Selim Emir Can

ECE3 22Su Final Report Content

Develop

The experimenters' plan was to execute the following:

1. calibrate the IR sensor fusion
2. implement the feedback system to control the car-road alignment
3. implement the logic that makes the car perform a donut and makes it stop at the end of the track.

If adding a new piece of code led to undesired behavior or desired but unplanned behavior, we decided that our code needed improvement. If adding a new piece of code led to the desired behavior, we decided that our code was correct and moved on.

Conduct

The parameters that we controlled in our code during our test runs were the starting point/orientation of the car, the conditions for detecting a crosspiece, base speed of the car, the time interval for which the car performs a donut, the coefficient of derivative control (K_d), and the coefficient of proportional control (K_p).

The controlled variables in our code were the FirstCrossPiece (boolean variable), SecondCrossPiece (boolean variable), DoDonut (boolean variable), DidDonut (boolean variable), left wheel speed and the right wheel speed of our car, our error value, and our error difference value (the formula is deterministic but the actual speed, error, error difference values are not).

The measured variables in our code during our final test run were the Infrared light hitting the IR sensors on the front of the car.

The IR sensors were calibrated and used to estimate proper K_d and K_p values. The pictures in figures 1 and 2 were printed with a laser printer. The IR_Sensor Example code from the ECE3 library was uploaded to the car. IR measurements were taken at each tick mark using the serial monitor. The distance from the center was calculated using the fact that the distance between each tick mark is 2cm.

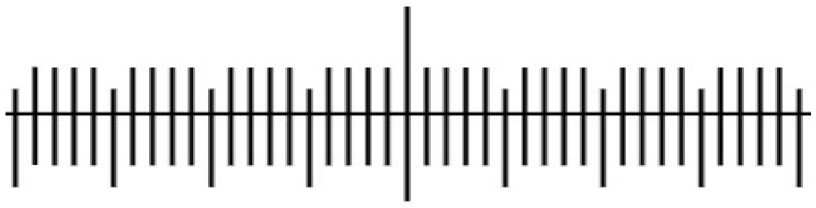


Figure 1: A calibration strip with tick marks – each 2cm apart.

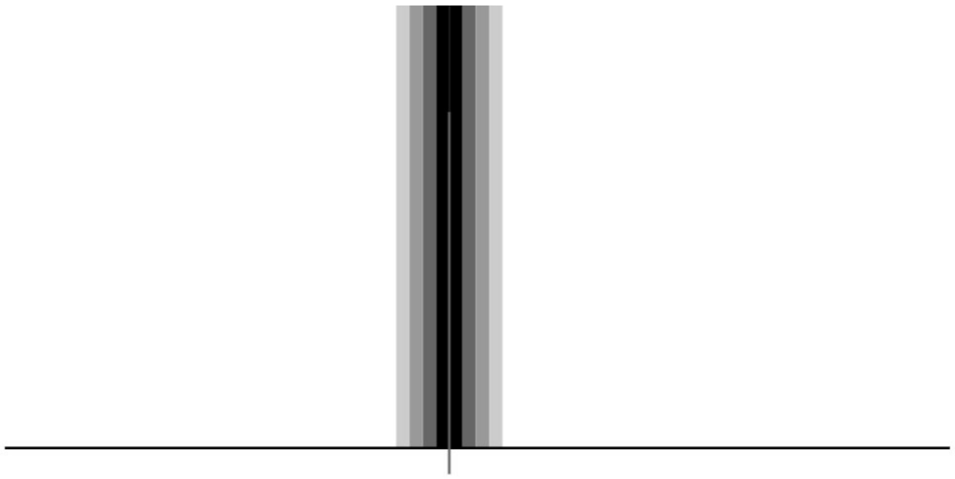


Figure 2: A calibration strip that replicates a typical track with no wears.

Distance from center of lane (cm)	IR Reading							
	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8
-40	1367.20	597.00	621.00	621.00	621.00	551.00	597.00	597.00
-36	2500.00	597.00	621.00	621.00	621.00	551.00	597.00	621.00
-32	2500.00	838.60	620.00	620.00	620.00	551.00	597.00	620.00
-28	1291.60	1809.80	620.00	620.00	620.00	551.00	597.00	620.00
-24	865.80	2409.80	695.60	629.60	634.40	555.60	615.40	620.00
-20	809.00	998.00	1568.00	620.00	620.00	551.00	610.80	620.00
-16	785.00	691.00	2472.00	667.00	644.00	555.60	620.00	620.00
-12	785.00	620.00	1547.80	1205.40	644.00	574.00	620.00	620.00
-8	785.00	620.00	851.20	2380.80	620.00	574.00	620.00	620.00
-4	785.00	597.00	667.00	2038.40	884.60	574.00	620.00	620.00
0	785.00	597.00	653.20	1025.80	1710.00	574.00	620.00	639.20
4	785.00	597.00	638.60	667.00	2229.20	695.00	620.00	633.80
8	785.00	597.00	620.00	643.00	1267.60	1186.00	620.00	643.00
12	784.60	597.00	620.00	643.00	770.20	2228.80	667.00	643.00
16	785.00	597.00	620.00	620.00	667.00	1595.80	1058.80	643.00
20	747.40	477.73	620.00	620.00	667.00	832.00	2138.00	643.00
24	785.00	597.00	620.00	620.00	644.00	597.00	2115.00	903.00
28	785.00	597.00	624.80	620.00	644.00	574.00	993.20	1886.40
32	786.00	597.00	620.00	620.00	620.00	551.00	668.00	2500.00
36	783.00	598.60	621.80	621.80	621.80	552.60	641.00	1343.80
40	787.00	597.00	620.00	620.00	620.00	551.00	620.00	763.00
Minimum Sensor Reading	747.40	477.73	620.00	620.00	620.00	551.00	597.00	597.00
Maximum Sensor Reading - Minimum Sensor Reading	1752.60	1932.07	1852.00	1760.80	1609.20	1677.80	1541.00	1903.00

Table 1: IR Sensor measurements taken for calibration

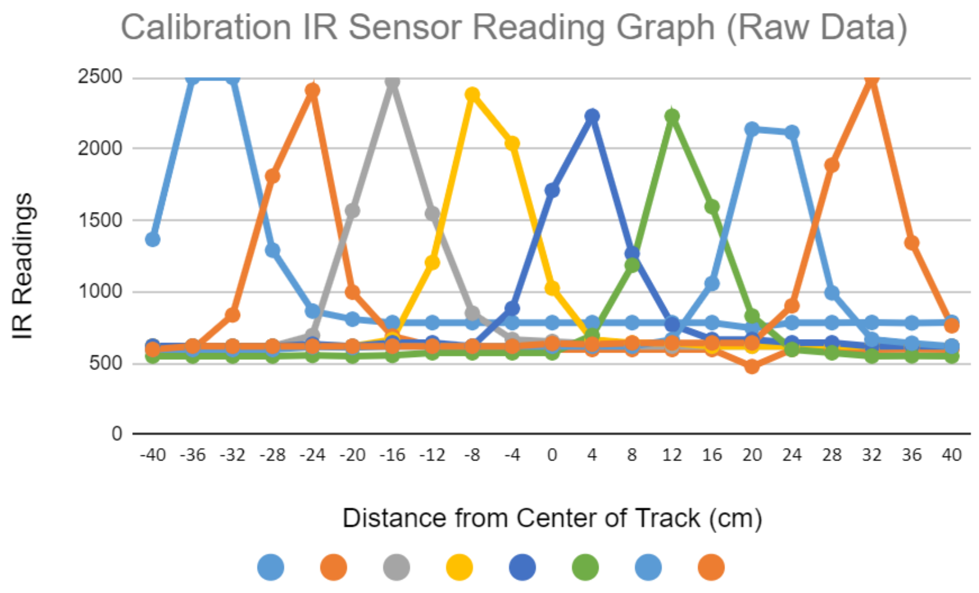


Figure 3: Graph of data in table 1. Each color corresponds to a sensor on the car.

