Line Following Robot

Dhara Patel and Argyris Kriezis

October 15, 2019

1 Background

The goal of this lab is to design, assemble and test a car that can autonomously follow a black-tape line. To achieve this goal we are going to combine mechanical, electrical and software tools. We laser cut hardboard to fabricate our chassis where the motors and sensors are mounted on. To control the car, we combine infrared sensors and PID (Proportional, Integral, and Derivative) control to keep the car on the black tape.

2 Electrical and Software Design

Properly calibrated infrared reflectance sensors are an integral part of achieving our goal of a line-following car because they are used to detect the black tape. An IR sensor has two components: a light emitting diode and a photo transistor. The LED emits light whose waves are reflected off the surface in front and detected by the photo transistor. The detected infrared light changes the flow of current between the collector and emitter on the photo transistor side according to the level of light reflected off the surface, meaning we can use the voltage drop across the photo transistor as a measure of reflectance. Figure 1 demonstrates how the IR sensor emitts and detects infrared light and how we can measure reflectance. The effectiveness of the sensor readings depends on the value of R_{sense}. Its value was calculated to be 10K Ω using Equation 1, where I_c is 1mA.

$$R_{sense} = \frac{V_{out} - 5V}{I_c} \tag{1}$$

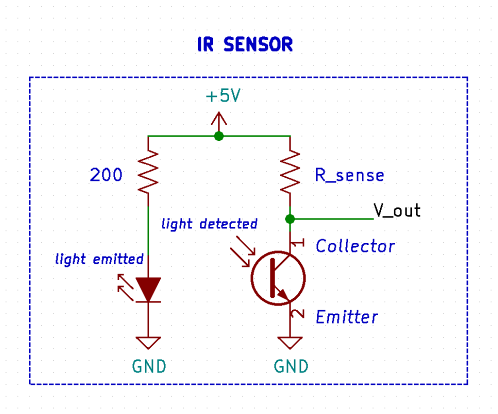


Figure 1: A schematic of an IR sensor depicts the LED, which emits light onto the surface in front, and the photo transistor, which detects the infrared light reflected off of the surface in front.

Black absorbs all light, which means no light is not reflected off the black tape we are trying to follow and a value close to 5V (1023 in Analog) is read. A dual IR sensor system can be implemented to keep the black tape between the two sensors as shown by Figure 2. As soon as a sensor detects the black tape we can adjust the speed of the motors accordingly using Proportional control.

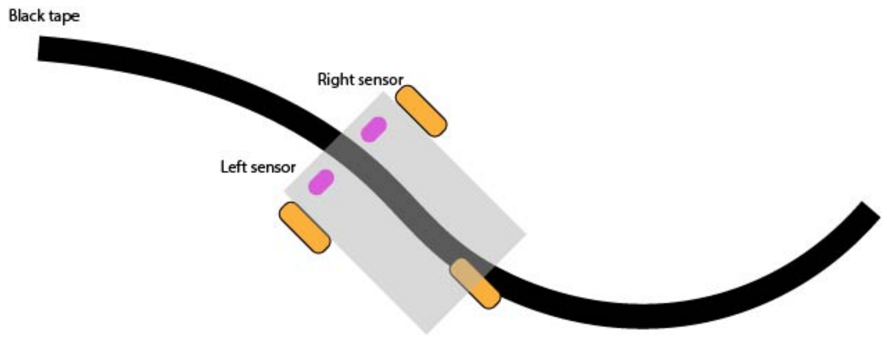


Figure 2: A dual IR sensor system is implemented as shown above in order to keep the mid-line of the car aligned with the black tape.

At every loop Proportional control measures the difference in the left and right sensor readings, computes a correction value, and changes the speed of the motors accordingly. The difference in readings, the *error*, is sent to `UpdatePID()` where the correction term is calculated by multiplying error and a proportional gain term, `pGain` (reference section 6 for all source code). The proportional gain term is tuning parameter used to scale the correction term down so that it can be applied to changes in motor speed. The error is also amplified on both ends of the spectrum before it is multiplied by `pGain` (see lines 67-74 in section 6.1). Error less than 100 is decreased by 30 and error greater than 300 is increased by 70. By amplifying error detection of tape makes a more significant difference in motor speed. Meager differences that may be due to lighting, dust, or floor patterns have less effect on motor speed.

If the returned correction term is positive, the right sensor is above the black tape and the left motor speed must increase to turn the car right. However, the opposite must happen when the correction term is negative. To account for this, we add and subtract correction to a base speed of 20 as shown in lines 38-45 in section 6.1.

