Name (UID): Marlee Kitchen (105580866), Jillian Pantig (305699949)
ECE3 Final Project Report - Prof. Mike Briggs
Fall 2021
12/11/2021

Develop

1. Familiarize with the key concepts of the project: Making the car follow a line and doing a donut whenever the car encounters a horizontal line.
2. Get an understanding of how the part of the car works independently to each other – pins, motors, and photoresistor sensors.
3. Understand the concepts of a PID control in order to write a code for the line-following car.
4. Do a calibration for and filtering or fusion of data from the car's photoresistor sensors.
5. Write a code with PD control, not applicable for the donut, using the filtered data from the photoresistor sensors that varies the motors' speeds independently depending on the input from the sensors in order for the car to follow the line.
6. After checking that the motors' properly respond to the sensors' data via inspection and through the Serial Monitor, run the car on track by starting at a low speed and while varying K_p and K_d values.
7. When optimal values for the PD control that dictated the car's ability to follow the line are determined, increase the speed and decide the K_p and K_d values appropriate for this speed – repeat this step until desired speed, 100, is attained.
8. After making sure that the car follows the line, write an additional code for the donut, using the sensor's maximum values and a time delay – the time that will make the car do a 180° turn, that will be triggered when the sensors of the car detect a horizontal black line at the turn around point of the track.
9. Following the successful integration of Step 6-8, alter the donut code in Step 8 to stop the car when the second horizontal black line is detected at the end of the track.

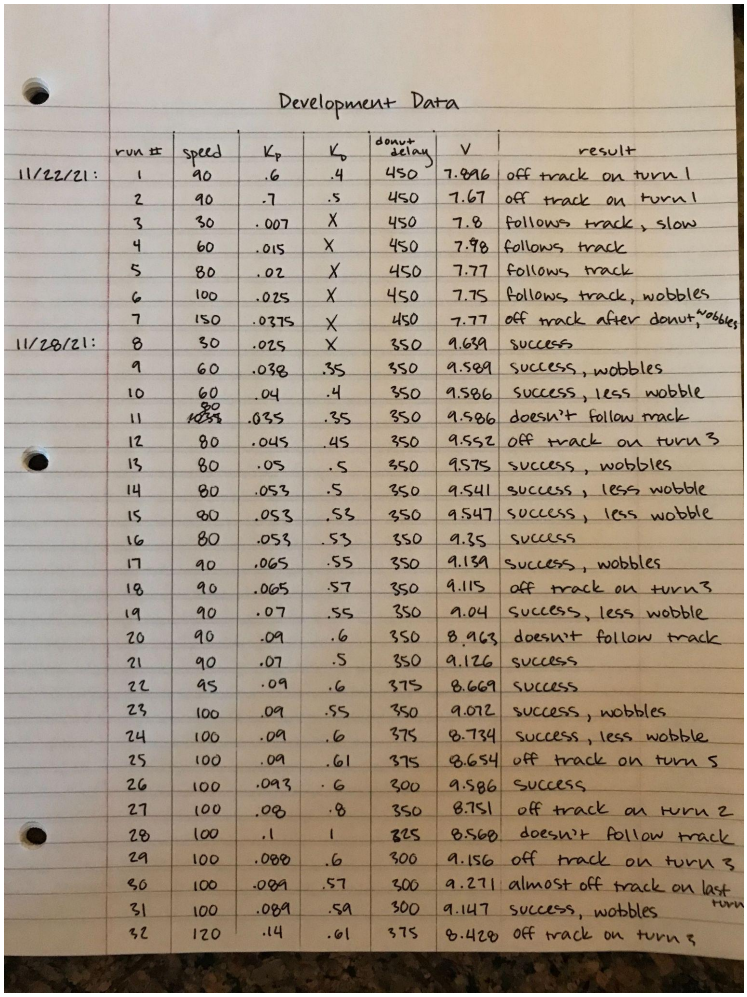
Conduct Tests

- The parameters controlled in this project were the motor speed, K_p and K_d values, the weighting scheme, and the starting position of the car.
- The variable measured in this project but did not control was the value of the car's voltage.
- Note: There is no test for Steps 1-4.
- Step 5 Test: When testing the response of the car with respect to the sensor data, we checked if the PD calculation is correct through the Serial Monitor and if the wheels of the car are moving in accordance with the line being detected by the sensors.
- Step 6 and 7 Test: When testing the car's ability to follow the track without the donut, we set the motor speed, K_p, and K_d values, measured the car's voltage, and then ran the car from Starting Position 2. Depending on the car's performance in the previous run, we would increase the K_p value if the car was not following the track tightly enough, decrease the K_d value when the car went off track on a turn, remeasure the voltage, and rerun the car, repeating this process until we optimized the car's performance.