	IR Reading								Error Term	
Distance from center of lane (cm)	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8	(15-14-12-8)/8	(8-4-2-1)/4
-40	353.65	61.73	0.54	0.57	0.62	0.00	0.00	0.00	-771.88	130.95
-36	1000.00	61.73	0.54	0.57	0.62	0.00	0.00	12.61	-1960.14	243.54
-32	1000.00	186.78	0.00	0.00	0.00	0.00	0.00	12.09	-2179.20	305.39
-28	310.51	689.45	0.00	0.00	0.00	0.00	0.00	12.09	-1766.09	383.35
-24	67.56	1000.00	40.82	5.45	8.95	2.74	11.94	12.09	-1886.73	528.61
-20	35.15	269.28	511.88	0.00	0.00	0.00	8.96	12.09	-1266.63	614.31
-16	21.45	110.39	1000.00	26.69	14.91	2.74	14.93	12.09	-1692.29	1071.94
-12	21.45	73.64	500.97	332.46	14.91	13.71	14.93	12.09	-1168.75	1165.08
-8	21.45	73.64	124.84	1000.00	0.00	13.71	14.93	12.09	-1287.01	2123.02
-4	21.45	61.73	25.38	805.54	164.43	13.71	14.93	12.09	-758.10	1627.69
0	21.45	61.73	17.93	230.46	677.36	13.71	14.93	22.18	360.00	430.68
4	21.45	61.73	10.04	26.69	1000.00	85.83	14.93	19.34	1001.10	25.06
8	21.45	61.73	0.00	13.06	402.44	378.47	14.93	24.17	880.26	-31.65
12	21.23	61.73	0.00	13.06	93.34	1000.00	45.43	24.17	1557.26	-32.70
16	21.45	61.73	0.00	0.00	29.21	622.72	299.68	24.17	1384.78	-59.77
20	0.00	0.00	0.00	0.00	29.21	167.48	1000.00	24.17	2075.75	-97.00
24	21.45	61.73	0.00	0.00	14.91	27.42	985.07	160.80	1933.16	-581.77
28	21.45	61.73	2.59	0.00	14.91	13.71	257.11	677.56	1603.69	-2546.98
32	22.02	61.73	0.00	0.00	0.00	0.00	46.07	1000.00	1806.30	-3777.63
36	20.31	62.56	0.97	1.02	1.12	0.95	28.55	392.43	638.28	-1463.22
40	22.60	61.73	0.00	0.00	0.00	0.00	14.93	87.23	39.28	-305.48

Table 2: IR Sensor measurements with the minimum subtracted and the maximum value normalized to 1000 and the resulting error term.

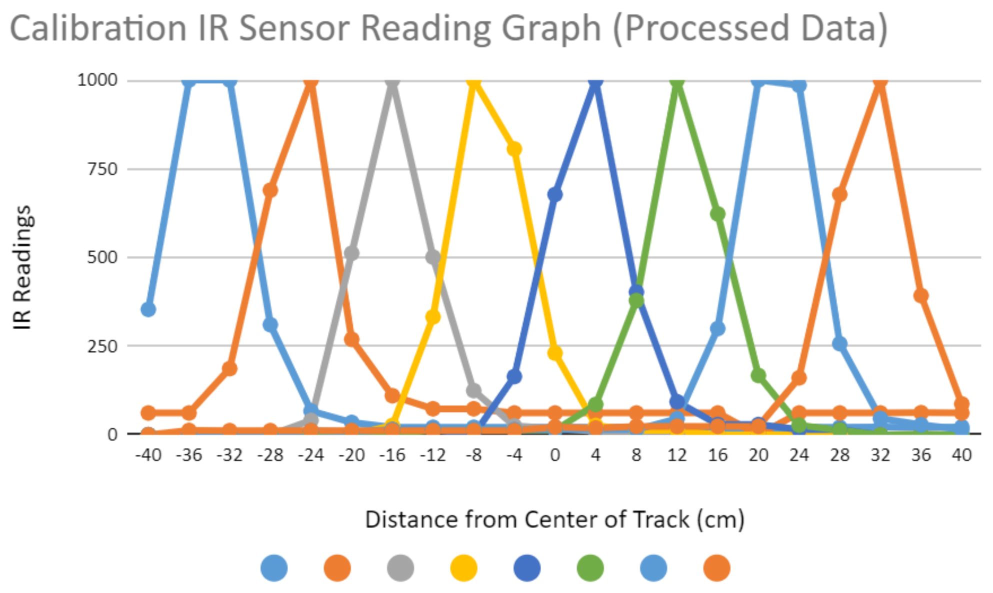


Figure 4: Graph of IR Reading data in table 2. Each color corresponds to a sensor on the car.

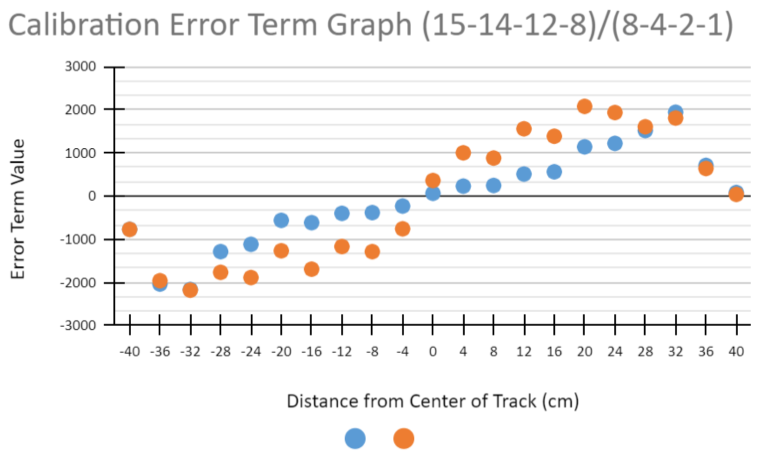


Figure 5: Graph of errors in table 2. The blue points are from the 8-4-2-1 scheme and the orange points are from the 15-14-12-8 scheme

15-14-12-8 scheme:

Error = $((\text{sensorValues}[7] - \text{sensorValues}[0]) * 15 + (\text{sensorValues}[6] - \text{sensorValues}[1]) * 14 + (\text{sensorValues}[5] - \text{sensorValues}[2]) * 12 + (\text{sensorValues}[4] - \text{sensorValues}[3]) * 8)/8$

8-4-2-1 scheme:

Error = $((\text{sensorValues}[7] - \text{sensorValues}[0]) * 8 + (\text{sensorValues}[6] - \text{sensorValues}[1]) * 4 + (\text{sensorValues}[5] - \text{sensorValues}[2]) * 2 + (\text{sensorValues}[4] - \text{sensorValues}[3]) * 1)/4$

