Department of Electrical and Electronic Engineering

EEEE1027

Applied Electrical and Electronic Engineering Construction Project

Report 2

Max Koo Yan Jie

20620542

December 2023

Abstract

Automation is undoubtedly one of the major themes driving technology innovation in the 21st Century. Among the most significant advancements are autonomous vehicles, from private personal transportation to ones used on an industrial scale. A good starting point to understand autonomous vehicles is a simple line-following vehicle. This report details and documents the construction of a line-following vehicle using two IR sensors and an Arduino UNO. Additionally, it goes through the implementation of several additional features, including distance calculation using a rotary encoder, ramp angle calculation using the MPU6050 Accelerometer Gyroscope, Bluetooth control using the HC-05 Bluetooth Serial Transceiver, obstacle avoidance using the HC-SR04 Ultrasonic Sensor, a piezoelectric buzzer, and travelling in a straight line with PID control using two rotary encoders. The report also presents the results obtained from each feature implemented and the limitations and potential improvements of this line-following vehicle project.

Table of Contents

1 Introduction

In the context of modern engineering, the pursuit of automation has ushered in a new era of technology capable of revolutionising the way humans work. Within this scope of automation, one of the most prominent fields lies in the advent of autonomous vehicles. Technology has advanced to the point where dreams of vehicles capable of navigation without human intervention are now possible. The increased safety, efficiency, and accessibility that autonomous vehicles provide [1] will no doubt shape the future of transportation.

Autonomous vehicles are an accomplishment built upon the combination of various different fields of engineering comprising electronics, robotics, artificial intelligence, and software engineering. The synthesis of advanced sensors, machine learning algorithms, and real-time control systems required for autonomous navigation presents a challenge that demands interdisciplinary expertise. This makes autonomous vehicles continually relevant in the broader engineering context. As such, they are a significant focus of various industries, most notably the automotive industry, which invests heavily in the research and development of these vehicles for the mass market.

Beyond the realm of personal transportation, autonomous vehicles also have a place in many other industries, such as logistics, manufacturing, and even healthcare. Automated guided vehicles (AGVs), which are a subset of autonomous vehicles that are most often used in industrial applications, are able to optimise material handling, transportation, and sorting in warehouses, distribution centres, and manufacturing facilities. These systems provide increased efficiency and improved safety, as it eliminates the possibility of human error and human injury [2]. In the healthcare sector, autonomous vehicles are present in the form of robotic surgical systems, no-contact medication distribution, patient transportation, and emergency medical vehicles [3]. This showcases just how versatile and impactful the technology is.

All of these make research into automation and autonomous vehicles a worthwhile endeavour. The current and potential benefits of autonomous vehicles cannot be understated, as they will only continue to be more relevant in every aspect of daily life.

1.1 Project Background

A line-following vehicle is a vehicle that is able to follow a line automatically without any human control. It represents one of the most fundamental types of autonomous vehicles. This project aims to construct a simple line-following vehicle using 2 IR sensors and a microcontroller. The IR sensor used is to be designed and constructed on a Veroboard. Using infrared light, the IR sensors can detect a line of high colour contrast, as the darker colour will reflect less IR light, and vice versa. The signal from both sensors is sent to an Arduino UNO R3 microcontroller programmed to make speed adjustments of the motors in real-time, thereby controlling the vehicle's trajectory to follow the line. By focusing on the fundamental concept of following a predefined path, this project lays the groundwork for comprehending the underlying principles of autonomous vehicle control.

In addition to following a line, this project also aims to use various sensors and components to increase the vehicle's capabilities. As the goal for autonomous vehicles is to be fully self-driving, it is essential to be able to design and integrate different sensors into a single, self-contained system. To showcase these capabilities, the vehicle must also complete specific tasks detailed in Section 1.2 of the report. With its challenges and complexities, this project serves as a valuable stepping stone in understanding the intricacies of autonomous navigation.

1.2 Project Objectives

This project was executed in fulfilment of the University of Nottingham Malaysia's Year 1 Autumn Semester Applied Electrical and Electronic Engineering Construction Project for the year 2023. The following list details the main objectives of the project:

- Construct a line-following vehicle using IR sensors.
- Calculate distance travelled using a rotary encoder.
- Calculate ramp incline using the MPU 6050 Accelerometer and Gyroscope Sensor.
- Obstacle avoidance using the HC-SR04 Ultrasonic Sensor.
- Bluetooth control using the HC-05 Bluetooth Serial Transceiver.

In addition to these main objectives, the vehicle must complete certain tasks as detailed below:

- Week 2 Objectives:
  - Build an IR sensor on a Veroboard.
  - Use the IR sensor to follow the line.
  - Complete the track within 60 seconds.
  - Display the distance travelled on the LCD.
- Week 3 Objectives
  - Task 1
    - Ascend the ramp and calculate the angle of the incline.
    - Stop for 4 seconds on the top of the ramp.
    - Rotate for 360°.
    - Descend the ramp and continue on to task 2.
  - Task 2
    - Follow the line for 80cm.
    - Stop for 3 seconds.
    - Continue to follow the line until the end of the track.
    - Display the distance and time elapsed.
  - Implement the HC-SR04 Ultrasonic Sensor for obstacle avoidance.
  - Implement the HC-05 Bluetooth Serial Transceiver for Bluetooth control.
  - Come up with and implement 2 additional advanced features.

