Department of Electrical and Electronic Engineering
EEEE1027/EEEE1031 Applied Engineering Construction Project

Report: Project Weeks 2&3 - Smart Robot Car

Name: Euluna Gotami
Student ID: 20113429
Course: Electrical and Electronic Engineering

Contents

Introduction

A robot is a machine that is programmed to automatically carry out complex series of actions. A line following robot is an example of such - where it is programmed to follow a path on the ground which can be a black line (visible) or a magnetic field (invisible) by using input from sensors.

There are a lot of uses for line following robots in the real world. For example, it is used in smart warehouses where machines and computers are used to complete warehouse operations such as keeping track of products. Line following robots are used to transport goods from one location to another.

In manufacturing they are used to transport products between manufacturing plants in different buildings. Trucks with human drivers are inefficient since it involves a lot of uncontrollable factors. Carts programmed to follow a line could be used here as it is much more cost effective and convenient compared to laying railway tracks.

In short, they are very useful in industrial applications where using rails and conveyor belts are inefficient.

Self-driving cars can also take advantage of the existing markings on the road to adjust its path accordingly.

The main goal of this project is to successfully build a smart robot car controlled by an Arduino Uno as its brain and a motor driver (L298N) to help supply current needed to power the 2 5V DC motors. All these components would be powered by a 7.4V Li-Po Battery and mounted on an acrylic frame to form the basics of the car.

The objectives for this project are:

- Making a DIY IR sensor.
- Designing the car to follow a line and make various degrees of turns.
- Programming the car to display time taken to complete the track on LCD.
- Designing the car to measure and display distance travelled on LCD using an encoder sensor.
- Designing the car to avoid obstacles using an ultrasonic sensor.
- Designing the car to be controlled wirelessly by using a Bluetooth module.

This project will help to improve understanding and develop skills in electronics and programming by applying theoretical knowledge to real world applications.

Description of Hardware and Software Design

Hardware Required

DIY IR Sensor

The Hardware required for DIY IR Sensor:

- IR LED
- Green LED
- Stripboard
- Photodiode
- LM358 dual op amp
- Various resistors
- Potentiometer 100K

The details are discussed below.

Photodiode

A photodiode – also called as Photodetector, photo sensor or light detector – is a semiconductor diode that converts light energy into electrical voltage or current. They can operate in different modes mainly Photovoltaic mode, Photoconductive mode, and Avalanche diode mode. (What is a Photodiode?, 2018)

For light detection circuits – such as the DIY IR Sensor - the Photoconductive mode is used, where it is set up as shown in Figure 1. In this mode the photodiode works in reverse bias condition where the p-side of the photodiode is connected to the negative terminal and the n-side to the positive terminal of the battery (refer to Figure 2). This mode of operation reacts quickly to changes in light and therefor, is more suitable for line following application compared to other modes.

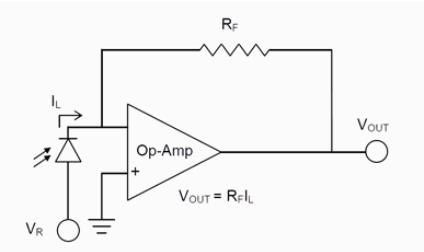


Figure 1: Photoconductive Mode

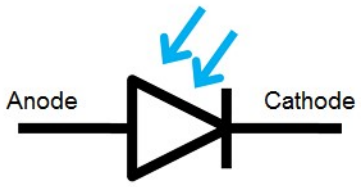


Figure 2: Photodiode Symbol

LM358 Dual Op Amp

The currents produced by a photodiode are usually small and thus may need to be amplified so that the output is on a usable level. This can be done by using an operational amplifier which is usually shortened to Op Amp.

An operational amplifier is an integrated circuit that can amplify weak electric signals. It has two input pins and one output pin. Its basic role is to amplify and output the voltage difference between the two input pins. Refer to Figure 3 for the diagram of an Op Amp. (What is an Operational Amplifier? - ABLIC Inc., n.d.)

LM358 is a dual op-amp IC integrated with two op-amps powered by a common power supply. Refer to Figure 4 for the LM358 diagram, or Appendix A for a more detailed version. The dual op-amp is used as it is more space efficient. (LM358 Dual OP-AMP IC, 2017)

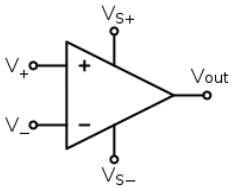


Figure 3: Circuit diagram symbol for an op amp

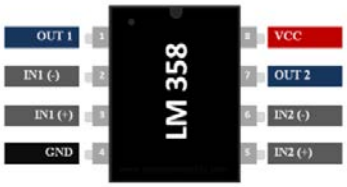


Figure 4: LM358 Diagram

DIY IR Sensor (comparator)

The DIY IR sensor is built on a breadboard based on the schematic in Figure 5. The materials needed were IR LED, green LED, photodiode, resistors (1M Ω 390 Ω, 100k Ω, and 1k Ω), and LM358 dual op amp.

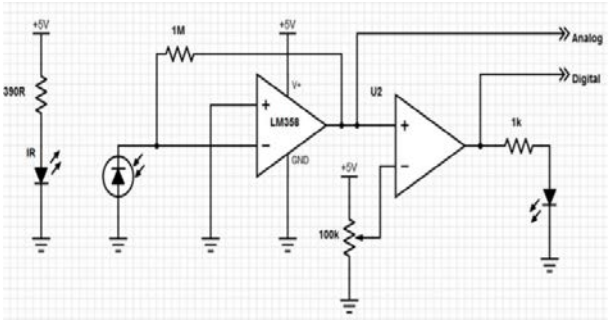


Figure 5: IR Sensor Schematic

When set up this way, the second op-amp compares the voltage levels between its inverting input and non-inverting input and gives an appropriate high and low output. In other words, it is used as a voltage comparator.