2.1 Testing and Calibration

Testing and calibration was an integral part of our design process. The proportional gain term was determined based on how well our sensors could detect black tape. Our decision to amplify and minimize error came about after realizing our tape didn't impact change in motor speed as much as it should. The distance between the sensors was also tested and changed accordingly. Ultimately, the sensors were placed so that they line the tape closely, allowing them to detect the black tape more quickly.

3 Mechanical Design

The mechanical design of the chassis was based on the idea that the car should be the closest possible to the ground. This way the sensors are going to be the ideal distance from the ground without the need of an extra base. In order to maintain this low height, the back wheel base had to be raised. Also, the motors of the wheels had to be placed on top of the chassis and this reduced the available area. The initial plan was for the Arduino to be placed between the back wheel and the motors, however the required clearance needed for the cables did not allow u to do it, so we placed the board on top of the back wheel.

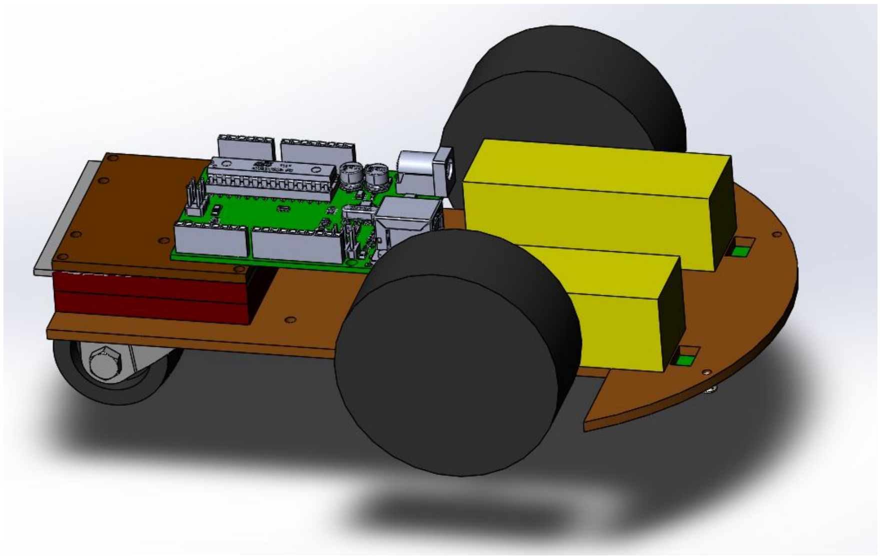


Figure 3: Rendering of the assembly of the chassis

The sensors are placed at the front-middle of the car, followed by the two big motors for each wheel. The Arduino is placed between the wheels and the raised base supporting the back wheel. The motors as well as the back wheel are screwed in the chassis and the Arduino board is held in place using a zip-tie.

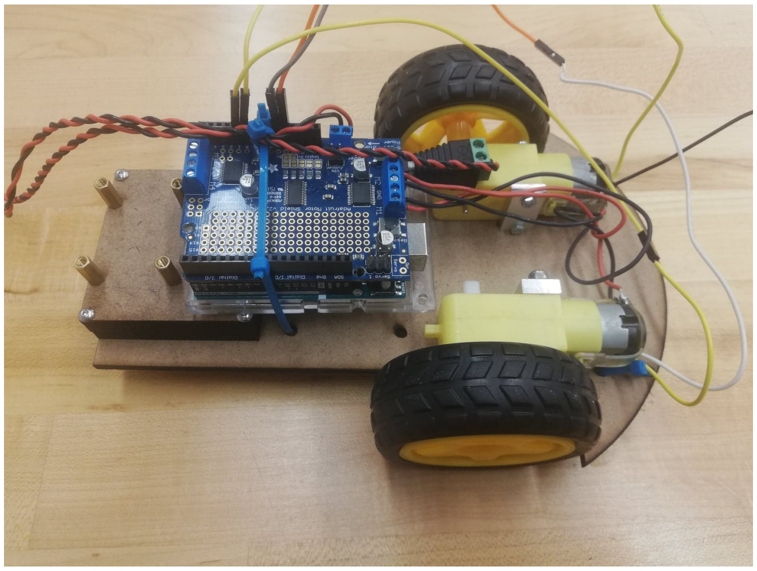


Figure 4: Picture of our assembled car

4 Results

After several iterations of testing and calibration, we were able to design a car that can follow black tape using infrared sensors and proportional control.

