EE3 Final Report

Lab 1D
Longtai Zhong (405121976), Zhiyuan Chen(605363281)
14 December 2019

I. Introduction and Background

The goal of this project was to program an electric car for it to be able to detect a black line and follow the track, and to complete the whole routine it'll do a turn at the end of the trajectory and go back to the original spot.

The sensors mounted at the bottom of the car read the black line properly. The car then adjusts its wheels speeds so that they let the cart turns according to the line. In addition, the sensors should also read a horizontal line, and when they read it for the first time, the cart should turn 180 degrees while centering the middle of the car. When the sensors read the line for the second time, the car should stop moving. The whole process was needed to be completed in under 75 seconds.

The design we selected was PID control. Initially our approach was to adjust wheel speed without feedback control system, but it didn't perform well and lacks flexibility, we then changed our design to PID.

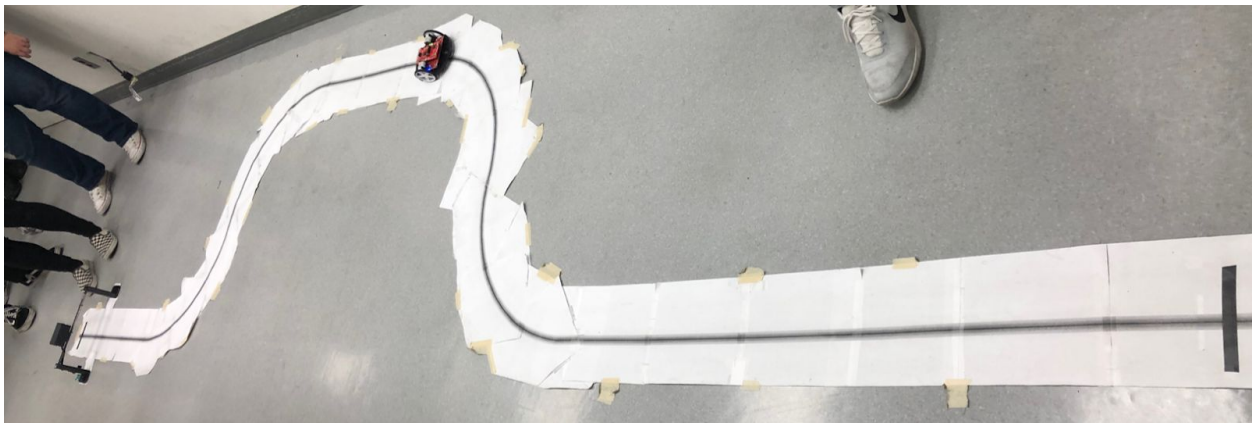


Figure 1: the track used for the race

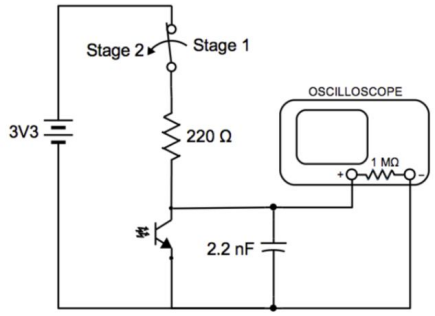
Basic Theory

At the bottom of the car there are 8 sensors. They detect the brightness of the color that is under the sensors and the larger the number they detect, the darker the brightness is. The numbers detected by those sensors are read by the line:

ECE3_read_IR(sensorValues);

and then stored in an array called sensorValues, and have 0 to 7 assigned to them. Looking from the bottom, they are 0 to 7 from right to left.

Figure 1 : the back of the car and sensor label



The path sensing system implements an important feature--phototransistor, which we can think of as variable resistors whose resistance changes with the incoming light intensity[1].

Figure 2: phototransistor circuit. This diagram was taken from EE3 Lab Manual week 3 "Related Phototransistor Circuit Experiment -- Theory".