The equation related to the input and output voltages is:

$$V_o = G(V_+ - V_-)$$

Where G = 100 000 for LM358

Therefore, it can be concluded that:
If $V_+ > V_-$ then V_o = 5V (Digital High Output is 1)
If $V_+ < V_-$ then V_o = 0V (Digital Low Output is 0)

The voltage at one of the inputs is generally fixed at a reference voltage. In the circuit in Figure 5, the voltage at the inverting input is set by the variable resistor (photodiode) to keep the output high when light is at a specific level. When the light is less, current through the photodiode decreases. This causes the voltage output from the transimpedance amplifier to decrease which then causes a decrease in voltage at the non-inverting input of the comparator. (Kumar Jha, 2016)

Therefore, when the voltage drops below the reference voltage on the inverting pin to the comparator, the output swings to a low state and the LED turns off.

If the sensor is working on the breadboard, we can conclude that the components are all working and are compatible. Transferring the circuit to a stripboard is optional but

recommended as it makes the sensor more compact and permanent. This will also make mounting the sensor on the car easier. Refer to the schematic in Figure 6 for the arrangement of components on a stripboard. The finished DIY IR Sensor should look like Figure 7.

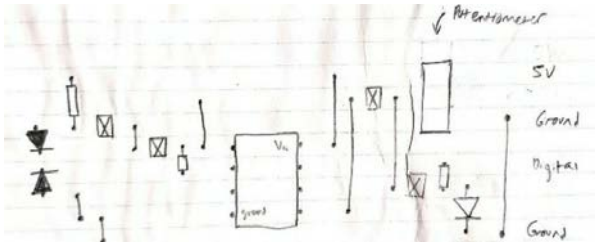


Figure 6: IR Sensor Stripboard Schematic

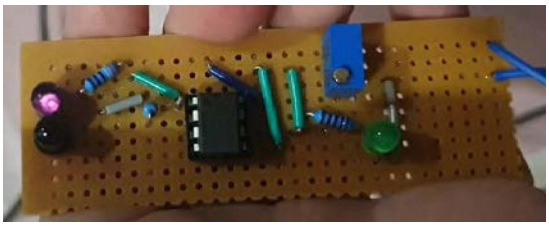


Figure 7: DIY IR Sensor

Smart Car/Robot

Hardware required for Smart Car/Robot:

- Arduino Uno
- Motor driver (L298N)
- 5V DC Motors
- Rotary Encoder Sensor
- Acrylic board
- LCD Shield
- Ultrasonic Sensor
- Bluetooth Module HC-05
- IR Sensor Modules
- Various resistors

Arduino Uno

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world. The heart of an Arduino is the microcontroller. For Arduino Uno, ATmega328 is used. It has specification of 8-bit CPU, 16 MHZ clock speed, 2 KB SRAM 32 KB flash Memory, 1 KB EEPROM. (Warren, Adams and Molle, 2011)

Refer to Appendix B for technical specifications and schematic of the Arduino Uno.

Motor Driver

The function of the motor driver is to act as an interface between motors and the controller. Motors need a high current to function while the controller need a low current. Therefore, the motor driver converts the low current signals to high to run the motor. It can also act as a Switching Device. Refer to Figure 8 for the diagram of L298N Motor Driver Module.

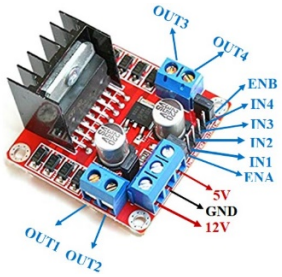


Figure 8: L298N Motor Driver Module diagram

DC Motor

A DC motor is an electrical machine that converts DC electrical energy into mechanical energy. Most commonly, they rely on the forces produced by magnetic fields and have an internal mechanism, either electromechanical or electronic to periodically change the direction of the current in part of the motor.

In this project, a 5V gear motor is used which usually comes in a set with a wheel (Figure 9). The speed of the motor can be controlled by changing the input voltage across the motor.

Figure 9: DC Gear Motor with Wheel

LCD Shield

The project requires the robot to display information such as the time taken to complete the track and distance travelled by the robot. The LCD Shield allows the robot to display information in a space of 2 rows and 16 columns (refer to Figure 10) and also allows the user to input information through the keypad. However, this function will not be used in the project. (Olfat, 2019)

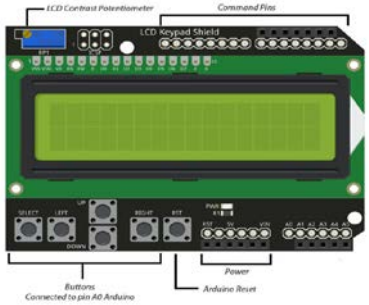


Figure 10: LCD Shield Diagram

IR Sensor module

An IR Sensor module consists of an IR emitter (IR LED) and receiver (photodiode). It works by measuring the amount of infrared light reflected from its own LED to determine if an object is in front of it. It will output a digital signal that can be received by the Arduino depending on what it is detecting. (IR Sensor Module, 2018)

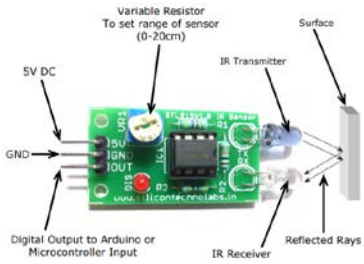


Figure 11: IR Sensor Module Diagram

Black objects absorb all light including infrared light. Using this understanding, it can be concluded that the IR Sensor will be able to differentiate the black line from the white ground.

