The University of Nottingham

Department of Electrical and Electronic Engineering

EEEE1027

Applied Electrical and Electronic Engineering Construction Project

# Autumn Project Logbook

Max Koo Yan Jie

20620542

Group 8

November/December 2023

## Project Details

| | |
|---|---|
| **Project Name:** | Line Following Vehicle |
| **Project Description:** | Build a line following vehicle that can complete a set of tasks as outlined in the objectives for each project week. The vehicle is assessed at the end of each week in a final track challenge. |

**Group Members:**

| | |
|---|---|
| **Team Leader:** | Max Koo Yan Jie |
| **Project Manager:** | Marcus Yew Min Jie |

# Table of Contents

# Week 2

**Description:** Build a line following vehicle using 2 IR sensors, and display the distance travelled on the LCD by using a rotary encoder.

**Objectives:**

1. Build an IR sensor on a Veroboard.
2. Use the IR sensor to follow the line.
3. Complete the track within 60 seconds.
4. Display distance travelled on the LCD

**Date:** 20/11/2023

**Time:** 0900H – 1700H

**Objective:** Resolder IR Sensor.

---

**Activity Description:**

The main objective for today was to resolder the IR sensor using new components, as the IR Sensor previously soldered in Project Week 1 was faulty and did not work. Following the same design, a new IR sensor is soldered.

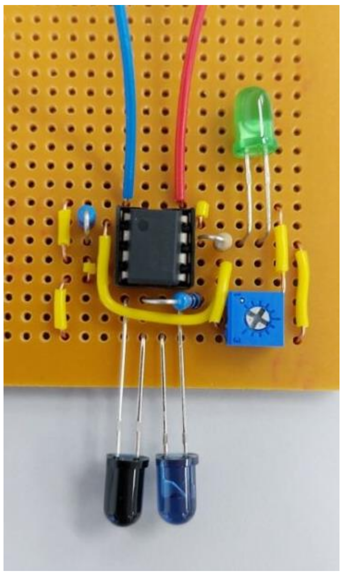


Fig. 1. Resoldered IR Sensor.

**Observations and Results:**

The new IR sensor is now functional. However, the output signal is quite weak despite an amplification of 1MΩ. Several tests were conducted to measure the output voltage and it was found that in a HIGH state, the output was only about 1.7V. Thus, it was concluded that the LM358 Op-Amp IC was not amplifying well. After changing the IC, the output signal became much stronger, and the LED lit up much brighter.

**Summary and Conclusion:**

The cause of the previous IR sensor not working is suspected to be a soldering issue, as the new sensor works despite using the exact same design. Much more practice and care are needed in future soldering work to ensure that all solder connections are solid.

**Date:** 21/11/2023

**Time:** 0900H – 1700H

**Objective:** Interface IR Sensor with Arduino for line following.

---

**Activity Description:**

With both IR sensors functional, it was attached to the front of the vehicle and connected to the Arduino via the analogue pins. The motor driver's ENA and ENB pins to control the speed of the motor was also connected to the available digital PWM pins on the Arduino, which are pins 3 and 11. Both IR sensors are powered up through a small breadboard that is connected to the 5V output of the motor driver.

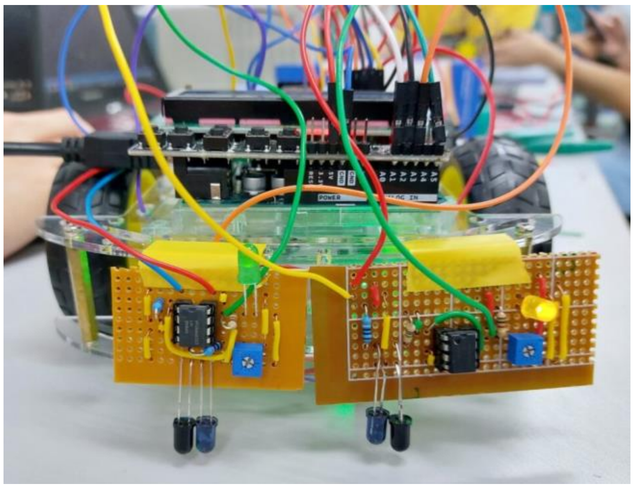
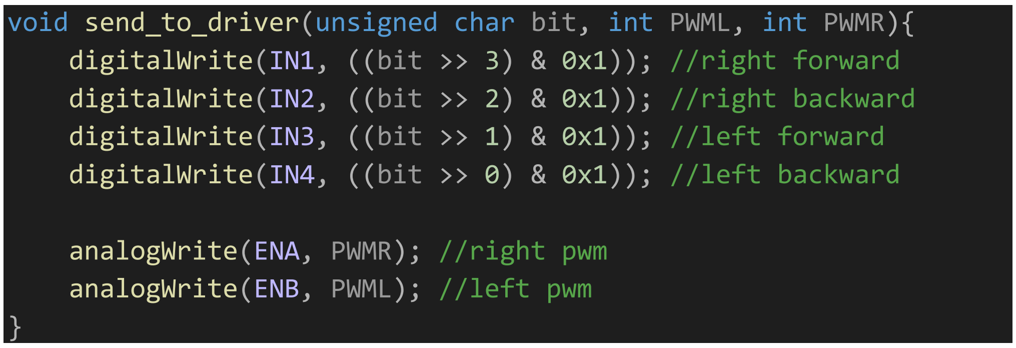


Fig. 2 Attached IR sensors on the vehicle.

For the code to control the motor driver, a `send_to_driver` function is created to accept a 4-bit HEX number as an argument. This HEX number is then converted to binary via bit-shifting and bit-masking and passed to the Arduino to control the 4 motor driver pins. It also accepts 2 PWM values sent to the ENA and ENB pins to control the speed of the motors. Creating this function allows for code deduplication and easier control over the motors.

