EG1311 Design and Make

B03 Team 6 Lab Report

Name	Matriculation Number
Chai Ching How	A0201445U
Chong Kai Jie Bryan	A0201477J
Chua Yu Rui	A0201437R
Chung Jun Yu Chester	A0201459J
Jeong Jaehun	A0201379H
Low Vic Yen	A0200497H

Table of Contents

1. Background

a. Goal

We are a team of engineers who have utilised the 5-stage Design Thinking process (Emphasise, Define, Ideate, Prototype and Test) to address the challenge of designing an all-terrain robot to support disaster relief.

b. Requirements

In order to support robot deployment in "areas that are uneven", our robot needs to be able to manoeuvre across a predefined obstacle course as described in Figure 1.

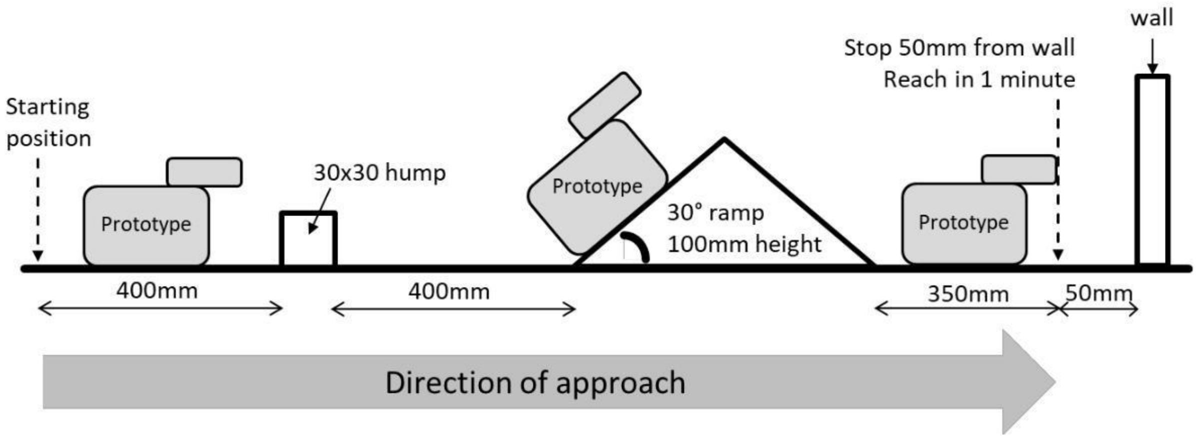


Figure 1: Predefined obstacle course

c. Key stakeholders and insights

Prior work on the identification of stakeholders, their key insights and needs was carried out.

<u>Target audience (key stakeholders):</u>

1. Trapped disaster victims
2. First responders
3. Local authorities

Key Insights	Needs
"The robot will be deployed in areas that are uneven"	The robot should possess enough stability and operate normally even when traversing through uneven terrains autonomously
"The robot should be friendly-looking as the victims will be in distress" "It shouldn't scare the victims when it gets close"	The robot's appearance must be friendly-looking and invoke a sense of assurance to the victims, providing a sense of calm and hope
"There should be flexibility to add more accessories to meet emergent challenges on the ground"	The robot should be able to handle loading of additional accessories to adapt and overcome any emerging challenges faced on the ground
"It has to transmit signals back regularly so that we know it is still working"	The robot must relay constant and consistent feedback of information back to the control station

2. **Ideation**
   a. **Generation of ideas**

Using Collaborative Sketch (C-Sketch), each team member independently sketched out their ideal solution based on the design requirement. The sketch was then passed on to the next member for inputs on improvement for the solution. This continued until every member had the opportunity to revise every sketch. The team then discussed and selected outstanding features to include in the prototype, narrowing it down to 3 distinctive ideas.

The 3 distinctive ideas generated are as follows:

✓ **Solution 1**: Rocker Bogie
As seen in Figure 1, this prototype sports independently moving front wheels, bearing a similarity to 2 front limbs. Coupled with high-friction wheel threads, this unique mechanism was designed to effectively clear obstacles twice the diameter of its wheel without compromising the stability of the vehicle.

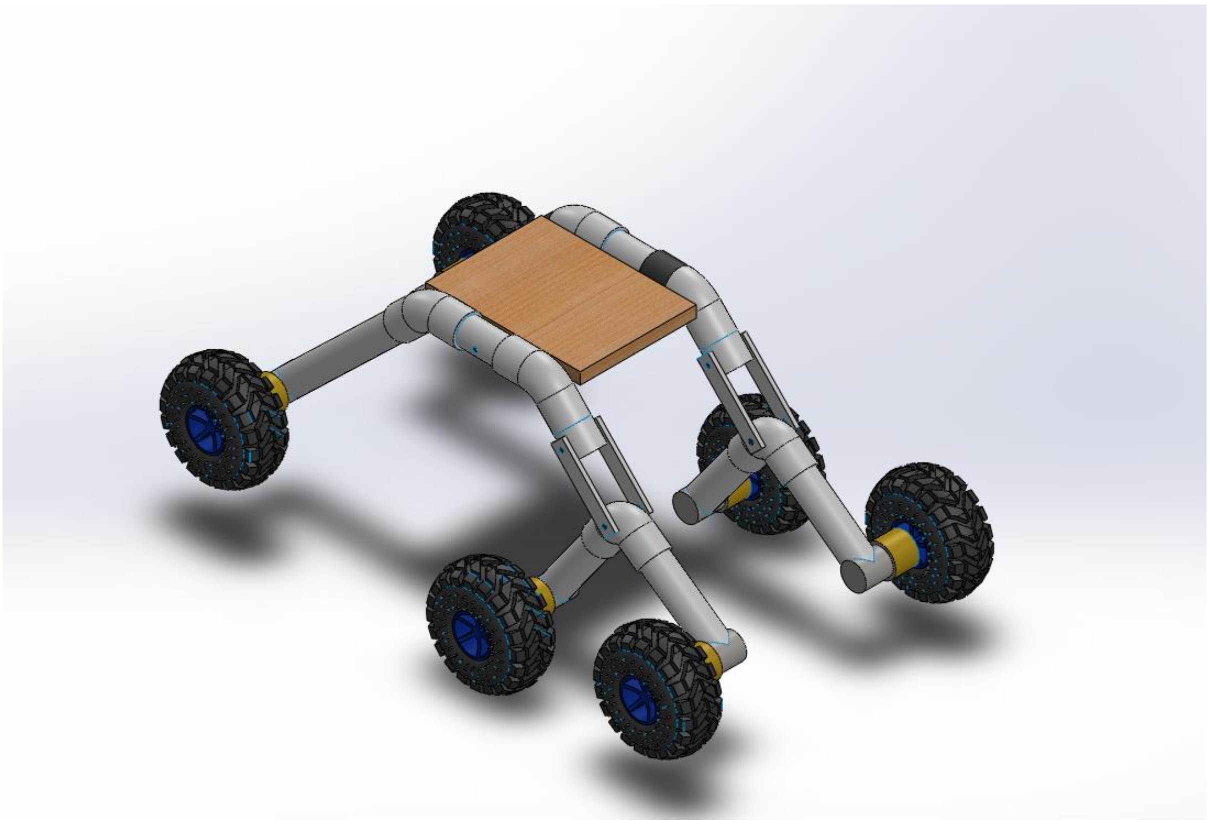


Figure 1: Rocker Bogie

✓ **Solution 2**: Monster Pram
As seen in Figure 2, this prototype is similar to a tractor, sporting small front wheels and significantly larger rear wheels. The small front wheels allow for sharper turning owing to a smaller turning

radius, while the big and broad rear wheels helps to distribute the vehicle’s weight across a larger surface area, reducing the pressure the vehicle applies on the surface it is travelling over, preventing it from sinking into soft ground. Furthermore, the large base area keeps the centre of gravity low, preventing the vehicle from falling over while going up inclined surfaces.