The 2 additional advanced features that was decided:

- Adding a buzzer to the vehicle.
- Implementing PID control using 2 rotary encoders.

2 Hardware Design

This section details all of the hardware used in the project and the decisions made behind each aspect of the hardware design.

2.1 Required Hardware

All of the hardware that is used in this project is listed in the Table I.

TABLE I
LIST OF HARDWARE

Component	Count
Arduino UNO R3	1
16x2 LCD Display	1
LCD Keypad Shield	1
IR Sensor	2
Optical Rotary Encoder Module	2
MPU6050 Gyroscope Accelerometer	1
HC-SR04 Ultrasonic Sensor	1
HC-05 Bluetooth Serial Transceiver	1
L298N Motor Driver	1
DC Motor	4
Mini Breadboard	1
3.7V Lithium Ion Battery	2
Battery Holder	1
Acrylic Chassis	1

2.2 Hardware Block Diagram

Figure 1 presents a visual representation of the system used in the line-following vehicle.

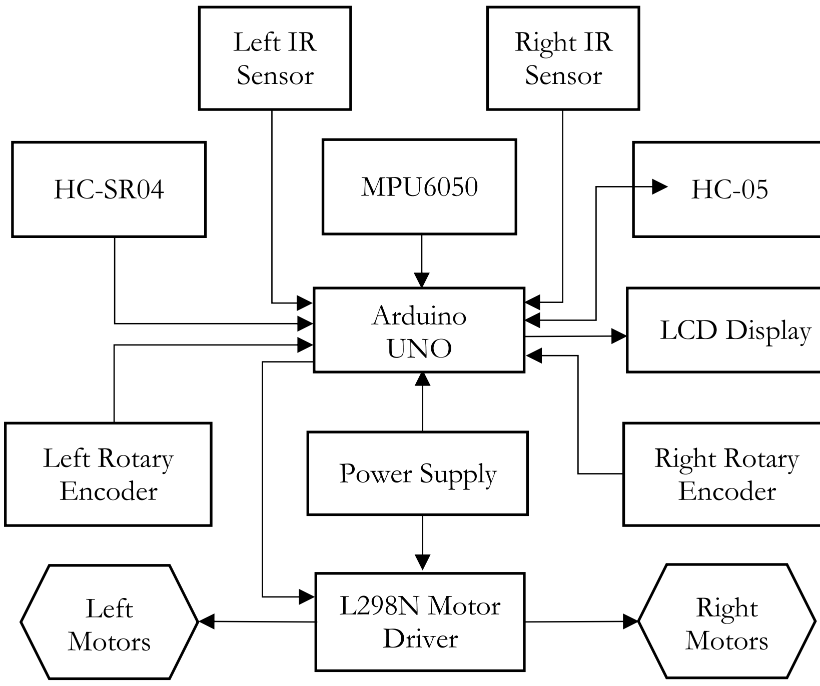


Fig. 1. Line-following Vehicle Block Diagram.

The physical placements of all the components changed throughout the project as more and more features were implemented. The changes can be found in the project logbook attached in Appendix A.

2.3 IR Sensor Design

The IR Sensor is built using an infrared LED and an infrared photodiode. At the centre of the IR Sensor is a LM358 dual operational amplifier integrated circuit, with the first Op-Amp being used as an inverting transimpedance amplifier to amplify the signal from the photodiode with an amplification of 1MΩ, and the second one used as a non-inverting comparator to produce a digital signal. The inverting terminal of the comparator is connected to a potentiometer to adjust the voltage threshold for the digital output. The output is then connected to an LED as an indicator of the IR sensor's signal. Figure 2 shows the schematic of the IR sensor in detail.

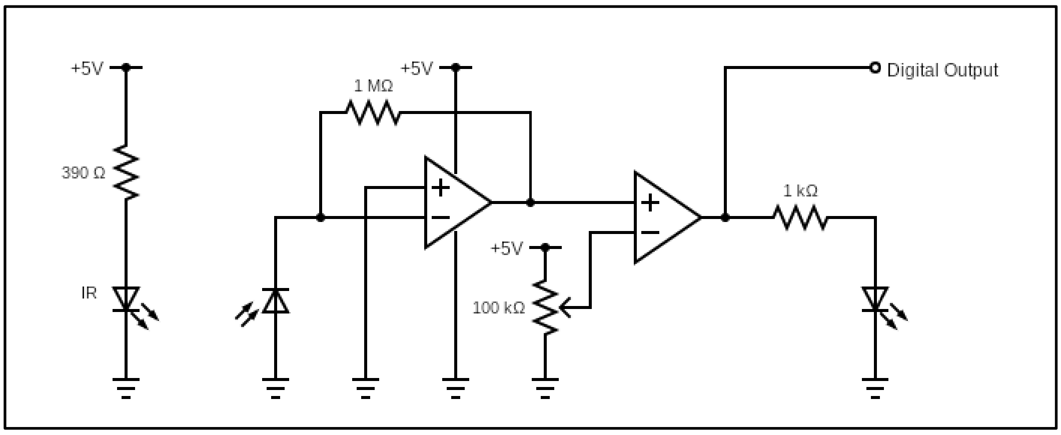


Fig. 2. IR sensor circuit schematic.

