ECE 3 Final Report

ECE 3, Fall 2020, December 19
Lab Section 1C

Yu-Wei Vincent Yeh [005123289], Philip Giang [505340657]

Introduction and Background

Project Goals

The goal of this project is to design an unmanned car that can move along a curved path. The car must be able to detect a change in the path (e.g. a bend) and react accordingly by changing the speeds of its wheels by various amounts in order to stay on the path. When the car reaches the end of the path, represented by a wide line perpendicular to the path, the car must be able to turn 180 degrees in order to traverse the path back to the starting point. This all must be done in the shortest time possible.

For this project, we chose to implement a PD controller (that differs from conventional setup, which will be explained in Results and Discussion and Future Work section) taking in the combined (fusioned) sensor input from the phototransistor circuits located on the bottom of the car. This decision meant that we needed to test which proportional and derivative constants we were to use in order for a successful traversal of the race track. [1] [2]

Path Sensing Circuitry

In order to detect a change in the path, we utilized a phototransistor circuit with the phototransistors facing the floor. The circuit can be split into two stages: one where the capacitor in the circuit is charged up, and one where it discharges (controlled by a switch).

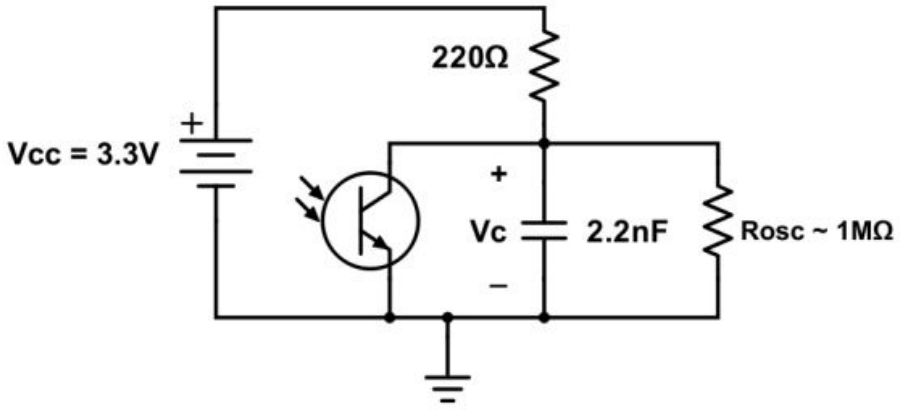


FIG. A: Stage 1 of phototransistor/path sensing circuit
Source: Professor Briggs ECE3 Lab Manual, pg. 71

This stage of the circuit ensures that the capacitor is fully charged regardless of light condition, giving a starting point to measure differences in brightness on the paper. The second stage gives us data for measuring the difference between light and dark.

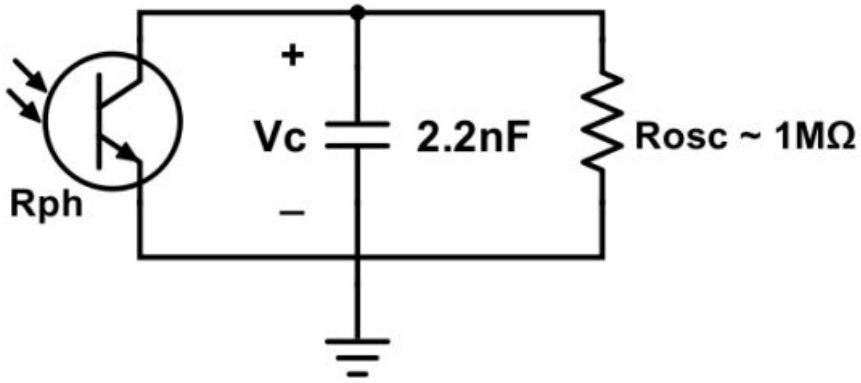


FIG. B: Stage 2 of phototransistor/path sensing circuit
Source: Professor Briggs ECE3 Lab Manual, pg. 73

In this stage, the capacitor starts at a set charge regardless of the brightness. The phototransistor acts as a variable resistor that depends on brightness, which means the capacitor discharges at a speed determined by the brightness/color of the paper directly underneath the phototransistor. [4]

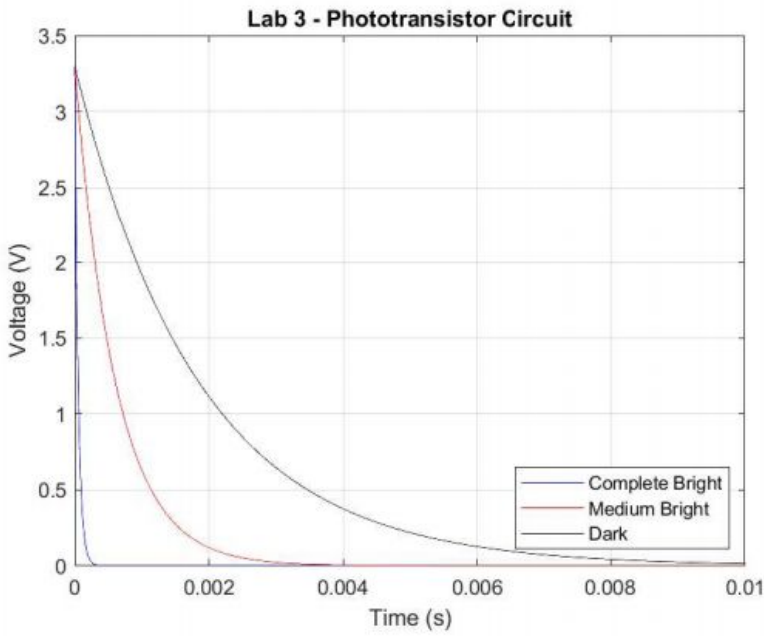


FIG. C: Discharging capacitor curves for variable brightness
Source: Data from ECE3 Lab 3, graph created in Matlab

$$V(t) = V_C e^{\frac{-t}{C(R_{Ph}//R_{OSC})}} \approx V_C e^{\frac{-t}{C*R_{Ph}}}$$