Ultrasonic Sensor

An ultrasonic sensor measures distance to an object using ultrasonic sound waves. It consists of a transmitter and receiver that sends out ultrasonic pulses (refer to Figure 12). The distance is calculated by using the time taken for the sound to bounce back, and the speed of sound in air. (Burnett, 2020)

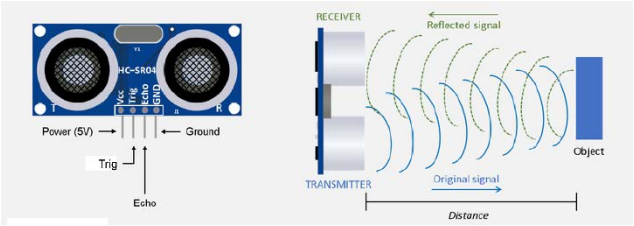


Figure 12: Ultrasonic Sensor Diagram

Bluetooth Module HC-05

The Bluetooth module allows communication between two devices - such as the Arduino and a smartphone - wirelessly through a Bluetooth connection. As shown in Figure 13, the RX pin needs an input of 3.3V. Since the Arduino outputs 5V, connecting it directly could potentially damage the module. To prevent this, a voltage divider circuit is necessary to lower the voltage.

Figure 13: Bluetooth Module

Rotary encoder Sensor

Rotary encoder is a type of sensor that is used to determine the angular position of a rotating shaft and generates an electrical signal accordingly. The photo sensor detects whether the light from the LED is coming through the holes in the disk (refer to Figure 14) and generates a square wave output. We can then find the number of times the reading changes state to calculate the total distance travelled by comparing it with the diameter of the wheel. (How Rotary Encoder Works, 2016)

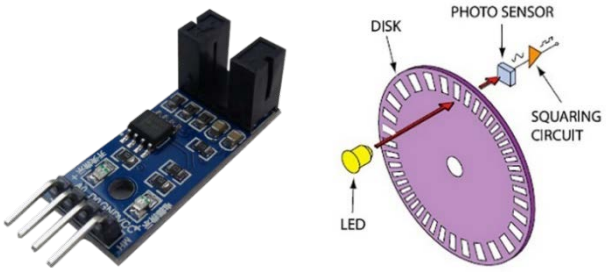


Figure 14: Encoder Sensor

Software Required

Arduino Version 1.8.42.0 was used for coding and uploading the sketches.

Hardware and Software Designs

Before starting the project, the basics of the car must first be assembled following the schematic in Figure 15. Figure 16 shows what the connections look like in real life.

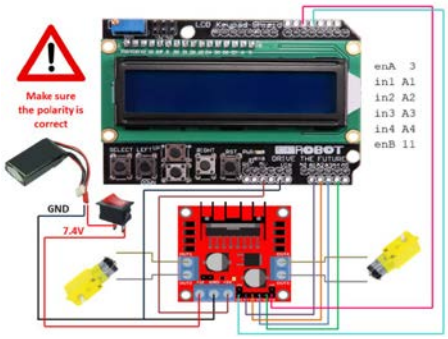


Figure 15: LCD to Motor Driver Connection

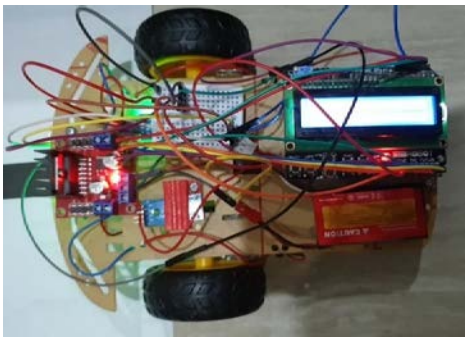


Figure 16: Basic Car

Line Following

Hardware and Logic

To enable the car to follow the line, IR sensors must be attached to the front of the robot. In this project, 2 IR sensor modules is used. Table 1 shows the actions the car will take depending on the inputs from the sensors.

Table 1: Arduino Working Logic

Input		State	Action
Left Sensor	Right Sensor		
0	0	Both sensors detect white	Forward
1	0	Left sensor detects black line	Turn Left
0	1	Right sensor detects black line	Turn Right
1	1	Both sensors detect black line	Stop

In words, the car will keep on moving forward as long as both sensors detect white. When the left sensor detects black, it will prompt the car to turn left until both sensors detect white. Then, it will proceed to move straight forward again. The opposite happens when the right sensor detects black. The car will stop when both sensors hit black.

Programming

The diagrams of Figure 17 and flow chart in Appendix C further displays the software requirements needed to follow the line. If else statements can be used to specify actions taken depending on the conditions. The final program can be found in Appendix D.