The IR sensor is soldered onto a Veroboard according to the schematic. To minimise the footprint of the IR sensor, all of the ground terminals were connected directly to a single rail at the bottom eliminating the use of extra wires. This also allows the IR LED and IR photodiode to protrude out of the bottom to improve its sensing performance. The resistors are soldered vertically instead of horizontally so that the body of it will only take up the space of 1 hole. A sketch of the Veroboard planning can be found in Appendix B.

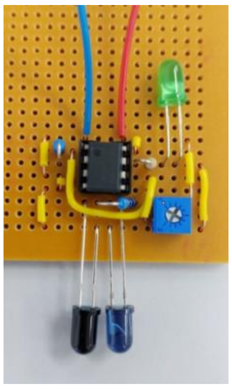


Fig. 3. The completed IR sensor.

2.4 Arduino Interface

The Arduino UNO has 14 digital pins and 6 analogue pins. Of the 14 digital pins, pins 0 and 1 are used for USART communication making it not advisable to use [4]. This leaves a total of 12 usable digital pins and 6 analogue pins for a total of 18 pins.

The LCD is connected to the Arduino via the keypad shield and uses pins 4, 5, 6, 7, 8, and 9. Since the keypad shield only uses pin 10 for the non-essential backlight control on the LCD [5], the entire keypad shield is detached from the Arduino and mounted on the back of the vehicle to free up pin 10. This also frees up pin A0 which is otherwise used for the buttons on the keypad shield. Pins 10 and 11 has PWM capabilities and were connected to the ENA and ENB pins on the L298N Motor Driver to control the speed of the motors. The remaining pins 13, 12, 3, and 2, were connected to the IN1, IN2, IN3, and IN4 pins on the L298N Motor Driver respectively to control the direction of the motors.

The pair of IR sensors were connected to the analogue pins A0 and A1. Pins A2 and A3 are then used for the 2 rotary encoders, leaving pins A4 and A5 for the MPU6050 I^2C communication, as these 2 pins are also the I^2C SCL and SDA lines on the Arduino.

The Arduino UNO did not have enough pins to implement everything concurrently, therefore some features were implemented as a standalone. The TX and RX pins on the HC-05 Bluetooth Serial Transceiver was connected to the digital pins 0 and 1, as it communicates via the USART protocol [6]. It is to be noted that connecting the HC-05 module will interfere with the communication between the Arduino and the computer, therefore it has to be disconnected before uploading any code. The echo pin on the HC-SR04 Ultrasonic Sensor is connected to pin A2 during testing. Finally, the buzzer utilised one of the digital pins to produce a square wave, again as a standalone feature.

3 Software Design

Software is an integral part of any automated system. It is what makes the vehicle a 'smart' autonomous vehicle. The code for the vehicle's Arduino UNO R3 is written in C++. All of the code developed for this project can be found in a GitHub repository attached in Appendix A.

3.1 L298N Motor Driver Library

A key element in making the line-following vehicle work is being able to control it. To make developing the algorithm for making the vehicle autonomous easier, a library was created to control the L298N Motor Driver with all of the key movement functions.

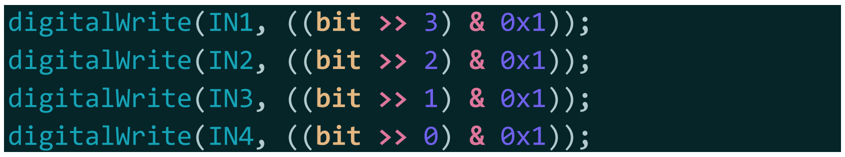
Four bits of information is needed to control the direction of the L298N Motor Driver along with 2 PWM values for controlling the speed. The `send_to_driver()` function was created to handle sending data to the 6 pins. The function has 3 parameters. The latter 2 parameters accept PWM values as arguments to control the speed of the left and right motors. As the data sent to the remaining 4 pins of the motor driver can be expressed in a 4-bit binary number, Binary-coded Hexadecimal (BCH) is used to convert a hexadecimal argument from the first parameter into a 4-bit binary number.

TABLE II
MOTOR DRIVER TRUTH TABLE

IN1	IN2	IN3	IN4	HEX	Effect
0	0	0	0	0x0	Both motors stop
0	1	0	1	0x5	Both motors backwards
0	1	1	0	0x6	Left forward, right backward
1	0	0	1	0x9	Right forward, left backward
1	0	1	0	0xA	Both motors forward

* Only essential logic is shown in this table as having either IN1 and IN2, or IN3 and IN4 set to high at the same time will produce a short.
* To have only 1 side of the motors turned on, a PWM value of 0 can simply be used, reducing the total amount of combinations.

Bit-shifting and bit-masking is then used to read the code to be sent to each individual pin. Creating this function allows for code deduplication and easier manipulation of the motor driver.



```
digitalWrite(IN1, ((bit >> 3) & 0x1));
digitalWrite(IN2, ((bit >> 2) & 0x1));
digitalWrite(IN3, ((bit >> 1) & 0x1));
digitalWrite(IN4, ((bit >> 0) & 0x1));
```

Fig. 4. Using bit-shifting and bit-masking.

Based on the `send_to_driver()` function, controls for the vehicle's movement can be easily coded. In each movement function, a delay of time t is added to adjust the amount of time the specific movement command will last. For the `turn()` function, the capability of accepting negative PWM values to change the direction of the motors individually was added so that the vehicle can have a wider range of rotational velocity.