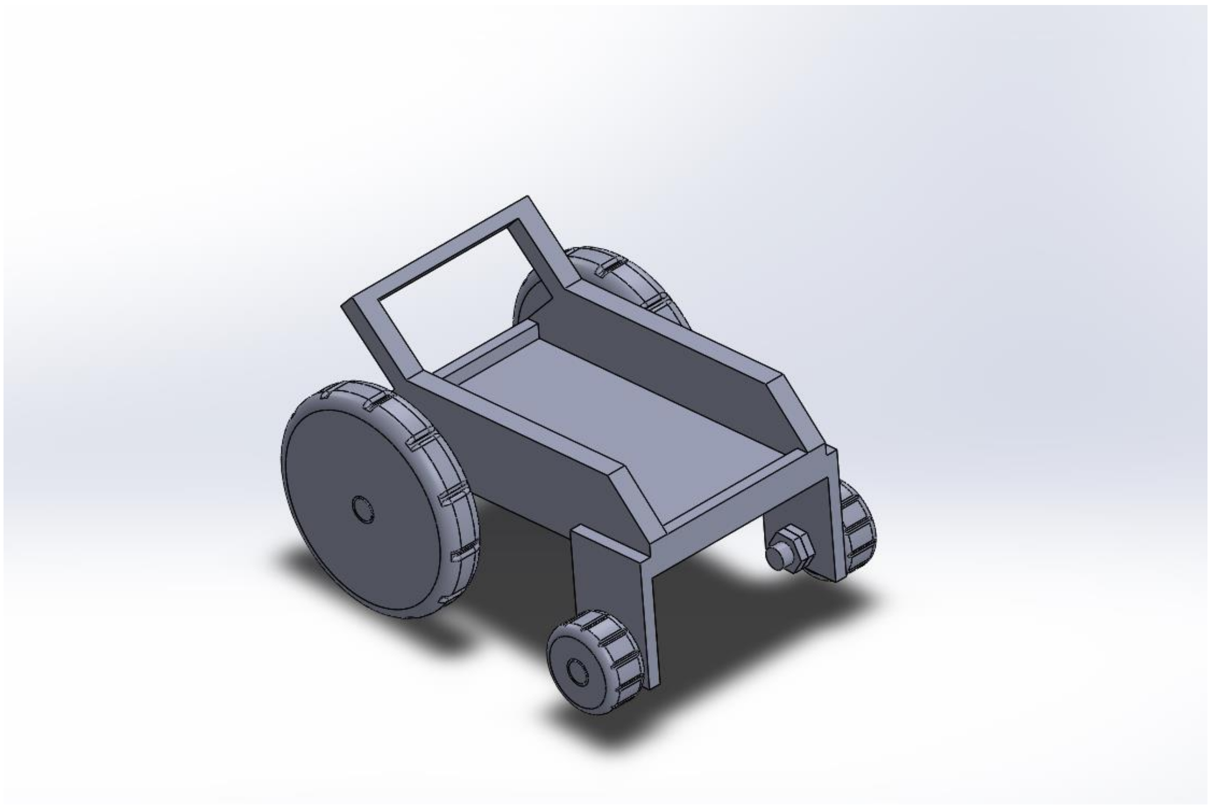


Figure 2: Monster Pram

- ✓ **Solution 3**: Mystery Machine
  As seen in Figure 3, this prototypes’ feature is the use of continuous tracks. Due to the increased area of contact, tracks provide higher traction compared to wheels and is also able to traverse difficult terrains with ease as chained links can grip and wrap around the uneven surfaces. Furthermore, with tracks, the base area of the vehicle is greatly increased, adding to its stability even at high speeds.

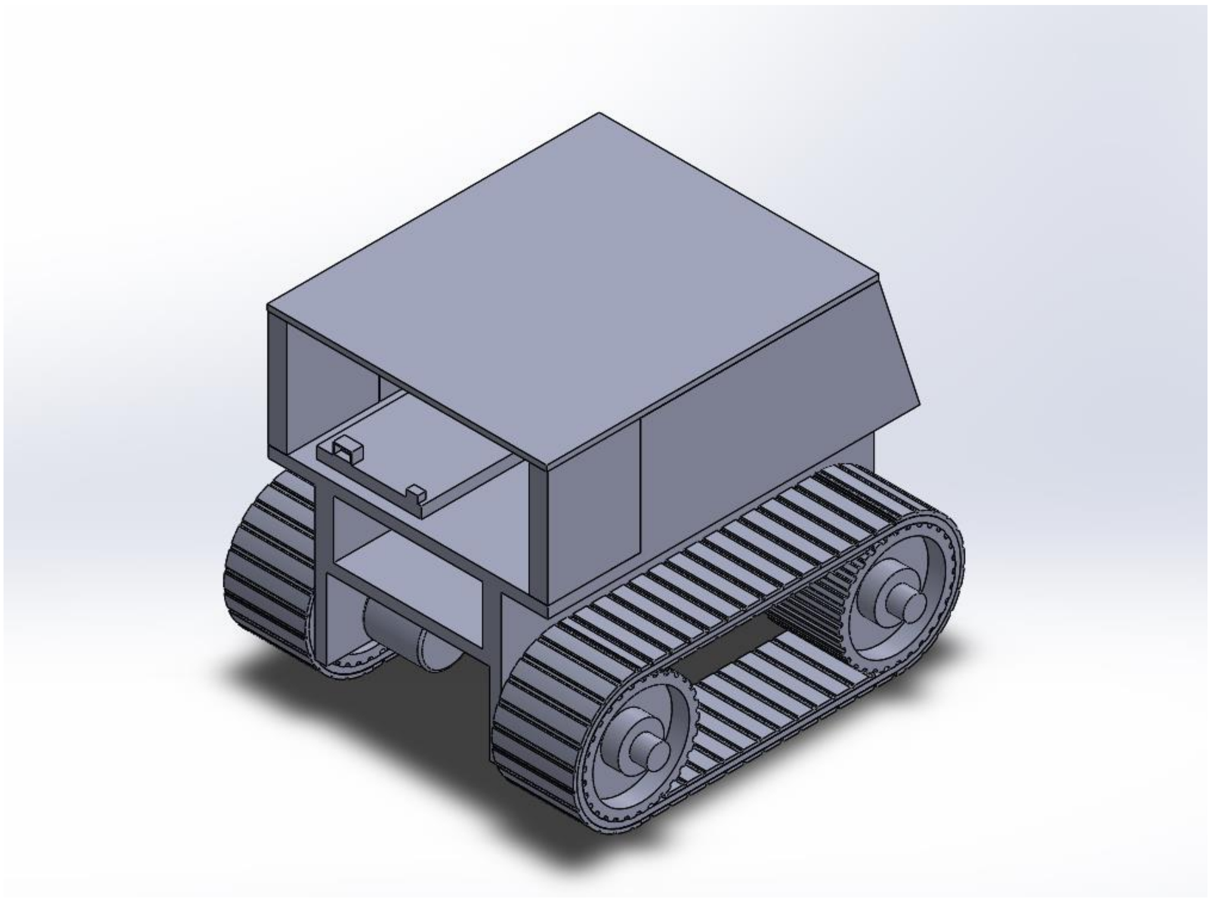


Figure 3: Mystery Machine

b. **Considerations**

- Solution 1 is unstable at high speeds as the front limbs will collide head-on with any obstacles if there was insufficient time for it to roll over the obstacle. The impulsive force from collisions will damage the front limbs, compromising its structural integrity in the long run.

- Solution 2 is also unstable at high speeds as any speed above the optimal speed limit would pose a severe risk of the vehicle toppling over. Furthermore, bigger wheels require a higher amount of torque and experience lesser rotations per minute compared to smaller wheels, thus it would be challenging to sync the motor speed using code given our level of expertise.

- Solution 3 which runs on continuous track would be costlier to maintain and upkeep as compared to vehicles on wheels.

- Filtering through the key insights, we came to a consensus that the main consideration should be the deployment of the robot in uneven areas. What enables the robot to tread undulating terrain successfully will thus be our focus. We then discussed extensively over the method of travel for the robot - whether it should be a tracked vehicle or just a simple vehicle with wheels.

c. Selection of ideas

All the ideas were assessed by each team member and evaluated based on the following criterion: Stability, Durability, Utility, Rationality, Feasibility and Aesthetics. As depicted in Figure 4, 5 and Table 1, while wheels provide higher speed, and ease of maintenance and repair, tracks provide higher traction, increase manoeuvrability, and are more mobile over diverse terrain, making it more suitable in the situation of a post-disaster environment. The results shown in Table 2 indicated that Solution 3, Mystery Machine, scored the highest. Coupled with careful analysis of the benefits provided by both tracks and wheels, our team selected Solution 3 to be our final design (Figure 6).

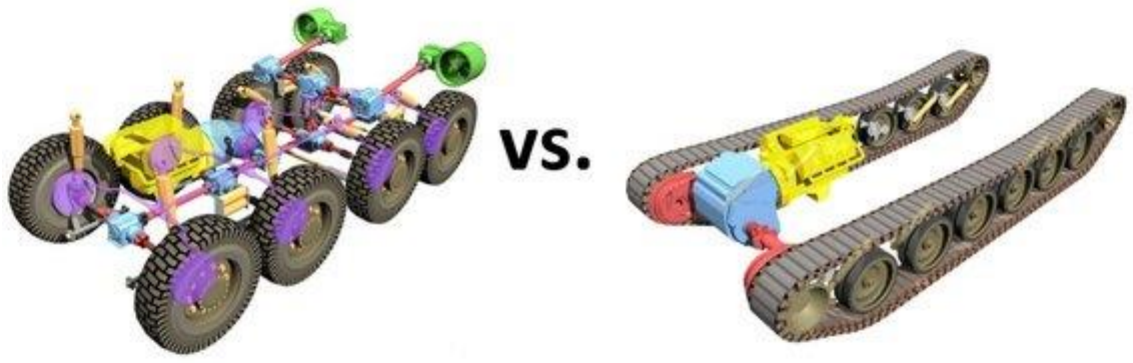


Figure 4: Wheels vs Tracks

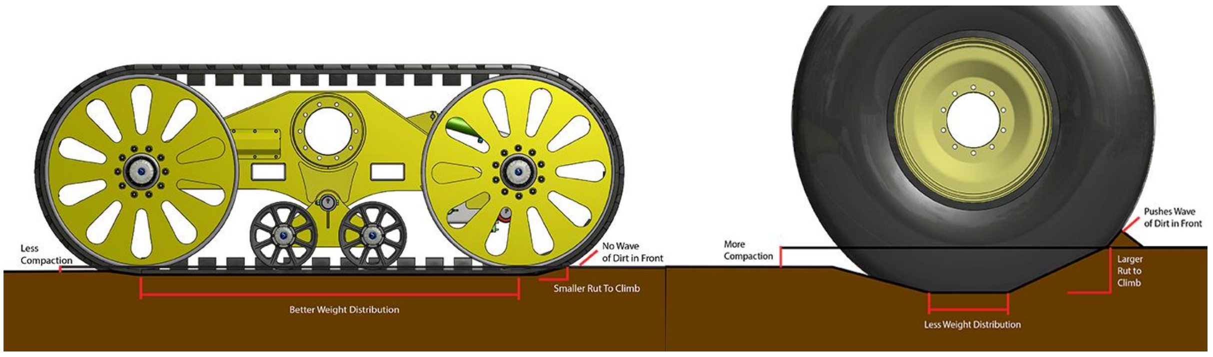


Figure 5: Comparison of compaction, weight distribution and rut

Track	Wheel
High traction	Higher speed
Mobility over diverse terrains	Simplicity (Maintenance and ease of repair)
Manoeuvrability (Pivot-steering)	