4.1 Sensor vs. Motor Speed

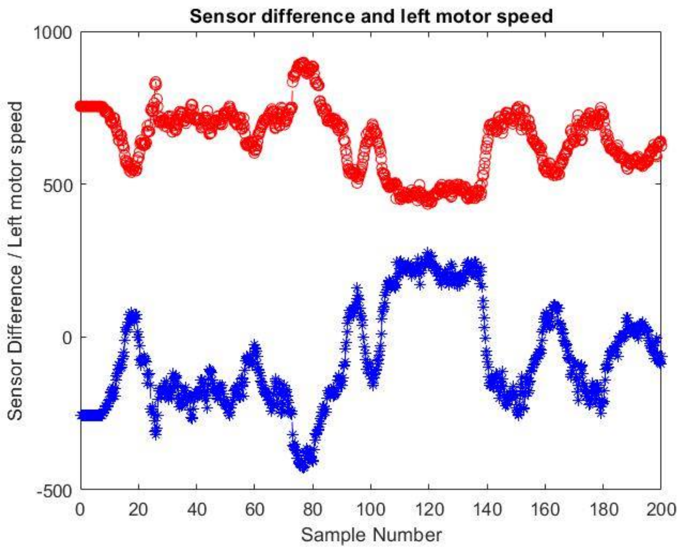


Figure 5: A plot of the left sensor data vs. the left motor speed over time as the car drives a track.

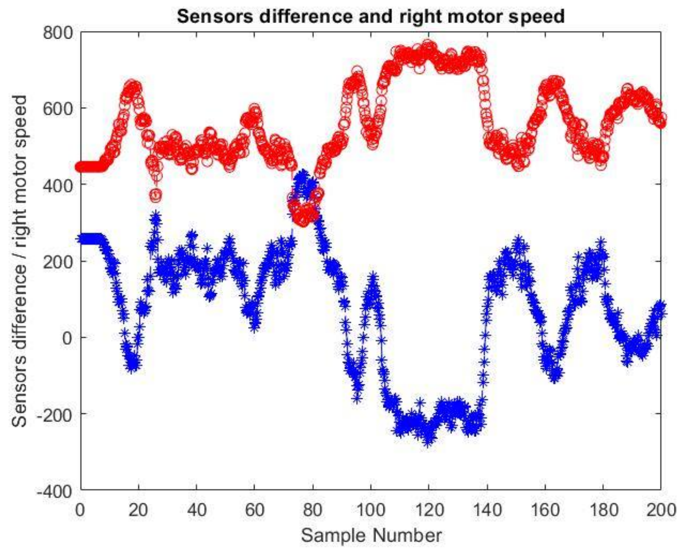


Figure 6: A plot of the right sensor data vs. the right motor speed over time as the car drives a track.

The graphs of the motor and sensor values are as we expected. The motor value, is scaled in order to be easier to compare it with the sensor output. The motor and sensor value follow opposite patterns.

4.2 Final Video

A video of our final line-following car can be found here: `https:/youtu.be/efYL0gq52b8`.

5 Discussion and Conclusion

5.1 Problems Encountered

During the design and fabrication of the car we encountered a number of different mecahnical problems. Because of our initial goal to have the car as close to the ground we could not just screw in the back wheel as the chassis would be lifted very high above the ground. To solve this problem we raised a section of the chassis so the back wheel can be screwed in without making the chassis unlevel. Furthermore, while assembling the car we realized that the cables did not have the required clearance when the Arduino board was placed between the motors and the back wheel. One more problem we encountered was with the electronic board that was holding the sensors system. The board had very small holes for the screws and we were not able to find a size of screws that could fit. To solve this problem we used the drill press to increase the hole size.