Equation 1: Voltage of discharging capacitor in phototransistor circuit

Based on the curves in FIG. C, it can be seen that covering up the phototransistor (comparable to having it point at a dark color) makes the capacitor discharge many times slower when compared to the phototransistor being exposed to light (comparable to having it point at a white point on the paper). This also agrees with (1) which relates

the rate of decay of the initial voltage with a lower R_{Ph} (a lower R_{Ph} being the result of a higher brightness). Thus, we can take the relative time it takes for the capacitor to discharge as a measurement of the color of the paper at that point.

Sensor Calibration, Fusion

The Arduino takes in the input from the sensors and translates it to a number ranging from 0 to 2500, with 0 being the highest brightness (sensor facing white paper) and 2500 being lowest brightness (sensor facing black paper). In order to correctly interpret the values received from the phototransistor circuits, we needed to make sure that the measurements from each individual phototransistor in the array were normalized. This was due to the fact that the phototransistors/capacitors/resistors in the circuit were not exactly identical due to factors such as tolerances, meaning different light sensing modules would read in different values for the same brightness intensity.

The calibration was done by first making the car print out each sensor's value at rapid intervals so we could see which sensor was detecting what. We then took a strip of black paper wide enough to cover two sensors and placed it to the left of the leftmost sensor. By slowly moving the paper across the line of sensors in ~2 millimeter increments and recording the output printed on the screen, we were able to get a measurement of what each sensor detected when on top of a black line. We took these values and then averaged them across several readings in order to produce the below graph.

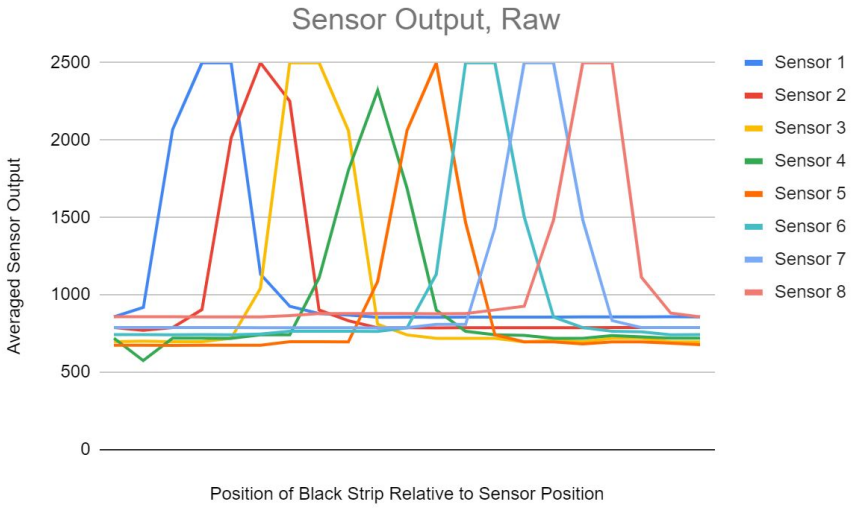


FIG. D: Raw averaged sensor outputs based on where the black strip is located under the car

Source: Created in Google Sheets

From the graph, it can be seen that each sensor has different minimum and maximum values despite the fact that the same black strip was used for all of the sensors. We took these values and normalized each sensor's readings by removing the minimum values from each data point (to set each sensor's minimum value to zero) and scaled each sensor's data points to their respective maximum values (so that all maximum values would be 1000 and all values between would be a percentage of that maximum). [3]

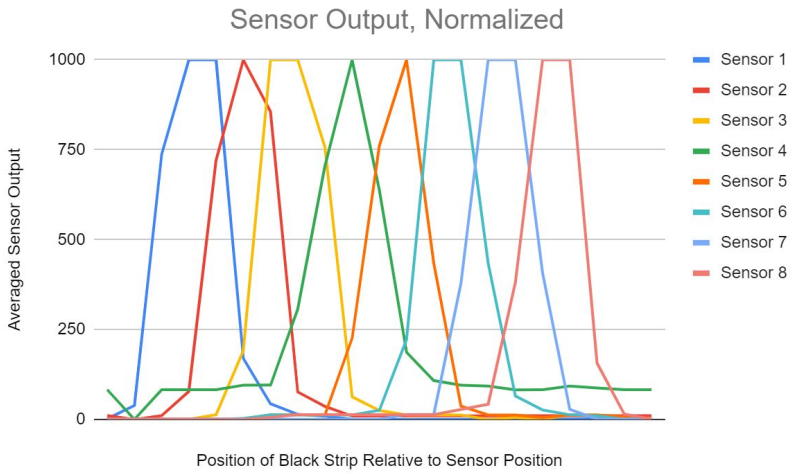


FIG. E: Normalized and averaged sensor outputs (minimums removed, scaled to 1000/maximum) based on where the black strip is located under the car
Source: Created in Google Sheets