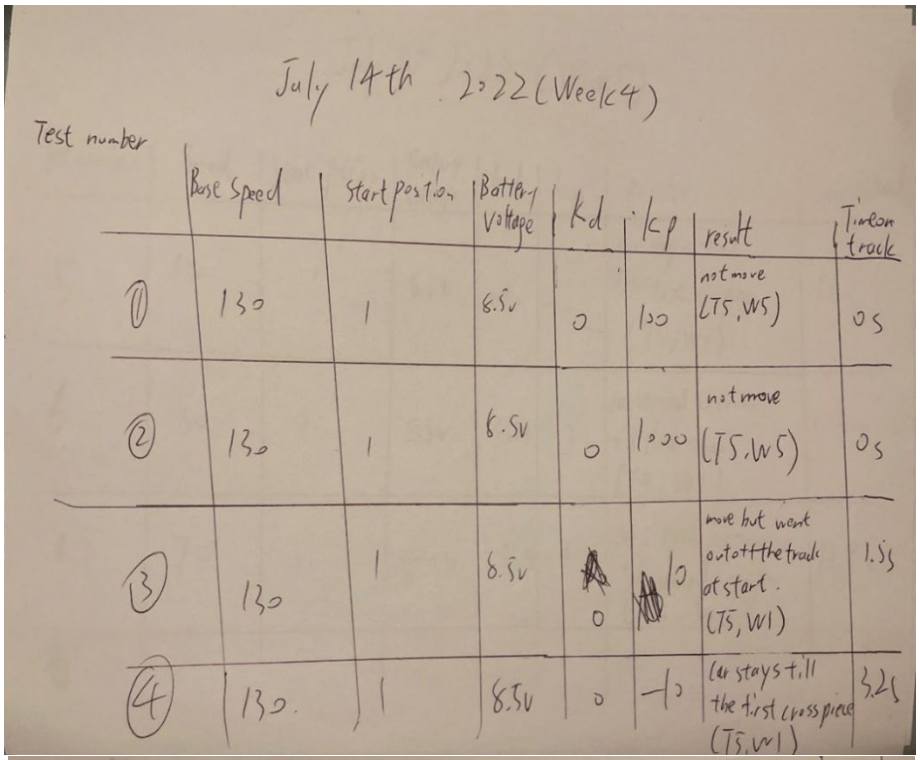
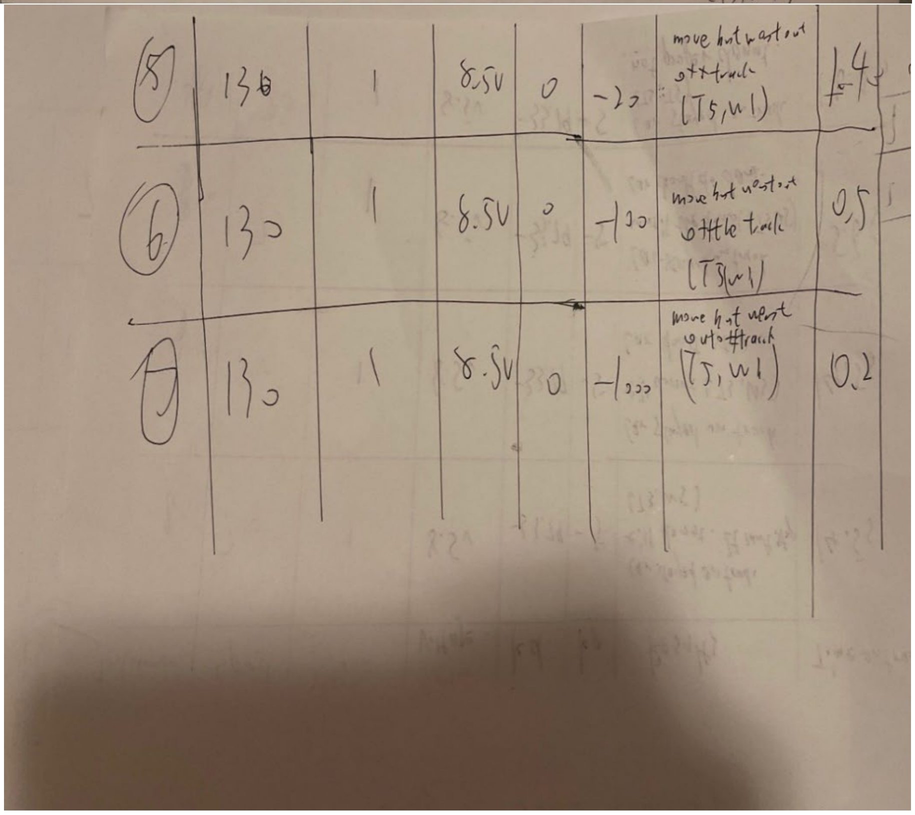
Table 1 and figure 3 shows the raw data we collected from the sensors while table 2 and figure 4 shows the data after the minimum IR reading for each sensor is subtracted from all the readings in that column and the resulting maximum value is used to normalize each column to an IR Reading of a 1000. Table 2 and figure 5 also show the resulting error curve for the 8-4-2-1 scheme and the 15-14-12-8 scheme. In figure 5, a positive error term means that the car is tilted/misaligned to the left side of the track and a negative error term means that the car is tilted/misaligned to the right side of the track. An ideal car would have close to zero error when it is perfectly aligned with the track (or off the track which the code should account for).

Note that the first black calibration strip we used was worn off/faded which caused error in our initial calibration. However, the calibration process was repeated with a better calibration strip and the data displayed above was obtained. After calibration was done, a feedback system that allows the car to follow the track was implemented. The parameters that were being used for the feedback system were the coefficient of derivative control, K_d, and the coefficient of proportional control, K_p. Test logs were collected and are given below along with a legend that describes the overall quality of the test runs.

W1	significant amount of weaving, car cannot stay on track
W2	a lot of weaving, car can barely stay on track
W3	fewer weaving, weaving hinders the effective speed
W4	little weaving, car operates relatively smooth
W5	no weaving (only weaves if there is big misalignment to realign the car)
T1	car turns very well, but the speed should not be increased further
T2	car turns well, the speed of the car should not be increased higher [than 130/255 PWM]
T3	the car turns sufficiently well; the speed can be further increased to reasonable value if wanted
T4	the car cannot turn very well; the speed should be lowered if reasonable
T5	the car cannot turn at all

Table 3: Legend used in determining the quality of the test run results

Test Log 1

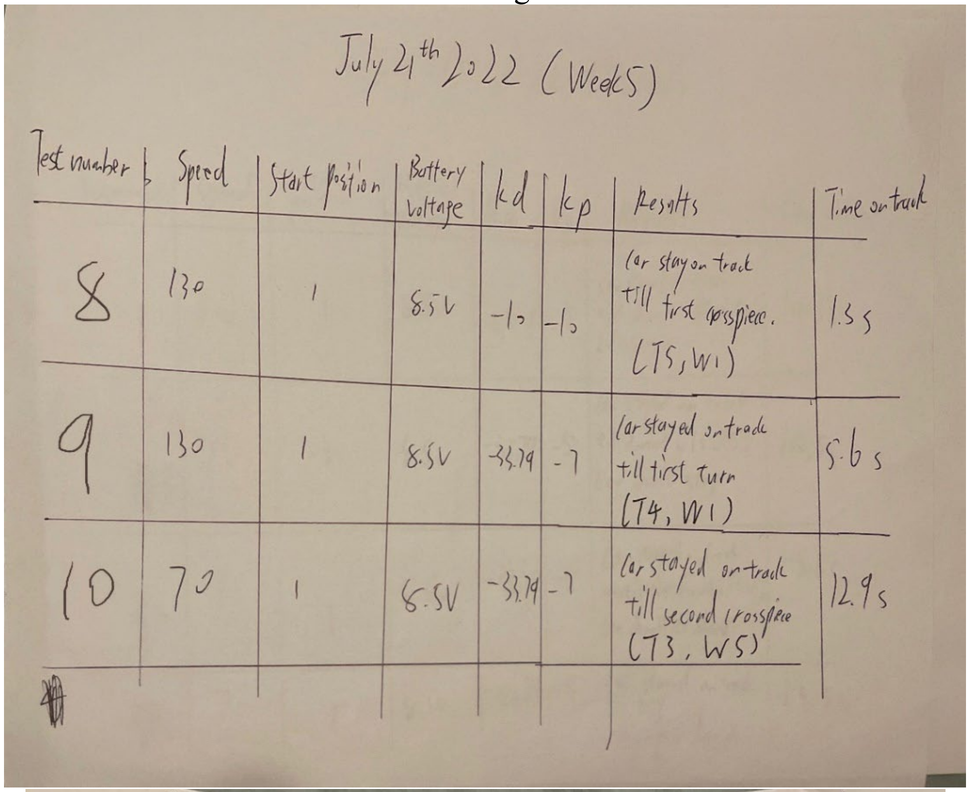
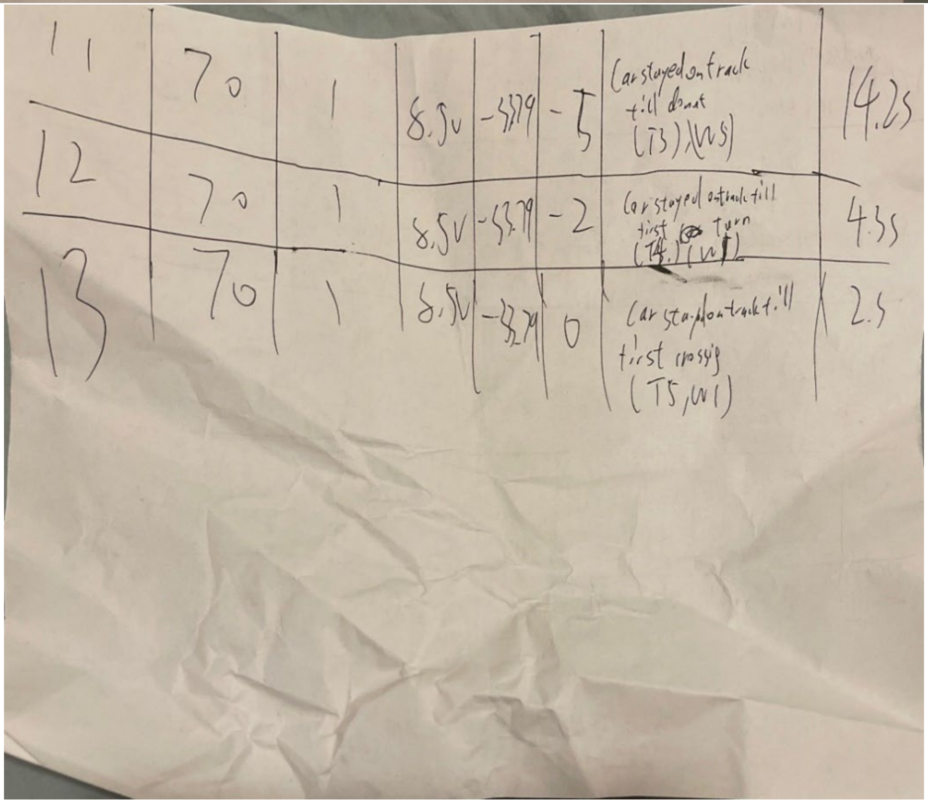