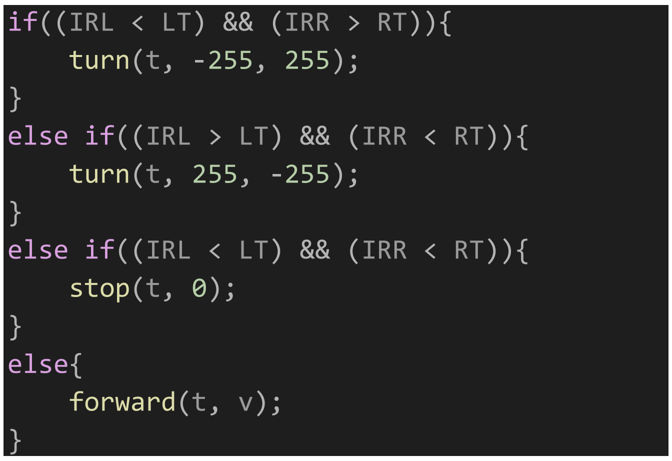


```
void send_to_driver(unsigned char bit, int PWML, int PWMR){
    digitalWrite(IN1, ((bit >> 3) & 0x1)); //right forward
    digitalWrite(IN2, ((bit >> 2) & 0x1)); //right backward
    digitalWrite(IN3, ((bit >> 1) & 0x1)); //left forward
    digitalWrite(IN4, ((bit >> 0) & 0x1)); //left backward

    analogWrite(ENA, PWMR); //right pwm
    analogWrite(ENB, PWML); //left pwm
}
```

Fig. 3. The `send_to_driver` function which accepts hexadecimal as a parameter.