Table 1: Benefits of tracks and wheels

Prototype	Stability	Durability	Utility	Rationality	Feasibility	Aesthetics	Score
Rocker Bogie	4	3	3	4	5	5	24
Monster Pram	3	5	4	3	5	2	22
Mystery Machine	4	5	5	5	5	3	28

Table 2: Voting results

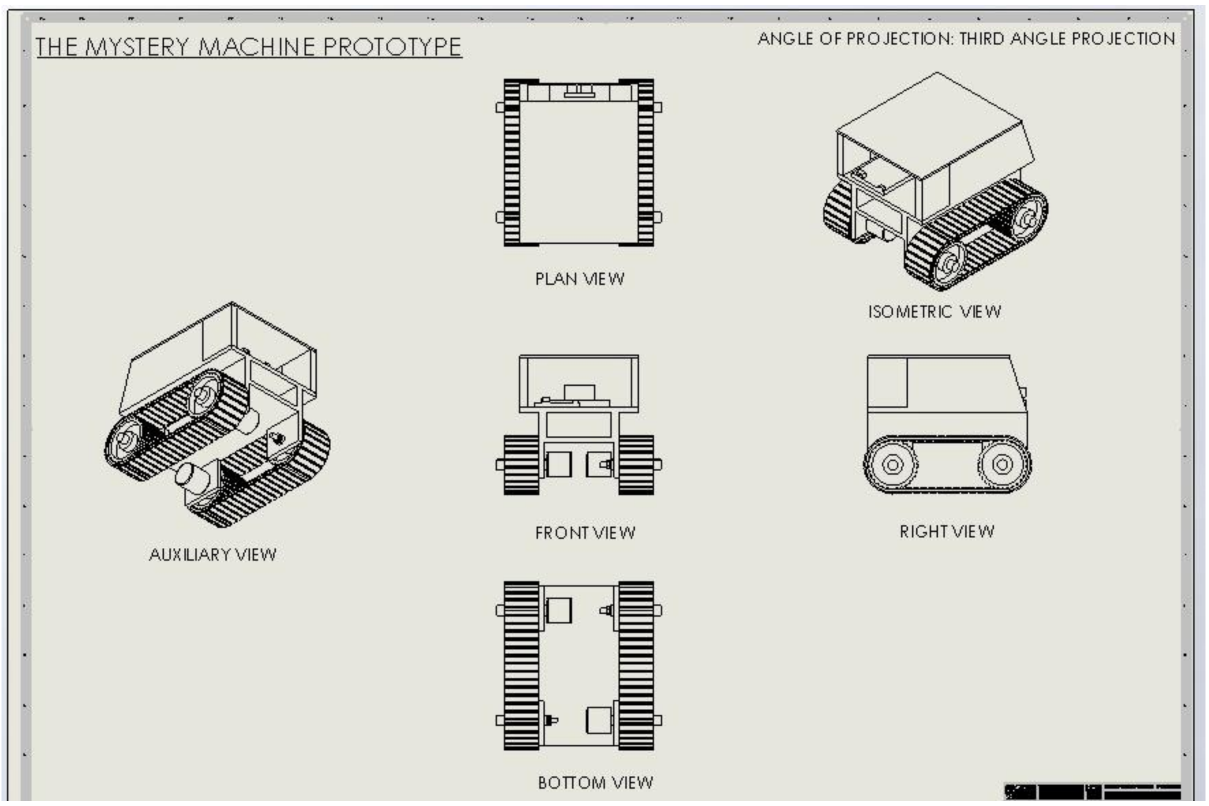

THE MYSTERY MACHINE PROTOTYPE
ANGLE OF PROJECTION: THIRD ANGLE PROJECTION
PLAN VIEW
ISOMETRIC VIEW
AUXILIARY VIEW
FRONT VIEW
RIGHT VIEW
BOTTOM VIEW

Figure 6: Final design

3. **Reflections**

The significant step in bringing us closer to our goal was continuous trial and error on the robot in order to satisfy the requirement of clearing the obstacle course. Trials helped us gain insight on unforeseen circumstances that simple calculations could not account for, highlighting the need for certain measures to be put in place to further improve our robot. We broke down our learning pointers into two general portions, hardware and software.

a. **Failures and Improvements (Hardware)**

× Initially, our team tried to clear the obstacles by only using the Arduino UNO onboard voltage (5V) as the voltage source for the motors. However, output voltage of 5V was insufficient to even clear the hump (Figure 7). Furthermore, the robot’s movement was extremely slow even when on flat terrain.

✓ Our team decided to find an alternative voltage source for the motors. With budget constraints, we settled with a 9V alkaline battery.

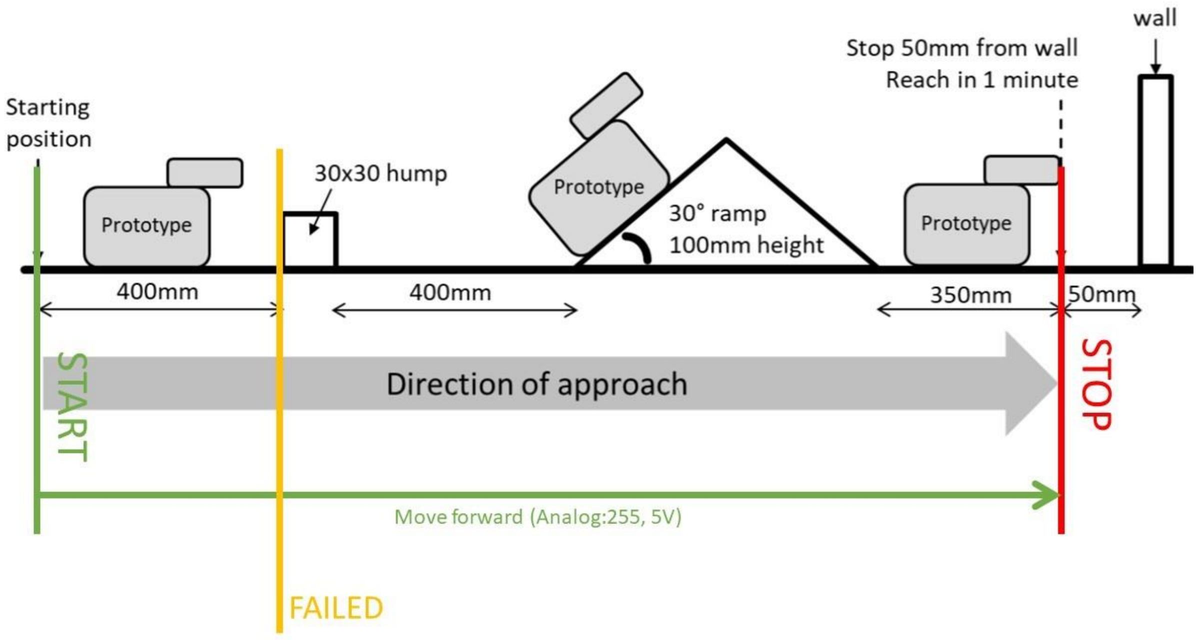


Figure 7: Failure to overcome the hump

× Our robot was unable to clear the ramp initially due to the lack of friction in the tracks since the tracks were made of plastic (Figure 8) given that it was meant for building prototypes.

✓ Initially, we tightened the tension of the tracks in hope that the difference in slack could power the vehicle brutely through the course. While we were successful in doing so, we were not contented as it did not target the root cause of the problem. Furthermore, in real-world context, obstacles may be steeper and slippier. Thus, we decided to wrap anti-slip mats around the tracks (Figure 8) to increase friction and grip of the vehicle.