July 14th 2022 (Week4)

Test number	Base Speed	Start position	Battery Voltage	kd	kp	result	Time on track
1	130	1	8.5v	0	100	not move (T5, W5)	0s
2	130	1	8.5v	0	1000	not move (T5, W5)	0s
3	130	1	8.5v	0	10	move but went out off the track at start. (T5, W1)	1.5s
4	130.	1	8.5v	0	-10	car stays till the first cross piece (T5, W1)	3.2s
5	130	1	8.5v	0	-20	move but went out off track (T5, W1)	1.4s
6	130	1	8.5v	0	-100	move but went out off the track (T5, W1)	0.5
7	130	1	8.5v	0	-1000	move but went out off track (T5, W1)	0.2

Table 4: This table shows the test run parameters, results, and the time the car spent on the track on 14th of July 2022.

Test Log 2

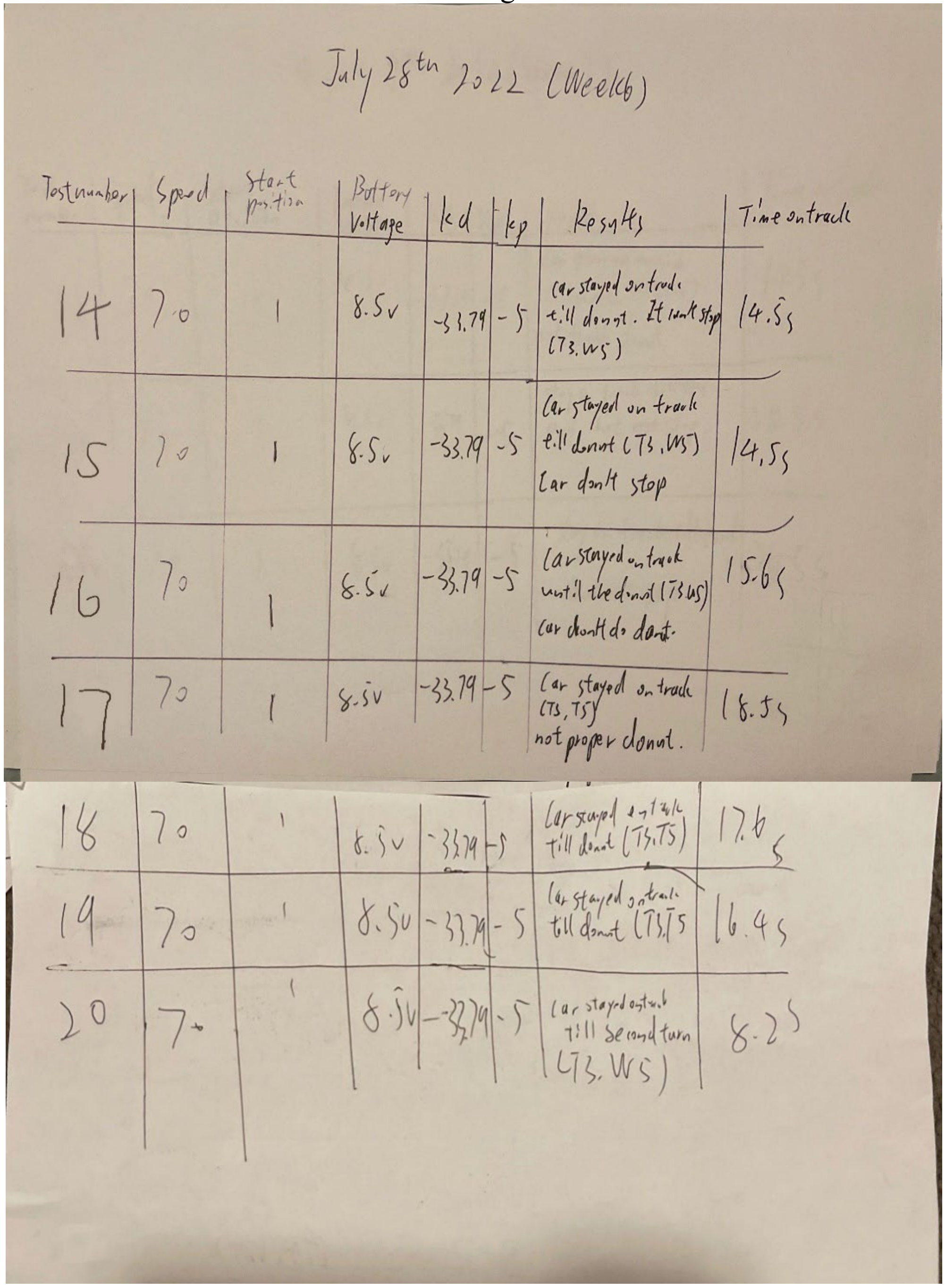



July 21th 2022 (Week5)

Test number	Speed	Start position	Battery voltage	kd	kp	Results	Time on track
8	130	1	8.5V	-10	-10	Car stay on track till first crosspiece. (T5, W1)	1.3 s
9	130	1	8.5V	-33.79	-7	Car stayed on track till first turn (T4, W1)	5.6 s
10	70	1	8.5V	-33.79	-7	Car stayed on track till second crosspiece (T3, W5)	12.9 s
11	70	1	8.5V	-33.79	-5	Car stayed on track till donut (T3), (W5)	14.2 s
12	70	1	8.5V	-33.79	-2	Car stayed on track till first turn (T4) (W1)	4.3 s
13	70	1	8.5V	-33.79	0	Car stayed on track till first crossing (T5, W1)	2 s

Table 5: This table shows the test run parameters, results, and the time the car spent on the track on 21st of July 2022.