Using the `send_to_driver` function, 4 movement control functions are created to move the car forwards and backwards, stopping the car, and turning the car. The `turn` function is set up to accept negative PWM values which allows the wheels on both sides of the car to move in opposite directions.



```
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. 4. Line following Algorithm.

Finally, these functions are called in the `loop()` function to create a set of movement instructions based on the reading of the 2 IR sensors.

**Observations and Results:**

While testing the code, it was noted that the IN4 pin was at a constant logic HIGH. After some testing with a multi-meter, the problem was found to originate from the Arduino pin 1, which is the TX pin. Changing it to the A5 pin solved the issue. The vehicle was then tested on the track. It managed to follow the straight section of the line but struggled with corners.

**Summary and Conclusion:**

Creating a library (attached in Appendix B) for controlling the motors allows a much simpler programming of the car and reduces the potential for bugs. More tuning is needed to ensure that the car can reliably follow the line.

**Date:** 22/11/2023

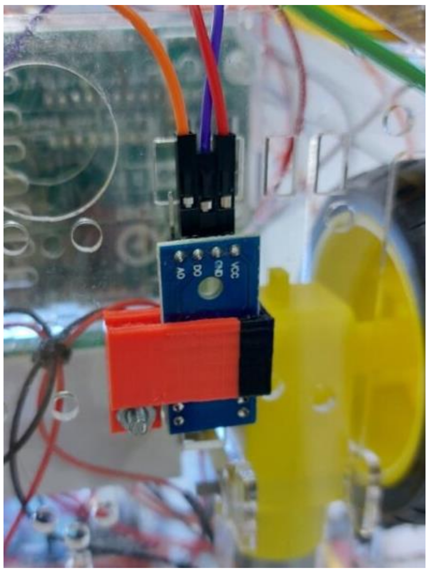
**Time:** 0900H – 1300H

**Objective:** Implementing the rotary encoder and tuning the line following component.

---

**Activity Description:**

A rotary encoder was attached to the wheel today. It was then coded to count the number of pulses which is multiplied by the arc length traversed with each pulse to get the total distance travelled by the car. The car was then tested on the track to test both the line following and distance components.



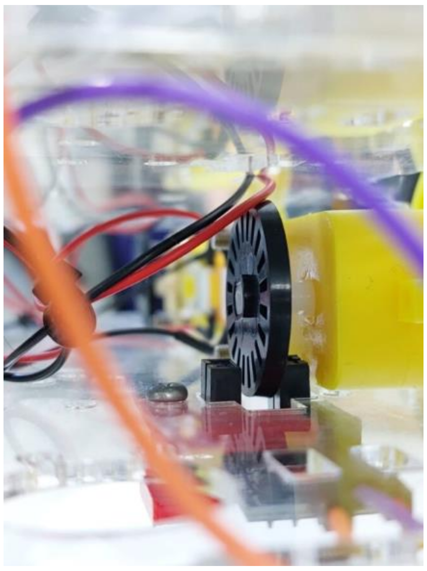


Fig. 5. Rotary encoder placement (left).

Fig. 6. Rotary encoder disc in between the optical sensors (right).

**Observations and Results:**

The first issue faced was that the vehicle could not turn at sharp corners and would instead just stop. The spacing of the 2 IR sensors did not give enough clearance between the lines, causing both sensors to be on the black line at the corner and stopping the vehicle. After increasing the spacing of the sensors, the vehicle can now turn at sharp corners.

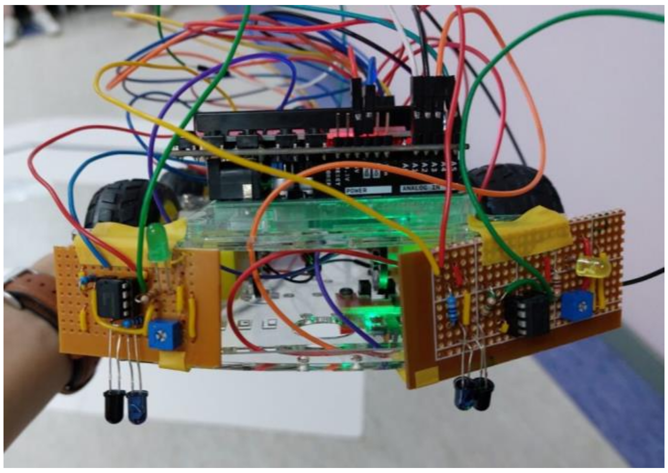


Fig. 7. Increasing the IR sensor spacing.

While tuning the speed of the motors, the vehicle was not responsive enough which caused it to go out of the line. It was eventually realised that a delay between subsequent measurements is not needed, as that causes the vehicle to continue its current course of action despite there being a change in the IR sensor output. After removing the delay, the vehicle became very responsive and would change directions immediately, making the line following more continuous.

**Summary and Conclusion:**

After some speed adjustments, the vehicle can now reliably follow the line and print the correct distance. The vehicle was then successfully assessed based on the objectives for the week.

## Week 3

**Description:** Adding advanced features to the line following vehicle.

**Objectives:**

1. Calculate ramp angle using the MPU 6050 Gyro Accelerometer sensor
2. Avoid obstacles using the HC-SR04 ultrasonic sensor
3. Bluetooth control using the HC-05 Bluetooth Serial Transceiver
4. Display the ramp angle, distance travelled, and the time elapsed on the LCD

### Task 1

Ascend the ramp and calculate the angle of the incline. Once on the ramp, stop for 4 seconds, then rotate for 360°. Descend the ramp and continue on to task 2

### Task 2

Follow the line for 80cm, and stop for 3 seconds. Then continue to follow the line until the end of the track. Display the distance and time elapsed.

**Date:** 23/11/2023

**Time:** 0900H – 1700H

**Objective:** Week 3 planning and using the MPU6050 to measure ramp angle.

---

**Activity Description:**

Week 2 objectives were finished ahead of schedule, so we decided to start planning out Week 3's schedule today. After some discussion, we began by investigating task 1 which required the MPU6050 to measure the ramp angle.

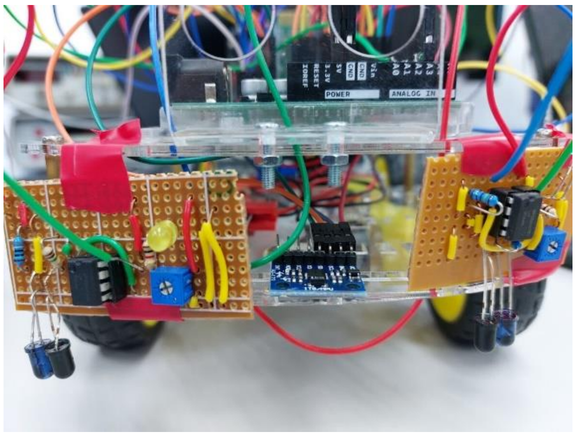


Fig. 8. Placement of the MPU6050.

Using the MPU6050 library, values for all 3 axes of the accelerometer and gyroscope can be read. Physical orientation of the sensor on the vehicle is very important as it determines the directions of all 3 axes. The angle of the ramp can be calculated using trigonometry:

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

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.

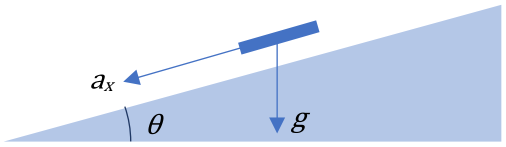


Fig. 9. Diagram showing the calculation of the ramp angle using the $a_x$ reading on the MPU6050.

**Observations and Results:**

The angle prints with fairly accurate values and the incline of the ramp was measured to be roughly 20°. To confirm the results, a protractor was used to measure the angle of the ramp, and the angle was also obtained to be around 20°.

**Summary and Conclusion:**

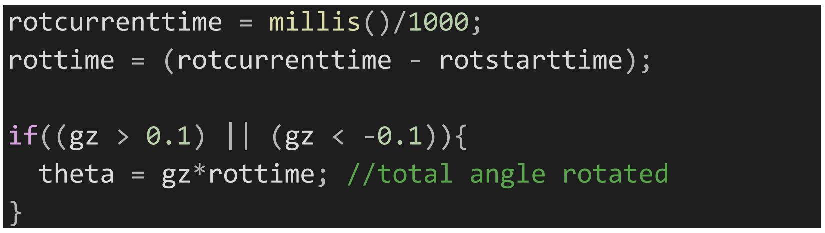
The MPU6050 sensor was tested and integrated into the car. With the ramp angle calculation implemented, the next steps are to code the sequence to complete task 1 and task 2.

|  |  |
|---|---|
| **Date:** | 24/11/2023 |
| **Time:** | 0900H – 1700H |
| **Objective:** | Measuring rotation using the gyroscope. |

---

**Activity Description:**

The rotating portion of the ramp sequence was coded today by using the reading of the gyroscope. As the reading of the gyroscope only gives the angular velocity and not the actual heading of the vehicle, the angle rotated by the vehicle needs to be calculated by taking the integral of angular velocity with respect to time. This can be done through code by multiplying the angular velocity with the time elapsed and summing all the values together. The rotation of the vehicle can then be limited to a certain angle to ensure that the vehicle only rotates 360° before going back down the ramp.



```
rotcurrenttime = millis()/1000;
rottime = (rotcurrenttime - rotstarttime);