3.2 Line-following Component

Using the motor driver library detailed in Section 3.1, the line-following algorithm can be coded easily. The logic behind the algorithm is whenever one of the IR sensors on either side of the vehicle senses the black line, it means that the line is tending towards that direction. The vehicle will then self-correct to turn towards that direction.

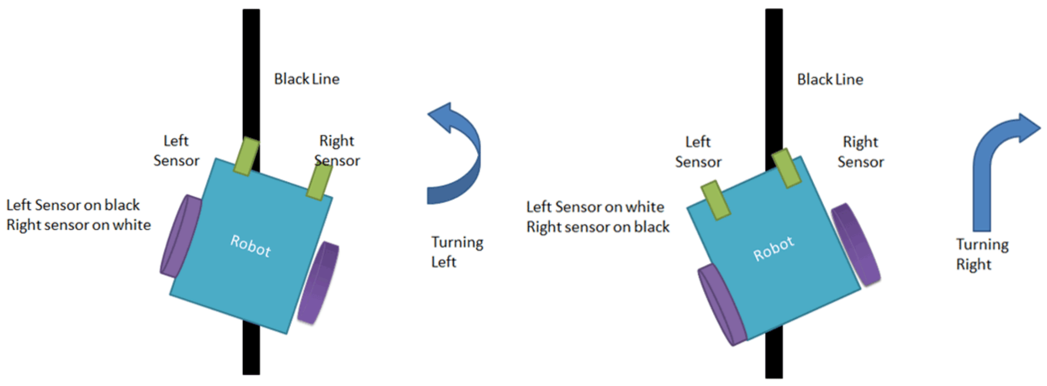
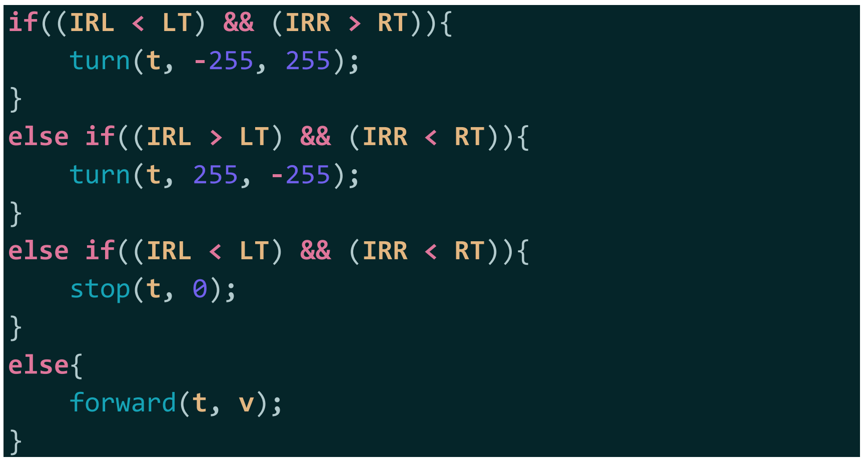


Fig. 5. Visual representation of the line-following algorithm.
Source: From [7].

Since analogue pins are used for the IR sensor, a threshold has to be set to make the signal digital. Whenever the IR sensor detects the black line, the signal will drop below the threshold and produce a digital LOW. Based on this, a set of instructions can be coded to continually check the output signal of the IR sensor to make decisions on the vehicle's movement.



```
if((IRL < LT) && (IRR > RT)){
    turn(t, -255, 255);
}
else if((IRL > LT) && (IRR < RT)){
    turn(t, 255, -255);
}
else if((IRL < LT) && (IRR < RT)){
    stop(t, 0);
}
else{
    forward(t, v);
}
```

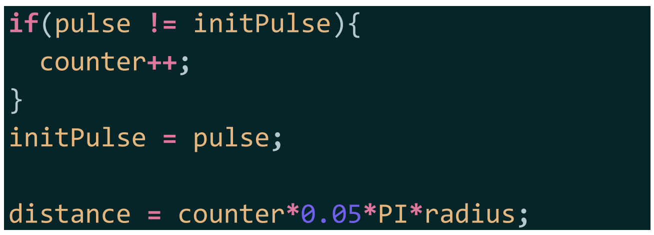
Fig. 6. Line-following algorithm.

The algorithm was encapsulated in a function so that it is more modular and can be incorporated wherever necessary.

3.3 Distance Calculation

The rotary encoder is used to measure rotation. The particular rotary encoder used in this project is an optical incremental encoder. To measure rotation, the number of pulses of the incremental encoder must be counted [8]. The disc used to produce the pulses for this rotary encoder contains 20 holes meaning that there will be 20 pulses per rotation of the wheel. With this information, we can simply calculate the distance travelled by multiplying the number of pulses with the circumference of the wheel.

The number of pulses is stored in the `counter` variable. It is incremented every time the signal from the rotary encoder changes. With 20 pulses per rotation, there will be a total of 40 signal changes as the signal rises and falls every pulse. Thus, the circumference of the wheel is divided by 40 to get the arc length in between each signal change.



```
if(pulse != initPulse){
  counter++;
}
initPulse = pulse;

distance = counter*0.05*PI*radius;
```

Fig. 7. Distance calculation using the number of pulses.

3.4 Ramp Angle Calculation

The ramp angle is measured using the MPU6050's accelerometer. The MPU6050 is placed flat with the x-axis pointing forward so that the pitch of the vehicle can be measured by reading acceleration along the x-axis. The angle of the ramp can be calculated using the following equation:

$$\theta = sin^{-1}\left(\frac{a_x}{g}\right) \tag{1}$$

where θ is the ramp angle, a_x is the acceleration along the x-axis of the sensor, and g is the acceleration due to gravity.