This circuit could be simplified to evaluate in two stages:

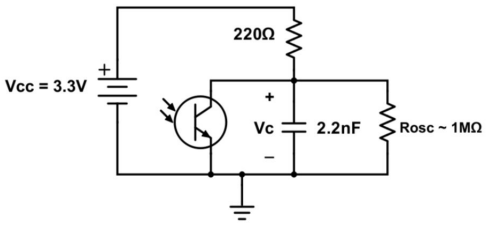


Figure 3: first stage. This diagram was taken from EE3 Lab Manual week 3

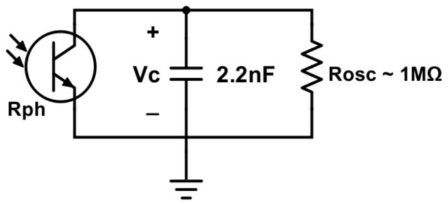


Figure 4: second stage. This diagram was taken from EE3 Lab Manual week 3

The phototransistor's resistance varies from ~5k ohms(light) to ~1M ohms(dark).

In stage 1, the voltage across the capacitor would be Vcc as the resistance of the parallel resistor set would be much larger than 220 ohms thus get all the voltage, and this sets the starting voltage for stage 2.

In stage2, we acquire:

$$V_c(t) = V_{cc}e^{-\frac{t}{C(R_{ph}//R_{OSC})}} \approx V_{cc}e^{-\frac{t}{C \cdot R_{ph}}}$$ [2]

Different lighting produces different sensor values, which is based quantitatively on the lighting condition of the phototransistors through the RC time constants[3].

II. Testing Methodology

Design we set up

We multiplied each obtained sensor value by different constants to make sensor weights. The purpose of making sensor weights was to distinguish if the line on the track was close to the edge or center, to determine how much the car should turn.

```
int a = 2*sensorValues[0]+1.875*sensorValues[1] + 1.5*sensorValues[2] + sensorValues[3];
int b = 2*sensorValues[7]+1.875*sensorValues[6] + 1.5*sensorValues[5] + sensorValues[4];
```

We had two integers a and b. Integer a is the sum of weighted sensor values of the left half of the sensors(looking from the top of the car), and integer b is the sum of weighted sensor values of the right half. We used the weights of 1, 1.5, 1.875, and 2, increasing toward the edge. It is increased toward the edge so that when the car is off the track a lot, a larger sum is produced for it to make a bigger turn.

Initial Plan

Initially, we had the integer a and b explained above, which were the sum of weighted sensor values from the left and the right sensors. We then subtracted one from the other to obtain an Integer c. c was obtained by b-a, meaning that C would be positive if the right sensors detect the dark line more, which means the car veers to the left, and be negative if the left sensors detect more. Based on the sign of C, we then determined whether the car should turn right or left.

In the set up section of the code, we have digitalWrite(left_dir_pin, LOW) and digitalWrite(right_dir_pin, LOW). They are responsible for determining the direction of the wheels. When we want to change the direction but have the same speed, we change LOW from HIGH and vice versa.

To change the speed of the wheels, we write

```
analogWrite(right_pwm_pin,170);
analogWrite(left_pwm_pin,170);
```

These analogWrite codes are responsible for the speed of the wheels, and the integer mean the speed of the wheels. The bigger the number, the faster the wheels turn.

We first set both of the speed of the wheel to 50, and then tried to find delay duration that will complete a turn. We took notes if the delay was too long or short. After finding best delay time for wheel speed of 50, we increased the speed to our desired speed, and tried to find the delay time for that speed.

- Donut Part

To test the donut, we launched a car from a close distance from the line (roughly 20cm) and checked if the car turns properly. We then also tested it from various distance to make sure the car doesn't detect a line where it is not supposed to.

To detect a solid horizontal line, we made an integer that was a sum of the raw sensor values. We then put a if statement of if that integer is above 14000, it will perform a donut. Since each maximum sensor value is 200, the maximum number of raw sensor value will be 2000*8 = 16000. We set the number little lower than 16000 so it detects a line properly.

To perform a donut, we set speeds of both wheels the same, and changed direction of right wheel, so that it would turn. We used a delay to determine a 180 degrees turn. In the test we tested different speeds and the delay number that match the speed.