if((gz > 0.1) || (gz < -0.1)){
  theta = gz*rottime; //total angle rotated
}
```

Fig. 10. Calculation of the total angle rotated using the MPU6050 gyroscope.

The vehicle was then reassembled by moving the wheels to the bottom of the car so that it has enough height clearance to go up the ramp. Using this opportunity, the positions of the MPU6050 module, HC-SR04 ultrasonic sensor, and the HC-05 Bluetooth Serial Transceiver were also planned out ahead of time to maximise space efficiency and tidiness of the wires.

**Observations and Results:**

The MPU6050 gyroscope was quite sensitive and would give small readings even though the car was not rotating. To solve this, a threshold was added to only read readings above a certain value.

The code only seemed to work when the Arduino is connected to the computer. When using battery power with the motors running, it would cause the Arduino to freeze. Through some educated guesses based on the fact that the Arduino would only freeze when the motors are turned on, we suspect that the issue could be because the motors are drawing too much power away from the Arduino. The battery was left to charge to see if it would solve the issue.

**Summary and Conclusion:**

By using the gyroscope on the MPU6050, we can ensure that the rotation of the vehicle is always consistent. The main focus now is to solve the freezing issue of the Arduino. This task is pushed to the next project week.

| | |
|---|---|
| **Date:** | 04/12/2023 |
| **Time:** | 0900H – 1700H |
| **Objective:** | Coding and testing the ramp sequence. |

**Activity Description:**

Continuing from the previous project week, the full ramp sequence was coded to initiate when detecting an increase in the incline to mark the start of the ramp. The following list details the full ramp sequence:

1. Detect the start of the ramp.
2. Ascend the ramp, calculate the angle, and display it.
3. Detect the end of the ramp.
4. Stop for 4 seconds on the ramp.
5. Rotate for 360°.
6. Descend the ramp.

Using the accelerometer on the MPU6050, we can detect when there is a change in the incline of the car, using it as a switch to start and end the sequence.

**Observations and Results:**

We originally only used 1 axis of the accelerometer, which is the x-axis, to detect a change in the incline. This proved to be problematic, as any sufficient acceleration of the car in that axis, including the initial acceleration, would trigger the ramp sequence. To solve this, the z-axis of the accelerometer was also used, as this axis would read a constant ~9.8ms$^{-1}$ whenever it is on flat ground regardless of any other acceleration.

Despite working without issues when the code was tested on the Arduino by itself, the Arduino still freezes when using battery power with the motors turned on, even after the batteries were fully charged.

Potential problems could involve:

1. Interruption of the power source by the motors.
2. Interruption of the I$^2$C communication through electromagnetic interference generated from the motors.
3. Arduino does not have enough processing power to continuously handle too many tasks at once.
4. Buffer overflow due to large amounts of data produced by the MPU6050.
5. Bug in the MPU6050 library.

**Summary and Conclusion:**

A ramp sequence was coded that only works when the motors are turned off. Through some research online and making educated guesses, a list of potential issues was compiled, which is to be tested tomorrow.

**Date:** 05/12/2023

**Time:** 0900H – 1700H

**Objective:** Troubleshooting the MPU6050.

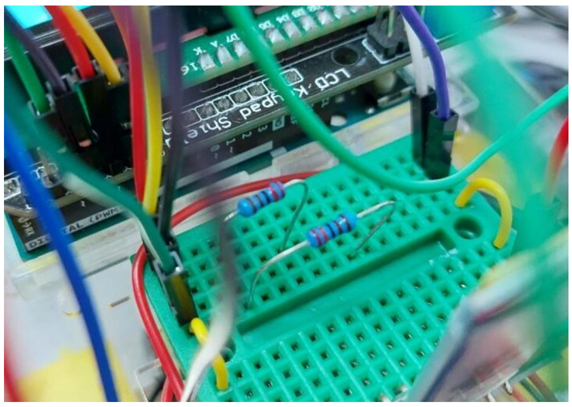
---

**Activity Description:**

The focus for today was to solve the freezing issue of the Arduino when connected with the MPU6050. Based on the list of potential problems, several methods were attempted to solve the issue.

The following list details all the methods we attempted:

- Using a separate power source (a power bank) to power the Arduino.
- Using a different MPU6050 library.
- Using 2 separate Arduinos, 1 for collecting and processing the MPU6050 data which is then sent to the 2nd Arduino via $I^2C$ to control the motors.
- Connecting two 2.2kΩ pull-up resistor to both the SCL and SDA $I^2C$ lines



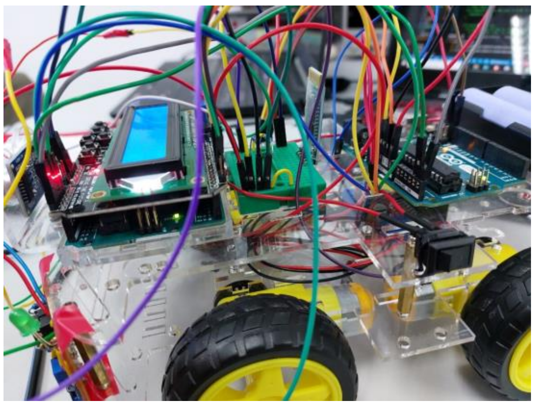


Fig. 11. $I^2C$ pull-up resistor (left).

Fig. 12. Using 2 Arduinos on the vehicle (right).

**Observations and Results:**

None of the methods we tried worked to solve the freezing issue. Using a separate power source only seemed to lengthen the time it took before the Arduino froze.

**Summary and Conclusion:**

Despite trying several different methods to fix the freezing issue, the problem still remained. As we were running out of time, we decided to push the focus towards completing the other objectives for Week 3 first, which were to implement the HC-SR04 Ultrasonic Sensor for obstacle detection, and the HC-05 Bluetooth Serial Transceiver for Bluetooth control.

**Date:** 06/12/2023

**Time:** 0900H – 1300H

**Objective:** Implementing the HC-SR04 Ultrasonic Sensor and the HC-05 Bluetooth Serial Transceiver.

**Activity Description:**

We decided to split the task to get it done faster. I was tasked with implementing the HC-05 Bluetooth Serial Transceiver on the car.

The Bluetooth transceiver was mounted straight onto the small breadboard to allow for easier wiring and more secure placement on the car. That location is also advantageous as it is elevated and further away from the motors, reducing any EMI caused by the motors, improving the reception and range of the transceiver.

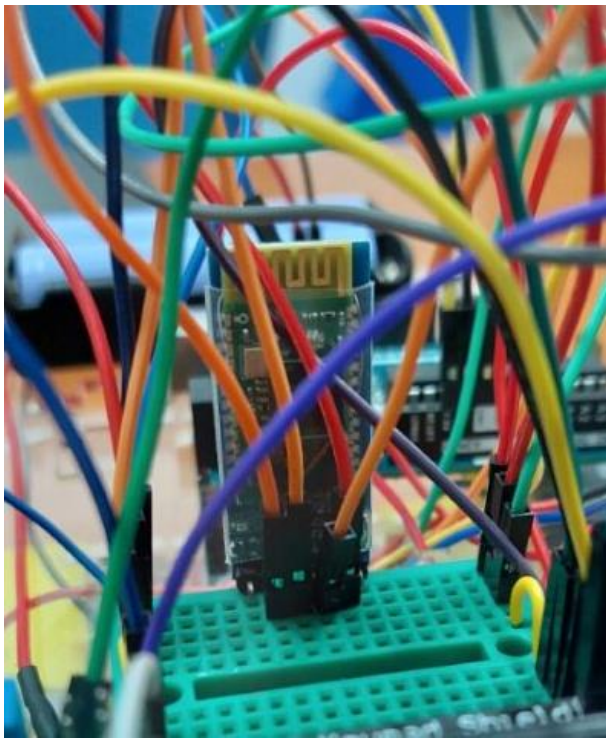
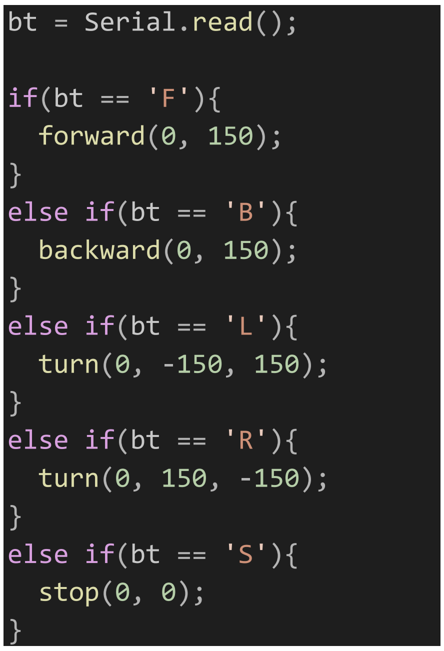


Fig. 13. Bluetooth transceiver mounted on the breadboard.

The Bluetooth transceiver is connected to the UART pins on the Arduino. Data from the Bluetooth transceiver can then be read using the `Serial.read()` function. Based on the data received, movement controls can then be implemented. A Bluetooth controller on a smartphone is used to control the car. It must be set up in such a way that the data sent matches the conditions coded.