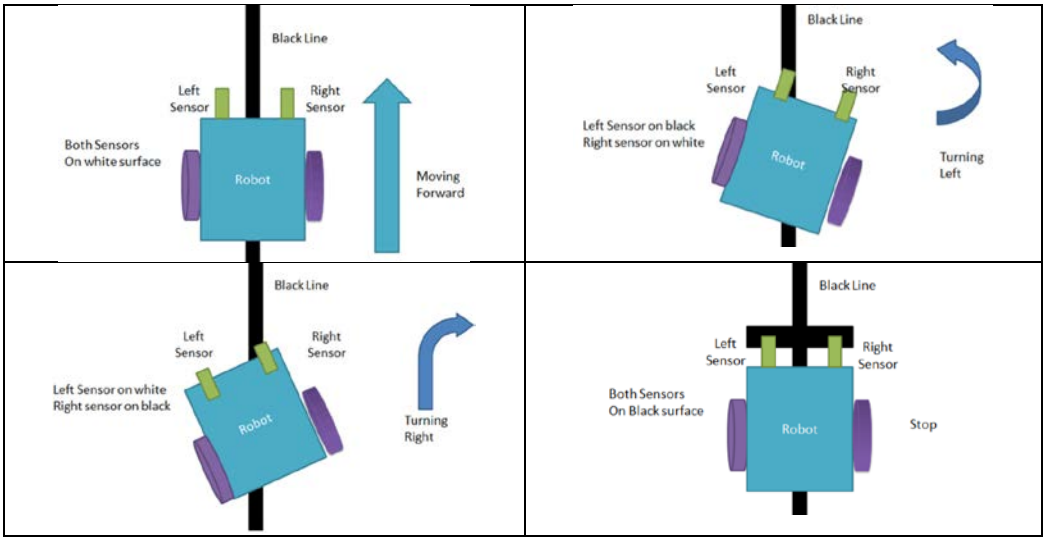


Figure 17: Line Following process

Line Following + Display Time

Hardware

This task uses the exact same hardware as the Line Following task.

Programming.

This task requires knowledge on using the Liquid Crystal library and Millis function. The Liquid Crystal library enables us to print information on the LCD shield while the Millis function returns the number of milliseconds passed since the Arduino board began running the program. (Aasvik, 2017)

The delay function would not work in this application as it would halt the rest of the program and interfere with the line following capabilities. Millis utilises the internal clock in the Arduino which means that it runs independently than the rest of the code. Therefore, it is much more accurate.

The programming requirements is clearly explained in the flowchart in Figure 18. The Exit function can be used to freeze the LCD. The final working code can be found in Appendix E.

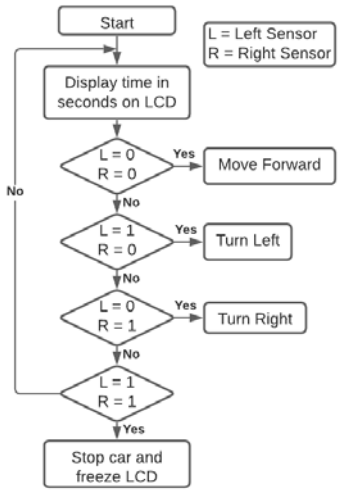


Figure 18: Line Following and Display Time Flow Char

Line Following + Display Distance Travelled

Hardware

This task requires the addition of a rotary encoder sensor to measure the total distance travelled. As shown in Figure 14, and as explained before, the encoder disk consists of slots. As the disk rotates with the wheel, the sensor will detect whether light is coming through the slots or not. The sensor will return a square wave. With this data, we can get the number of slots that have passed the sensor.

Encoder sensors must be connected to interrupt pins – which have higher priority - to work properly. However, the Arduino Uno only contains 2 interrupt pins where one is already being used by the LCD shield. This means that we are limited to only 1 encoder sensor. It should not be a problem in situations where the left and right wheel travel the same distance.

Programming

The flow chart of the programming of this task can be seen in Figure 19. The task requires knowledge of how to use interrupt pins. In this case, whenever the interrupt pin detects that its input is rising, it will execute its commands which in this case are to increase the step counter, and to calculate and display the new total distance travelled.

The step counter is the total number of slots that have passed by the encoder sensor.

We can obtain the circumference of the wheel by measuring its diameter and using the formula $circumference = diameter \times \pi$. Once we have these values, we can count the number of slots in the encoder disk - which may vary depending on brand - and use it to calculate the distance travelled for every slot passed.

Therefore we can use formula $Total\ distance = Step\ counter \times distance/slot$. The complete functioning code can be found in Appendix F

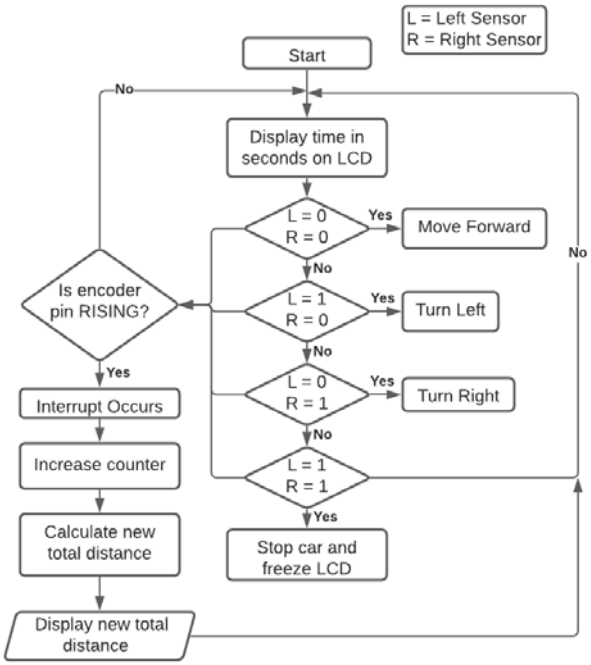


Figure 19: Line Following and Display Distance Flow Chart

Obstacle Avoiding Car

Hardware

This task requires the use of an ultrasonic sensor that is placed facing forward, in the front of the car. The ultrasonic sensor sends out ultrasound pulses and measures the time taken for the pulse to be reflected back. It will then use the speed of sound in the air to calculate its distance to an object.

As shown in Figure 12, the ultrasonic sensor contains 4 pins which are Vin, ground, echo and trig. The echo and trig pins can be connected to either the digital or analog pins.

Software

To check for distance, we will first program the sensor to send a pulse and wait for it to be bounced and received again. When the pulse is received, it will cause the echo pin to return HIGH. The time taken for the pulse to be received is returned by the sensor.

Using this information, we can convert it to centimetres by dividing it by 58.

The basics of the program is explained in Figure 20. The car is programmed to keep moving forward as long as the nearest object is more than 20 cm away. Once the distance gets to less than 20 cm, the car will reverse slightly and turn right to avoid it. The final working code can be found in Appendix G.