Test Log 3

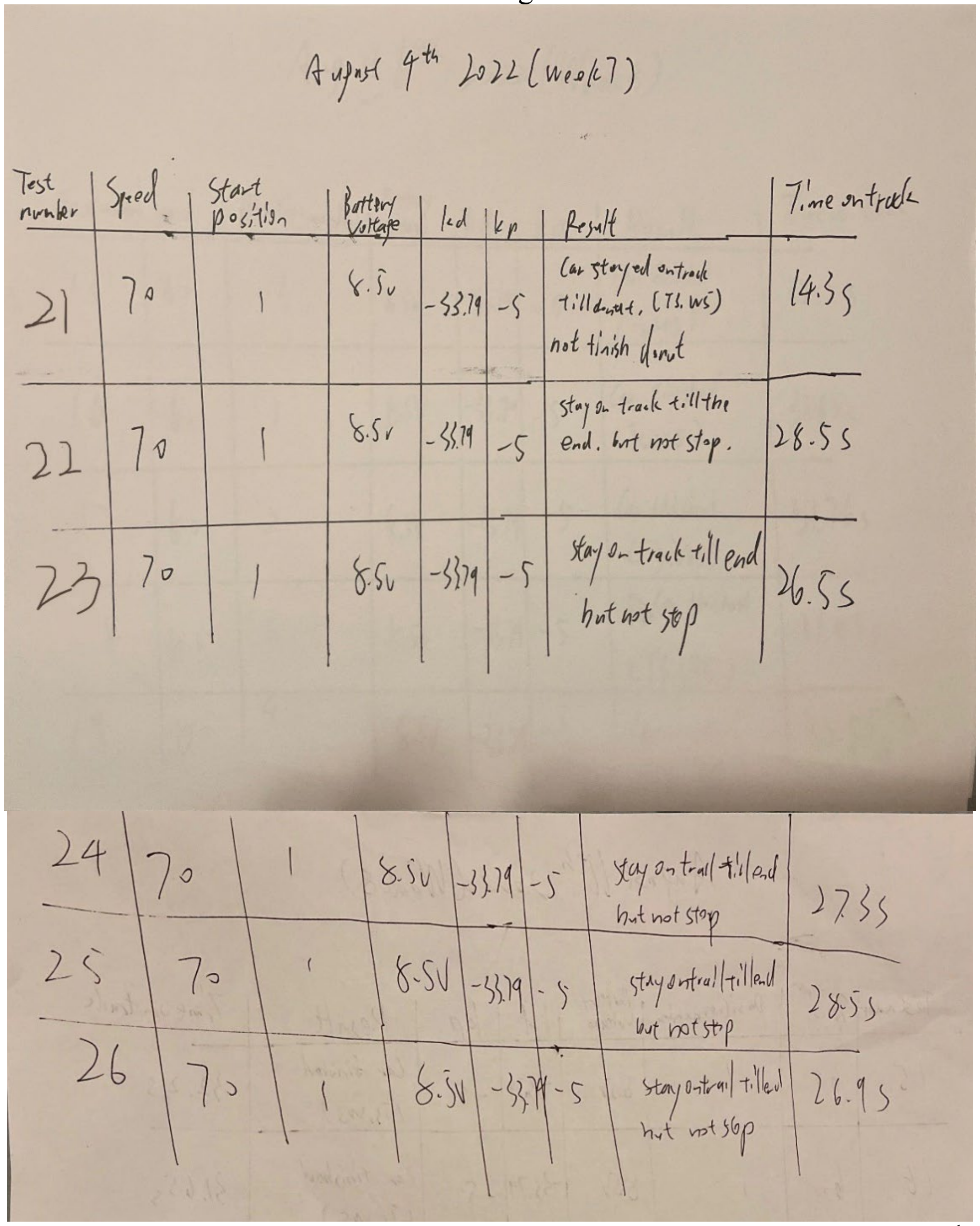


July 28th 2022 (Week6)

Testnumber	Speed	Start position	Battery Voltage	kd	kp	Results	Time ontrack
14	70	1	8.5v	-33.79	-5	car stayed on track till donut. It won't stop (T3, W5)	14.5s
15	70	1	8.5v	-33.79	-5	Car stayed on track till donut (T3, W5) Car don't stop	14.5s
16	70	1	8.5v	-33.79	-5	Car stayed on track until the donut (T3 W5) car don't do donut.	15.6s
17	70	1	8.5v	-33.79	-5	Car stayed on track (T3, T5) not proper donut.	18.5s
18	70	1	8.5v	-33.79	-5	Car stayed on track till donut (T3, T5)	17.6s
19	70	1	8.5v	-33.79	-5	Car stayed on track till donut (T3, T5	16.4s
20	70	1	8.5v	-33.79	-5	Car stayed on track till second turn (T3, W5)	8.2s

Table 6: This table shows the test run parameters, results, and the time the car spent on the track on 28th of July 2022.

Test Log 4



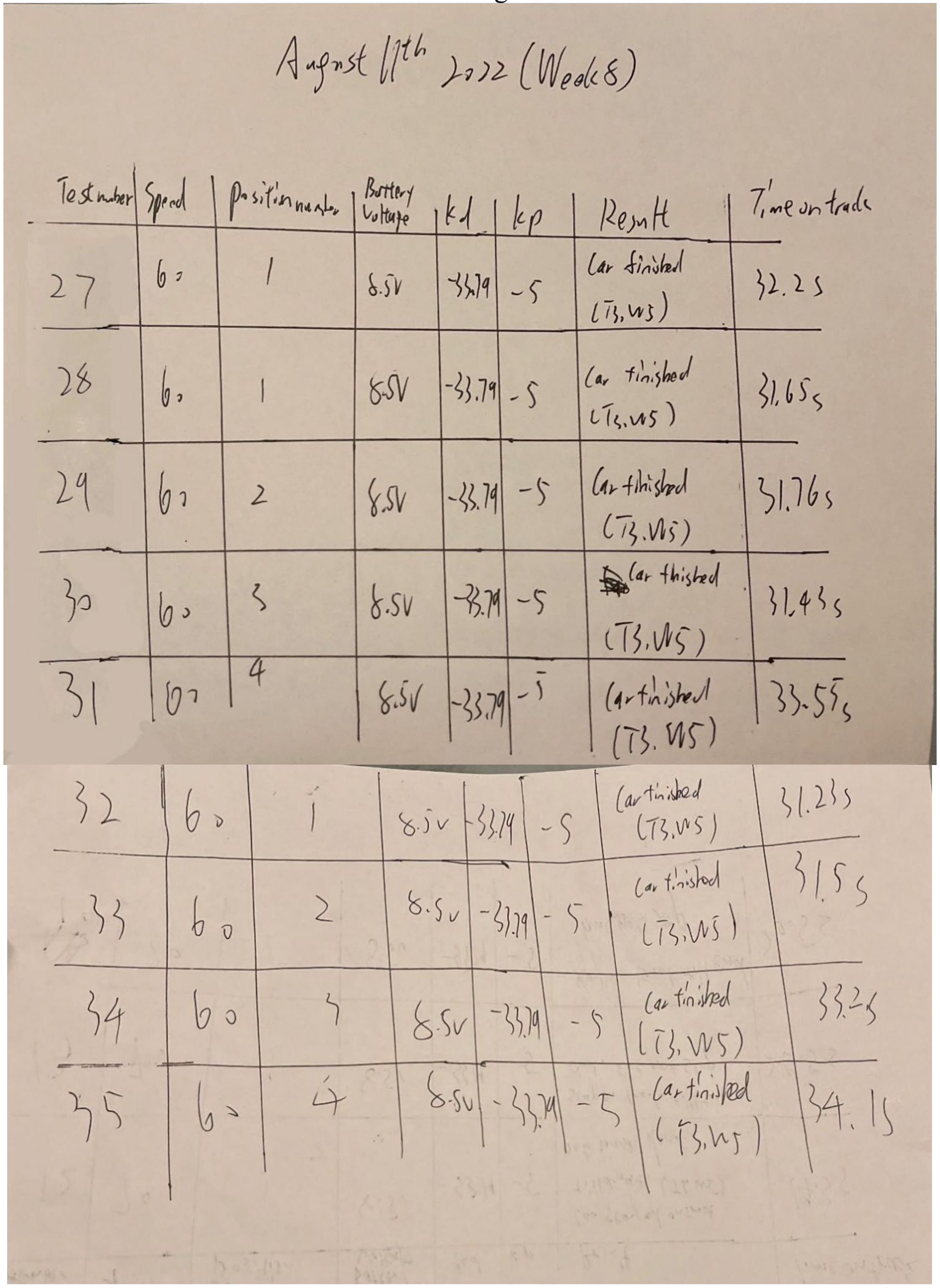
August 4th 2022 (week7)

Test number	Speed	Start position	Battery Voltage	kd	kp	Result	Time on track
21	70	1	8.5v	-33.79	-5	Car stayed on track till donut. (TS. WS) not finish donut	14.3s
22	70	1	8.5v	-33.79	-5	stay on track till the end. but not stop.	28.5s
23	70	1	8.5v	-33.79	-5	stay on track till end but not stop	26.5s
24	70	1	8.5v	-33.79	-5	stay on trail till end but not stop	27.3s
25	70	1	8.5V	-33.79	-5	stay on trail till end but not stop	28.5s
26	70	1	8.5V	-33.79	-5	stay on trail till end but not stop	26.9s

Table 7: This table shows the test run parameters, results, and the time the car spent on the track on 4th of August 2022.