Table 1: Wheel Speed vs Delay to complete 180 degrees turn

Wheel Speed	Delay	Car behavior
50	500	delay too short
50	700	delay too short
50	1000	delay too short
50	1200	180 degrees, but we wanted to make the donut go faster
80	700	speed too slow
120	700	speed too slow
150	700	speed too slow
170	700	Desired speed, but delay was too long
170	500	delay too long
170	200	delay too short
170	300	desired speed, desired duration

After testing, we found that the wheel speed of 170 and delay time of 300 will complete a 180 degree turn in desired speed.

- PID part

```
Input=c;
Output=Compute(Input);
//Serial.print(rightSpd-Output);
//Serial.print('\t');
//Serial.print(leftSpd+Output);

analogWrite(right_pwm_pin,rightSpd-Output);
analogWrite(left_pwm_pin,leftSpd+Output);


}
  double Compute(double inp)
{

   /*Compute all the working error variables*/
   double error = Setpoint - inp;

   double dErr = (error - lastErr);

   /*Compute PID Output*/
   double out = kp * error + kd * dErr;

   /*Remember some variables for next time*/
   lastErr = error;


   return out;
}
```

We ditched Ki and used PD control, our new velocity would be based on the value of c overtime.

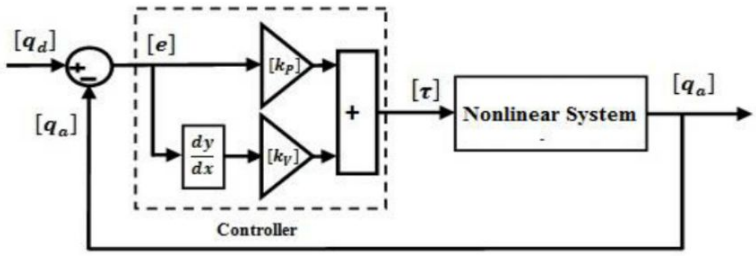


Figure: PD control. Image from https://www.researchgate.net/figure/Block-Diagram-of-PD-Control_fig3_289531089

Table 2: Different values of kp and how the car moves when kd is 0

kp	kd	Car behavior
0.1	0	goes off track immediately
0.01	0	stays on track the longest
0.02	0	Can not complete a loop, goes off track at curve immediately
0.03	0	goes off track on straight line
0.012	0	goes off track at curve
0.005	0	goes off track at curve
0.008	0	goes off track at curve

The data showed that when we have large number of kp, the car oscillates a lot and goes off the track immediately, however, when we have low number of kp the car can not fully turn.
From this, we decided to use 0.01 for our kp and began to adjust kd.

Table 3: Different values of kp and how the car moves when kd is 0.01

kp	kd	Car behavior
0.01	0.01	Oscillates too much
0.01	0.05	Oscillates too much
0.01	1	can’t turn at curve
0.01	0.5	can’t turn at curve
0.01	0.1	Completes a loop in 45 seconds

When we have no kd, the car oscillates too much. As we have increased number of kd, the car oscillates less. However, when kd is too large the car can not make a turn.
We found a value of kp and kd which was 0.01 and 0.1. With these values, We were able to finish the circuit in 70 seconds. In order to achieve 45 seconds, we needed to make the car go faster. We changed the speed from 70 to 100. When we made it to 100, however, the car could not turn properly. We then increased the kd value to 0.3 and kp to 0.015, and the problem got fixed.

- Other Problems Encountered

Even though we found desired speeds and time for the loop, we have encountered several problems on the way and we needed to make some adjustments to solve these issues.