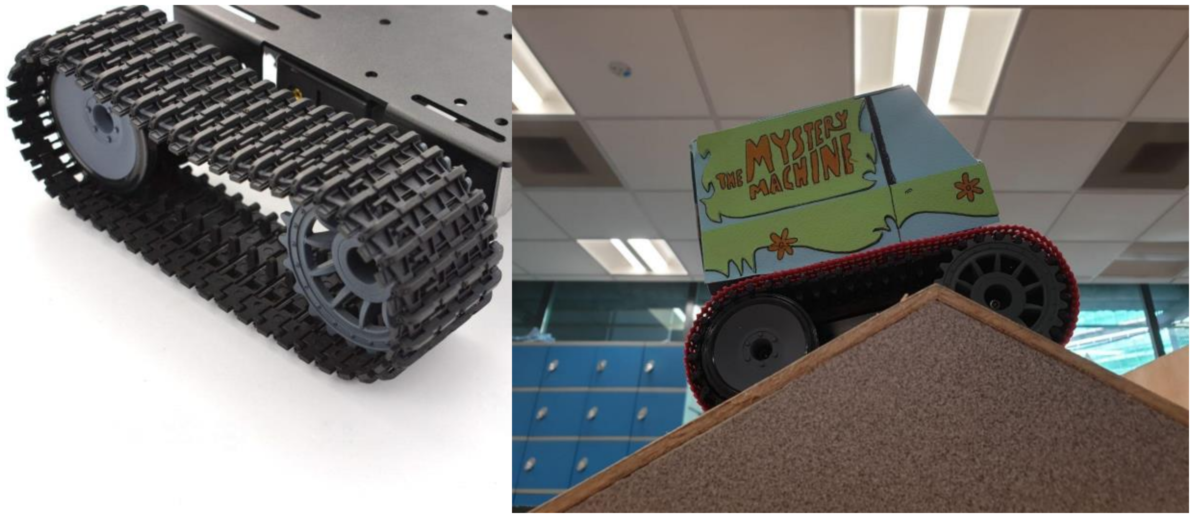


Figure 8: Enhancement made to increase friction and grip of vehicle

- × With a stop range of 50±5mm, our team finally got some successful runs as the robot could clear all the obstacles and stop at the desired position. However, we were not satisfied with the success rate as the robot was unable to stop exactly at the 50mm mark most of the time (Figure 9).
- ✓ There were two reasons for such failure. One of it was that the distance was not measured properly in the subsequent runs since the robot's direction was shifted after it had cleared the obstacles. The slanted robot results in the ultrasonic sensor (front of the receiver and transmitter) not being aligned to the surface of the wall. Another reason could be the design of the track itself. Since we had chosen to use existing tracks on the market, given that they are mass produced, most of it are not perfectly manufactured, resulting in slight errors in dimension, balance, and alignment. To solve these two possibilities, we added a gyroscope (MPU-6050), which helps in auto-calibrating the robot's direction.

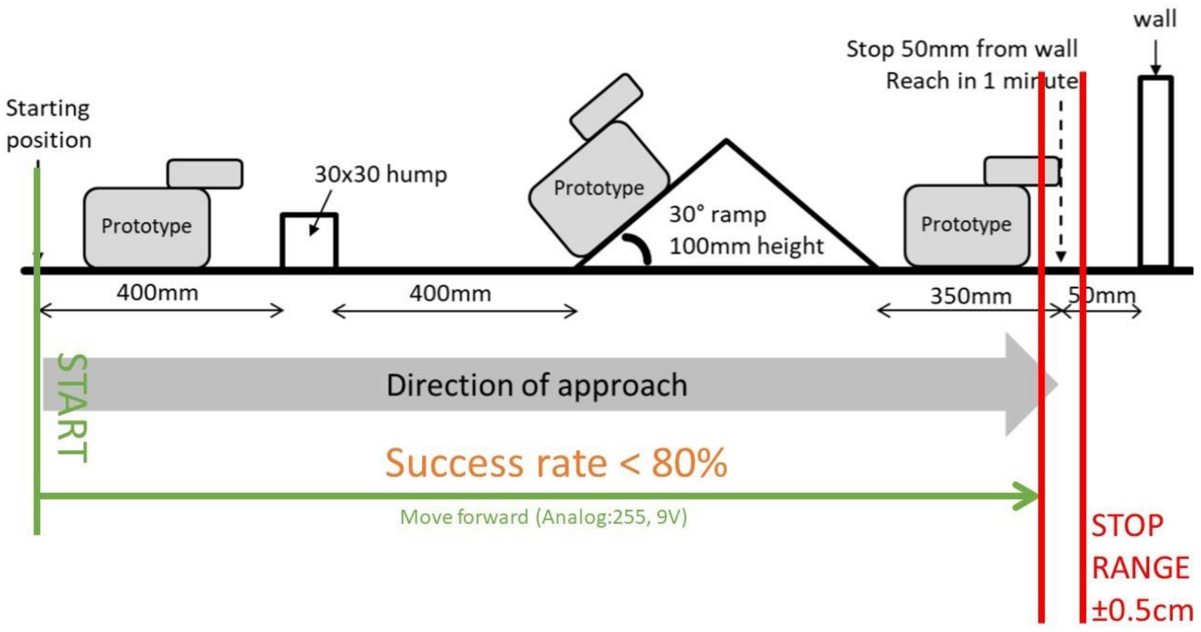


Figure 9: Low success rate of stopping exactly 50mm before the wall

- × The gyro-meter will return parameters including the yaw value of the robot, and we wish to keep our yaw value to 0 as much as possible through the whole obstacle course by controlling our robot's left and right track's motor voltage output deliberately to adjust the direction of the robot. However, when the adjusting (in the event of the robot's yaw value exceeding the set range) happens concurrently during the clearing of obstacles, our robot does not have enough power to clear the obstacles (Figure 10) as one of the motors would be slowed down during the adjusting process.
- ✓ Initially we wanted to solve the issue with the implementation of a gearbox, since it would reduce the speed and increase the torque of

the robot to clear the obstacles. However, we did not want to compromise our speed throughout the obstacle course and we just needed to make sure that the robot has enough power to clear the obstacle when it encounters them. Hence, we went ahead with increasing our alternative voltage source to 18V (2*9V batteries connected in series)

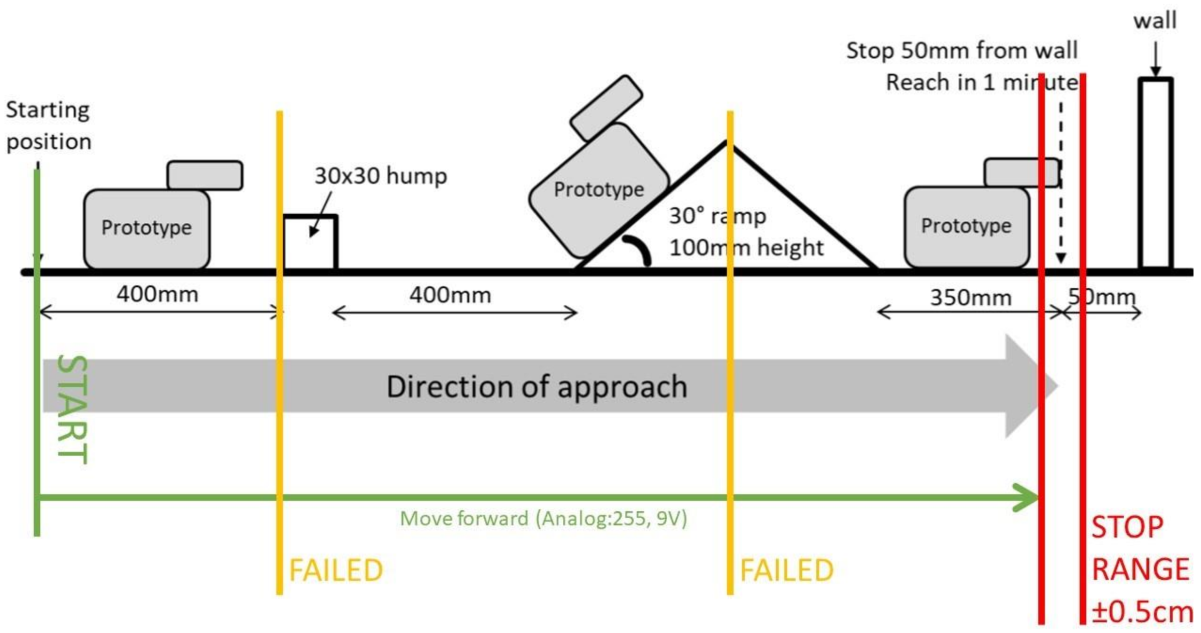


Figure 10: Failure to clear obstacles with the addition of a gyroscope

b. Failures and Improvements (Software)

× Our robot could not stop exactly 50mm before the wall.

✓ Our team figured out that the issue was due to overcompensation of movement of vehicle while making small adjustments, since there was no leeway provided in the code, thus, we decided to include a stopping range of ±5mm.

× With an extensive voltage output (18V), our team was worried that prolonged high input would greatly reduce the motor's lifespan since the motor was 12V rated.

✓ Our team decided to reduce the maximum Analog output in the code from 255 to 200 and breakdown the clearing of the obstacle course into two stages. With reference to Figure 11, both sides of the tracks would run under the same voltage (Analog: 200, 14.11) to clear the two obstacles (hump and ramp), after which, the robot would start to slow down (Analog:100, 7V) and begin calibrating after it has passed through the 300mm mark from the wall. This implementation achieved a high success rate in stopping exactly 50mm from the wall while reducing damage to the motors.