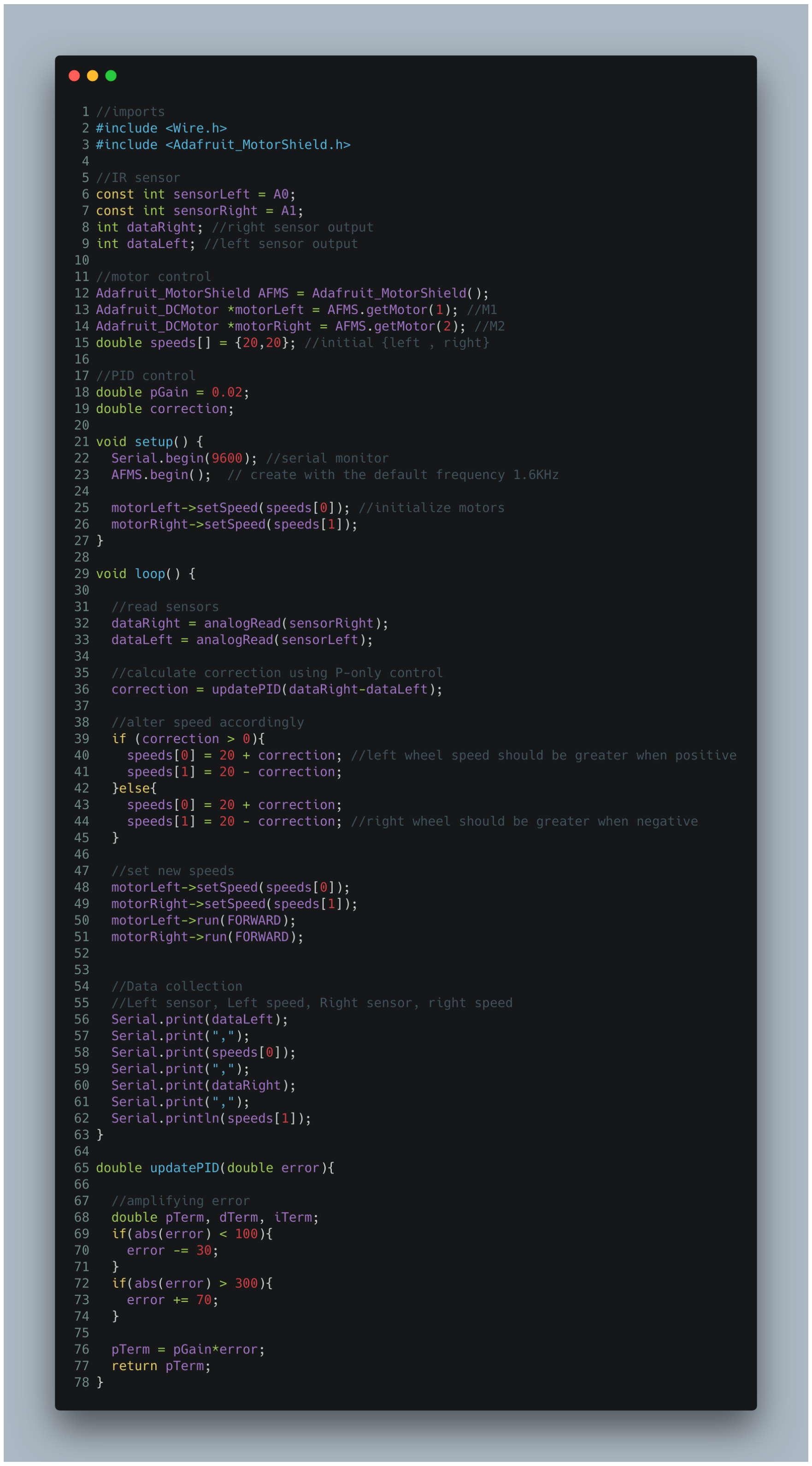
Throughout our process of testing and calibration, we encountered several electrical and software problems too. First, we realized our IR sensors were too far apart to correct motor speeds fast enough. To solve this problem, we re-aligned the sensors so that they line the outer edge of the black tape. The detection of tape also didn't impact the speed of the motors to a great degree. To increase the impact on the motors we amplified large errors and minimized small errors.

5.2 Reflection

Starting this lab we both had some experience with electrical, software and mechanical design from previews lab. However, we still encountered many problems during the design and test of the system. We believe we both got better at debugging as we spent a big part of the project trying to find the problems of our code. Moreover, we gained valuable experience in how to integrate mechanical and electrical systems together as sometime it is harder than it seems to be. The skills gained in this lab are going to be very useful to both of us for our final project that we are now starting.

