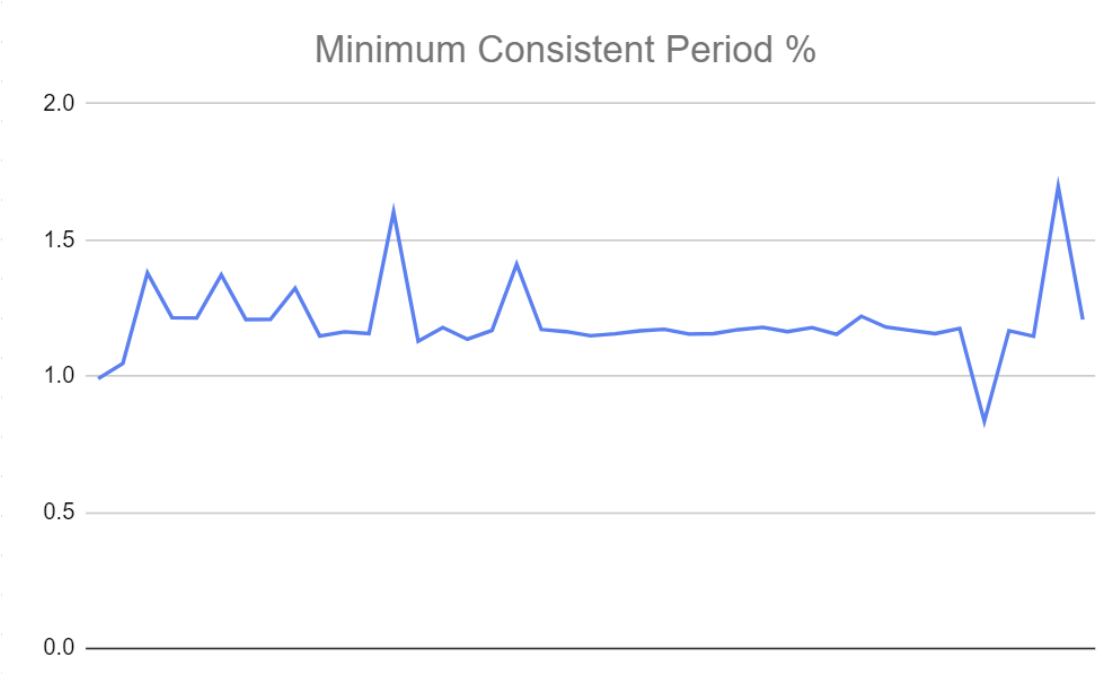
PID: small track

Using all PID terms, we found that setting our speed to 70, Kp = 0.06, Ki = 1/10000, and Kd = 4 gave us the best results. Although adding in Ki didn't have a noticeable performance impact, we were able to record our fastest solution of 5.6835 seconds.

Again, if we analyze the Time vs Error graph, we can estimate the robot settles back on the line in about 0.5 seconds.