Selim Emir Can

Test Log 5



August 11th 2022 (Week 8)

Test number	Speed	Position number	Battery Voltage	kd	kp	Result	Time on track
27	60	1	8.5V	-33.79	-5	Car finished (T3, W5)	32.2s
28	60	1	8.5V	-33.79	-5	Car finished (T3, W5)	31.65s
29	60	2	8.5V	-33.79	-5	Car finished (T3, W5)	31.76s
30	60	3	8.5V	-33.79	-5	Car finished (T3, W5)	31.43s
31	60	4	8.5V	-33.79	-5	Car finished (T3, W5)	33.55s
32	60	1	8.5V	-33.79	-5	Car finished (T3, W5)	31.23s
33	60	2	8.5V	-33.79	-5	Car finished (T3, W5)	31.5s
34	60	3	8.5V	-33.79	-5	Car finished (T3, W5)	33.2s
35	60	4	8.5V	-33.79	-5	Car finished (T3, W5)	34.1s

Table 8: This table shows the test run parameters, results, and the time the car spent on the track on 11th of August 2022.

Analyze

Certain test runs were taken from the test logs and organized into a table to analyze and interpret the data (For convenience we re-numbered the test runs and will be referring to these numbers only in the Interpret section):

Data Entry 1:

14th of July 2022 (Week 4)							
Test Number	Speed	Start Position	Battery Voltage	K_d	K_p	Results	Time on Track
1	130	Position 1	8.5V	0	100	Car did not move (T5, W5)	0s
2	130	Position 1	8.5V	100		Car did not move (T5, W5)	0s
3	130	Position 1	8.5V	0	10	Car moved but went off track at the start (T5, W1)	1.5s
4	130	Position 1	8.5V	0	-10	Car stayed on track until the first crosspiece (T5, W1)	3.2s

Table 9: This derived table shows some of the test log data selected from table 4 which consists of the test run parameters, results, and the time the car spent on the track on 14th of July 2022.

Data Entry 2:

21th of July 2022 (Week 5)							
Test Number	Speed	Start Position	Battery Voltage	K_d	K_p	Results	Time on Track
5	130	Position 1	8.6V	-10	-10	Car stayed on track until the first crosspiece (T5, W1)	1.3s
6	130	Position 1	8.6V	-33.79	-7	Car stayed on track until the first turn (T4, W1)	5.6s
7	70	Position 1	8.6V	-33.79	-7	Car stayed on track until the second crosspiece (T3, W5)	12.9s

Table 10: This derived table shows some of the test log data selected from table 5 which consists of the test run parameters, results, and the time the car spent on the track on 21st of July 2022.

Data Entry 3:

28th of July 2022 (Week 6)							
Test Number	Speed	Start Position	Battery Voltage	K_d	K_p	Results	Time on Track
8	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the donut (T3, W5). The car could not stop	14.5s
9	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the donut (T3, W5). The car could not stop	14.5s
10	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the donut (T3, W5). The car could not start the donut	15.6s
11	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the donut (T3, W5). The car could not properly do a donut	18.5s

Table 11: This derived table shows some of the test log data selected from table 6 which consists of the test run parameters, results, and the time the car spent on the track on 28th of July 2022.

Data Entry 4:

4th of August 2022 (Week 7)							
Test Number	Speed	Start Position	Battery Voltage	K_d	K_p	Results	Time on Track
12	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the donut (T3, W5). The car could not finish the donut	14.3s
13	70	Position 1	8.5V	-33.79	-5	Car stayed on track until the end of the track, but it could not stop. (T3, W5)	28.5s
14	70	Position 1	8.5V	-33.79	-5	Car stayed on track but did not stop at the end of the track (T3, W5)	26.5s

Table 12: This derived table shows some of the test log data selected from table 7 which consists of the test run parameters, results, and the time the car spent on the track on 4th of August 2022.

Data Entry 5:

11th of August 2022 (Week 8)							
Test Number	Speed	Start Position	Battery Voltage	K_d	K_p	Results	Time on Track
15	60	Position 1	8.5V	-33.79	-5	Car finished the whole track (T3, W5).	32.2s
16	60	Position 1	8.5V	-33.79	-5	Car finished the whole track (T3, W5).	31.65s
17	60	Position 2	8.5V	-33.79	-5	Car finished the whole track (T3, W5).	31.76s
18	60	Position 3	8.5V	-33.79	-5	Car finished the whole track (T3, W5).	31.43s
19	60	Position 4	8.5V	-33.79	-5	Car finished the whole track (T3, W5).	33.55s

Table 13: This derived table shows some of the test log data selected from table 8 which consists of the test run parameters, results, and the time the car spent on the track on 11th of August 2022.

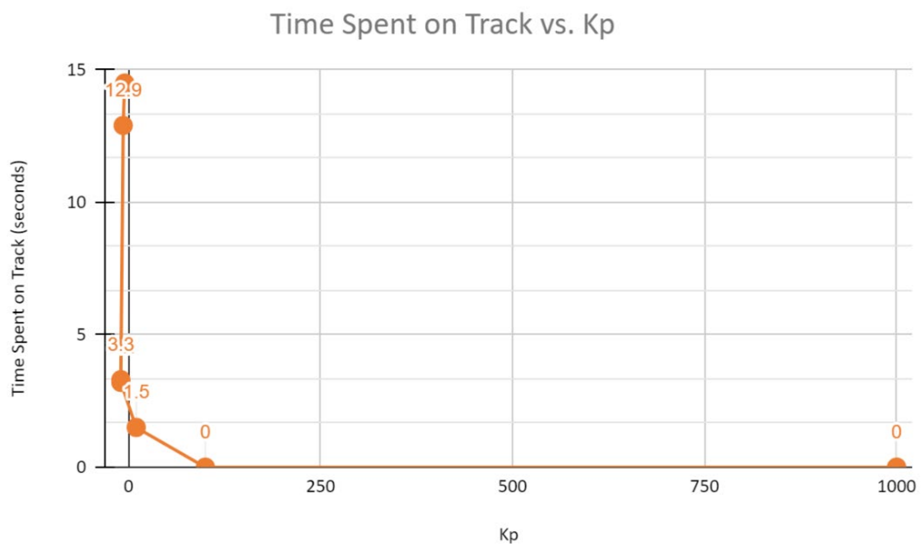


Figure 6: Graph derived from test logs displaying the relationship between K_p and time spent on track.

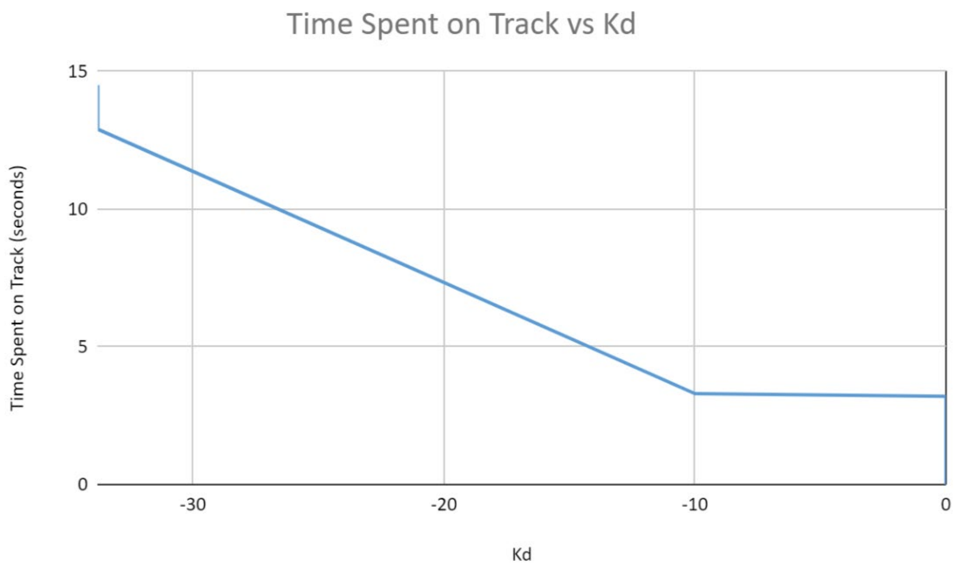


Figure 7: Graph derived from test logs displaying the relationship between K_d and time spent on track.