With these normalized values, it was possible to accurately combine them together in order to determine both where the track was located underneath the car (either left or right) and how much action the car needed to take in order to correct its path. In order to do this, we assigned two different weighting schemes to rate the values gathered from each sensor. For example, with an 8-4-2-1 weighting scheme, the leftmost and rightmost sensors would have their output multiplied by 8, the sensors next to those sensors would have their output multiplied by 4, etc, and differentiating between the left and right sensors by multiplying the left sensors with a negative constant. The single value we obtained from adding the scaled values together would then be used as our positional indicator relative to the track. [3]

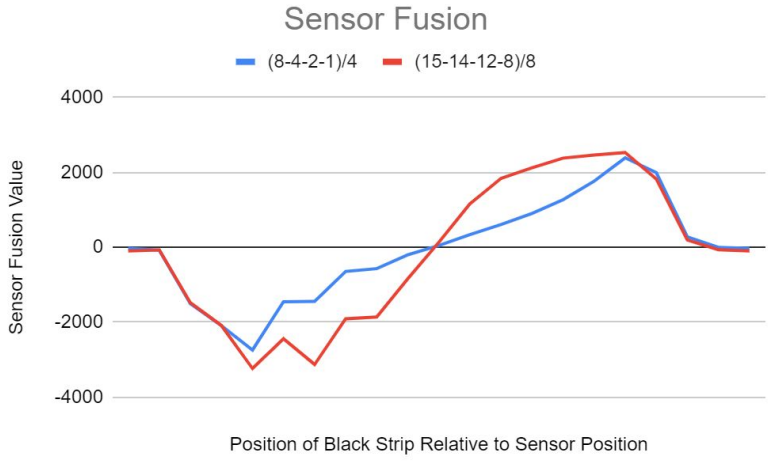


FIG. F: Sensor fusion values for two different weighting schemes
Source: Created in Google Sheets

Our final design choice was to choose the 8-4-2-1 weighting scheme, which gave nearly proportional response values compared to the 15-14-12-8 weighting scheme, which gave more extreme response values for smaller changes in position from the middle.

PD Controller

$$u(t) = K_P e(t) + K_d \frac{de(t)}{dt}$$

Equation 2: General PD controller equation, with e(t) representing the error and u(t) representing function return

The control system that enabled our car to move along the track was a PD controller. A PD controller is a feedback loop subsystem that takes in the system's current error and adjusts the overall system's input variables based on a proportional and time-varying response to the error. For this project, the system used the car's sensor fusion value as the error and made changes to the wheel speeds depending on a proportional response to the sensor fusion value itself as well as how the fusion value changed over time. A more complicated version of the PD controller exists called the PID controller which implements an integral response to the error, but we decided that it would be more efficient to test our vehicle based on two error variables (Kp and Kd) rather than three (Kp, Kd, Ki). [6]

On-Track Development Methodology

a. Test setup:

We first located our car loaded with the IR Sensor Example under the lighting condition that we would be operating under during the race day to obtain accurate base values for our sensor calibration. We then designed a simple algorithm for the straight track that intended to let us get a feel of how the wheels steer in response to different sensor values - the process and setup are fully showcased in part b of this section. After obtaining a full understanding of the path sensing system, we then proceeded to implement the PD controller - the logic and algorithmic design were also demonstrated through diagrams and flowcharts in part b.

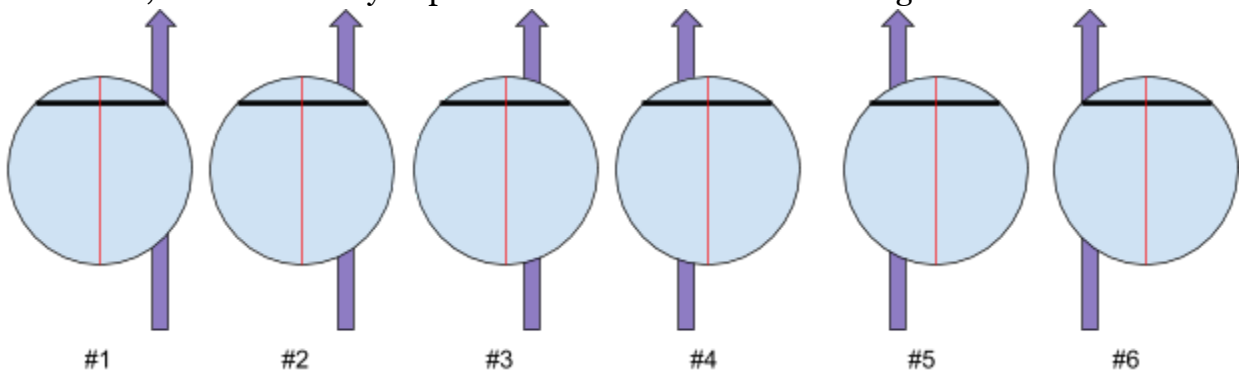
b. How the tests were conducted:

The very first step in our development was to get a feel of how the sensors work and how the wheels should respond when the sensed values change by implementing specified thresholds instead of utilizing PID control. That is, our simple testing algorithm has six cases (shown as FIG. G) in which the car would steer at different levels:

1. For cases #1 and #6, the car diverged significantly from the path, so we increased the speed of left wheel and right wheel, respectively, by a significant amount to make the car steer towards the track.
2. For cases #2 and #5, the divergence wasn't as serious as that of cases #1 and #6, so we increased the speed of left wheel and right wheel, respectively, by some amount to drive the car back to the track.
3. For cases #3 and #4, the car only diverged from the track by a little, so we only made minor adjustments to the speed of left and right wheel respectively for two cases.