https://drive.google.com/file/d/1SQavkJ7AOaBUkNPNiZTtqBM7PluuW0E9/view?usp=s



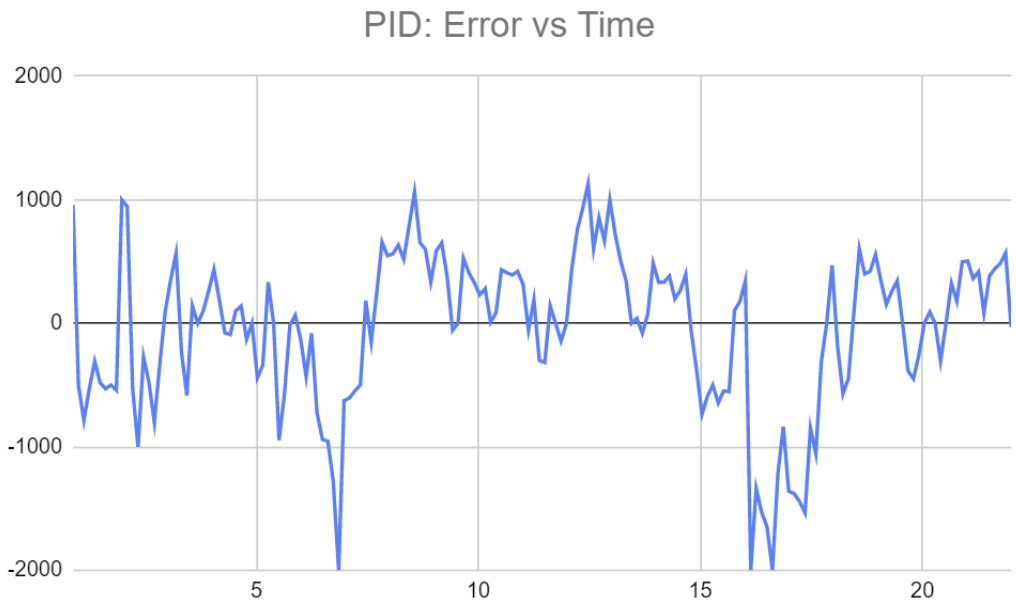


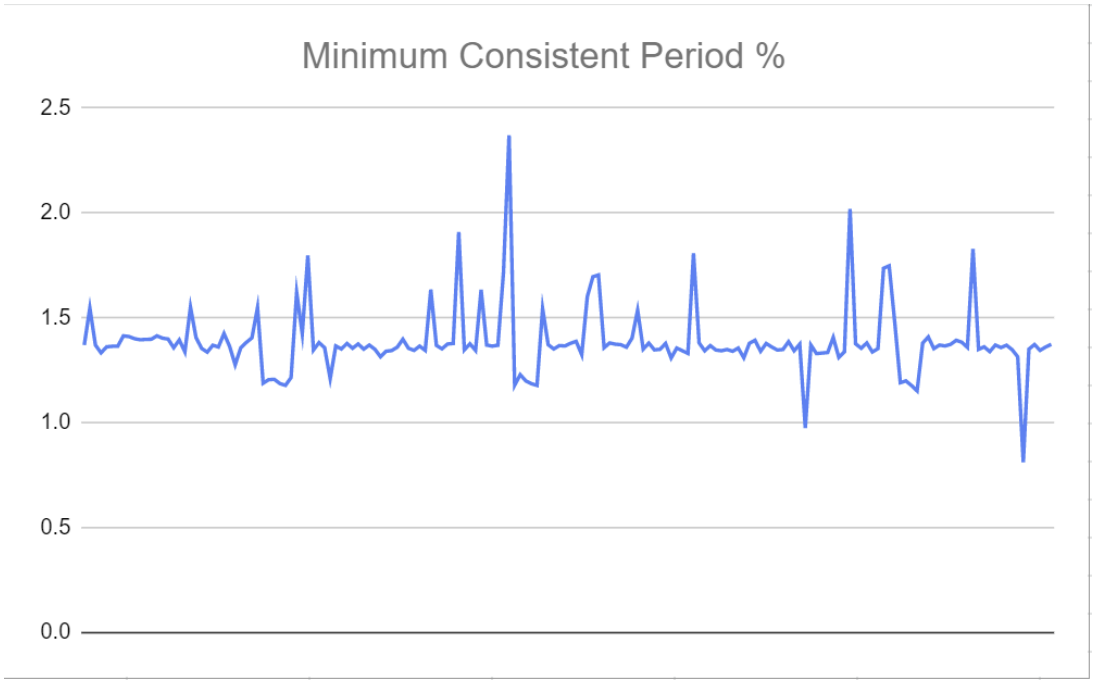
Assuming that the goal of this project is to come up with the fastest tracking alphabot. Use data to make a recommendation for the optimal control loop strategy. You should also include a video of your solution.

PID: big track

The values we found to be the most efficient when following the lines for PID were: Maximum speed = 80, Kp = .04, Ki = 1/10000, and Kd = 6. Along with the terms, we had the frequency of our control loop set to 166.7 Hz. Using this data our fastest and most consistent lap time around the big track was 21.1 seconds. Thus making those our recommended values to use for the fastest tracking Alphabot

https://drive.google.com/file/d/1Nm_NrrYHVrWrEjSOvqan1hh-6h4IIMLz/view?usp=sharing




Minimum Consistent Period %
2.5
2.0
1.5
1.0
0.5
0.0