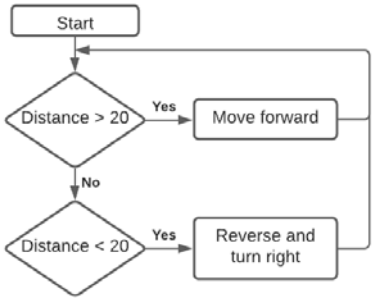


Figure 20: Obstacle Avoiding Car Flow Chart

Bluetooth Controlled Car

Hardware

For this task, a Bluetooth module is to be connected to the car. This will allow an external device to send inputs or commands to the robot while the program is running. The Bluetooth module used can be seen in Figure 13. As shown, the RX module only needs 3.3V. Therefore, the voltage divider shown in Figure 21 has to be used.

The Module has 6 pins but only 4 – Vin, ground, RX and TX - will be needed for the task. The RX pin is to be connected to the TX pin on Arduino, and the TX pin to the RX pin on Arduino.

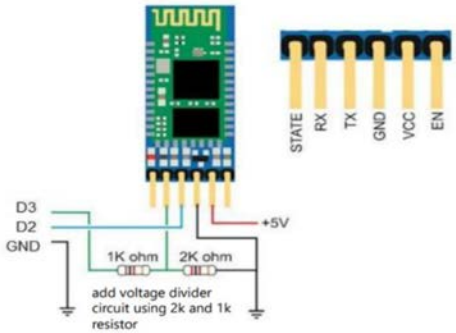


Figure 21: Bluetooth Voltage Divider

Software

A remote controller app needs to be downloaded on the smartphone used. The app used here was “Arduino Bluetooth Robot Car - Remote Controller” by Bluino Electronics. Figure 21 shows the interface of the app where each arrow and arrow combination will send out a corresponding letter to the Arduino when pressed.

Figure 21: Bluetooth RC App

The letters used will be different for every app and can be found by either using the serial monitor or checking the tutorial section of the app. Table 1 shows the actions the robot will take depending on the arrows pressed.

Table 1: Arduino Working Logic

Arrows pressed	Command	Action
When no arrow pressed	S	Stop
↑	F	Forward
↓	B	Backward
←	L	Left
→	R	Right
↑ + →	I	Forward Right
↑ + ←	G	Forward Left
↓ + →	J	Backward Right
↓ + ←	H	Backward Left

With the information shown on the table, we can use a switch case or if statements to code the conditions. The working code can be found in Appendix H where the switch case statements were used.

Results, Analysis and Discussion

DIY IR Sensor

The placement of the components must be done accurately and precisely. Or else, there is a risk of components shorting and burning. To test if the circuit is working, first make sure that the pot is not wound to the max or min since it would affect the results. Next is to put an object or hand in front of the IR LED and photodiode. If the LED lights up, it means that the circuit is detecting the object. Else, try to rotate the pot – which will change the sensitivity - and try again.

Issues encountered could be:

- IR LED and Photodiode not close enough
- Loose connection
- Shorted or faulty components.

It is much more convenient to use a ready-made sensor since it is much more reliable and compact. In addition, they are relatively inexpensive to buy.

Line Following

When testing the car on the track, I observed that the car was completely ignoring the track. From understanding of how IR sensors work, I concluded that the path was not dark enough since the IR sensors were still detecting the reflected light. The solution to this was to make the path darker by either reprinting it with darker ink or colouring it with a marker which was what I did. The speed was also lowered so that the car has more time to react.

The next tests showed that the sensor was still giving inaccurate readings. This time it was because the track was not plane enough. And since the sensors were very close to the ground, the creases from the paper were enough to physically cover the sensors. This issue was resolved by rebuilding the track more carefully and making sure that it is completely flat.

In the following tests, the car was able to follow the line in the first half of the track. However, it had a hard time when it reached the S curve since there was a smaller turning radius which meant that both sensors were hitting black line (the condition for the car to stop). This was resolved by making the opposite wheel reverse while turning to reduce the turning radius of the car. In addition, the distance between the sensors were adjusted so that it is wider than the width of the track but not wide enough that it ignores the curves.

In the following tests, the car was able to complete the track with no issues 6/8 times. This is because the car is also influenced by other factors such as the lighting in the room or the track was not dark enough in some spots.

Line Following + Display Time

The tests first showed that the car was able to successfully complete the course. However, when timed with a stopwatch, the time displayed was slower than the actual. This was because the delay function was used. Delay is not suitable here as the program takes some time to run its commands. Which means that if for example, the program takes 1ms to run

and a delay of 10ms was used, the actual time passing would be 11ms, but only 10ms was added to the counter. The Millis function is more suitable for this since it uses the internal clock of the Arduino and runs independently (not influenced by code).

The accuracy of the timer was tested by allowing the car to run for 4 minutes (above ground) and comparing the results with a stopwatch. The idea behind this is that inaccuracies in the timer will be more obvious after more time has passed. There was only a 2 second difference in the stopwatch and display. This was most likely due to human error.

Line Following + Display Distance

Before attempting to combine the codes, the distance measuring capabilities were tested first. This is to help make debugging the code easier since there are less variables to worry about.

To do this, the motors were first disabled, and the car was pushed along the length of a 30cm ruler. The results of the first few tests displayed 122 cm, 121 cm, and 118 cm. This was about 4 times the actual distance which was odd. To fix this, I divided the outputs by 4. Now the outputs were very close to actual.

In the next tests, a greater distance was to be measured with the idea that inaccuracies would become more obvious. The car was pushed along a 69cm straight section of a track and displayed an average of 67cm which was pretty close to actual.

Since the measuring section of the code was working well, I proceeded to merge it with the line following function.

