Denver Brown UID 305-547-343

Brian Hardell UID 705-593-998

ECE3 20F Final Report

Introduction and Background

We were issued a project car which had two wheels, an electric motor for each wheel, an energeia control board and eight phototransistors, each connected to an RC circuit, on the bottom. The board was also able to control other circuits which would increase the power to the wheels, change their direction of spin and apply/remove current to their motors. Our goal was to design a system by which the circuit board would interpret the results of the photoreceptors in order to follow a black line on a white piece of paper as accurately and quickly as possible.

Specifically, we had to make the car follow an official track complete with curves, reach a horizontal black line indicating the end of the track and double back to the beginning of the track. Initially the time we were looking to beat was fifteen seconds but later we were able to make the extra credit goal of finishing the track in under eleven seconds. We decided on a design by which we would use the proportional difference between the cars current position and the center of the track to make logical decisions and steer the vehicle. You can also make a system where the car makes logical decisions based on the derivative (rate of change) of the cars position in relation to the center of the track by simply comparing your current sensor data to the immediately preceding data (calculating the derivative) and making your steering decisions based on that.

The phototransistors are each connected to a two stage R/C circuit. The key to understanding how they work is to remember the behavior of capacitors in R/C circuits. The phototransistors on the project car can be modelled as a two-stage R/C circuit such as that shown in the lab manual.:  
O.M. Stafsudd 2013  
  
  
  
 In a forced function where capacitors are connected to a voltage source such as in Stage 1, the capacitors build up a Voltage in opposition to the current passing through them until there is no current flow, this is “charging”. Stage 2 is analogous to a natural response where the capacitor attempts to prevent a change in voltage by acting as a voltage source, this voltage decreases as the capacitor “discharges” until it reaches zero. How quickly this occurs depends on how much resistance is in the Stage 2 circuit and this is where the light detection actually occurs. Depending on how much current enters the base of the phototransistor via light energy, it applies different amounts of resistance to the circuit (approximately 5E3 Ohms for pure light and 1E6 Ohms for pure dark). The board then measures the time it takes for the phototransistor to discharge and interprets it as a number between 0 (very light) and 2500 (very dark).

These sensors do not always agree with each other regarding the intensity of light they detect, and this is where sensor calibration comes in. This is a process by which we normalize the output of the sensors so that the data becomes easy to work with and meaningful. This can be accomplished by connecting the board to a PC running the energeia software and running the following code:

#include <ECE3.h>

uint16\_t sensorValues[8];

void setup()

{

ECE3\_Init();

Serial.begin(9600); // set the data rate in bits per second for serial data transmission

delay(2000);

}

void loop()

{

ECE3\_read\_IR(sensorValues);

for (unsigned char i = 0; i < 8; i++)

{

Serial.print(sensorValues[i]);

Serial.print('\t'); // tab to format the raw data into columns in the Serial monitor

}

Serial.println();

delay(50);

}

This will print out the values to the serial monitor which can be accessed from the energeia bar under Tools. Next you place a solid black strip under each sensor to record its maximum value (take the average of five readings for increased accuracy) and then move it under the next sensor and do it again. The idea then is to modify the output of each sensor so that its minimum value returns a 0 (white paper) while its maximum value returns a 1000 (black paper). This can be accomplished with the following code:

sensorValues[X] = abs((sensorValues[X]-MinimumValue)\*(1000/MaximumValue));