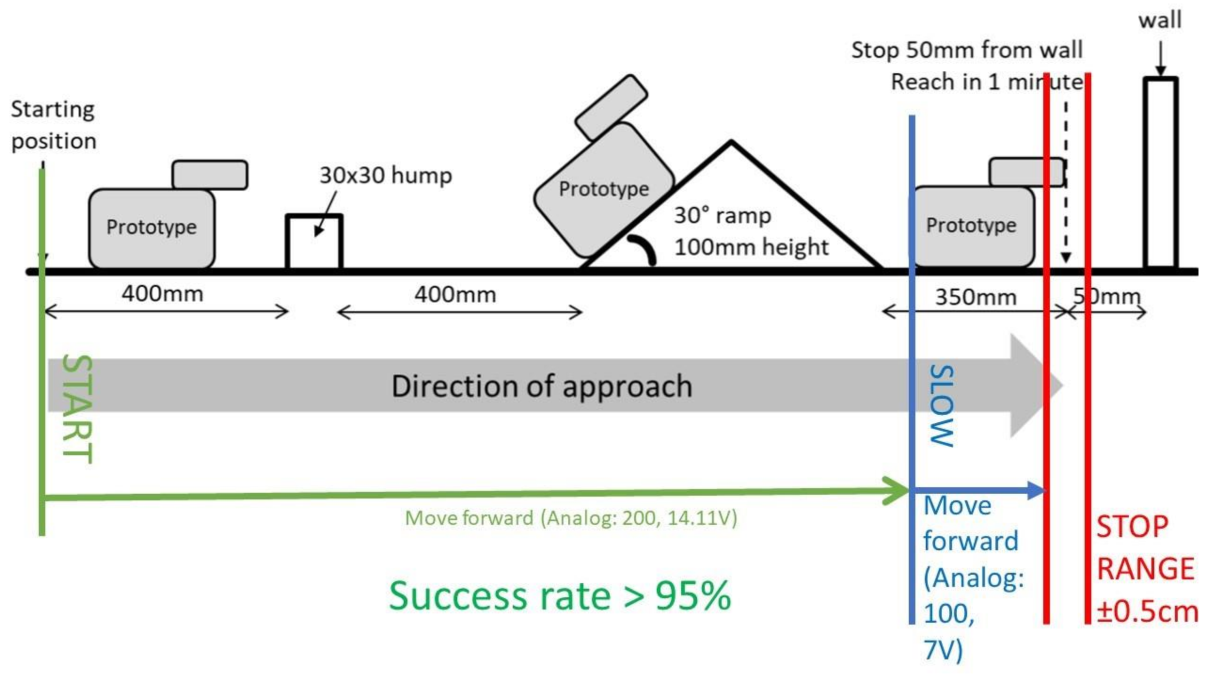


Figure 11: Clearing of obstacle course into two stages

4. **Summary**
   a. **Final design (specifications)**
   After fine-tuning our prototype based on the lessons learnt in the 2 trial runs, the final design of our Mystery Machine was built as follows:
      i. **Arduino UNO**
      Our team was contemplating between using Arduino UNO or Raspberry PI. Eventually, we decided to go with Arduino UNO because of three main factors. Firstly, it is a cheaper alternative compared to Raspberry PI, which is budget-friendly if required to be mass-produced. Secondly, the Arduino platform is more user-friendly and easier to understand as it is very simple to interface sensors and other electronic components to Arduino. However, for Raspberry PI, it requires complex tasks like installing libraries and software for interfacing sensors and other components. Lastly, Arduino UNO is easier to power as it is compatible with most battery packs in the market. Therefore, we decided to go with Arduino UNO.
      ii. **L293D with 12V DC Motor powered by 2*9V Alkaline Battery**
      Since the Arduino UNO can only deliver currents up to 40mA, our team have decided to use the L293D to amplify the input current as well as the torque of the motor, without sacrificing the speed of the motors. Furthermore, the L293D has a large voltage drop, providing the motor with less voltage than the power voltage applied to the driver. Therefore, it can be used with small 12V DC motors, which provides enough torque to move the vehicle in a post-disaster environment and can be overclocked under extreme conditions.

Due to the size and weight of the Mystery Machine, two 9V batteries are required to power it through the obstacle course, particularly when going up the ramp.

iii. **Gyrometer (MPU-6050)**
Our team attached a gyrometer to ensure that the vehicle does not veer off course. Data on the yaw, pitch, roll and acceleration was collected and used to self-correct its movement throughout the course, by adjusting the speed of the tracks accordingly.

iv. **Ultrasonic Sensor**
Our team attached an ultrasonic sensor to accurately detect distance from obstacles so as to avoid collision with the wall at the end of the obstacle.

v. **WIFI Module (ESP32)**
Our team attached a WIFI module that operates on Access Point mode (router) to allow the control station to access the sensor data of the robot (Figure 12). Through this method of relaying information, we can tell if the robot is still functioning properly out there. Furthermore, with a well-designed HTML website, we would be able to control the robot manually should the need arise. We chose WIFI over Bluetooth as WIFI allows for multiple connections and data sharing, has faster connection and better range.

vi. **'Mystery Machine' Design**
Our team decided to adapt the Mystery Machine design from the famous childhood cartoon, "Scooby Doo", to calm the victims and provide a sense of hope, given that in a post-disaster scenario, the victims would be traumatized and in a state of shock. Furthermore, the Mystery Machine is brightly coloured, which increases its visibility.

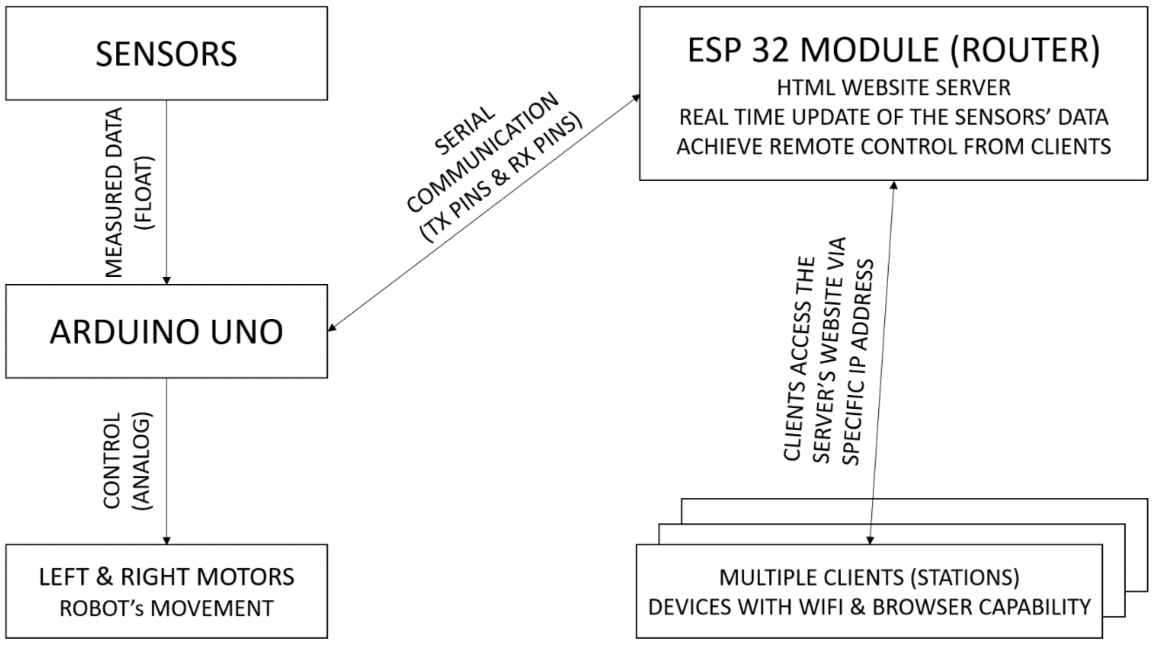
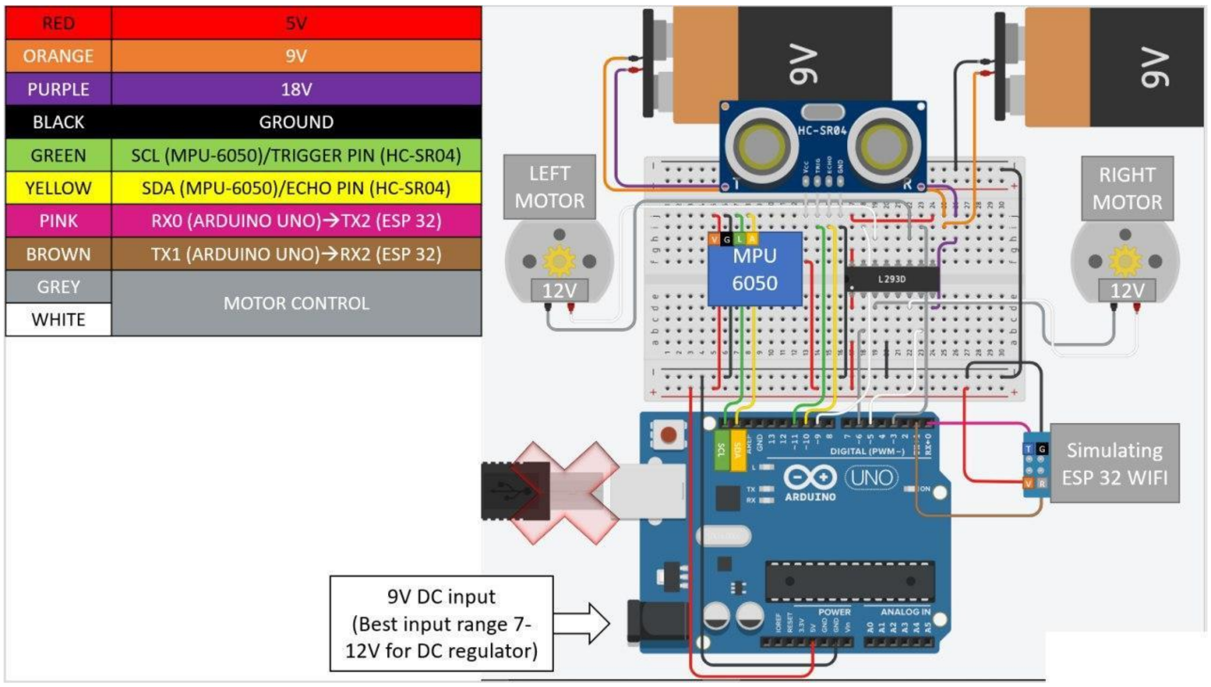

SENSORS
MEASURED DATA (FLOAT)
ARDUINO UNO
CONTROL (ANALOG)
LEFT & RIGHT MOTORS
ROBOT's MOVEMENT
SERIAL COMMUNICATION (TX PINS & RX PINS)
ESP 32 MODULE (ROUTER)
HTML WEBSITE SERVER
REAL TIME UPDATE OF THE SENSORS' DATA
ACHIEVE REMOTE CONTROL FROM CLIENTS
CLIENTS ACCESS THE SERVER'S WEBSITE VIA SPECIFIC IP ADDRESS
MULTIPLE CLIENTS (STATIONS)
DEVICES WITH WIFI & BROWSER CAPABILITY

Figure 12: ESP32 Connectivity

b. Final configuration

i. Circuit

Figure 13 depicts our final circuit configuration on the breadboard.