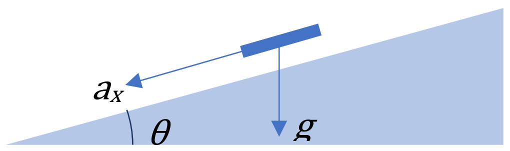
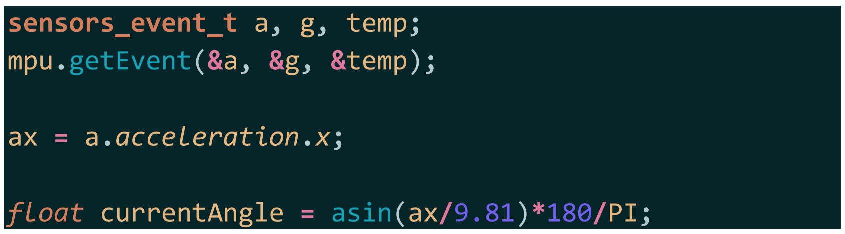


Fig. 8. Diagram showing the calculation of the ramp angle using the a_x reading on the MPU6050.



```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

ax = a.acceleration.x;

float currentAngle = asin(ax/9.81)*180/PI;
```

Fig. 9. Reading and calculating the ramp angle.

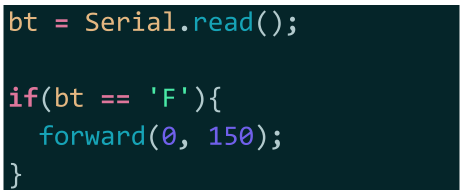
3.5 Additional Features

As part of expanding the capabilities of the vehicle beyond simply following a line, several additional features were added, including Bluetooth control and obstacle avoidance features. Their implementation is detailed in this section.

3.5.1 Bluetooth Control

The HC-05 Bluetooth Serial Transceiver receives and transmits serial data to the Arduino via the USART pins 0 and 1 [6]. From here, different data can be sent to the Arduino to run certain actions. Using a Bluetooth remote control app on a smartphone, the vehicle is able to be controlled remotely.

The app used to control the car in this project is named "Arduino Car" developed by ONE DAY OF CODE and can be found on the Google Play Store [9]. Pressing any button in this app will send a character to the connected HC-05 module. Using the `Serial.read()` function, this character can be stored into a variable. A series of if else statements determine the character and executes the corresponding movement command.

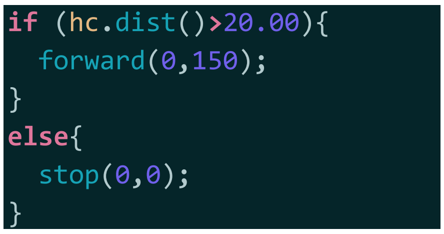


```
bt = Serial.read();

if(bt == 'F'){
  forward(0, 150);
}
```

Fig. 10. Short snippet of the forward command.

3.5.2 Obstacle Avoidance

The HC-SR04 sensor uses ultrasound to detect obstacles placed in front of it. Using the time taken for the transmitted ultrasound to be reflected back to the receiver, and the speed of ultrasound in air, it is also able to determine the distance of the object from the sensor [10]. An if statement is used to allow the car to move forward only if there are no obstacles detected within a specified distance of the sensor.



```
if (hc.dist()>20.00){
  forward(0,150);
}
else{
  stop(0,0);
}
```

Fig. 11. Condition to stop if there is an obstacle within 20cm of the sensor.

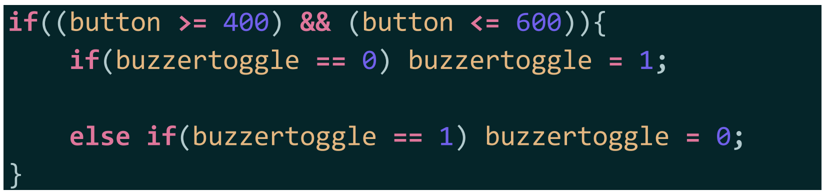
3.6 Advanced Features

Two additional advanced features were added to the vehicle to add depth to the line-following vehicle. The first is the addition of a buzzer, and the second enables the vehicle with PID control.

3.6.1 Buzzer Implementation

The piezoelectric buzzer uses the piezoelectric effect to generate sound. This can be easily done by applying a periodic voltage signal across the buzzer to generate tones of specific frequencies [11]. Expanding upon this, simple music can be played using the buzzer. GitHub provides a good repository for preprogrammed songs that can be uploaded to the Arduino easily [12].

To control when the buzzer is activated, one of the buttons on the Arduino keypad shield was utilised to trigger the buzzer. Using the A0 pin as an input, pressing specific buttons will produce a corresponding signal within a certain voltage range [13]. This can be used to activate or deactivate the portion of code for the buzzer so that it will only play after the button is pressed .