- Step 8 Test: When testing the car's function on the turn around point of the track, we set the speed and delay time for the donut, measured the car's voltage, and ran the car, altering the delay time on each new run until we determined the proper value corresponding to a 180° turn.
- Step 9 Test: The code for stopping the car is incorporated with the donut code such that a second condition, first condition for the donut turn, is added when the car encounters the second horizontal black line which is tested simply by checking if the car stops on the last horizontal black line – the end of the track.

Analyze

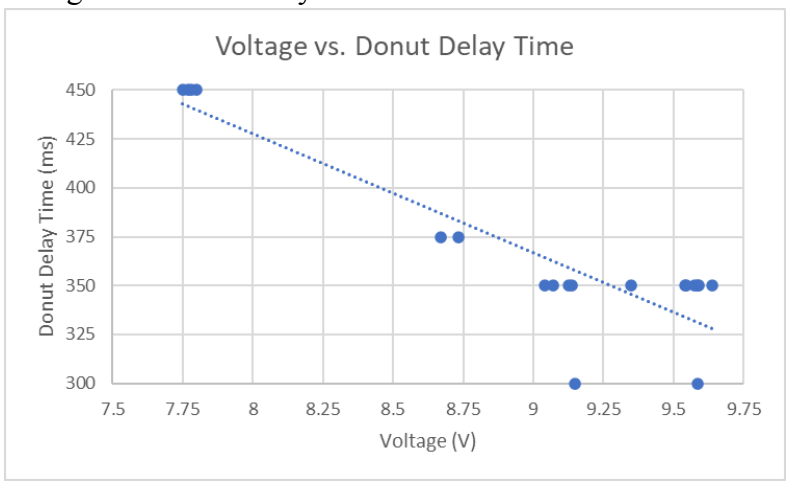
- Development Data:



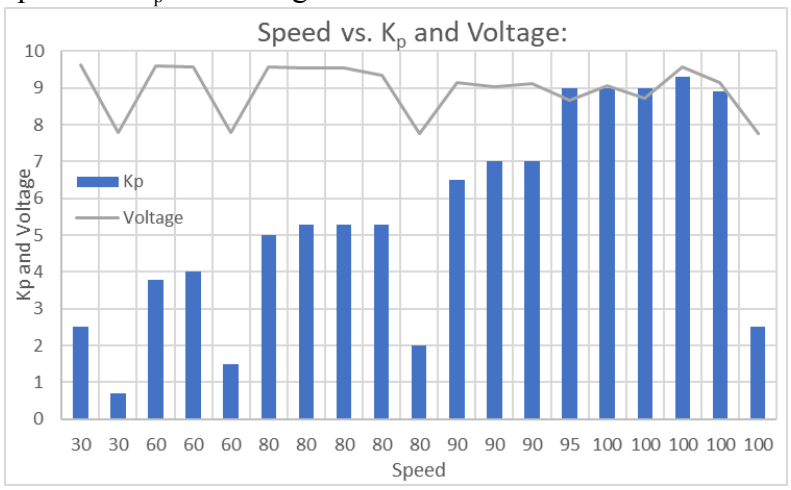
Development Data

	run #	speed	K_p	K_D	donut delay	V	result
11/22/21:	1	90	.6	.4	450	7.896	off track on turn 1
	2	90	.7	.5	450	7.67	off track on turn 1
	3	30	.007	X	450	7.8	follows track, slow
	4	60	.015	X	450	7.98	follows track
	5	80	.02	X	450	7.77	follows track
	6	100	.025	X	450	7.75	follows track, wobbles
	7	150	.0375	X	450	7.77	off track after donut, wobbles
11/28/21:	8	30	.025	X	350	9.639	success
	9	60	.038	.35	350	9.589	success, wobbles
	10	60	.04	.4	350	9.586	success, less wobble
	11	~~80~~ 60	.035	.35	350	9.586	doesn't follow track
	12	80	.045	.45	350	9.552	off track on turn 3
	13	80	.05	.5	350	9.575	success, wobbles
	14	80	.053	.5	350	9.541	success, less wobble
	15	80	.053	.53	350	9.547	success, less wobble
	16	80	.053	.53	350	9.35	success
	17	90	.065	.55	350	9.139	success, wobbles
	18	90	.065	.57	350	9.115	off track on turn 3
	19	90	.07	.55	350	9.04	success, less wobble
	20	90	.09	.6	350	8.963	doesn't follow track
	21	90	.07	.5	350	9.126	success
	22	95	.09	.6	375	8.669	success
	23	100	.09	.55	350	9.072	success, wobbles
	24	100	.09	.6	375	8.734	success, less wobble
	25	100	.09	.61	375	8.654	off track on turn 5
	26	100	.093	.6	300	9.586	success
	27	100	.08	.8	350	8.751	off track on turn 2
	28	100	.1	1	825	8.568	doesn't follow track
	29	100	.088	.6	300	9.156	off track on turn 3
	30	100	.089	.57	300	9.271	almost off track on last turn
	31	100	.089	.59	300	9.147	success, wobbles
	32	120	.14	.61	375	8.428	off track on turn 3

