Project Report

Obstacle Avoidance for Ground Robot

-Aanvik Bhatnagar, Divyansh Pandey, Hemang Jain, Mohak Somani

1. Introduction

The project involves the implementation of an intelligent obstruction-avoidance ground robot using the ESP32 micro-controller. The system integrates ultrasonic sensors, motor drivers, edge-avoidance, radar (real-time obstacle visualization through a radar unit) , and a Bluetooth Remote control. The embedded system employs a seamless data flow architecture to facilitate communication and control between various components, enabling both autonomous and manual modes of operation for the ground robot. The Radar Unit and Edge Avoidance enhance the overall functionalities of the ground robot from the perspective of safety and applications

2. Component List

1. ESP 32 (x2) -
2. Chasis with Tyres (x4) and Motors (x2)
3. Motor Driver (x1)
4. HC-SR04 Ultrasonic Sensor (x4)
5. Servo Motor (x1)
6. IR Sensor (x2)
7. LEDs & Breadboards
8. Resistances & Jump Wires

9. Miscellaneous Hardware & Testers

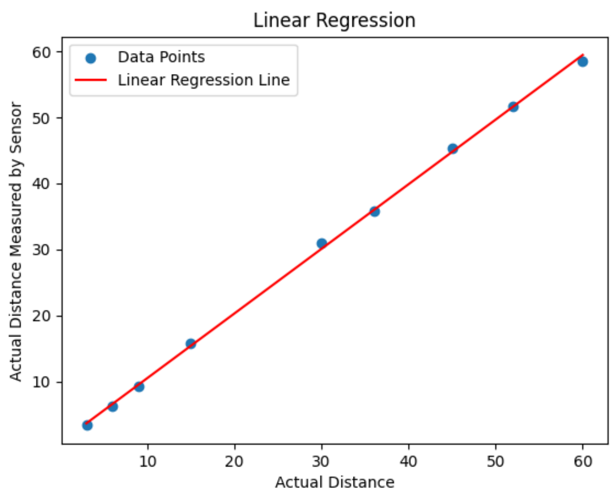
3. Calibration

3.1 Calibration of IR sensor

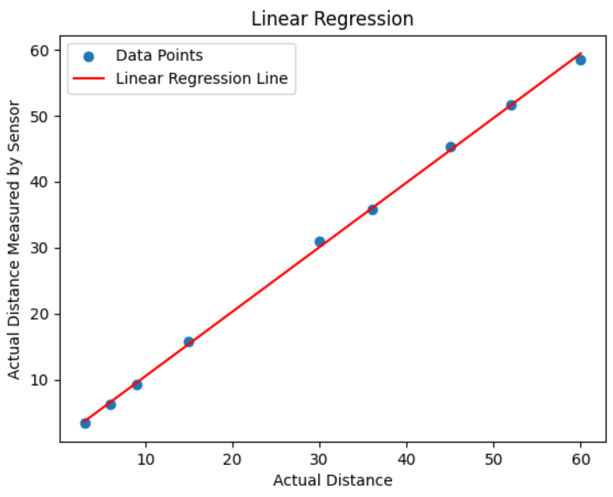
- The IR sensors on the module need to be aligned in the correct direction and the transmitter - receiver pair should be at appropriate distance so as to get the correct reading.
- The IR sensor consists of a screw which is adjusts the threshold for the IR sensor. Thus we need to physically rotate the screw accordingly so as to get the desired threshold for the IR sensor. The height of the Robot body from the ground was about 5.1cm and the IR was calibrate for a height of 5.5 cm (So that the robot does not avoid small pits or craters and only avoided larger drops in depth)
- Finally, 2 IR sensors are used and we need to calibrate them as a unit so as to ensure that both the sensors work independently and according to the desired threshold to raise an exception and ask the ground robot to move backward
- According to the datasheet of the IR sensors
    1. Wavelength $\Rightarrow$ 700nm to 1mm
    2. Light Emitting angle $\Rightarrow$ 20 to 60 degrees
    3. I/O pins $\Rightarrow$ 5 V and 3.3 V compatible
    4. Range $\Rightarrow$ upto 20 cm
    5. Current Supply $\Rightarrow$ Max 20mA

3.2 Calibration of Ultrasonic Sensor

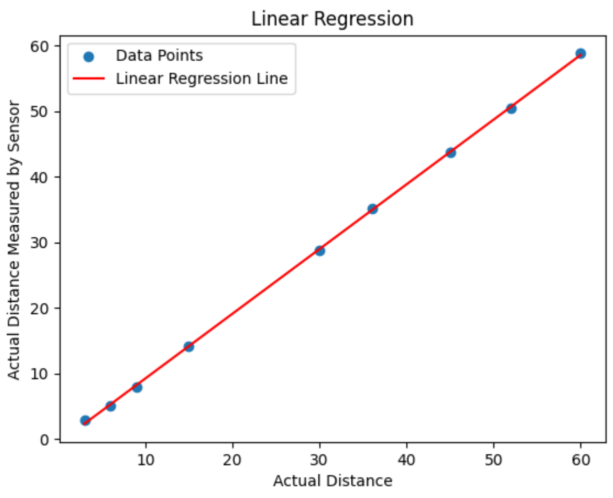
- First get a map of the actual distances of the obstacle and that recorded by the ultrasonic sensor in a table (Set the obstacle at some predetermined intervals over the range of the sensor)
- We then implement a Linear Regression code to get the best fit line so as to get a linear function of actual distance (y) in terms of measured distance (x)
- The code provides a plot and outputs the equation in terms of y and x. The obtained plots for the 4 Ultrasonic sensors is given below:



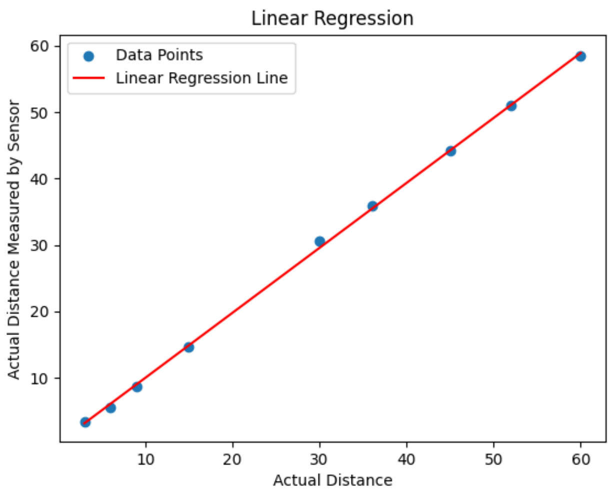
$$US_1 : y = 0.98x + 0.40$$



$$US_2 : y = 0.98x + 0.73$$



$$US_3 : y = 0.99x - 0.65$$



$$US_4 : y = 0.98x + 0.23$$

- According to the datasheet of the ultrasonic sensors:
  1. Max Range ⇒ about 4 m
  2. Min Range ⇒ 2 cm
  3. Working Frequency ⇒ 40 Hz
  4. Working Current ⇒ 40mA
  5. Measuring Angle ⇒ 15 degrees

3.3 Calibration of Motor Driver

- The motor driver draws current from an external sources and using the 5V outlet , the ESP is powered however to control the voltage of the Motor driver it also consist off a voltage regulator. This allows the user to control the input Voltage in the motor driver irrespective of the power supply
- ESP being powered by the outlet of the Motor driver , the current may not lead to an overload condition and thus we need to connect a resistor in parallel so as to limit the current in the ESP
- According to the datasheet of the motor driver:
  - Motor Supply Voltage (Maximum) ⇒ 46 V
  - Motor Supply Current (Maximum) ⇒ 2 A
  - Logic Voltage ⇒ 5 V
  - Driver Voltage ⇒ 5 V to 35 V
  - Driver Current ⇒ 2 A
  - Logical Current ⇒ 0mA to 36mA
  - Maximum Power (W) ⇒ 25 W

3.4 Calibration of Servo Motor

- The servo motor may have an initial offset relative to its body which may lead to incorrect angle of rotation with respect to the body of the ground robot
- Adjusting the initial angle of the servo motor according to the initial offset is required so as to get the desired angle of rotation i.e 15 degrees to 165 degrees.
- According to the datasheet of the Servo motor:
  - Operating Voltage ⇒ +5V
  - Torque ⇒ 2.5kg/cm
  - Operating speed ⇒ 0.1s/60°
  - Rotation ⇒ 0°-180°
  - Weight of motor ⇒ 9gm

4. Implementation

The implementation of the obstacle avoidance robot involved a systematic approach, starting with the assembly of hardware components followed by firmware development. This included integration of sensor readings, data communication, motor control, and implementation of autonomous and manual control algorithms.

The autonomous control algorithm allowed the robot to manoeuvre around obstacles by using data from the ultrasonic sensors and IR sensors, and reacting accordingly. The manual control was implemented via a Bluetooth remote control, enabling direct user control over the robot's movement.

4.1 Obstacle Avoidance

We planned to mount an array of Ultrasonic sensors in front of the car to increase the persistent field of view of the car. We initially planned to mount a sensor in each direction of the chassis that is one facing forwards , one backwards , one on each side. This idea was soon discarded keeping in mind the fact that the car is fully capable of rotating about it's axis without any translatory motion. Assuming a near square structure of the frame , we can conclude that the car can rotate in it's own space (does not require additional space to rotate). Therefore , the purpose served by the left,right and backward ultrasonic sensors can be redundant which can be handled by covering the field of view of the car in one particular direction say facing forward.

After deciding the orientation of the sensors , we mounted them and tested them again to ensure proper connections to the ESP board. After that we started working with the obstacle avoidance algorithm which works as follows:

- Scan the vicinity using the ultrasonic sensors.
- If the ultrasonic sensor (primary sensor) detects an object it should alert the car so that the car can take necessary decision.
- The secondary sensors act as an indicator which inform the car about the availability of a free path.