Interpret

Data Entry 1:

The car did not move for tests 1 and 2 because the K_d and K_p values were unreasonably high. The car immediately moved away from the track for test 3 because the sign of K_p was wrong. Because the displacement error is positive, the steering must be negative to correct the error. Similarly, the experimenters realized that the sign of K_d must be negative because the velocity error is negative, and the steering must be positive to make the derivate of the error less negative. Inferring from the calibration data and this information, the experimenters were able to get a longer test run on test 4 at the cost of significant weaving at the first cross piece. This is most likely because the code did not robustly check for crosspieces making the donut code trigger.

Data Entry 2:

The car only reached the first crosspiece on test 5 because the car was weaving due to a bad choice of K_d. The experimenters referred to the calibration data to improve the steering of the car for test 6. This time the car was spent more time on the track but due to its speed, it could not steer fast enough on the first turn. In test 7 the flawed cross piece recognition code caused the car to stop at the second crosspiece first time around the track. Crosspiece detection relied on seeing "all black" once and therefore was not robust enough to get past the crosspiece to perform a donut.

Data Entry 3:

The code used in tests 8 and 9 made the car perform a donut when it saw a white reading once immediately after a single "all black" reading. This method was not robust and resulted in the car going off the track. On the rest of the tests our car relied on seeing "all black" readings multiple times. Because the car code looked for multiple "all black" readings but not consecutive "all black" readings, the "all black" counter did not function properly when it was time to perform a donut and hence the car stopped early in tests 10 and 11.

Data Entry 4:

With the new code, the car relied on seeing "all black" readings consecutively as opposed to only seeing "all black" multiple times. This allowed the car to successfully stop for the donut but, in test 12, the car did half a donut which made it go off track. Consequently, the car was programmed using a constant donut speed and using the tick encoder to have a consistent donut in test 13. Nevertheless, the car could not stop at the end of the track because the stop code was incorrectly placed in the code where it would not trigger in both test 13 and 14.

Data Entry 5:

Once the stop code was fixed the experimenters were able to get a successful run (test 15). Different initial positions (3 and 4) were tested, and the experimenters determined the car speed needed to be decreased or otherwise the car would run off of the track at the start. Consequently, test runs 16 to 19 were performed, confirming that the car and the code (Appendix I) was ready for race day.

K_p and K_d graphs:

As it can be seen from figures 6 and 7, the ideal K_p and K_d values were negative and were around -5 and -33 respectively. When these values were positive, the time spent on the track was around 0 seconds. It should be noted that these graphs are very limited in the sense that there are few data. This undersampling implies that our relationship may not be correct and that there may be a better K_p and K_d value that makes the car perform faster and finish the track.

Appendix I: Completion Code

```
#include <ECE3.h>

uint16_t sensorValues[8]; // right -> left, 0 -> 7
const int r = 60;
const int l = 60;
const int left_nslp_pin = 31; // nslp ==> awake & ready for PWM
const int left_dir_pin = 29;
const int left_pwm_pin = 40;
int leftSpd;
int rightSpd;
const int LED_RF = 41;
const int right_nslp_pin = 11; // nslp ==> awake & ready for PWM
const int right_dir_pin = 30;
const int right_pwm_pin = 39;

int wheelSpd = 80;
int distance = 800;
///////////////////////////////////
void setup() {
    // put your setup code here, to run once:
    pinMode(left_nslp_pin, OUTPUT);
    pinMode(left_dir_pin, OUTPUT);
    pinMode(left_pwm_pin, OUTPUT);
    pinMode(right_nslp_pin, OUTPUT);
    pinMode(right_dir_pin, OUTPUT);
    pinMode(right_pwm_pin, OUTPUT);

    digitalWrite(right_nslp_pin, HIGH);
    digitalWrite(left_nslp_pin, HIGH);
    //  digitalWrite(left_nslp_pin,LOW);

    pinMode(LED_RF, OUTPUT);

    ECE3_Init();

    // set the data rate in bits/second for serial data transmission

    resetEncoderCount_left();
    resetEncoderCount_right();
    delay(2000); //Wait 2 seconds before starting
}

float error_prev = 0;
float error_dif = 0;
float K_d = -6.7586679 * 5;
float K_p = -5;
bool DoDonut = false;
bool DidDonut = false;
bool FirstCrossPiece = false;
bool SecondCrossPiece = false;
int counter = 0;

void changeWheelSpeeds(int initialLeftSpd, int finalLeftSpd, int initialRightSpd, int
finalRightSpd) {
    /*
     *   This function changes the car speed gradually (in about 30 ms) from initial
     *   speed to final speed. This non-instantaneous speed change reduces the load
     *   on the plastic geartrain, and reduces the failure rate of the motors.
     */
    int numSteps = 5;
    int pwmLeftVal = initialLeftSpd; // initialize left wheel speed
    int pwmRightVal = initialRightSpd;  // initialize right wheel speed
    int deltaLeft = (finalLeftSpd - initialLeftSpd) / numSteps; // left in(de)crement
    int deltaRight = (finalRightSpd - initialRightSpd) / numSteps;  // right
in(de)crement

    for (int k = 0; k < numSteps; k++) {
        pwmLeftVal = pwmLeftVal + deltaLeft;
        pwmRightVal = pwmRightVal + deltaRight;
        analogWrite(left_pwm_pin, pwmLeftVal);
        analogWrite(right_pwm_pin, pwmRightVal);
        delay(60);
    } // end for int k
} // end void changeWheelSpeeds

int average()  //average pulse count
{
    int getL = getEncoderCount_left();
    int getR = getEncoderCount_right();
    //  Serial.print(getL);Serial.print("\t");Serial.println(getR);
    return ((getEncoderCount_left() + getEncoderCount_right()) / 2);
}

void loop()
{
    if (((sensorValues[0] + sensorValues[1] + sensorValues[2] + sensorValues[3] +
sensorValues[4] + sensorValues[5] + sensorValues[6] + sensorValues[7])/8) > 2200)
    {
      counter++;
      if((FirstCrossPiece == false) && (counter > 3))
      {
       FirstCrossPiece = true;
       counter = 0;
      }

      if((SecondCrossPiece == false) && (counter > 3))
      {
       SecondCrossPiece = true;
       counter = 0;
      }

      if((DoDonut == false) && (counter > 3))
      {
       changeWheelSpeeds(leftSpd, 0, rightSpd, 0);
       if(DidDonut == true)
         {
          while(true)
          {
            analogWrite(left_pwm_pin,0);
            analogWrite(right_pwm_pin,0);
          }
         }
       digitalWrite(left_dir_pin,LOW);
       digitalWrite(right_dir_pin, HIGH);
       int in = getEncoderCount_left();
         while((getEncoderCount_left() - in) < 290)
         {
          analogWrite(left_pwm_pin,70);
          analogWrite(right_pwm_pin,70);
         }
         changeWheelSpeeds(leftSpd, 0, rightSpd, 0);
         digitalWrite(left_dir_pin,LOW);
         digitalWrite(right_dir_pin,LOW);
         changeWheelSpeeds(0, 70, 0, 70);
         FirstCrossPiece = false;
         SecondCrossPiece = false;
         DoDonut = false;
         DidDonut = true;
         counter = 0;
      }
   }
   else
   {
    counter = 0;
   }

    ECE3_read_IR(sensorValues); // get sensor readings
    float error = ((sensorValues[7] - sensorValues[0]) * 15 + (sensorValues[6] -
sensorValues[1]) * 14 + (sensorValues[5] - sensorValues[2]) * 12 + (sensorValues[4] -
sensorValues[3]) * 8) / 8;
    // calculate error
    // either donut, finish, cross piece or phantom crosspiece
    error_dif = error - error_prev;
    error_prev = error;
    leftSpd = l + (K_p * error / 255) + (K_d * error_dif / 255);
    rightSpd = r - (K_p * error / 255) - (K_d * error_dif / 255);
    analogWrite(left_pwm_pin, leftSpd);
    analogWrite(right_pwm_pin, rightSpd);
}
```