Please note that the purpose of the above setup was only intended for us to familiarize ourselves with the steering of right and left wheels, as we would implement the PID control (we only deployed P and D) in later stages. After obtaining a rough idea as to

how the sensors worked, and allowed the car to follow a straight path, we tested out various speeds that we would like the car to do donuts at the end of the curve. Our eventual parameters for donut were 170 speed and steer for 300ms. After performing the above procedures, we were able to get the car to go back and forth on the straight curve. Now, we could finally implement the actual PID control algorithm.



*Legend**:** Circular object - The car / Red line - Line that bisects the car / Black line - Sensors*

FIG. G: This diagram shows the six cases for the simple testing algorithm
Source: Created in Google Docs

	Case #1	Case #2	Case #3	Case #4	Case #5	Case #6
Cutoffs	val < -1200	-1200 < val < -700	-700 < val < -200	300 < val < 800	800 < val < 1300	val > 1300
Speed for steering (Left / Right)	85 / 70	77 / 70	73 / 70	70 / 73	70 / 77	70 / 85

Table 1: The threshold values set for each case in FIG. G

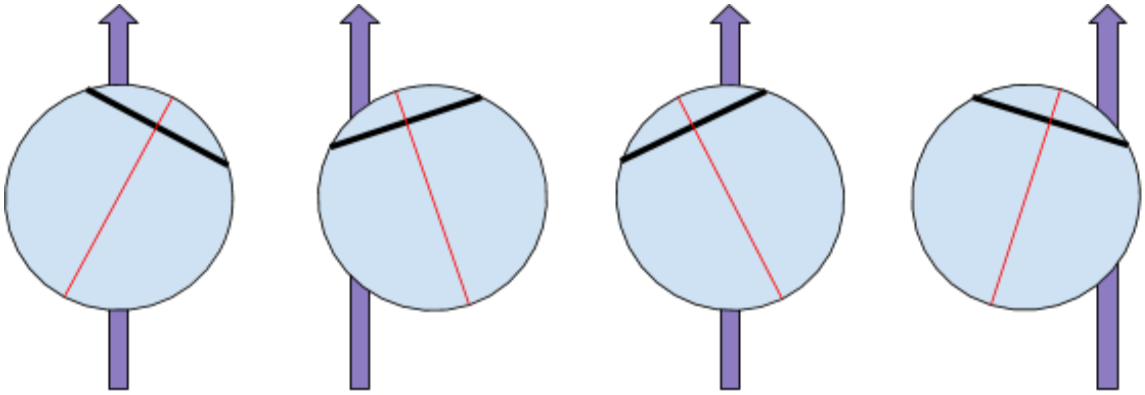
We then started to develop the much more complicated PD version. Compared to typical PID control where all terms were combined into a single steering command value, our algorithm controls left and right wheels with different values (either P or D term) depending on the scenario. Our algorithm considered the following four cases as shown in FIG. H:

1. Case #1: The fusion value would be positive since the track is to the left of the car, and the change in position (current fusion value - previous value) would be positive as the fusion value is getting more positive. (car diverging from track) In this case, since the car is getting away from the track towards the right, the D term would like to steer the car towards the left, and since the track is on the left of the car, the P term would also like to steer the car towards the left. (P and D agree)
2. Case #2: The fusion value would be positive since the track is to the left of the car, and the change in position (current fusion value - previous value) would be negative as the fusion value is getting less positive. (car approaching the track) In

this case, since the car is approaching the track, the D term would like to steer the car towards right, and since the track is on the left of the car, the P term would like to steer the car towards the left. (P and D disagree)

3. Case #3: The fusion value would be negative since the track is to the right of the car, and the change in position (current fusion value - previous value) would be negative as the fusion value is getting more negative. (car diverging from track) In this case, since the car is getting away from the track towards the left, the D term would like to steer the car towards the right, and since the track is on the right of the car, the P term would also like to steer the car towards the left. (P and D agree)
4. Case #4: The fusion value would be negative since the track is to the right of the car, and the change in position (current fusion value - previous value) would be positive as the fusion value is getting less negative. (car approaching the track) In this case, since the car is approaching the track, the D term would like to steer the car towards left, and since the track is on the right of the car, the P term would like to steer the car towards the right. (P and D disagree)

With these four cases in mind, we developed an algorithm shown in FIG. I.



Legend — Circular object: The car / Red line: Line that bisects the car / Black line: Sensors

FIG. H: This diagram shows the four cases for the complicated PID control algorithm
Source: Created in Google Docs