```
bt = Serial.read();

if(bt == 'F'){
  forward(0, 150);
}
else if(bt == 'B'){
  backward(0, 150);
}
else if(bt == 'L'){
  turn(0, -150, 150);
}
else if(bt == 'R'){
  turn(0, 150, -150);
}
else if(bt == 'S'){
  stop(0, 0);
}
```

Fig. 14. Bluetooth controller code.

**Observations and Results:**

As there were not enough pins for all of the components, the LCD Keypad shield was detached from the Arduino and mounted independently on the back of the vehicle. This freed up quite a few pins that were obscured by the keypad shield.

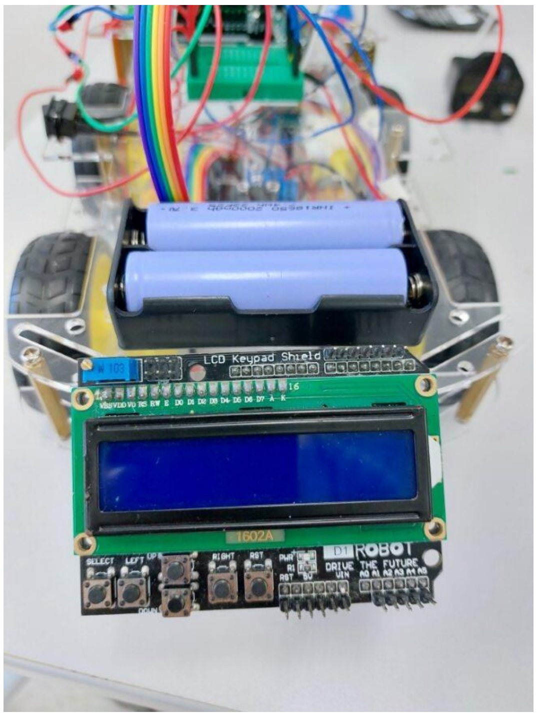


Fig. 15. Mounting the LCD Keypad Shield on the back of the vehicle.

It is also to be noted that the Bluetooth transceiver must be disconnected from the UART pins before uploading the code, otherwise it will interfere with the serial communication between the Arduino and the computer. Otherwise, the Bluetooth control worked flawlessly.

**Summary and Conclusion:**

Both the Bluetooth control and obstacle avoidance features were successfully implemented today. The focus can now be placed on finding an alternative solution that completes Task 1 without causing the Arduino to freeze.

**Date:** 07/12/2023

**Time:** 0900H – 1700H

**Objective:** Coding an alternative solution for task 1 and task 2.

---

**Activity Description:**

Since the MPU6050 constantly causes the Arduino to freeze, we decided to find an alternative solution that only uses the MPU6050 once to measure the ramp angle, instead of doing a continuous reading. The approach was entirely rethought, and the following list details the new ramp sequence:

1. Ascend the ramp halfway and stop.
2. Read the accelerometer to measure the angle once and display it.
3. Continue up the ramp.
4. Stop on the top of the ramp for 4 seconds.
5. Rotate for 360°.
6. Descend the ramp.

Once task 1 was completed, task 2 can be done by simply using an if statement to follow the line until the distance measured is equal to or greater than 80cm. The car is then made to stop for 3 seconds before continuing to the end of the track. The time elapsed is printed on the LCD using the `millis()` function below the total distance.

**Observations and Results:**

The car managed to ascend the ramp and calculate its angle without freezing. A delay before the accelerometer reading was added to ensure that the car is stationary when the angle is being measured.

After the 360° rotation, the car was sometimes misaligned, and would cause it to go in unpredictable directions when descending the ramp. A line following component was added to realign the car before its descension.

No issues were found with the task 2 portion of the code.

**Summary and Conclusion:**

By reading the accelerometer only once instead of doing it continuously, the freezing issue of the Arduino can be circumvented. Task 1 and task 2 were successfully completed today.

**Date:** 08/12/2023

**Time:** 0900H – 1700H

**Objective:** Implementing advanced features.

**Activity Description:**

For the advanced features, we have decided to do a Proportional-Integral-Derivative (PID) control to move the vehicle in a straight line. Normally, the vehicle can travel in a straight line for short distances, but the longer it is allowed to move, it will tend to move towards one direction or another causing it to go off course. To solve this, 2 rotary encoders are used, 1 on the wheels of each side of the vehicle. For the car to travel in a straight line, the distance on both sides must be equal. Using the number of pulses, the rotary encoders measure the difference in the distance travelled by the wheels on each side of the car which is the error. Based on that error value, the speed of the wheels can be adjusted accordingly using a PID controller so that the car travels in a straight line.

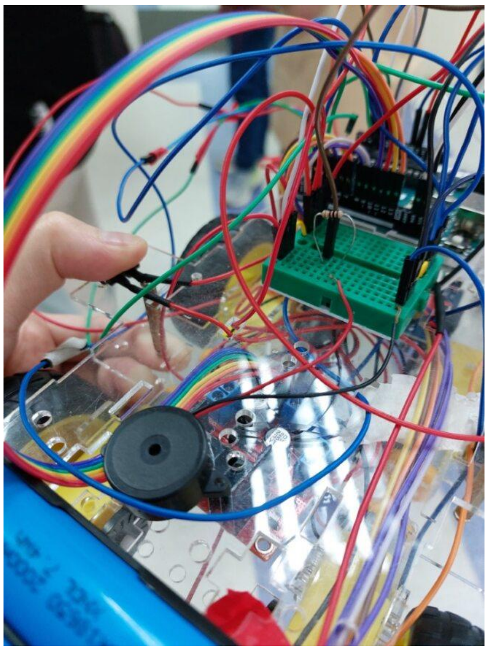
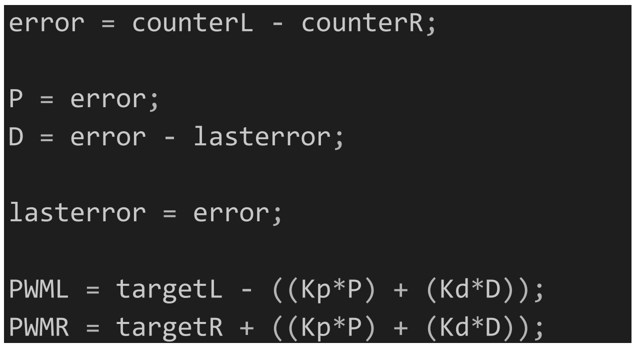


Fig. 16. Adding the buzzer onto the vehicle.

The second advanced feature we decided on was implementing a buzzer with multiple functionalities, such as a horn, a siren, or a music player. Using the `tone()` function, sound waves of different frequencies can be played on the buzzer.

**Observations and Results:**

The first prototype of the PID controller only included the proportional component. This worked okay in terms of keeping the vehicle in a straight line, but it tended to oscillate too much. Therefore, a derivative component which compares the current error with the previous error was also added to dampen the oscillations over time. After a bit of tuning, the vehicle can now travel in a straight line even after long stretches. As it was quite troublesome to show the difference between the movements with and without the PID control, a button on the LCD Keypad Shield was used to activate the PID control whenever needed.