Problem 1: The car performs loop even when there is no line

The issue was that the sensors were detected that there was a horizontal line, even though there wasn't, which means the sum of the raw sensor values surpassed 14000 for a tick. To solve this issue, we made sure that the sensor was detecting the line for more than single tick, meaning that the sum of raw sensor values surpassed 14000 for continuous ticks. We set a counter so that when the counter is incremented when the sensors continuously detect dark line. This prevents random detection of a line and only performs when the car detects a line with appropriate thickness.

```
  if (d > 14000)
{
  i++;
} else {
  i=0;
}

if (i>2 && count == 0) {
  count = 1;
  analogWrite(right_pwm_pin,170);
  analogWrite(left_pwm_pin,170);
  digitalWrite(right_dir_pin,HIGH);
  delay(300);
  digitalWrite(right_dir_pin,LOW);
   analogWrite(right_pwm_pin,100);
  analogWrite(left_pwm_pin,100);
  delay(200);
```

Figure 5: i counter code solution

Problem 2: The cart performs loop twice on the same spot

We had this error when we increased the speed of the car on the track. The cart would perform a loop on the line, and then does it again. We assumed that the error started to happen because since the speed of the car is increased, the car would detect the line at a deeper place than before, meaning that after it completes a turn it still detects a line. To prevent this issue, we just had to make it so that when it completes a loop the sensors are not on the line.

The line after we changed the wheel direction at digitalWrite(right_dir_pin, LOW),weI changed the wheel speeds to 100, and set a delay for 200ms. This would make the car move forward for a little bit after the turn, making sure that the horizontal line is not under sensors when the car completes a turn.

Extra Credit: Odometry

Requirement: When the average encoder count of right and left wheels are above 15000, turn the blue light on.

Code:

```
avgEncoderCount = (getEncoderCount_left()+getEncoderCount_right())/2;
if(avgEncoderCount>15000){
        digitalWrite(BLUE_LED, HIGH);
}
```

I named a value avgEncoderCount which takes the average encoder count of right and left wheels, and put digital Write that makes the blue LED light on when that variable is above 15000.

III. Results and Discussion

Test Discussion

All the data we collected are useful for our project goals, it helped us fix the range for proper constant values we need.

In Table 1 we were able to estimate and continue testing different delay values based on the ones we already tested. In Table 2 we can see how Kp should change, when we set Kp at a larger value, the car would oscillates too much on the track but we want it to move smooth, when the Kp value's low, the car doesn't turn enough thus it's hard for it to follow the track. In Table 3, with a fixed Kp value, we found the proper Kd in the same way, and from the data we collected, we can see smaller value leads to oscillation and for larger value, unable to turn at the curve.

With the data and interpretations, we were able to find the correct value for our car based on former performance.

Race Day Discussion

On race day, our car performed as we expected. It could follow the track smoothly, turn at the curves, do a donut and stop at the very end. The routine was completed in 25.27s, the performance was just as our training during the lab. Performance video: https://youtu.be/2GGUB0ltFoY

The limitations include: 1) the PID control was able to adjust the speed in order to keep the car on track, but as a feedback system using constant parameters, it was hard to define the speed directly, for example, if we want the speed to be faster on straight line and slower on the curve; 2) the car, though operating smoothly, we could still see the tiny trembles when it runs; 3) the track is smooth and it'll be harder if it has sharp angles such as a right angle, or a circular turnaround.

We encountered a project design change during the process, to conduct the project differently, we ditched the method that used initially where different fixed speed would be altered regarding different c values. But we changed to PID control. If we were still to use the original design, we would build multiple threshold values for c and implement them in a few if statements. In addition, for the car to run without the tiny trembles, we would establish other if statements in the code on the base of PID, like within some c range, we would have the speed fixed.

IV. Conclusions and Future Work

Our goal was to program a line-following robot car, and it's main features include turning at the end of the track, going back to the starting line and stop, finishing the process within 45 seconds. Our design met our goals well, the car was able to accomplish these smoothly. We learned a lot

and gained engineering hands-on experience. Through the testing process, we encountered problems and solved them, and when a current system's not working, we adopt another method. We learned about arduino programming under a practical circumstance, utilizing the sensor system, PID control, as well as team collaboration.

Since the project now is detecting color black and on a horizontal surface, some extensions we would like to do are sensing different colors and going uphill. For color sensing, we'd test the car on different colors and record the sensor values in serial monitor. For going uphill, we'd modify the code since the speed in our current code might not be enough to support the car on the incline, code modification could be increasing both wheels' speed when the threshold is under certain value, and then test the car on different inclines.

V. References

[1] ,[2], [3] EE3 Laboratory Manual week 3