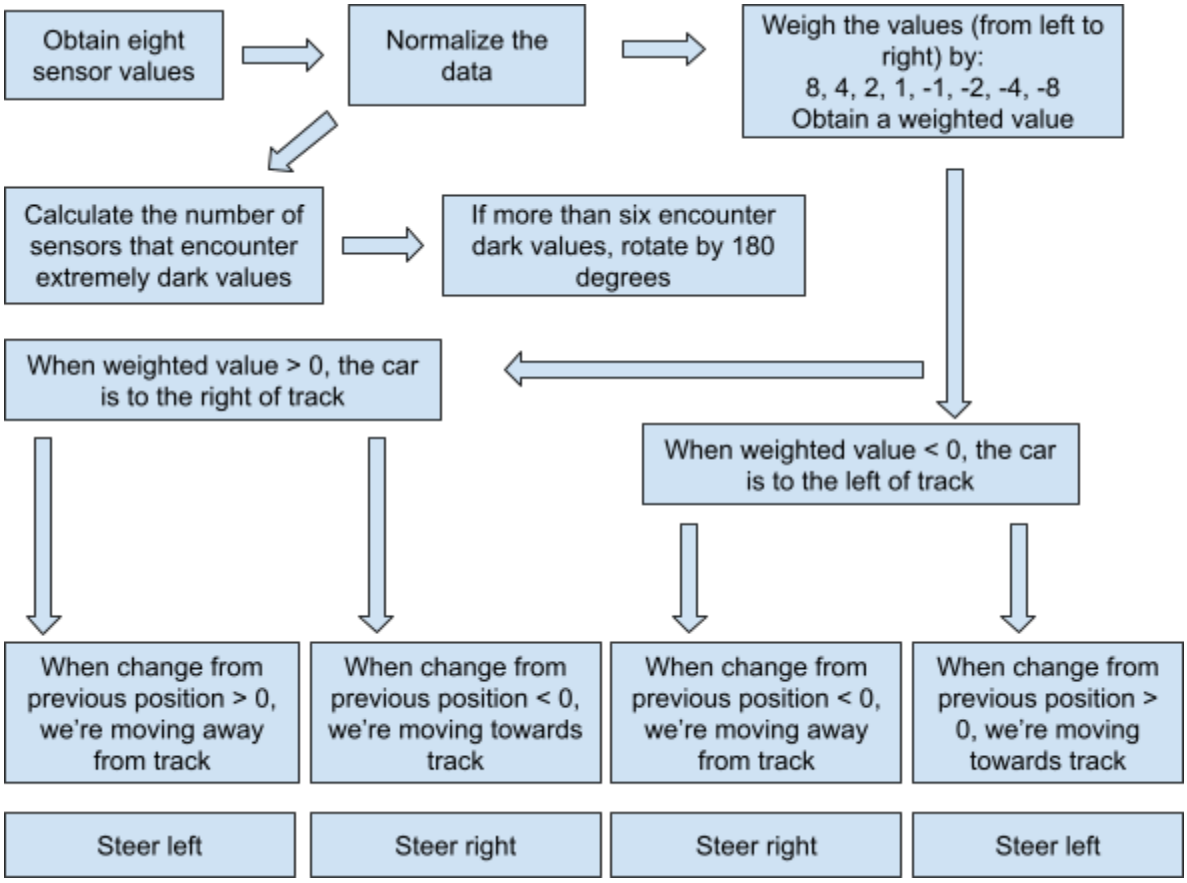


FIG. I: Flow chart of PD algorithm
Source: Created in Google Docs

Before the trial runs, the earliest parameters that we evaluated were the base sensor value for our calibration process, and we did so by the process described in part a. Earlier on in the testing process (also shown in part c and d) we decided to evaluate the values of Kp and Kd first since they were critical to the behavior of the cars along the track as opposed to the donut delay, which could easily be determined based on the speed of rotation. For Kp, we first put the car at various positions on the track and made sure it steered correctly at a reasonable rate. As for Kd, we simply let the car run on the straight track to ensure that it worked well. There weren't any specific values that we were looking for since the way we did it was to first test out a set of relatively reasonable values (Kp = 0.1 and Kd = 2), then adjust their relative magnitude accordingly. Since adjusting two parameters could produce an overwhelmingly large set of results, we eventually set Kp to a specific value of 0.03 then adjust Kd accordingly.

When the car was on the track, we looked for qualities including the car's steering response to rapid change, the location in which the car ran out of track (either due to over steering or insufficient steering), and in which ways was the car destabilized. We also looked for characteristics of the track itself - factors such as lighting on the track, wrinkles on the paper, and the flawed connection between two papers which could significantly undermine our effort and cause the car to derail. Our test trials mostly consisted of runs that start at position 1 since the primary factor that dictated the success of starting at other locations (such as 3 and 4) was the first steering response of

the car when it was just turned on--as long as the car didn't oversteer or understeer before reaching the right direction, starting location didn't make much difference as the car usually achieved stability before reaching the bend. After obtaining the proper parameters that worked for location 1, we tested those parameters on other locations and made minor adjustments.

c. Data Analysis (*The Result/Time column contains categorization labels included at the end of this subsection)

We conducted many tests in order to find the correct Kp and Kd to use. Parameters used by each partner vary due to different characteristics of the car we received. Each trial was slightly different - we either changed the Kp coefficient, Kd coefficient, base wheel speed, or donut speed. The appropriate working donut speed and delay was already determined before we recorded the data. Due to unnoticed flawed logics in our algorithm, failed trial data that were collected prior to the runs included in the following tables were omitted as they were worthless to the adjustment of our parameter values. Furthermore, certain sets of parameters in the following table were tested multiple times consecutively to ensure stability, but we decided to include only the result from one of the tries. Some trials have two results in the Result/Time column due to complicated behavior that temporarily drove the car away from the track. Along the way, there were multiple sets of parameters that could work, but due to their instability, they were not used. Fewer tests were conducted by Philip Giang as a result of prior testing; parameters tested in Philip Giang's table were based primarily on the successfully tested parameters shown in Table 2.