I decided to measure a greater distance and used the beginner track (refer to Appendix I). It was roughly measured with a ruler to be 270 cm. When the car ran the track 3 times, it showed an average of 265 cm which was very close.

The inaccuracies were most likely because the ruler could not measure the curves accurately.

Obstacle Avoiding Car

The car and code were first tested by using the serial monitor to display the distance between the car and object. The car was placed 20 cm from the wall using a ruler to measure. The serial monitor displayed 21 cm which was acceptable.

When tested on the ground, the car collided with the wall. This was most likely because the car was too fast, and the minimum distance was set too low. I lowered the speed of the car and increased the minimum distance.

Bluetooth Controlled Car

The Bluetooth commands have been discussed in Table 1. The benefits of controlling the car via Bluetooth is that there is no need to use cables to transmit data to the car. However, wired connections are still generally more reliable. This was evident during testing.

There was slight difficulty getting the smartphone and car to connect. After restarting both devices, it was connected. Sometimes during testing the car would lose connection to the phone and collide to a wall.

When the connection was good, the car was able to respond the commands correctly.

Bluetooth Controlled Car + Obstacle Avoiding

Since there were multiple times when the Bluetooth car lost connection and hit the wall, it would be a good idea to add obstacle avoiding capabilities to the car. The previous codes were merged, and the car can now avoid head on collisions by reversing when an object is near.

Conclusion

The line following robot has the ability to follow a path – that is dark and unreflective - and make the necessary curves and turns to remain on it. It detects its position relative to the path using 2 IR sensors. However, the car is dependant on the surrounding conditions such as lighting of the room.

Additional features such as timer and distance measuring capabilities can be added to the car. The timing function is very accurate since it uses the internal clock of the Arduino to keep track of time and the distance measuring capabilities is also accurate with an inaccuracy of less than 2%.

The car can also be modified to avoid obstacles using an ultrasonic sensor, and to have Bluetooth functionality by using a Bluetooth module. A more useful application is to combine these two functions to have a Bluetooth car that can avoid head on collisions.

I found that this project was a fun way of getting more exposure to real world use of electronics since it had a mix of industrial (such as the line following robot) and consumer applications (Bluetooth controlled car) which was interesting to learn. The project also helps to problem solving skills and the ability to self-learn since it encourages us to find solutions on our own. Furthermore, it made us start to learn how to solder (DIY IR Sensor).

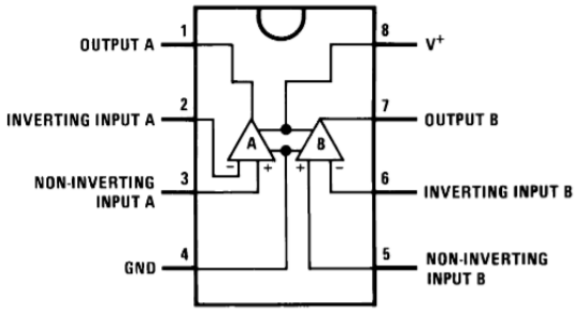
For future work, I would improve the line following capabilities of the robot by including a PID controller which was not done due to time constraints of the project. Furthermore, currently the obstacle avoiding car is only able to avoid hear on collisions since there is only one sensor. Next time I would add sensors on its side so that it can avoid diagonal collisions.

Reference

1. Aasvik, M., 2017. Arduino Tutorial: Using Millis() Instead Of Delay(). [online] Norwegiancreations.com. Available at: <https://www.norwegiancreations.com/2017/09/arduino-tutorial-using-millis-instead-of-delay/> [Accessed 5 January 2021].
2. Burnett, R., 2020. Understanding How Ultrasonic Sensors Work | Maxbotix Inc.. [online] MaxBotix Inc. Available at: <https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm.> [Accessed 5 January 2021].
3. HowToMechatronics. 2016. How Rotary Encoder Works. [online] Available at: <https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/> [Accessed 5 January 2021].
4. Components101. 2018. IR Sensor Module. [online] Available at: <https://components101.com/sensors/ir-sensor-module> [Accessed 5 January 2021].
5. Kumar Jha, S., 2016. Project Report Line Following Robot. [online] Available at: <https://www.researchgate.net/publication/327814718_PROJECT_REPORT_LINE_FOLLOWING_ROBOT> [Accessed 5 January 2021].
6. Components101. 2017. LM358 Dual OP-AMP IC. [online] Available at: <https://components101.com/ic-lm358-pinout-details-datasheet> [Accessed 5 January 2021].
7. Olfat, S., 2019. Connecting Arduino To Firebase To Send & Receive Data [By ESP8266]. [online] Electropeak. Available at: <https://electropeak.com/learn/connect-arduino-esp8266-firebase-send-receive-data/> [Accessed 5 January 2021].
8. Warren, J., Adams, J. and Molle, H., 2011. Arduino Robotics. [New York]: Apress, pp.51-83.
9. https://www.electronicshub.org. 2018. What Is A Photodiode? Working, Characteristics, Applications. [online] Available at: <https://www.electronicshub.org/photodiode-working-characteristics-applications/> [Accessed 5 January 2021].
10. ABLIC Inc. n.d. What Is An Operational Amplifier? - ABLIC Inc.. [online] Available at: <https://www.ablic.com/en/semicon/products/analog/opamp/intro/> [Accessed 5 January 2021].