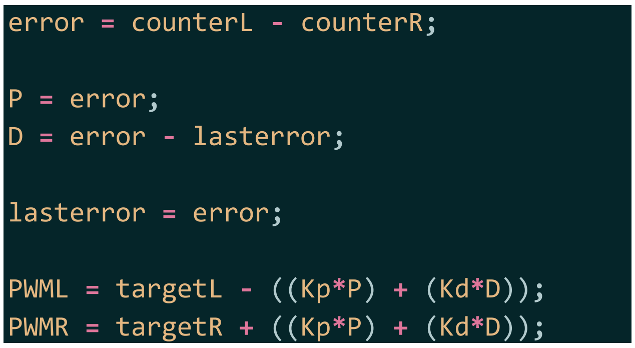
```
if((button >= 400) && (button <= 600)){
    if(buzzertoggle == 0) buzzertoggle = 1;

    else if(buzzertoggle == 1) buzzertoggle = 0;
}
```

Fig. 12. Code to toggle the buzzer when button is pressed.

3.6.2 PID Implementation

As the motors and wheels on both sides of the vehicle are not exactly the same, variations in the speed they spin means that the vehicle will inevitably tend to one direction or the other instead of going perfectly straight. This is especially noticeable when the vehicle travels for long stretches. To counteract this issue, 2 rotary encoders are used to measure the distance traversed by both sides of the vehicle, and a PID control is implemented to correct the error, ensuring that the vehicle travels in a straight line.

For this vehicle, only the proportional and derivative controllers are used, as the integral controller is not necessary in this simple system. The error value used is the difference between the counter value of the rotary encoder for the left side and the right side. Whenever the left side moves faster than the right, the error will be positive, and vice versa. For calculating the derivative of the error, the previous error is subtracted from the current error. Each controller error is then multiplied by a corresponding scaling factor to obtain a correction value. This scaling factor is to be tuned during testing through trial and error to find the optimum value. The sum of the correction values will give the PID control variable. This control variable can be applied as an increase or decrease in the PWM values of the motors [14], [15].



```
error = counterL - counterR;

P = error;
D = error - lasterror;

lasterror = error;

PWML = targetL - ((Kp*P) + (Kd*D));
PWMR = targetR + ((Kp*P) + (Kd*D));
```

Fig. 13. PID calculation for correcting motor speed.

To make the feature demonstrable, another button on the LCD keypad shield was used to make the feature toggleable. This makes showcasing the difference between having the PID control versus normal forward movement much easier without having the upload the code repeatedly.

4 Results and Discussion

The line-following vehicle was able to successfully complete all of the objectives and tasks listed in Section 1.2. This section discusses the results obtained in this project.

4.1 Following a Line

The IR sensor that was soldered initially was not very sensitive despite being functional. With a specified reverse light current of around 30μA - 60μA [16], an amplification of 1MΩ should give an expected output voltage of 30V - 60V capped at around 5V by the voltage supply to the LM358 Op-Amp. However, the measured voltage output was only at around 1.7V leading to a rather weak signal. After some troubleshooting, it was concluded that the LM358 Op-Amp used was faulty and not amplifying well. Changing the component solved the issue, and the output signal became much stronger as evidenced by the brighter LED.

In the initial test of the line-following algorithm, the vehicle barely managed to follow the line. There were 2 issues here. The first is that the spacing of the two IR sensor were too close, causing it to not have enough clearance to make sharp turns. Both the IR sensors would end up on the black line making the vehicle stop. Increasing the spacing solved this problem. The second problem is the reaction time of the vehicle. It was initially thought that a delay is needed in between each reading of the IR sensors to properly function. This wasn't the case as it slowed down the reaction time of the vehicle causing it to miss a turn in the line occasionally. Removing the delay solved this issue.

Lastly, the speed of the vehicle while following the line had to be tuned through trial and error. The goal is to complete the track as fast as possible, but too fast and the vehicle will overshoot the line. The ideal PWM value found for this particular vehicle was around 60, which is roughly a 23% duty cycle.

4.2 Calculating Distance and Time

The length of the entire track calculated by the rotary encoder was quite consistent at around 9 meters. It is to be noted that this measurement is the linear distance rotated by the wheel on one side of the vehicle and does not reflect the length of the track as if measured using a ruler. The time it took to complete the track was at around 43 seconds.

4.3 Calculating Ramp Angle

The initial measurements of the ramp angle were very inconsistent, occasionally giving negative values even though the accelerometer reading should give a positive value in its orientation. The reason was because the vehicle was still in motion on the ramp when the measurement was taken, which caused inaccurate readings. To solve this, a delay was added to make sure that the vehicle is stationary on the ramp before the reading is taken.

The incline of the ramp was measured to be around 15° - 20° which falls within the range of the value that was measured manually through trigonometry which works out to be roughly 18°.

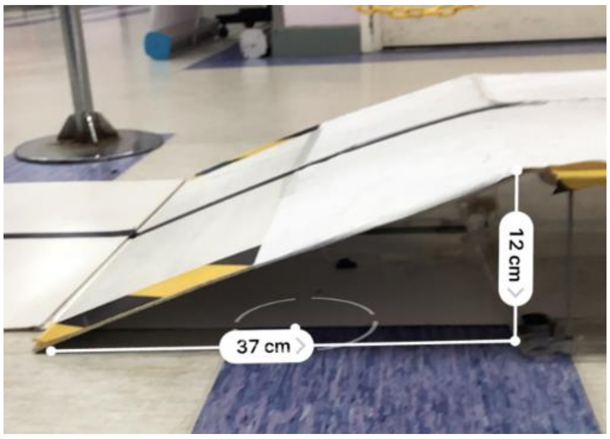


Fig. 14. Manual ramp angle calculation.