Yu-Wei Vincent Yeh's Tests and Final Values

Trial	Position	Fusion Coeff	Control Coeff		Speeds		Donut Delay	Result/Time
			Kp	Kd	Base	Donut		
1.	1	±8, 4, 2, 1	0.1	2	70	70	850	C4
2.	1	Same	0.1	3	70	70	850	C4
3.	1	Same	0.1	1.5	70	70	850	C3
4.	1	Same	0.1	5	70	70	850	C3
5.	1	Same	0.1	5	40	70	850	C4
6.	1	Same	0.05	5	40	70	850	A[11.12]+C4
7.	1	Same	0.08	5	40	70	850	A[11.59]+C4
8.	1	Same	0.03	5	40	70	850	A[12.34]
9.	1	Same	0.03	5	45	70	850	C4+A
10	1	Same	0.03	4	45	70	850	D3+A

11.	1	Same	0.03	7	45	70	850	C4
12	1	Same	0.03	3	45	70	850	A[11.25] + D1
13	1	Same	0.03	3	43	70	850	A[10.74]
14	1	Same	0.03	4	43	70	850	H+C3+A
15	1	Same	0.03	3.6	43	70	850	A[11.01]+E
16	1	Same	0.03	3.8	45	70	850	D1
17.	2	Same	0.03	3.2	45	70	850	A [10.91]
18	3	Same	0.03	3.2	47	70	850	G
19	3	Same	0.03	3.5	45	70	850	A [11.73s] + G
20	3	Same	0.03	3.5	43	70	850	A [11.84s]
21	4	Same	0.03	3.5	43	70	850	A [11.51s]
22	1	Same	0.03	3.5	43	70	850	A [11.84s]
23	1	Same	0.03	3.5	45	70	850	A [11.61s]
24	1	Same	0.03	3.2	45	70	850	A [11.53s]
25	1	Same	0.03	3.2	55	70	850	F
26	1	Same	0.03	2.5	55	70	850	F
27	1	Same	0.03	2.0	55	70	850	C1
28	1	Same	0.03	2.3	55	70	850	E
29	1	Same	0.03	2.3	55	70	800	C1
30	1	Same	0.04	2.3	55	70	800	C3
31	1	Same	0.035	2.3	55	70	800	C3
32	1	Same	0.033	3.5	50	70	800	C1
33	1	Same	0.033	3.7	50	70	800	B
34	1	Same	0.02	2.3	50	70	800	C1
35	1	Same	0.03	3.2	48	70	800	A [11.49s]
36	1	Same	0.03	3.2	42	70	800	A [12.38s]
37	1	Same	0.03	3.2	44	140	500	E

38	1	Same	0.03	3.2	44	140	360	A [10.78]
39	1	Same	0.03	3.2	47	140	360	A [10.23]
40	1	Same	0.03	3.2	47	210	250	E
41	1	Same	0.03	3.2	47	170	300	A [10.31s]
42	1	Same	0.05	3.0	70	170	300	C1
43	1	Same	0.05	3.9	70	170	300	F
End of Trials: Eventually went for trial 24 but with 170 donut and 300 delay								

**Trials after 24 are attempts made for speeding up the car*
**The speed-up turned out to be unsuccessful so trial 23's parameters with faster donut was used during race day*

Table 2: Data from trials to determine PD, wheel speeds
Source: Yu-Wei Vincent Yeh

Philip Giang's Tests and Final Values

Trial	Position	Fusion Coeff	Control Coeff		Speeds		Donut Delay	Result/Time
			Kp	Kd	Base	Donut		
1	1	±8, 4, 2, 1	.02	3.2	50	70	850	C1
2	1	Same	.025	3.2	50	70	850	C3
3	1	Same	.030	3.2	50	70	850	C2
4	1	Same	.035	3.2	50	70	850	C2
5	1	Same	.025	4.0	50	70	850	C2
6	1	Same	.025	3.5	50	70	850	A [14.53]
7	1	Same	.025	3.55	50	70	850	A [15.65]
8	1	Same	.025	3.45	50	70	850	A [13.85]
9	1	Same	.025	3.30	50	70	850	C3
10	1	Same	.025	3.45	50	70	850	D3
11								D3
12								A [13.24]

13	1	Same	.025	3.45	45	70	850	A [13.88]
14								A [13.43]
15								A [13.65]
16	2	Same	.025	3.45	45	70	850	A [13.94]
17								A [13.82]
18	3	Same	.025	3.45	45	70	850	A [14.24]
19								A [13.64]
20	4	Same	.025	3.45	45	70	850	A [14.05]
21								A [14.71]
22	1	Same	.025	3.45	47	70	850	A [12.98]
23								A [13.18]
24	2	Same	.025	3.45	47	70	850	A [12.86]
25								A [13.21]
26	3	Same	.025	3.45	47	70	850	A [12.61]
27								A [12.76]
28	4	Same	.025	3.45	47	70	850	A [12.98]
29								A [12.45]
30	1	Same	.025	3.45	47	170	300	A [11.28]
31								A [11.53]
End of Trials: Eventually went for trial 30								

**Trials with a lack of variable input copy the same variables from the above test (i.e. repeat of above test)*
**Uses same legend as above for results*

Table 3: Data from trials to determine Kp, Kd values for user's own car, based on trials done in (Table 2).
Source: Philip Giang

Result of each run is selected from following scenarios:
A. [Time] (Means succeeded)
B. Totally destabilized without reaching the bended portion
C1. Ran out of track (or almost) when encountered the bend due to insufficient steering (forward)