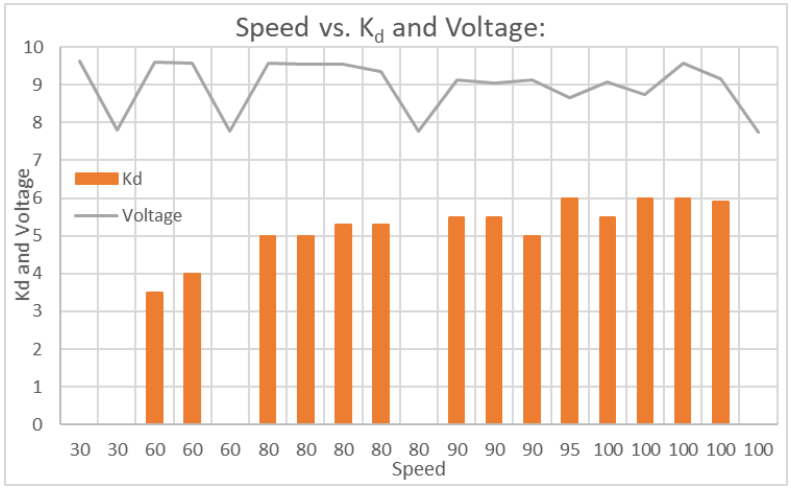
- Voltage vs. Donut Delay Time:



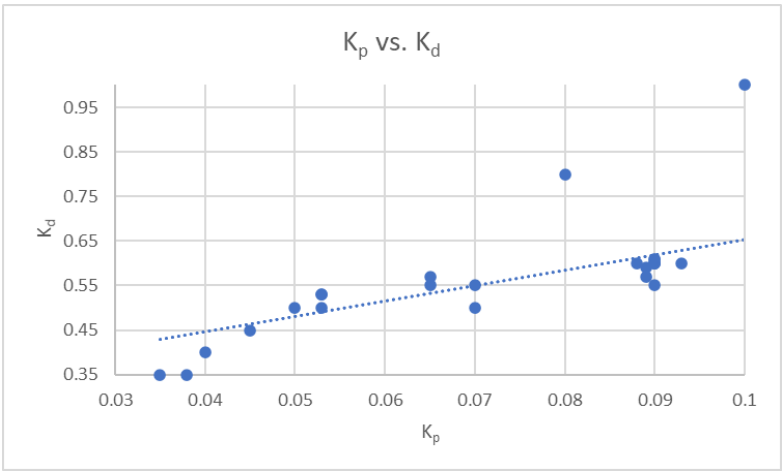
- Speed vs. K_p and Voltage:



- Speed vs. K_d and Voltage:



- K_p vs. K_d:



Interpret

- Voltage vs. Donut Delay Time: This graph portrays the negative relationship between the voltage and delay time that was required in our donut code to ensure that the car completed a full 180° turn which is logical given that our donut code kept the speed constant, therefore the voltage was the only other factor affecting the speed of the turn, so whenever the voltage was lower, a higher delay time was necessary for the car to complete the full turn.
- Speed vs. K_p and Voltage: This graph shows that there is a direct relationship between K_p (K_p normalized in order to make the values more visible in the graph), voltage (V), and speed in a way that when there is higher voltage and speed, the K_p is set higher in order for the car to accurately follow the line, and vice versa.
- Speed vs. K_d and Voltage: In this graph, it can be seen that there is a direct relationship between K_d (K_d normalized in order to make the values more visible in the graph), voltage, and speed such that increasing or decreasing the speed and voltage correspond to an increase or a decrease in the K_d value, respectively, resulting in the car having the right response correction with respect to its current and last position and allowing it to properly follow the track.
- K_p vs. K_d: This graph serves as a visual representation of the positive correlation between K_p and K_d that corresponded to the subsequent increase in K_d required when K_p was increased concurrently with the speed in order to maintain the car's line following ability.