```
error = counterL - counterR;

P = error;
D = error - lasterror;

lasterror = error;

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

Fig. 17. PID Control code using the difference in readings of the 2 rotary encoders.

The buzzer was implemented without much trouble. We decided to use another one of the LCD Keypad Shield buttons to activate the buzzer so that we can have more control over when it turns on.

**Summary and Conclusion:**

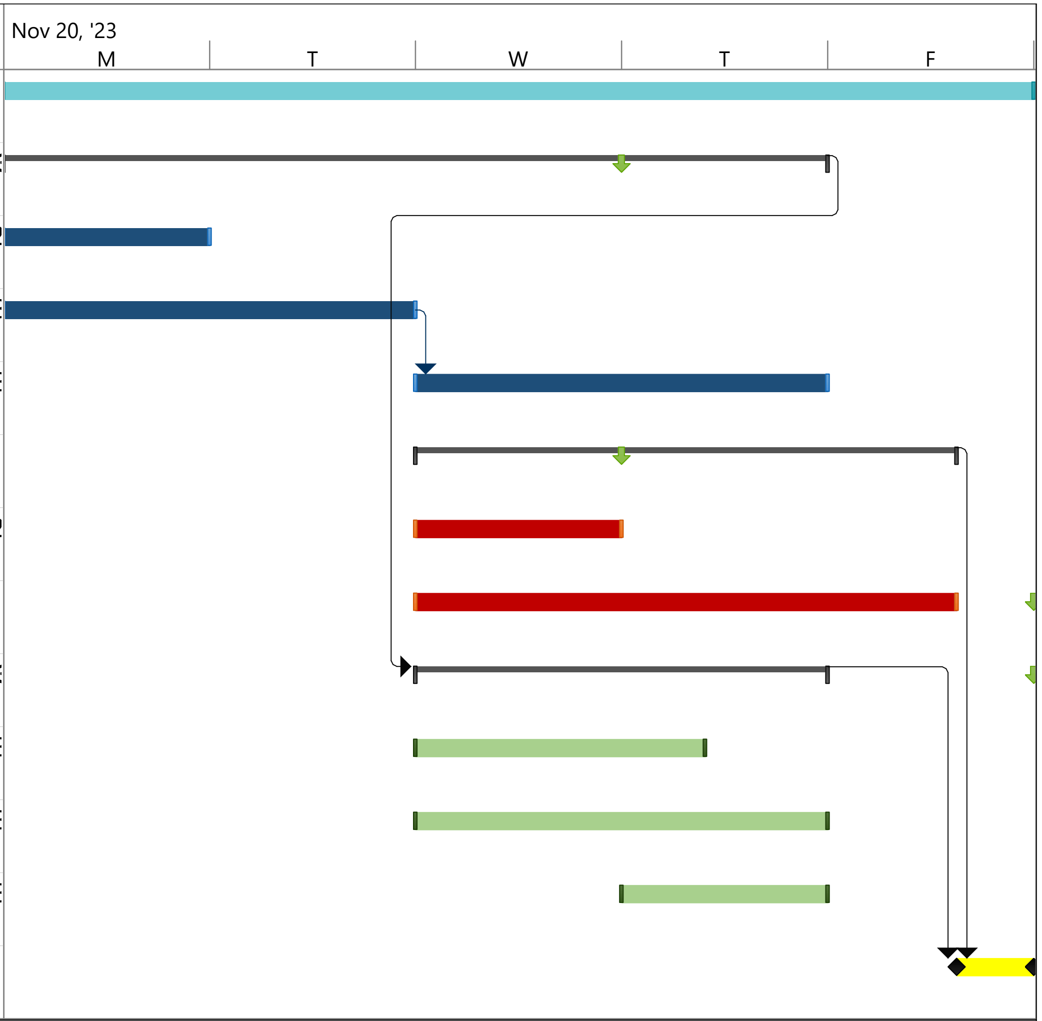
The advanced features were successfully implemented today. With the PID control and 2 rotary encoders, the vehicle is now able to move in a straight line especially for long stretches of movement. The buzzer is also a fun and functional addition to the vehicle.