6 Source Code

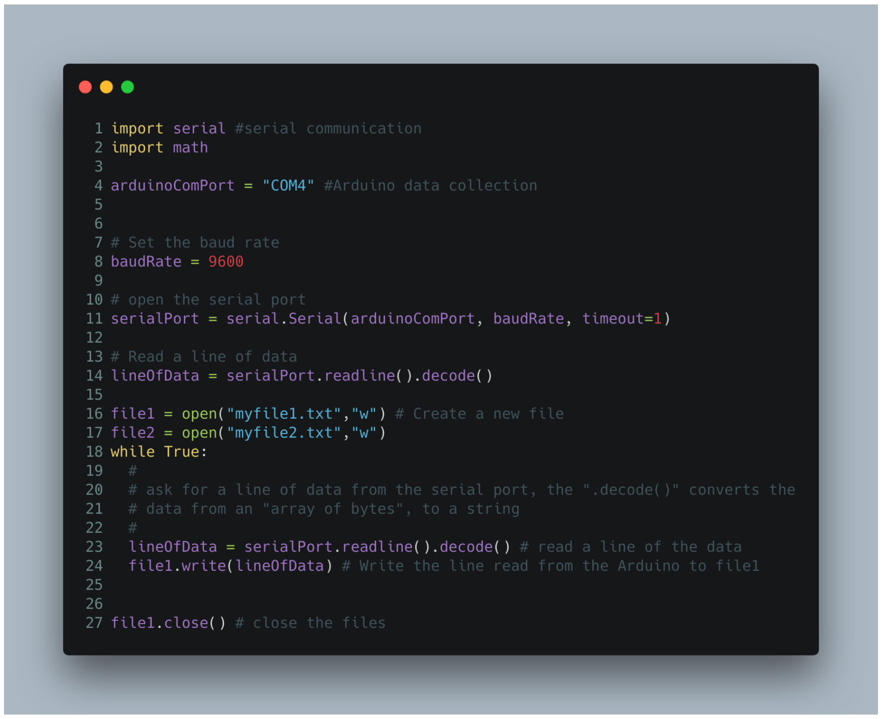
6.1 Sensor Application Code in C



```c
//imports
#include <Wire.h>
#include <Adafruit_MotorShield.h>

//IR sensor
const int sensorLeft = A0;
const int sensorRight = A1;
int dataRight; //right sensor output
int dataLeft; //left sensor output

//motor control
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *motorLeft = AFMS.getMotor(1); //M1
Adafruit_DCMotor *motorRight = AFMS.getMotor(2); //M2
double speeds[] = {20,20}; //initial {left , right}

//PID control
double pGain = 0.02;
double correction;

void setup() {
  Serial.begin(9600); //serial monitor
  AFMS.begin();  // create with the default frequency 1.6KHz

  motorLeft->setSpeed(speeds[0]); //initialize motors
  motorRight->setSpeed(speeds[1]);
}

void loop() {

  //read sensors
  dataRight = analogRead(sensorRight);
  dataLeft = analogRead(sensorLeft);

  //calculate correction using P-only control
  correction = updatePID(dataRight-dataLeft);

  //alter speed accordingly
  if (correction > 0){
    speeds[0] = 20 + correction; //left wheel speed should be greater when positive
    speeds[1] = 20 - correction;
  }else{
    speeds[0] = 20 + correction;
    speeds[1] = 20 - correction; //right wheel should be greater when negative
  }

  //set new speeds
  motorLeft->setSpeed(speeds[0]);
  motorRight->setSpeed(speeds[1]);
  motorLeft->run(FORWARD);
  motorRight->run(FORWARD);


  //Data collection
  //Left sensor, Left speed, Right sensor, right speed
  Serial.print(dataLeft);
  Serial.print(",");
  Serial.print(speeds[0]);
  Serial.print(",");
  Serial.print(dataRight);
  Serial.print(",");
  Serial.println(speeds[1]);
}

double updatePID(double error){

  //amplifying error
  double pTerm, dTerm, iTerm;
  if(abs(error) < 100){
    error -= 30;
  }
  if(abs(error) > 300){
    error += 70;
  }

  pTerm = pGain*error;
  return pTerm;
}
```

Figure 7

6.2 Data Collection Code in Python



```python
import serial #serial communication
import math

arduinoComPort = "COM4" #Arduino data collection


# Set the baud rate
baudRate = 9600

# open the serial port
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

# Read a line of data
lineOfData = serialPort.readline().decode()

file1 = open("myfile1.txt","w") # Create a new file
file2 = open("myfile2.txt","w")
while True:
  #
  # ask for a line of data from the serial port, the ".decode()" converts the
  # data from an "array of bytes", to a string
  #
  lineOfData = serialPort.readline().decode() # read a line of the data
  file1.write(lineOfData) # Write the line read from the Arduino to file1


file1.close() # close the files
```

Figure 8