C2. Ran out of track (or almost) when encountered the bend due to over steering (forward)
C3. Ran out of track (or almost) in the middle of the bend (forward)
C4. Ran out of track (or almost) right after finishing the bend (forward)
D1. Ran out of track (or almost) when encountered the bend due to insufficient steering (back)
D2. Ran out of track (or almost) when encountered the bend due to over steering (back)
D3. Ran out of track (or almost) in the middle of the bend (back)
D4. Ran out of track (or almost) right after finishing the bend (back)
E. Destabilized due to the donut
F. Ran out of track (or almost) right after the donut
G. Ran out of track (or almost) in the beginning (location 3)
H. Ran over the cross piece section

d. Test Data Interpretation

As shown in Table 2, we've noticed some correlation between the ratio between Kp and Kd with respect to the success of our trial runs. For trials such as #6, #8, and #19, we realized that ratios of around 100 could produce more feasible parameters for our algorithm. (Although later, right before the race day, we did encounter parameters with much smaller Kd to Kp ratio that would lead to smoothness and stability, but we didn't have time to tune those parameters) For earlier runs with greater speeds (70), the car rarely behaved stably and would often get destabilized, which lead to the decision of lowering the speed to ensure that the car actually runs. As for the Kd values, one of the problems we encountered was when Kd value was too large and the car diverged significantly from the track, not only would the P term drive the car to a greater speed, the error change would also become too great as a result. Thus, this leads to smaller Kd values used in later trials. The donut speed that we used was rarely changed, at least not until later runs, since we were only interested in the behavior along the track. In later runs, a higher donut speed as well as the delay associated with it were determined for faster runs.

For trials done in Table 3, trials #8, #11, #12, #13 show varying results with the same parameters; while the Kp, Kd, and speed parameters worked for #8 and #12, they would give inconsistent results and the car would derail. This meant that the configuration was unstable, which was fixed by lowering the speed to a certain floor where the PD controller would work with those values and then slowly incrementing the speed.

From the data collected in the table, we eventually decided that the appropriate Kp value would be between 03~0.4, the appropriate Kd value would be between 3.2~4.0, and the appropriate base speed for the above parameters would be between 43~47.

Results and Discussion

a. Test Discussion:

Our results did align with our project goals in terms of getting a car to traverse the track in a timely fashion. However, while we did manage to get a stable range for a certain speed, we could have gotten a faster result if we had tested and determined the correct ranges of Kp and Kd for higher speeds. The results of our testing showed that the variables worked together when in a certain range of each other, i.e. a set Kp and Kd would work for a certain speed range and the time could be optimized based on that

range. Given this information, we could limit the amount of testing needed to be done for given ranges of either speed, Kp, or Kd. More work is needed in order to accurately determine the exact effect of Kp and Kd on how well the car reacts to changes in the path for certain speeds; although we did notice that a higher Kd value would sometimes result in a quicker deviation from the track, we would need to test more to see this effect in tandem with changes in Kp. Furthermore, as mentioned in various points in our report, having a more balanced Kp to Kd ratio could yield smoother runs, which would require more thorough testing. Some trials resulted in the car successfully traversing the path going forwards, but veering off at the bend when returning to the starting point. This means that we would also need to work more on testing exactly why this result occurred, i.e. replicating this issue until we figured the culprit variables that caused an issue when returning.

b. Race Day Discussion:

i. Race day result:

On the race day, we achieved results of 10.8 and 11.2 seconds respectively. For both partners' cars, the first trial was unstable with the cars veering off the bend, but this was expected since we encountered similar problems during our testing - the cars usually behave weirdly after being inactive for a while. In the end, our cars were able to complete the runs at two random locations successfully without much distablization on the bend or at the cross piece section.

ii. Limitations:

Some of the limitations of our code include the lack of the I component for PID control that caused the car to wobble and the inflexibility to account for calibration values under different conditions - those values were only set to fit the condition of our testing environment. Although the car runs loop() about 150 times each second, it would still be better, although almost impossible, for it to scan at a rate closer to continuous sampling. For our track, one of the limitations would be its overly steep turn that limits the car to run at a lower speed - however, this is supposed to be the intent of this project. We initially encountered a problem where our car would accidentally scan the wrinkles and gaps between the papers, causing it to do a donut halfway. However, this issue was later fixed by making the requirements for doing donuts more strict - we eventually let the cars scan the cross piece two additional times before doing a donut.

iii. What we would do differently:

During the development process, we wasted a significant amount of time testing on codes with false logics that we misimplemented. If we have a chance to redo the project, we would focus more on going over the design of our code instead of going straight to testing. Furthermore, after making sure that our code has correct logic and choosing a sensible Kp value, we only varied Kd value to fit that particular Kp. Thus, the control system that we implemented uses an extremely large Kd value to respond to the error changes, and Kp in our system only does minor work in steering.

Towards the end, we realized that letting our P term contribute more to the steering and D term contribute less (relative to our current parameter), actually improved the smoothness of our runs, but it was already too late to test for perfect values. Thus, if we were to conduct the project differently, we would attempt to find more sets of

parameters that work instead of only fixing one variable and adjusting the others accordingly. Lastly, even though we used the concept of PID control to implement our algorithm, we did manipulations on right and left speed separately with different values. Therefore, we would try to create a single steering command if we get a chance to do this project again.

Conclusions and Future Work

In conclusion, our design did meet our goal as the car was able to complete the path without anomalies. With this project, we've learned the essence of scientific approach towards project development through constant failing and adjusting parameter values as well as the code itself. Furthermore, PID control is an idea that we've never encountered prior to this course, and this project did teach us a lot about the subject - the fact that sensor values could lead to such an application was indeed fascinating.