Appendix II: Extra Credit Code

```
// Base code.
//
// *  NOTE: this code will do only three things:
// *    --rotate one wheel, and
// *    --blink the right front mainboard LED.
// *
// *  You will need to add more code to
// *  make the car do anything useful.
//

#include <ECE3.h>

uint16_t sensorValues[8]; // right -> left, 0 -> 7
 int r = 70;
 int l = 70;
const int left_nslp_pin = 31; // nslp ==> awake & ready for PWM
const int left_dir_pin = 29;
const int left_pwm_pin = 40;
int leftSpd;
int rightSpd;
const int LED_RF = 41;
const int right_nslp_pin = 11; // nslp ==> awake & ready for PWM
const int right_dir_pin = 30;
const int right_pwm_pin = 39;

int wheelSpd = 80;
int distance = 800;
//////////////////////////////////
void setup() {
    // put your setup code here, to run once:
    pinMode(left_nslp_pin, OUTPUT);
    pinMode(left_dir_pin, OUTPUT);
    pinMode(left_pwm_pin, OUTPUT);
    pinMode(right_nslp_pin, OUTPUT);
    pinMode(right_dir_pin, OUTPUT);
    pinMode(right_pwm_pin, OUTPUT);

    digitalWrite(right_nslp_pin, HIGH);
    digitalWrite(left_nslp_pin, HIGH);
    //  digitalWrite(left_nslp_pin,LOW);

    pinMode(LED_RF, OUTPUT);

    ECE3_Init();

    // set the data rate in bits/second for serial data transmission

    resetEncoderCount_left();
    resetEncoderCount_right();
    delay(2000); //Wait 2 seconds before starting

}

float error_prev = 0;
float error_dif = 0;
float K_d = -6.7586679 *5;
float K_p = -5;
bool DoDonut = false;
bool DidDonut = false;
bool FirstCrossPiece = false;
bool SecondCrossPiece = false;
int counter = 0;
int whitecounter = 0;

void changeWheelSpeeds(int initialLeftSpd, int finalLeftSpd, int initialRightSpd, int
finalRightSpd) {
    /*
     *   This function changes the car speed gradually (in about 30 ms) from initial
     *   speed to final speed. This non-instantaneous speed change reduces the load
     *   on the plastic geartrain, and reduces the failure rate of the motors.
     */
    int numSteps = 5;
    int pwmLeftVal = initialLeftSpd; // initialize left wheel speed
    int pwmRightVal = initialRightSpd;  // initialize right wheel speed
    int deltaLeft = (finalLeftSpd - initialLeftSpd) / numSteps; // left in(de)crement
    int deltaRight = (finalRightSpd - initialRightSpd) / numSteps;  // right
in(de)crement
```

```
    for (int k = 0; k < numSteps; k++) {
        pwmLeftVal = pwmLeftVal + deltaLeft;
        pwmRightVal = pwmRightVal + deltaRight;
        analogWrite(left_pwm_pin, pwmLeftVal);
        analogWrite(right_pwm_pin, pwmRightVal);
        delay(60);
    } // end for int k
} // end void changeWheelSpeeds

int average()  //average pulse count
{
    int getL = getEncoderCount_left();
    int getR = getEncoderCount_right();
    //  Serial.print(getL);Serial.print("\t");Serial.println(getR);
    return ((getEncoderCount_left() + getEncoderCount_right()) / 2);
}

void loop()
{
    if (getEncoderCount_left() < 600) {
      l = 60;
      r = 60;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 600 && getEncoderCount_left() < 2000) {
      l = 180;
      r = 180;
      K_d = -6.7586679 *4;
      K_p = -4;
    }
    if (getEncoderCount_left() > 2000 && getEncoderCount_left() < 3800) {
      l = 85;
      r = 85;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 4000 && getEncoderCount_left() < 5400) {
      l = 180;
      r = 180;
      K_d = -6.7586679 *4;
      K_p = -4;
    }
    if (getEncoderCount_left() > 5400 && getEncoderCount_left() < 6000) {
      l = 100;
      r = 100;
      K_d = -6.7586679 *5;
      K_p = -5;
    }

    if (getEncoderCount_left() > 6000 && getEncoderCount_left() < 6100) {
      l = 85;
      r = 85;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 6100 && getEncoderCount_left() < 6800) {
      l = 70;
      r = 70;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 6800 && getEncoderCount_left() < 7200) { //changed
      l = 90;
      r = 90;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 7200 && getEncoderCount_left() < 8400) { //changed
      l = 190;
      r = 190;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 8400 && getEncoderCount_left() < 10100) {
      l = 90;
      r = 90;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 10100 && getEncoderCount_left() < 11200) {
      l = 190;
      r = 190;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 11200 && getEncoderCount_left() < 11600) {
      l = 120;
      r = 120;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if (getEncoderCount_left() > 11600) {
      l = 80;
      r = 80;
      K_d = -6.7586679 *5;
      K_p = -5;
    }
    if ((((sensorValues[2] + sensorValues[3] + sensorValues[4] + sensorValues[5])/4) <
700) && (whitecounter > 3))
    {
      whitecounter++;
      changeWheelSpeeds(leftSpd, 0, rightSpd, 0);
      while(true)
      {
        analogWrite(left_pwm_pin,0);
        analogWrite(right_pwm_pin,0);
      }
    }
    else
    {
      whitecounter = 0;
    }
    if (((sensorValues[0] + sensorValues[1] + sensorValues[2] + sensorValues[3] +
sensorValues[4] + sensorValues[5] + sensorValues[6] + sensorValues[7])/8) > 2000)
    {
      counter++;
      if((FirstCrossPiece == false) && (counter > 2))
      {
       FirstCrossPiece = true;
       counter = 0;
      }

      if((SecondCrossPiece == false) && (counter > 2))
      {
       SecondCrossPiece = true;
       counter = 0;
      }

      if((DoDonut == false) && (counter > 2))
      {
       changeWheelSpeeds(leftSpd, 0, rightSpd, 0);
       if(DidDonut == true)
         {
          while(true)
          {
            analogWrite(left_pwm_pin,0);
            analogWrite(right_pwm_pin,0);
          }
         }
       digitalWrite(left_dir_pin,LOW);
       digitalWrite(right_dir_pin, HIGH);
       int in = getEncoderCount_left();
         while((getEncoderCount_left() - in) < 240)
         {
          analogWrite(left_pwm_pin,170);
          analogWrite(right_pwm_pin,170);
         }
         changeWheelSpeeds(leftSpd, 0, rightSpd, 0);
         digitalWrite(left_dir_pin,LOW);
         digitalWrite(right_dir_pin,LOW);
         changeWheelSpeeds(0, 70, 0, 70);
         FirstCrossPiece = false;
         SecondCrossPiece = false;
         DoDonut = false;
         DidDonut = true;
         counter = 0;
      }
   }
   else
   {
    counter = 0;
   }

    ECE3_read_IR(sensorValues); // get sensor readings
    float error = ((sensorValues[7] - sensorValues[0]) * 15 + (sensorValues[6] -
sensorValues[1]) * 14 + (sensorValues[5] - sensorValues[2]) * 12 + (sensorValues[4] -
sensorValues[3]) * 8) / 8;
    // calculate error
    // either donut, finish, cross piece or phantom crosspiece
    error_dif = error - error_prev;
    error_prev = error;
    leftSpd = l + (K_p * error / 255) + (K_d * error_dif / 255);
    rightSpd = r - (K_p * error / 255) - (K_d * error_dif / 255);
    analogWrite(left_pwm_pin, leftSpd);
    analogWrite(right_pwm_pin, rightSpd);
}
```