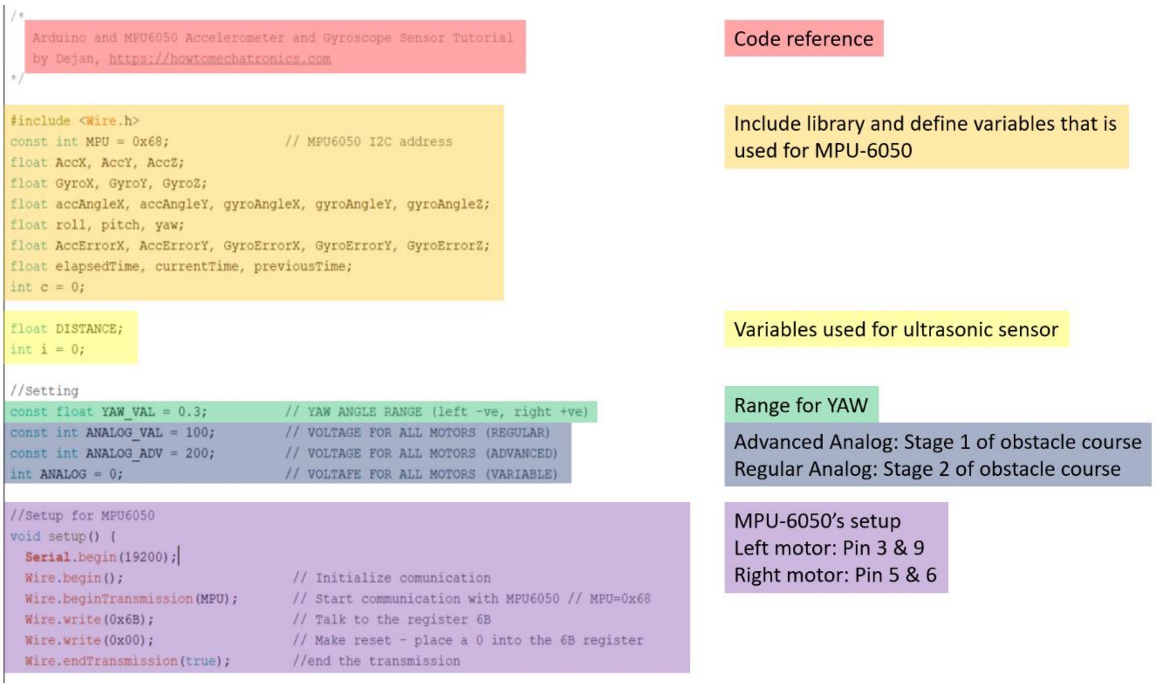


RED	5V
ORANGE	9V
PURPLE	18V
BLACK	GROUND
GREEN	SCL (MPU-6050)/TRIGGER PIN (HC-SR04)
YELLOW	SDA (MPU-6050)/ECHO PIN (HC-SR04)
PINK	RX0 (ARDUINO UNO)→TX2 (ESP 32)
BROWN	TX1 (ARDUINO UNO)→RX2 (ESP 32)
GREY	MOTOR CONTROL
WHITE	

Figure 13: Final circuit configuration

ii. Code

We split up the explanation of the code into two segments, one code is for the vehicle (Figure 14), and the other code is for the Gyroscope (Figure 15). Explanations of code are colour coded for easy reference.



```
/*
  Arduino and MPU6050 Accelerometer and Gyroscope Sensor Tutorial
  by Dejan, https://howtomechatronics.com
*/

#include <Wire.h>
const int MPU = 0x68;               // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

float DISTANCE;
int i = 0;

//Setting
const float YAW_VAL = 0.3;          // YAW ANGLE RANGE (left -ve, right +ve)
const int ANALOG_VAL = 100;         // VOLTAGE FOR ALL MOTORS (REGULAR)
const int ANALOG_ADV = 200;         // VOLTAGE FOR ALL MOTORS (ADVANCED)
int ANALOG = 0;                     // VOLTAFE FOR ALL MOTORS (VARIABLE)

//Setup for MPU6050
void setup() {
  Serial.begin(19200);
  Wire.begin();                      // Initialize comunication
  Wire.beginTransmission(MPU);       // Start communication with MPU6050 // MPU=0x68
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true);        //end the transmission
```

Code reference

Include library and define variables that is used for MPU-6050

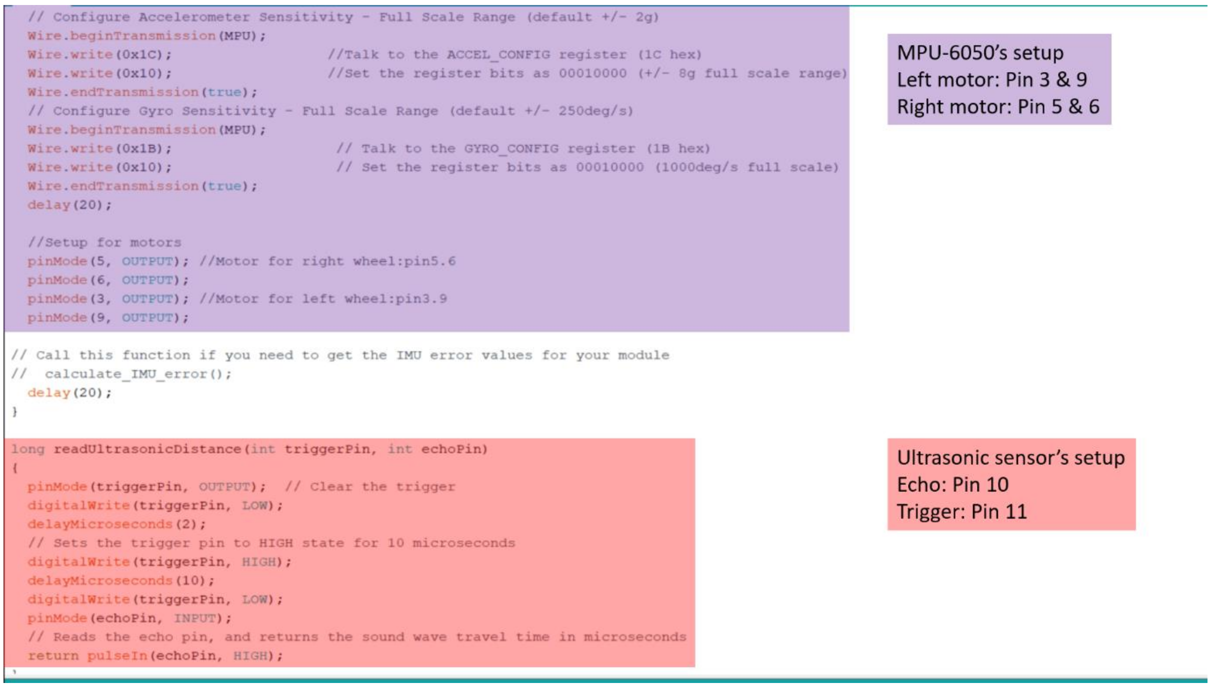
Variables used for ultrasonic sensor

Range for YAW

Advanced Analog: Stage 1 of obstacle course
Regular Analog: Stage 2 of obstacle course