For this project, as mentioned in the previous section, we've only implemented the P and D component of the PID controller since P and D are sufficient for our car to run on the track efficiently. Also, despite having successfully made the car follow the path under 11 seconds, our overall time was still relatively unimpressive when compared to some other groups'. For future work, we would like to implement the I term for the smoothness of the control system and possibly discover more suitable parameters for our control system for faster speed and better reaction time. Furthermore, during the development process, we attempted to make the base speed varying instead of fixed to increase stability but eventually failed. Thus, we would also like to explore the possibility of having a helper function that could determine our base speed. Ultimately, we intend to implement our algorithm with advanced optimization methods such as concave and convex optimization. [7]

References

[1]https://www.omega.com/en-us/resources/how-does-a-pid-controller-work#:~:text=PID%20Controller%20Working%20Principle,and%20applied%20to%20the%20input.
[2] https://en.wikipedia.org/wiki/PID_controller
[3]ECE 3 Fall 2020 Lab Section 5 Notes – Feedback Control
[4]Professor Briggs Lecture/Notes: RSLK PATH DETECTION CIRCUIT
[5]Professor Briggs Lecture/Slides: ECE 3 Project Tips
[6]Professor Briggs Lecture/Slides: A Conceptual Description of the PID Controller
[7]https://web.stanford.edu/~boyd/papers/pdf/pid_tuning_ecc.pdf

Code

```
#include <ECE3.h>
#include <cmath>
uint16_t sensorValues[8]; // right -> left, 0 -> 7
float fsenseVals[8];
const float offsetweight[8] = {-8,-4,-2,-1,1,2,4.5,8};
const float minVal[8] = {820, 770, 680, 620, 620, 620, 690, 670};
const float maxVal[8] = {1750, 1780, 1500, 1270, 1300, 1650,
1680,1800};
int leftSpd;
int rightSpd;
int basePWM = 70;
float Kp = 0.03;
float Kd = 3.2;
float errorChange;
float lastVal;
int somespeed = 0;
int donutCount = 0;
const int left_nslp_pin=31;
const int left_dir_pin=29;
const int left_pwm_pin=40;
const int right_nslp_pin=11;
const int right_dir_pin=30;
const int right_pwm_pin=39;

const int LED_RF = 41;

////////////////////////////////////
void setup() {
  pinMode(left_nslp_pin,OUTPUT);
  pinMode(left_dir_pin,OUTPUT);
  pinMode(left_pwm_pin,OUTPUT);

  digitalWrite(left_dir_pin,LOW);
  digitalWrite(left_nslp_pin,HIGH); //let left wheel on

 pinMode(right_nslp_pin,OUTPUT);
  pinMode(right_dir_pin,OUTPUT);
  pinMode(right_pwm_pin,OUTPUT);

  digitalWrite(right_dir_pin,LOW); //high voltage or low voltage
  digitalWrite(right_nslp_pin,HIGH);

  pinMode(LED_RF, OUTPUT);

  ECE3_Init();
  // set the data rate in bits/second for serial data transmission
  //Serial.begin(9600);
  delay(2000); //Wait 2 seconds before starting
  errorChange = 0;
  lastVal = 0;
```

```
}

void loop() {
  leftSpd = 170;
  rightSpd = 170;
  basePWM = 45;
  digitalWrite(right_dir_pin,LOW);
  float weighted = 0;
  ECE3_read_IR(sensorValues);
  int endingDonut = 0;
  for (unsigned char i = 0; i < 8; i++)
  {
    fsenseVals[i] = sensorValues[i]; //convert uint to float
    fsenseVals[i] -= minVal[i];
    fsenseVals[i]/= maxVal[i];
    fsenseVals[i] *= 1000.00;
    if (fsenseVals[i] >= 950){
      endingDonut ++;
    }
    weighted += fsenseVals[i]*offsetweight[i];
  }
  //obtained the weighted error value of sensors
  weighted/=4.0;
  errorChange = weighted - lastVal;
  somespeed = Kd*(errorChange);
  if(endingDonut >= 7){
    donutCount++;
    //if sensed the cross section twice, do a donut
    if(donutCount == 2){
      digitalWrite(right_dir_pin,HIGH);
      analogWrite(left_pwm_pin,leftSpd); //0-255
      analogWrite(right_pwm_pin,rightSpd);
      delay(300);
    }
    else if (donutCount > 2){ //if sensed the cross section again,
then stop
      digitalWrite(left_nslp_pin,LOW);
      digitalWrite(right_nslp_pin,LOW);
    }
  }else{
    //code for the four different scenarios described in the flow
diagram
    if (weighted > 0){
      if (errorChange > 0){
        rightSpd = basePWM+somespeed;
        basePWM -= Kp*weighted;
        leftSpd = basePWM;

      }else{
         basePWM += Kp*weighted;
         rightSpd = basePWM;
         leftSpd = basePWM+somespeed;
```

```
    }
  }
  else {
    if (errorChange < 0){
      leftSpd = basePWM-somespeed;
      basePWM += Kp*weighted;
      rightSpd = basePWM;
    }else{
      basePWM -= Kp*weighted;
      leftSpd = basePWM;;
      rightSpd = basePWM-somespeed;
    }
  }
}
//update the previous error value to obtain error change
lastVal = weighted;
analogWrite(left_pwm_pin,leftSpd); //0-255
analogWrite(right_pwm_pin,rightSpd);
digitalWrite(LED_RF, HIGH);
digitalWrite(LED_RF, LOW);
}
```