4.4 Obstacle Avoidance and Bluetooth Control

The obstacle avoidance of the vehicle worked, but the momentum of the vehicle itself would still cause it to hit the obstacle. The threshold distance for stopping was increased to counteract this. The Bluetooth control worked well to control the vehicle without any issues.

4.5 Buzzer

The buzzer worked well without any issues and was able to play music. The button integration made it easier to turn the buzzer on or off without having to reupload the code.

4.6 PID Control

The PID control was tuned using the method as described in [17]. The K_p value was tuned to 0.4, while the K_d value was tuned to 0.2. This produced a stable control loop that kept the car moving in a straight line.

5 Conclusion

The line-following vehicle constructed in this project achieved all of the objectives and goals set out in Section 1.2. It is able to successfully follow a line fully autonomously. Aside from that, the vehicle was also built with features such as distance measurement, ramp angle measurement, Bluetooth control, obstacle avoidance, buzzer implementation, and PID control. The integration of all these features expands the capabilities and complexity of the vehicle beyond just following a line.

This project has provided beneficial insights into the systems governing an autonomous vehicle. As an educational endeavour, it has also provided an ideal platform to learn essential engineering skills and explore the complexities of embedded systems, programming, sensor integration, automation algorithms, and project management, all of which are crucial knowledge in engineering.

5.1 Limitations of the Line-following Vehicle

The line-following vehicle constructed in this project does not come without limitations. The initial idea for Task 1 in the Week 3 objective was to have the MPU6050 accelerometer detect the start and end of the ramp, as well as measuring the 360° rotation using the gyroscope. However, the MPU6050 kept causing the Arduino to freeze whenever it was read continuously. After exhausting all possible solutions, a compromise was made to read the sensor only once for the angle calculation. To ascend the ramp, a timing delay was used which meant that it was hard-coded and not autonomous. The 360° rotation was also done using timing delays which is not an ideal way to do it as the rotation is inconsistent.

The obstacle avoidance algorithm used was quite basic and only stop when encountering an obstacle instead of searching for a path around the obstacle. As for the Bluetooth control, its movement control is limited to only 4 commands: forward, backward, turn left, and turn right. Turning cannot happen in tandem with moving forward or backward. The number of pins on the Arduino UNO also did not allow for the full integration of all the features together. Instead, each of the additional features were made as a separate standalone feature.

5.2 Future Work

While the line-following vehicle was succeeded in completing all the objectives, many improvements to the line-following vehicle could be made to address the limitations detailed in Section 5.1. Primarily, a more powerful microcontroller board such as the Arduino MEGA can be used as it has 54 digital pins and 16 analogue pins [18], enough to integrate all the features concurrently. A more robust obstacle avoidance algorithm can be designed so that the vehicle will path-find its way around an obstacle instead of just stopping. The Bluetooth control can also be coded to allow a wider range of motion.

No solution has been found to solve the freezing issue between the MPU6050 and the Arduino UNO. Perhaps using an Arduino MEGA could solve this problem, but more testing and experimentation would be needed.

References

[1] ‘Automated Vehicle Safety | NHTSA’. Accessed: Dec. 25, 2023. [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[2] ‘Automated Guided Vehicles (AGV) Meaning & Types | 6 River Systems’. Accessed: Dec. 25, 2023. [Online]. Available: https://6river.com/what-are-automated-guided-vehicles/

[3] P. Marmaglio, D. Consolati, C. Amici, and M. Tiboni, ‘Autonomous Vehicles for Healthcare Applications: A Review on Mobile Robotic Systems and Drones in Hospital and Clinical Environments’, *Electronics 2023, Vol. 12, Page 4791*, vol. 12, no. 23, p. 4791, Nov. 2023, doi: 10.3390/ELECTRONICS12234791.

[4] ‘UNO R3 | Arduino Documentation’. Accessed: Dec. 26, 2023. [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3

[5] ‘LCD_KeyPad_Shield_For_Arduino_SKU__DFR0009-DFRobot’. Accessed: Dec. 26, 2023. [Online]. Available: https://wiki.dfrobot.com/LCD_KeyPad_Shield_For_Arduino_SKU__DFR0009

[6] ‘Arduino and HC-05 Bluetooth Module Complete Tutorial’. Accessed: Dec. 26, 2023. [Online]. Available: https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/

[7] ‘Arduino Line Follower Robot Code and Circuit Diagram’. Accessed: Dec. 27, 2023. [Online]. Available: https://circuitdigest.com/microcontroller-projects/line-follower-robot-using-arduino

[8] M. Murray, ‘How Rotary Encoders Work - Electronics Basics - The Geek Pub’. Accessed: Nov. 04, 2023. [Online]. Available: https://www.thegeekpub.com/245407/how-rotary-encoders-work-electronics-basics/

[9] ‘Arduino Car – Apps on Google Play’. Accessed: Dec. 27, 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.electro_tex.bluetoothcar&hl=en-MY

[10] ‘How HC-SR04 Ultrasonic Sensor Works & How to Interface It With Arduino’. Accessed: Nov. 04, 2023. [Online]. Available: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/

[11] 'How Do Piezo Buzzers Work | RS'. Accessed: Dec. 27, 2023. [Online]. Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/how-do-piezo-buzzers-work

[12] 'GitHub - robsoncouto/arduino-songs'. Accessed: Dec. 27, 2023. [Online]. Available: https://github.com/robsoncouto/arduino-songs

[13] 'LCD & Keypad Shield Quickstart Guide | Freetronics'. Accessed: Dec. 28, 2023. [Online]. Available: https://www.freetronics.com.au/pages/16x2-lcd-shield-quickstart-guide

[14] M. O'Hanlon, 'Going straight with PID'.