Appendixes

Appendix A - LM358 Dual Operational Amplifier Datasheet



Appendix B - Technical Specifications and schematic of Arduino Uno

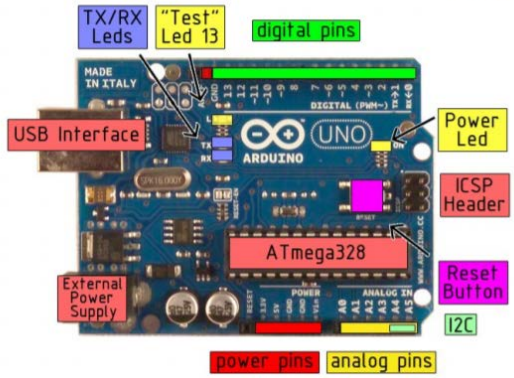
Technical Specification

EAGLE files: arduino-duemilanove-uno-design.zip Schematic: arduino-uno-schematic.pdf

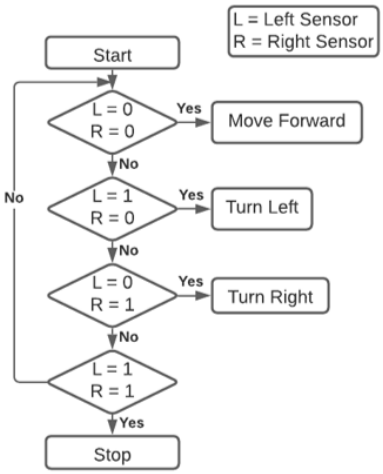
Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

the board



Appendix C - Line Following Car Flow Chart



Appendix D – Line Following Code (Include functions from Appendix J)

```
#define enA  3
#define in1 A1
#define in2 A2
#define in3 A3
#define in4 A4
#define enB 11

#define sensorA 12
#define sensorB 2

void setup() {
  Serial.begin(9600);
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(sensorA,INPUT);
  pinMode(sensorB,INPUT);
}

void loop() {
  Serial.println(digitalRead(sensorA));
  Serial.println(digitalRead(sensorB));

  if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==LOW){
    motorForward();

    analogWrite(enA, 60); // Send PWM signal to
motor A
    analogWrite(enB, 60); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==LOW){
    // Turn left
    motorAReverse();
    motorBForward();

    analogWrite(enA, 72); // Send PWM signal to
motor A
    analogWrite(enB, 87); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==HIGH){
    // Turn right
    motorAForward();
    motorBReverse();

    analogWrite(enA, 72); // Send PWM signal to
motor A
    analogWrite(enB, 87); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==HIGH){
    motorStop();
    exit(0);
  }
}
```

Appendix E – Line Following + Display Time Code (Include functions from Appendix J)

```
#include <LiquidCrystal.h>
#define pin_RS 8
#define pin_EN 9
#define pin_d4 4
#define pin_d5 5
#define pin_d6 6
#define pin_d7 7
#define pin_BL 10
#define enA  3
#define in1 A1
#define in2 A2
#define in3 A3
#define in4 A4
#define enB 11
#define sensorA 12
#define sensorB 2
LiquidCrystal lcd( pin_RS,  pin_EN,  pin_d4,
pin_d5,  pin_d6,  pin_d7);

float time1;

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("Timer:");
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(sensorA,INPUT);
  pinMode(sensorB,INPUT);
}

void loop() {
  Serial.println(digitalRead(sensorA));
  Serial.println(digitalRead(sensorB));

  if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==LOW){
    motorForward();

    analogWrite(enA, 60); // Send PWM signal to
motor A
    analogWrite(enB, 60); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==LOW){
    // Turn left
    motorAReverse();
    motorBForward();

    analogWrite(enA, 72); // Send PWM signal to
motor A
    analogWrite(enB, 87); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==HIGH){
    // Turn right
    motorAForward();
    motorBReverse();

    analogWrite(enA, 72); // Send PWM signal to
motor A
    analogWrite(enB, 87); // Send PWM signal to
motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==HIGH){
    motorStop();
    exit(0);
  }

  lcd.setCursor(0,1);
  time1 = int(millis()/1000);
  lcd.print(int(time1));

  lcd.setCursor(3,1);
  lcd.print("Seconds");
}
```

Appendix F - Line Following + Display Distance Travelled

(Include functions from Appendix J)

```
#include <LiquidCrystal.h>
#include "TimerOne.h"

#define pin_RS 8
#define pin_EN 9
#define pin_d4 4
#define pin_d5 5
#define pin_d6 6
#define pin_d7 7
#define pin_BL 10

LiquidCrystal lcd( pin_RS,  pin_EN,
pin_d4,  pin_d5,  pin_d6,  pin_d7);

#define enA  3
#define in1 A1
#define in2 A2
#define in3 A3
#define in4 A4
#define enB 11

//IR Sensors
#define sensorA 12
#define sensorB 13

//Encoder
#define encoder 2
#define pi 3.142
#define diskSlots 20.00   //Number of slots
in encoder disk
#define diameter 6.40     //Wheel diameter
in cm

volatile int counter = 0;

int distanceInCM(int counter){

  float result;
  float circumference = diameter*pi;
  float CMPerSlot =
circumference/diskSlots; //distance
traveled by car every slot

  //counter for 1 rotation should be 20.
  //for unknown reasons, when tested, 1
rotation adds 80 to counter.
  //to correct for this error, result is divided
by 4 below.

  CMPerSlot = CMPerSlot/4.00;

  float f_result = counter * CMPerSlot;
  result = (int) f_result; //rounded down

  return result;
}

void ISR_timer(){
  Timer1.detachInterrupt();

  lcd.setCursor(0,1);
  int distance = distanceInCM(counter);
  lcd.print(int(distance));

  lcd.setCursor(4,1);
  lcd.print("CM");

  Timer1.attachInterrupt(ISR_timer);
}

void ISR_count(){
  counter++;
}

void setup() {
  Serial.begin(9600);

  Timer1.initialize(1000000);
```

```
attachInterrupt(digitalPinToInterrupt(enco
der), ISR_count, RISING); //will run
ISR_count when encoder pin detects 0 to 1
  Timer1.attachInterrupt(ISR_timer);

  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(sensorA,INPUT);
  pinMode(sensorB,INPUT);

  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("Distance:");

  motorStop();
  delay(2000);

  motorForward();
  motorSetSpeed(100);
  delay(200);
}

void loop() {
  Serial.println(digitalRead(sensorA));
  Serial.println(digitalRead(sensorB));

  if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==LOW){
    motorForward();

    analogWrite(enA, 70); // Send PWM
signal to motor A
    analogWrite(enB, 70); // Send PWM
signal to motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==LOW){
    // Turn left
    motorAReverse();
    motorBForward();

    analogWrite(enA, 70); // Send PWM
signal to motor A
    analogWrite(enB, 80); // Send PWM
signal to motor B
  }
  else if(digitalRead(sensorA)==LOW &&
digitalRead(sensorB)==HIGH){
    // Turn right
    motorAForward();
    motorBReverse();

    analogWrite(enA, 80); // Send PWM
signal to motor A
    analogWrite(enB, 70); // Send PWM
signal to motor B
  }
  else if(digitalRead(sensorA)==HIGH &&
digitalRead(sensorB)==HIGH){
    motorStop();
  }

}
```