MPU-6050's setup
Left motor: Pin 3 & 9
Right motor: Pin 5 & 6

Figure 14(a): Code for Operation

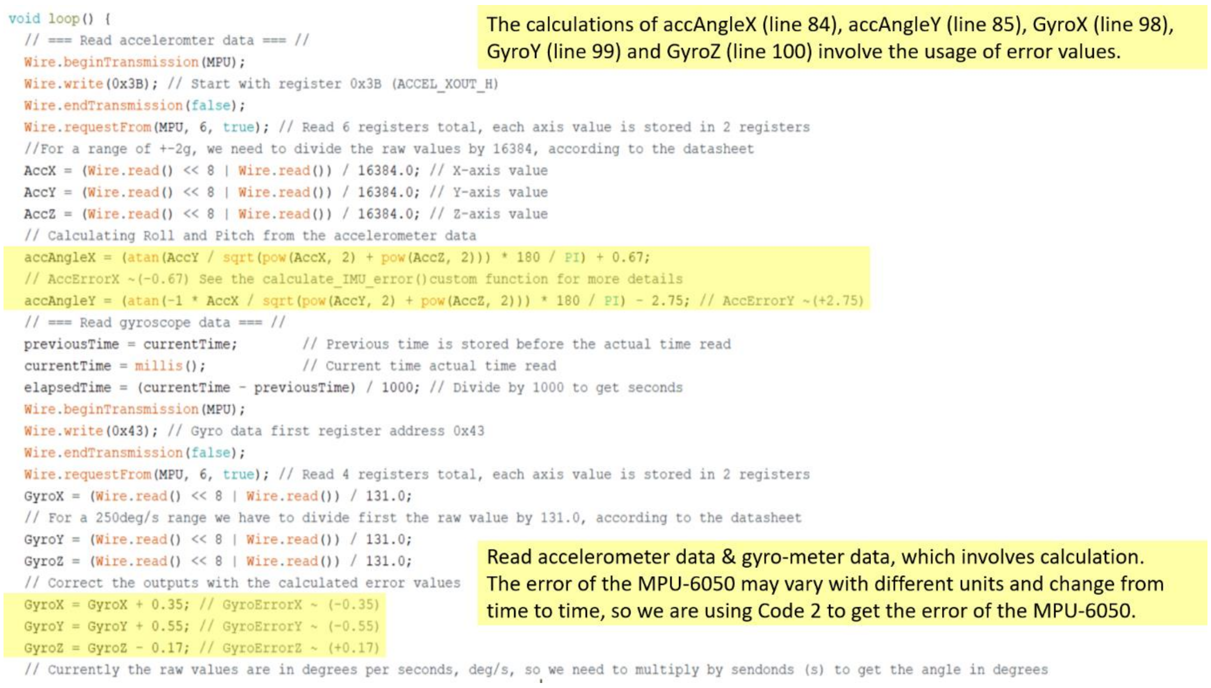


```
  // Configure Accelerometer Sensitivity - Full Scale Range (default +/- 2g)
  Wire.beginTransmission(MPU);
  Wire.write(0x1C);                  //Talk to the ACCEL_CONFIG register (1C hex)
  Wire.write(0x10);                  //Set the register bits as 00010000 (+/- 8g full scale range)
  Wire.endTransmission(true);
  // Configure Gyro Sensitivity - Full Scale Range (default +/- 250deg/s)
  Wire.beginTransmission(MPU);
  Wire.write(0x1B);                   // Talk to the GYRO_CONFIG register (1B hex)
  Wire.write(0x10);                   // Set the register bits as 00010000 (1000deg/s full scale)
  Wire.endTransmission(true);
  delay(20);

  //Setup for motors
  pinMode(5, OUTPUT); //Motor for right wheel:pin5.6
  pinMode(6, OUTPUT);
  pinMode(3, OUTPUT); //Motor for left wheel:pin3.9
  pinMode(9, OUTPUT);

// Call this function if you need to get the IMU error values for your module
//  calculate_IMU_error();
  delay(20);
}

long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microseconds
  return pulseIn(echoPin, HIGH);
```

MPU-6050's setup
Left motor: Pin 3 & 9
Right motor: Pin 5 & 6

Ultrasonic sensor's setup
Echo: Pin 10
Trigger: Pin 11

Figure 14(b): Code for Operation

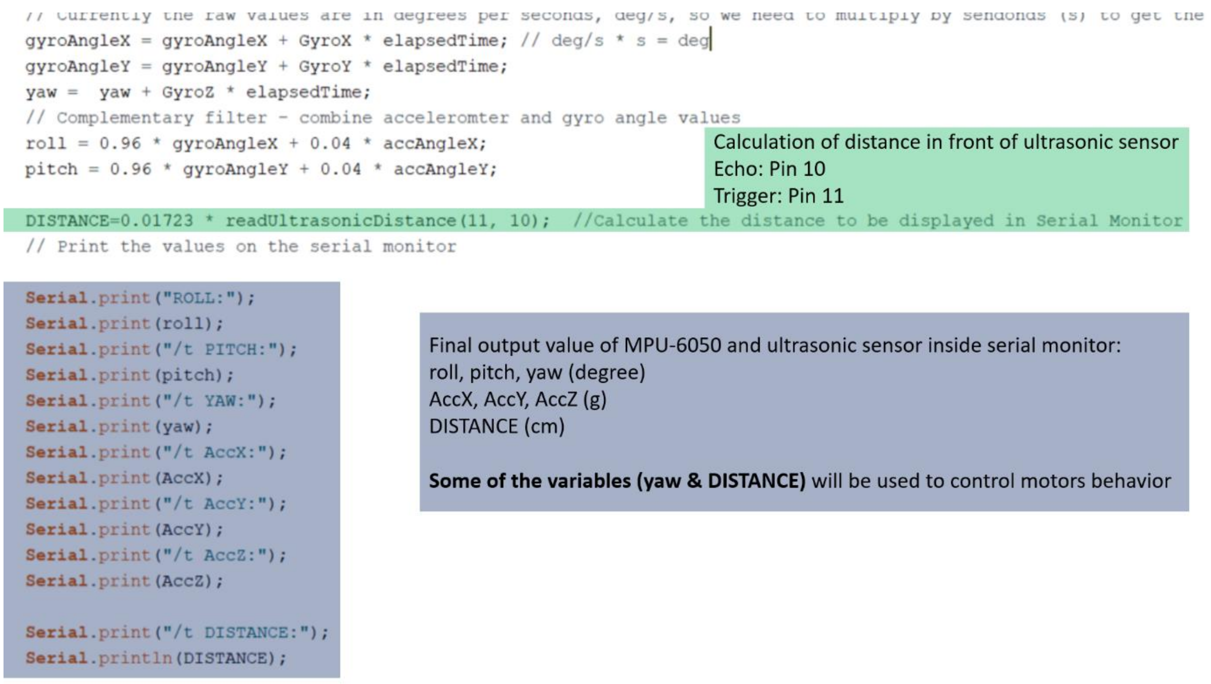


```
void loop() {
  // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
  //For a range of +-2g, we need to divide the raw values by 16384, according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
  // Calculating Roll and Pitch from the accelerometer data
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) + 0.67;
  // AccErrorX ~(-0.67) See the calculate_IMU_error()custom function for more details
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) - 2.75; // AccErrorY ~(+2.75)
  // === Read gyroscope data === //
  previousTime = currentTime;        // Previous time is stored before the actual time read
  currentTime = millis();            // Current time actual time read
  elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to get seconds
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0;
  // For a 250deg/s range we have to divide first the raw value by 131.0, according to the datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + 0.35; // GyroErrorX ~ (-0.35)
  GyroY = GyroY + 0.55; // GyroErrorY ~ (-0.55)
  GyroZ = GyroZ - 0.17; // GyroErrorZ ~ (+0.17)
  // Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by sendonds (s) to get the angle in degrees
```

The calculations of accAngleX (line 84), accAngleY (line 85), GyroX (line 98), GyroY (line 99) and GyroZ (line 100) involve the usage of error values.

Read accelerometer data & gyro-meter data, which involves calculation. The error of the MPU-6050 may vary with different units and change from time to time, so we are using Code 2 to get the error of the MPU-6050.

Figure 14(c): Code for Operation



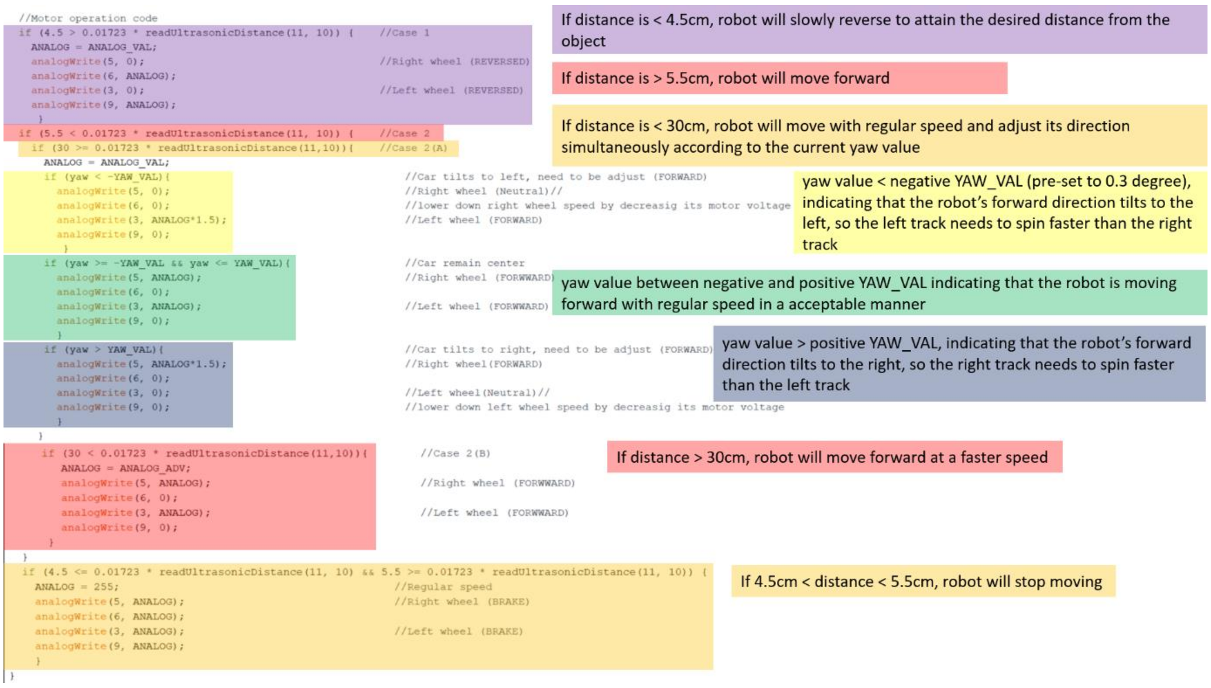
```
// Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by sendonds (s) to get the
gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
gyroAngleY = gyroAngleY + GyroY * elapsedTime;
yaw =  yaw + GyroZ * elapsedTime;
// Complementary filter - combine acceleromter and gyro angle values
roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

DISTANCE=0.01723 * readUltrasonicDistance(11, 10);  //Calculate the distance to be displayed in Serial Monitor
// Print the values on the serial monitor

Serial.print("ROLL:");
Serial.print(roll);
Serial.print("/t PITCH:");
Serial.print(pitch);
Serial.print("/t YAW:");
Serial.print(yaw);
Serial.print("/t AccX:");
Serial.print(AccX);
Serial.print("/t AccY:");
Serial.print(AccY);
Serial.print("/t AccZ:");
Serial.print(AccZ);

Serial.print("/t DISTANCE:");
Serial.println(DISTANCE);
```