Note that the absolute value function is to prevent any of the values from going below 0 if they somehow get a reading lower than their minimum. Here is the data from our sensor calibration to illustrate:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ORIGINAL DATA (AVERAGE OF FIVE READINGS)** | | | | | | | | | | | | | | | | | | | | | | |
| **ERROR** | S1 | | | S2 | | | S3 | | | S4 | | S5 | | | S6 | | | S7 | | S8 | | |
| -4 | 2028.2 | | | 562.6 | | | 630.8 | | | 630.8 | | 562.6 | | | 677.2 | | | 677.2 | | 723.6 | | |
| -3 | 837.8 | | | 1769.6 | | | 675.4 | | | 652.6 | | 562.2 | | | 675.4 | | | 675.4 | | 721.8 | | |
| -2 | 630.6 | | | 722.4 | | | 2143.4 | | | 675.6 | | 562.6 | | | 675.6 | | | 675.6 | | 722.4 | | |
| -1 | 631 | | | 549.2 | | | 1024 | | | 1713.6 | | 562.4 | | | 676.4 | | | 676.4 | | 741.6 | | |
| 1 | 630.8 | | | 540.4 | | | 630.8 | | | 1116.8 | | 1466.4 | | | 676.6 | | | 676.6 | | 746.2 | | |
| 2 | 630.6 | | | 540.4 | | | 630.6 | | | 653.6 | | 1047.6 | | | 1769.6 | | | 653.6 | | 746.2 | | |
| 3 | 631.2 | | | 540.2 | | | 631.2 | | | 654.4 | | 562.8 | | | 1674 | | | 1627.4 | | 723.4 | | |
| 4 | 630.4 | | | 540 | | | 630.4 | | | 654 | | 562.4 | | | 677.2 | | | 2048.4 | | 1302.8 | | |
| **Minimum** | 630.4 | | | 540 | | | 630.4 | | | 630.8 | | 562.2 | | | 675.4 | | | 653.6 | | 721.8 | | |
| **DATA WITH MINIMUM VALUE SUBTRACTED** | | | | | | | | | | | | | | | | | | | | | | | |
| **ERROR** | | | S1 | | S2 | | | S3 | | | S4 | | S5 | | | S6 | | | S7 | | | S8 | |
| -4 | | | 1397.8 | | 22.6 | | | 0.4 | | | 0 | | 0.4 | | | 1.8 | | | 23.6 | | | 1.8 | |
| -3 | | | 207.4 | | 1229.6 | | | 45 | | | 21.8 | | 0 | | | 0 | | | 21.8 | | | 0 | |
| -2 | | | 0.2 | | 182.4 | | | 1513 | | | 44.8 | | 0.4 | | | 0.2 | | | 22 | | | 0.6 | |
| -1 | | | 0.6 | | 9.2 | | | 393.6 | | | 1082.8 | | 0.2 | | | 1 | | | 22.8 | | | 19.8 | |
| 1 | | | 0.4 | | 0.4 | | | 0.4 | | | 486 | | 904.2 | | | 1.2 | | | 23 | | | 24.4 | |
| 2 | | | 0.2 | | 0.4 | | | 0.2 | | | 22.8 | | 485.4 | | | 1094.2 | | | 0 | | | 24.4 | |
| 3 | | | 0.8 | | 0.2 | | | 0.8 | | | 23.6 | | 0.6 | | | 998.6 | | | 973.8 | | | 1.6 | |
| 4 | | | 0 | | 0 | | | 0 | | | 23.2 | | 0.2 | | | 1.8 | | | 1394.8 | | | 581 | |
| **Maximum** | | | 1397.8 | | 1229.6 | | | 1513 | | | 1082.8 | | 904.2 | | | 1094.2 | | | 1394.8 | | | 581 | |
| **DATA NORMALIZED TO 1000** | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ERROR** | | S1 | | | | S2 | | | S3 | | | | | S4 | | | S5 | | | | S6 | | | | S7 | S8 |
| -4 | | 1000.0 | | | | 18.4 | | | 0.3 | | | | | 0.0 | | | 0.4 | | | | 1.6 | | | | 16.9 | 3.1 |
| -3 | | 148.4 | | | | 1000.0 | | | 29.7 | | | | | 20.1 | | | 0.0 | | | | 0.0 | | | | 15.6 | 0.0 |
| -2 | | 0.1 | | | | 148.3 | | | 1000.0 | | | | | 41.4 | | | 0.4 | | | | 0.2 | | | | 15.8 | 1.0 |
| -1 | | 0.4 | | | | 7.5 | | | 260.1 | | | | | 1000.0 | | | 0.2 | | | | 0.9 | | | | 16.3 | 34.1 |
| 1 | | 0.3 | | | | 0.3 | | | 0.3 | | | | | 448.8 | | | 1000.0 | | | | 1.1 | | | | 16.5 | 42.0 |
| 2 | | 0.1 | | | | 0.3 | | | 0.1 | | | | | 21.1 | | | 536.8 | | | | 1000.0 | | | | 0.0 | 42.0 |
| 3 | | 0.6 | | | | 0.2 | | | 0.5 | | | | | 21.8 | | | 0.7 | | | | 912.6 | | | | 698.2 | 2.8 |
| 4 | | 0.0 | | | | 0.0 | | | 0.0 | | | | | 21.4 | | | 0.2 | | | | 1.6 | | | | 1000.0 | 1000.0 |

Finally, we want to take all our data and interpret it in order to make navigation decisions, we did this through sensor fusion. The idea is to combine your sensor data in a way that one number comes out which can tell you if you are to the left of the center, to the right of the center, on the center, or at the end of the track. We did this by summing all the sensor data using a weighted system. Sensors to the left of center return negative values while sensors to the right return positive values, that way if the total sum is negative you know that the line is under the left part of the car and you need to turn that way to get to center and vice-versa for positive sums. The outermost circuits (sensors 0 and 7) we scaled by a factor of 8, followed by a factor of 4 for the next sensors(sensors 1 and 6), a factor of 2 for the third set(sensors 2 and 5) and a factor of 1 for the innermost sensors(sensors 3 and 4). This scaling was done so that the car correctly interprets the data from the outer sensors as being further and therefore more likely to require a correcting steer. The implementation is as follows:

int weightedSum = 0;

weightedSum += sensorValues[4];

weightedSum += sensorValues[5]\*2;

weightedSum += sensorValues[6]\*4;

weightedSum += sensorValues[7]\*8;

weightedSum -= sensorValues[0]\*8;

weightedSum -= sensorValues[1]\*4;

weightedSum -= sensorValues[2]\*2;

weightedSum -= sensorValues[3];