Appendix G – Obstacle Avoiding Car

```
#define enA  3
#define in1 A1
#define in2 A2
#define in3 A3
#define in4 A4
#define enB 11

#define echo 12
#define trig 13

long duration, distance;

void setup() {
  Serial.begin(9600);

  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  pinMode(echo, INPUT);
  pinMode(trig, OUTPUT);
}

void loop() {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);
  distance = duration /(2*29.1); //distance
in CM

  if (distance<20){
    motorReverse();
    analogWrite(enA, 100); // Send PWM
signal to motor A
    analogWrite(enB, 100); // Send PWM
signal to motor B
    delay(250);


    motorAForward();
    analogWrite(enA, 100); // Send PWM
signal to motor A
    analogWrite(enB, 100); // Send PWM
signal to motor B

    motorBReverse();
    analogWrite(enA, 100); // Send PWM
signal to motor A
    analogWrite(enB, 100); // Send PWM
signal to motor B
    delay(100);
  }

  else{
    motorForward();
    analogWrite(enA, 80); // Send PWM
signal to motor A
    analogWrite(enB, 80); // Send PWM
signal to motor B
  }
}
```

Appendix H – Bluetooth Controlled Car

```
#define enA  3
#define in1 A1
#define in2 A2
#define in3 A3
#define in4 A4
#define enB 11

char command;

void setup(){
  Serial.begin(9600);
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  analogWrite(enA, 85); // Send PWM
signal to motor A
  analogWrite(enB, 85); // Send PWM
signal to motor B
}

void loop(){
  while (Serial.available() > 0)
  {
    command = Serial.read();
    Serial.println(command);
  }

  if ( command == 'F') // Forward
  {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  }

  else if (command == 'B') // Backward
  {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }

  else if (command == 'L') //Left
  {
    // left
    // Set Motor A Reverse
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // Set Motor B Forward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  }

  else if (command == 'R') //Right
  {
    // right
    // Set Motor A Forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    // Set Motor B Reverse
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }

  else if (command == 'S') //Stop
  {
    // Set Motor A B Stop
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
  }

  else if (command == 'I') //Forward Right
  {
    // right
    // Set Motor A Forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
```

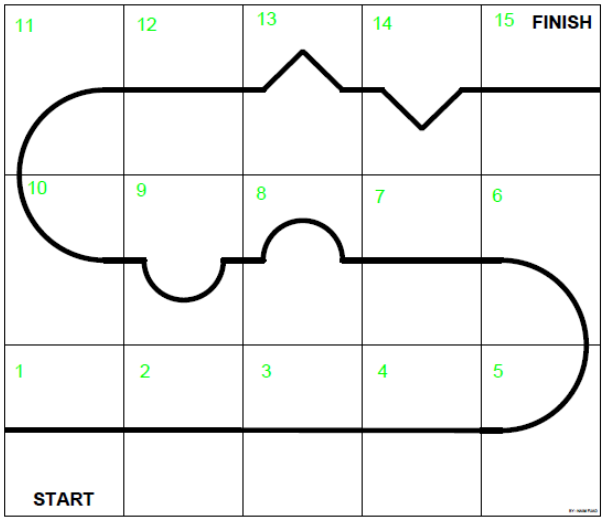
```
    // Set Motor B Reverse
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }

  else if (command == 'G') //Forward Left
  {
    // left
    // Set Motor A Reverse
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // Set Motor B Forward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  }

  else if (command == 'J') //Backward
Right
  {
    // right
    // Set Motor A Forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    // Set Motor B Reverse
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }

  else if (command == 'H') //Backward Left
  {
    // left
    // Set Motor A Reverse
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // Set Motor B Forward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
  }
}
```

Appendix I – Beginner Track



Appendix J – Car Movement Functions

```
void motorAForward(){
  // Set Motor A Forward
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
}

void motorBForward(){
  // Set Motor B Forward
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

void motorAReverse(){
  // Set Motor A Reverse
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
}

void motorBReverse(){
  // Set Motor B Reverse
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}

void motorForward(){
  // Set Motor Forward
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

void motorReverse(){
  // Set Motor Reverse
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}

void motorSetSpeed(int motorSpeed){
  // Set Motor Speed
  analogWrite(enA, motorSpeed); // Send
PWM signal to motor A
  analogWrite(enB, motorSpeed); // Send
PWM signal to motor B
}

void motorStop(){
  // Set Motor Stop
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  analogWrite(enA, 0);
  analogWrite(enB, 0);
}
```