# Appendix A

Project planning Gantt Chart for week 2 and week 3.

See next page.

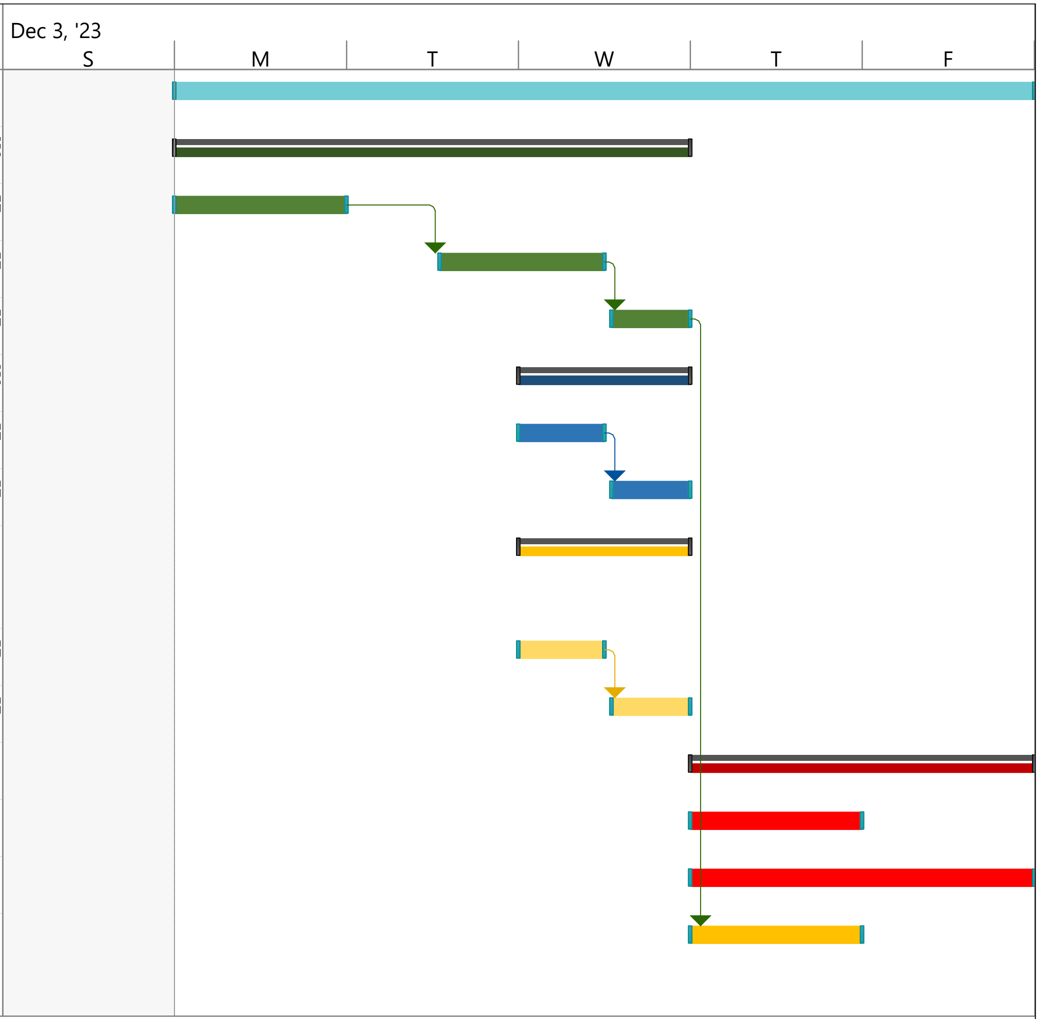
| ID | ℹ | Task Mode | Task Name | Duration | Start | Finish |
|---|---|---|---|---|---|---|
| 1 | | 📌 | PROJECT WEEK 2 | 5.57 days | Mon 11/20/2 | Fri 11/24/23 |
| 2 | ⚠ | ➡ | **Light Sensor** | **4.57 days** | **Mon 11/20/2** | **Thu 11/23/2** |
| 3 | | 📌 | Build on Breadboa | 1.14 days | Mon 11/20/2 | Mon 11/20/2 |
| 4 | | 📌 | Build on Veroboar | 2.29 days | Mon 11/20/2 | Tue 11/21/2 |
| 5 | | 📌 | Soldering | 2.29 days | Wed 11/22/2 | Thu 11/23/2 |
| 6 | ⚠ | ➡ | **LCD (Display Distanc** | **3.15 days** | **Wed 11/22/2** | **Fri 11/24/23** |
| 7 | | 📌 | Interface Arduino | 1.14 days | Wed 11/22/2 | Wed 11/22/2 |
| 8 | | 📌 | Debug Code | 3.15 days | Wed 11/22/2 | Fri 11/24/23 |
| 9 | | ➡ | **Interface Light Senso** | **2.29 days** | **Wed 11/22/2** | **Thu 11/23/2** |
| 10 | | 📌 | Wire Connections | 1.39 days | Wed 11/22/2 | Thu 11/23/2 |
| 11 | | 📌 | Program Code | 2.29 days | Wed 11/22/2 | Thu 11/23/2 |
| 12 | | 📌 | Debug Code | 1.43 days | Wed 11/22/2 | Thu 11/23/2 |
| 13 | | 📌 | Final Track Challenge | 1.16 days | Fri 11/24/23 | Fri 11/24/23 |