[15] R. Toulson and T. Wilmshurst, 'An Introduction to Control Systems', *Fast and Effective Embedded Systems Design*, pp. 273–295, Jan. 2012, doi: 10.1016/B978-0-08-097768-3.00013-1.

[16] Vishay Semiconductors, 'BPV10NF Silicon PIN Photodiode'. [Online]. Available: https://www.vishay.com/doc?81503

[17] 'How to Tune a PID Controller • PID Explained'. Accessed: Dec. 28, 2023. [Online]. Available: https://pidexplained.com/how-to-tune-a-pid-controller/

[18] 'Mega 2560 Rev3 | Arduino Documentation'. Accessed: Dec. 28, 2023. [Online]. Available: https://docs.arduino.cc/hardware/mega-2560

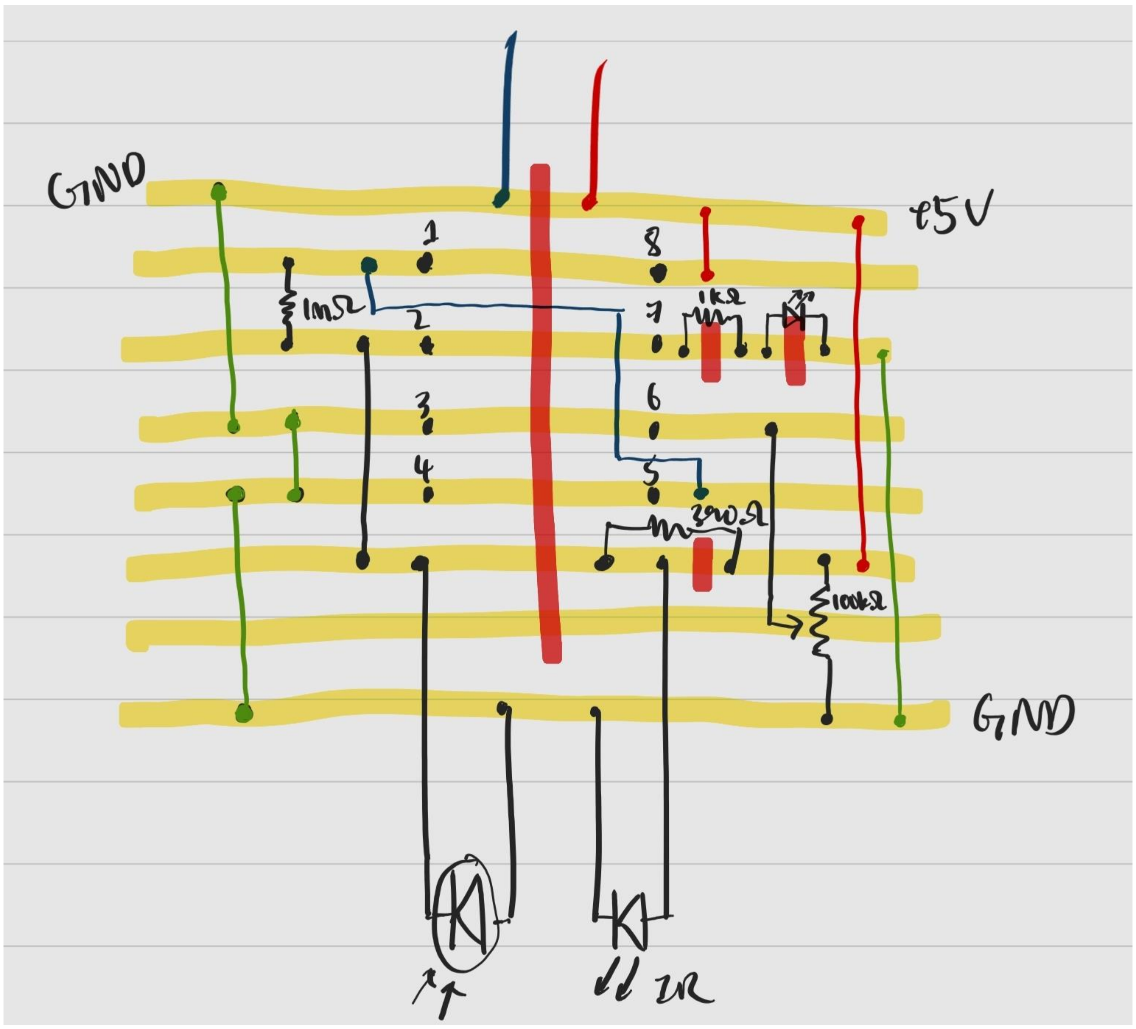
Appendix A

All of the code produced for this project can be found in the GitHub repository linked below. Along with the code, there is also the project logbook and a copy of Report 1 detailing the sensors used in this project.

EEEE1027 Autumn Construction Project GitHub Repository

Appendix B

A sketch of the IR sensor to plan out the positions of the components on the Veroboard.


GND
+5V
1
2
3
4
5
6
7
8
1MΩ
1kΩ
330Ω
100kΩ
GND
IR