Code for Line-Following Car (Arduino)

```
//ECE 3 - Marlee Kitchen and Jillian Pantig
#include <ECE3.h>

#define N_SENS 8                //Number of Sensors
#define R_SENS        1000    //Normalize sensor data to this
#define MOTORSPEED 90

const int left_nslp_pin=31; // nslp ==> awake & ready for PWM
const int left_dir_pin=29;
const int left_pwm_pin=40;
const int right_nslp_pin=11; // nslp ==> awake & ready for PWM
const int right_dir_pin=30;
const int right_pwm_pin=39;
const int LED_RF = 41;

uint16_t sensorValues[8];

long maxOfSumOfSensors = 12540; //Sum of Max Sensor Data
int sens_max[N_SENS] = {1687, 1637, 1694, 1359.2, 1376, 1468, 1530.8,
1788}; //Max sensor data for each sensor (Detects black)
int sens_min[N_SENS] = {805, 735, 806, 656.8, 712, 643, 675.2, 712};
//Min sensor data for each sensor (Detects white)
int weight_val[N_SENS] = {-8,-4,-2,-1,1,2,4,8}; //Weighting scheme
for data

const float kp = 0.065;
const float kd = 0.5;

float line_pos = 0;
float prev_line_pos = 0;

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(left_dir_pin,LOW);
  digitalWrite(right_dir_pin,LOW);

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

  ECE3_Init();
  //Serial.begin(115200); // set the data rate in bits per second for
serial data transmission (For debugging purposes)
  delay(300);
}

float get_line_pos(int last_dir){
    float sens_scaled[N_SENS];
    float line = 0;
    int line_detected = 0;
    float avg_num = 0;
    float avg_den = 0;

    for(int x = 0; x < N_SENS; x++){
        //Scale from 0 to R_SENS
        sens_scaled[x] = sensorValues[x] - sens_min[x];
        sens_scaled[x] *= R_SENS;
        sens_scaled[x] /= sens_max[x];
        sens_scaled[x]  *= weight_val[x];

        avg_num += sens_scaled[x];
        avg_den += sens_scaled[x];
    }

    line = avg_num / 4;

    return line;
}
```

```
float get_PID_correction(float line, float last_line, float kp, float
kd){
  float proportional = line;
  float derivative = (line - last_line);
  float correction = (kp * proportional + kd * derivative);

  return correction;
}

static int x = 0;
unsigned long startS = 0;
unsigned long endS = 0;

void loop() {
  ECE3_read_IR(sensorValues);

  int sumSensors = 0;
  for(int i = 0; i < N_SENS; ++i)
    sumSensors += sensorValues[i];

  endS = millis();
  if (sumSensors >= (maxOfSumOfSensors - 100)) //Code for donut and
stop on Horizontal black lines
  {
    switch (x) {
      case 0:
        {
        startS = millis();
        analogWrite(right_pwm_pin, 140);
        analogWrite(left_pwm_pin, 140);
        digitalWrite(left_dir_pin,HIGH);
        delay(275);
        digitalWrite(left_dir_pin,LOW);
        analogWrite(left_pwm_pin, MOTORSPEED);
        analogWrite(right_pwm_pin, MOTORSPEED);
        ++x;
        break;
        }
      case 1:
```

```
        if(endS - startS >= 600){
          analogWrite(left_pwm_pin, 0);
          analogWrite(right_pwm_pin, 0);
          digitalWrite(left_nslp_pin,LOW);
          digitalWrite(right_nslp_pin,LOW);
          }
        break;
    }
  }

  prev_line_pos = line_pos;
  line_pos = get_line_pos(prev_line_pos>0);

  float PID_CORR = get_PID_correction(line_pos, prev_line_pos, kp,
kd);

    if(PID_CORR > 0){
        analogWrite(left_pwm_pin, MOTORSPEED +5);
        analogWrite(right_pwm_pin, MOTORSPEED + PID_CORR);
    }
    else{
        analogWrite(left_pwm_pin, MOTORSPEED + 5 - PID_CORR);
        analogWrite(right_pwm_pin, MOTORSPEED);
    }

  }
```