Calculation of distance in front of ultrasonic sensor
Echo: Pin 10
Trigger: Pin 11

Final output value of MPU-6050 and ultrasonic sensor inside serial monitor:
roll, pitch, yaw (degree)
AccX, AccY, AccZ (g)
DISTANCE (cm)

Some of the variables (yaw & DISTANCE) will be used to control motors behavior

Figure 14(d): Code for Operation



```
//Motor operation code
if (4.5 > 0.01723 * readUltrasonicDistance(11, 10)) {     //Case 1
  ANALOG = ANALOG_VAL;
  analogWrite(5, 0);                                        //Right wheel (REVERSED)
  analogWrite(6, ANALOG);
  analogWrite(3, 0);                                        //Left wheel (REVERSED)
  analogWrite(9, ANALOG);
  }
if (5.5 < 0.01723 * readUltrasonicDistance(11, 10)) {     //Case 2
  if (30 >= 0.01723 * readUltrasonicDistance(11,10)){     //Case 2(A)
    ANALOG = ANALOG_VAL;
    if (yaw < -YAW_VAL){                                      //Car tilts to left, need to be adjust (FORWARD)
      analogWrite(5, 0);                                      //Right wheel (Neutral)//
      analogWrite(6, 0);                                      //lower down right wheel speed by decreasig its motor voltage
      analogWrite(3, ANALOG*1.5);                             //Left wheel (FORWARD)
      analogWrite(9, 0);
      }
    if (yaw >= -YAW_VAL && yaw <= YAW_VAL){                   //Car remain center
      analogWrite(5, ANALOG);                                 //Right wheel (FORWWARD)
      analogWrite(6, 0);
      analogWrite(3, ANALOG);                                 //Left wheel (FORWWARD)
      analogWrite(9, 0);
      }
    if (yaw > YAW_VAL){                                       //Car tilts to right, need to be adjust (FORWARD)
      analogWrite(5, ANALOG*1.5);                             //Right wheel(FORWARD)
      analogWrite(6, 0);
      analogWrite(3, 0);                                      //Left wheel(Neutral)//
      analogWrite(9, 0);                                      //lower down left wheel speed by decreasig its motor voltage
      }
    }
    if (30 < 0.01723 * readUltrasonicDistance(11,10)){        //Case 2(B)
      ANALOG = ANALOG_ADV;
      analogWrite(5, ANALOG);                                 //Right wheel (FORWWARD)
      analogWrite(6, 0);
      analogWrite(3, ANALOG);                                 //Left wheel (FORWWARD)
      analogWrite(9, 0);
     }
  }
  if (4.5 <= 0.01723 * readUltrasonicDistance(11, 10) && 5.5 >= 0.01723 * readUltrasonicDistance(11, 10)) {
   ANALOG = 255;                                            //Regular speed
   analogWrite(5, ANALOG);                                  //Right wheel (BRAKE)
   analogWrite(6, ANALOG);
   analogWrite(3, ANALOG);                                  //Left wheel (BRAKE)
   analogWrite(9, ANALOG);
   }
}
```

If distance is < 4.5cm, robot will slowly reverse to attain the desired distance from the object

If distance is > 5.5cm, robot will move forward

If distance is < 30cm, robot will move with regular speed and adjust its direction simultaneously according to the current yaw value

yaw value < negative YAW_VAL (pre-set to 0.3 degree), indicating that the robot's forward direction tilts to the left, so the left track needs to spin faster than the right track

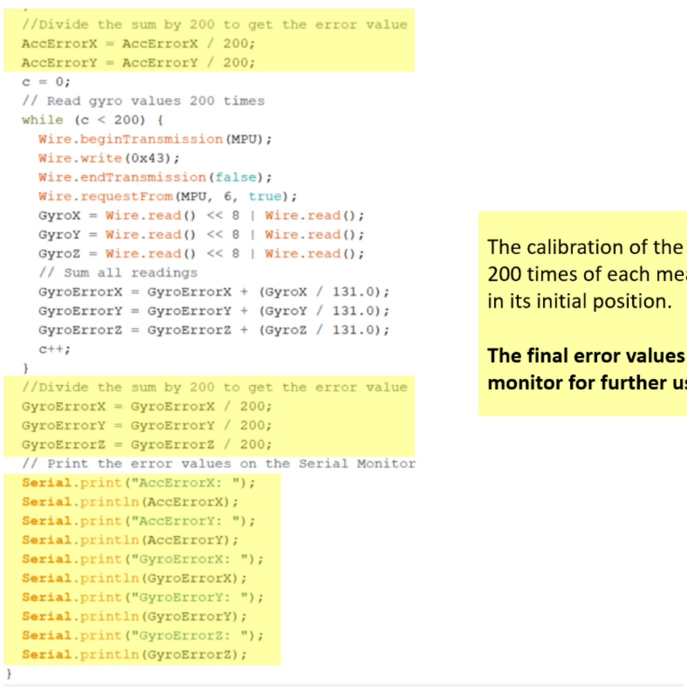
yaw value between negative and positive YAW_VAL indicating that the robot is moving forward with regular speed in a acceptable manner

yaw value > positive YAW_VAL, indicating that the robot's forward direction tilts to the right, so the right track needs to spin faster than the left track

If distance > 30cm, robot will move forward at a faster speed

If 4.5cm < distance < 5.5cm, robot will stop moving

Figure 14(e): Code for Operation



```
  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  // Print the error values on the Serial Monitor
  Serial.print("AccErrorX: ");
  Serial.println(AccErrorX);
  Serial.print("AccErrorY: ");
  Serial.println(AccErrorY);
  Serial.print("GyroErrorX: ");
  Serial.println(GyroErrorX);
  Serial.print("GyroErrorY: ");
  Serial.println(GyroErrorY);
  Serial.print("GyroErrorZ: ");
  Serial.println(GyroErrorZ);
}
```

The calibration of the MPU-6050 is done by getting the average value of 200 times of each measurements (for all parameters) when the robot is in its initial position.

The final error values of all parameters will be displayed at the serial monitor for further use (Code 1).

Figure 15: Code for Gyroscope

c. Conclusion

Throughout the journey of designing our robot, we have gained some valuable insights.

Firstly, we learnt to be meticulous in our work, given that a slight miscalculation can lead to a drastic difference in outcome. Such is so for the case of the code, where the robot was able to stop 50mm before the wall only after we added a ±5mm stopping range to the code.

Secondly, working with a budget constraint, we had to plan our purchase properly and efficiently in order to minimise excessive wastage. This could only be achieved through thorough research and brainstorming, which we benefitted greatly through adopting the 5-stage Design Thinking process. In addition, we learned that there was no perfect design, and the only way was to find the optimal solution for the problem while working under the constraints present.

In the aftermath of a natural disaster, there are usually many victims who require urgent medical attention and essential supplies to sustain. Under harsh circumstances, the movement of emergency response vehicles is impeded, and this necessitates the utility of robots to aid in disaster relief in areas where conventional vehicles are unable to gain access. While our robot was able to fulfil the basic task requirements of manoeuvring through uneven terrains within a stipulated timing of 1 minute as well as having a friendly appearance, there are further modifications that can improve its functionality. Table 3 indicates a list of changes we propose to improve its performance and capabilities.

Modifications	**Purpose**
Floodlight	Illuminate the surroundings, since visibility is low under pile of rubbles
Suspension system	Not only will it reduce wear and tear of the fragile parts of the vehicle, it also addresses the issue of a less bumpy ride while travelling at high speed, which can benefit fractured victims that require immediate attention
Camera with thermal sensor	Allows for scanning and spotting of potentially trapped victims in areas that are inaccessible
2-way sound system	Not only can the speaker broadcast messages to victims who are trapped, 2-way sound system also allows for communication between victims and rescue workers who are based in the command centre
Global Positioning System (GPS)	Having a GPS would allow for accurate authentication of location that requires additional back up
Amphibious capabilities	Natural disasters are not only limited to land and there may be a need to carry out rescue operations at sea, thus it would be beneficial if the vehicle was amphibious
LuminAID / All Terrain Solar Trailer (ATST)	Given the high demand for electrical power due to the many electrical components attached on the vehicle, having a LuminAID or ATST would ensure that the vehicle can function normally during power outages.

Table 3: Proposed improvements to robot