Project: Week2 Ganatt Chart
Date: Wed 12/13/23

| | | | | | |
|---|---|---|---|---|---|
| Task | ▬ | Inactive Summary | ▭ | External Tasks | ▬ |
| Split | ┈ | Manual Task | ▬ | External Milestone | ◇ |
| Milestone | ◆ | Duration-only | ▬ | Deadline | ⇩ |
| Summary | ┏━┓ | Manual Summary Rollup | ▬ | Progress | ━ |
| Project Summary | ┏━┓ | Manual Summary | ┏━┓ | Manual Progress | ━ |
| Inactive Task | ▭ | Start-only | [ | | |
| Inactive Milestone | ◇ | Finish-only | ] | | |

| ID | ℹ | Task Mode | Task Name | Duration | Start | Finish |
|---|---|---|---|---|---|---|
| 1 | | 📌 | PROJECT WEEK 3 | 5 days | Mon 12/4/23 | Fri 12/8/23 |
| 2 | | 📌 | **Interface MPU-6050** | **3 days** | **Mon 12/4/2** | **Wed 12/6/2** |
| 3 | | 📌 | Coding | 1 day | Mon 12/4/23 | Mon 12/4/23 |
| 4 | | 📌 | Debug | 1 day | Tue 12/5/23 | Wed 12/6/23 |
| 5 | | 📌 | Wiring | 0.5 days | Wed 12/6/23 | Wed 12/6/23 |
| 6 | | 📌 | **Integrate Bluetooth** | **1 day** | **Wed 12/6/2** | **Wed 12/6/2** |
| 7 | | 📌 | Coding | 0.5 days | Wed 12/6/23 | Wed 12/6/23 |
| 8 | | 📌 | Debug | 0.5 days | Wed 12/6/23 | Wed 12/6/23 |
| 9 | | 📌 | **Interface Ultrasonic Sensor** | **1 day** | **Wed 12/6/23** | **Wed 12/6/23** |
| 10 | | 📌 | Coding | 0.5 days | Wed 12/6/23 | Wed 12/6/23 |
| 11 | | 📌 | Debug | 0.5 days | Wed 12/6/23 | Wed 12/6/23 |
| 12 | | 📌 | **Additional Features** | **2 days** | **Thu 12/7/23** | **Fri 12/8/23** |
| 13 | | 📌 | Buzzer | 1 day | Thu 12/7/23 | Thu 12/7/23 |
| 14 | | 📌 | PID Control | 2 days | Thu 12/7/23 | Fri 12/8/23 |
| 15 | | 📌 | Final Track Challenge | 1 day | Thu 12/7/23 | Thu 12/7/23 |



Project: poroject wwek 3
Date: Wed 12/13/23

| | | | | | |
|---|---|---|---|---|---|
| Task | ▬ | Inactive Summary | ▭ | External Tasks | ▬ |
| Split | ┈ | Manual Task | ▬ | External Milestone | ◆ |
| Milestone | ◆ | Duration-only | ▬ | Deadline | ⇩ |
| Summary | ┏━┓ | Manual Summary Rollup | ▬ | Progress | ━ |
| Project Summary | ┏━┓ | Manual Summary | ┏━┓ | Manual Progress | ━ |
| Inactive Task | ▭ | Start-only | [ | | |
| Inactive Milestone | ◇ | Finish-only | ] | | |

# Appendix B

L298N motor driver library.

```
#include <Arduino.h>
#include <car.h>


/*
 *   Hex to Binary Cheat Sheet
 *  ==========================================
 *   A = 1010 -> both motors forward
 *   9 = 1001 -> right forward, left backward
 *   6 = 0110 -> left forward, right backward
 *   5 = 0101 -> both motors backward
 */

/* accepts input as a user defined literal hex operator and converts to 4 bit
binary, each bit for each of the 4 motor driver pins  */
void send_to_driver(unsigned char bit, int PWML, int PWMR){
    digitalWrite(IN1, ((bit >> 3) & 0x1)); //right forward
    digitalWrite(IN2, ((bit >> 2) & 0x1)); //right backward
    digitalWrite(IN3, ((bit >> 1) & 0x1)); //left forward
    digitalWrite(IN4, ((bit >> 0) & 0x1)); //left backward

    analogWrite(ENA, PWMR); //right pwm
    analogWrite(ENB, PWML); //left pwm
}

/* forward for time t in milliseconds */
void forward(int t, int PWM){
    send_to_driver(0xA, PWM, PWM);
    delay(t);
}

/* backward for time t in milliseconds */
void backward(int t, int PWM){
    send_to_driver(0x5, PWM, PWM);
    delay(t);
}

/* stop for time t in milliseconds */
void stop(int t, int PWM){
    send_to_driver(0x0, PWM, PWM);
    delay(t);
}

/* turns the vehicle for time t in milliseconds */
```

```
void turn(int t, int PWML, int PWMR){
    if(PWML < 0){   //left motor backward, right motor forward
        send_to_driver(0x9, -PWML, PWMR);
        delay(t);
    }
    else if(PWMR < 0){  //right motor backward, left motor forward
        send_to_driver(0x6, PWML, -PWMR);
        delay(t);
    }
    else if((PWML < 0) && (PWMR < 0)){  //both motor backward
        send_to_driver(0x5, -PWML, -PWMR);
        delay(t);
    }
    else{   //both motor forward
        send_to_driver(0xA, PWML, PWMR);
        delay(t);
    }
}

void followLine(int IRL, int IRR, int LT, int RT, int t, int v){
    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);
    }
}
```