Based on observations , we decided to set an initial threshold for the sensor ie 10 cm for the primary sensor and 15 cm for the secondary sensor. With testing and calibration , these thresholds were constantly updated till a particular optimum pair was found.

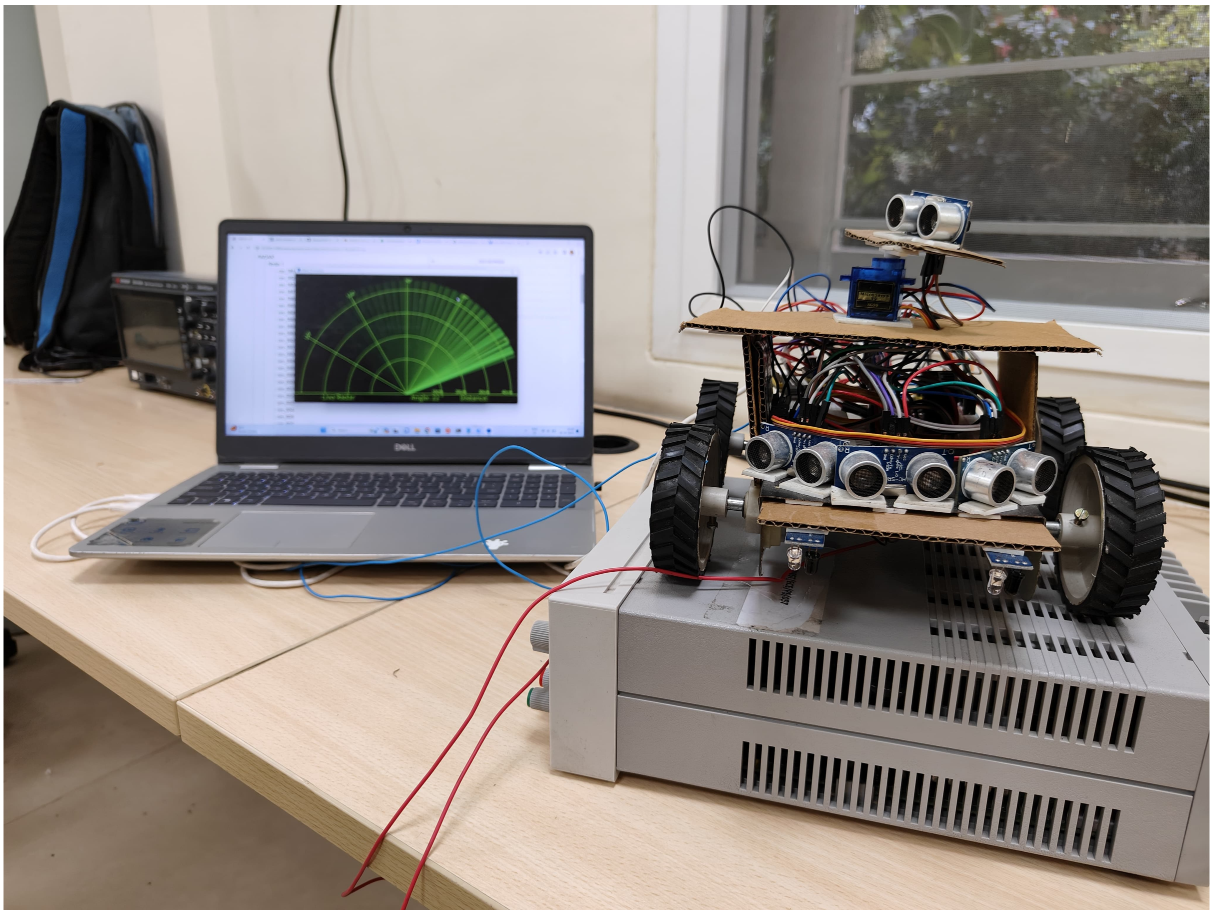
The algorithm implemented had the following major features:

- It checks the distance recorded by each ultrasonic sensor.
- If a sensor records a value less than the threshold distance , the car immediately takes a sharp reverse to immediately get away from the obstacle after which it finds the direction of most free path by comparing distance recorded and then turns in that direction.
- This process keeps repeating infinitely. If no such alert is raised the car moves in forward direction by default.
- The movements are controlled by the abstraction of controlling motors provided by the motor driver.

4.2 Edge avoidance

The IR sensors were calibrated to nearly the height of the car chassis from the ground. This was achieved through methods of trial and error.The recorded height of the car frame was 5.1 cm which was used for calibrating the sensors.

The IR sensors give an alert when one of the sensor goes into HIGH state ie the distance measured is higher than the threshold level which implies there is no potential surface under the car in an oblique direction of the car's line of motion.If an alert is detected , the car takes a sharp reverse and then finds a free path countering the potential edge.



4.3 Radar

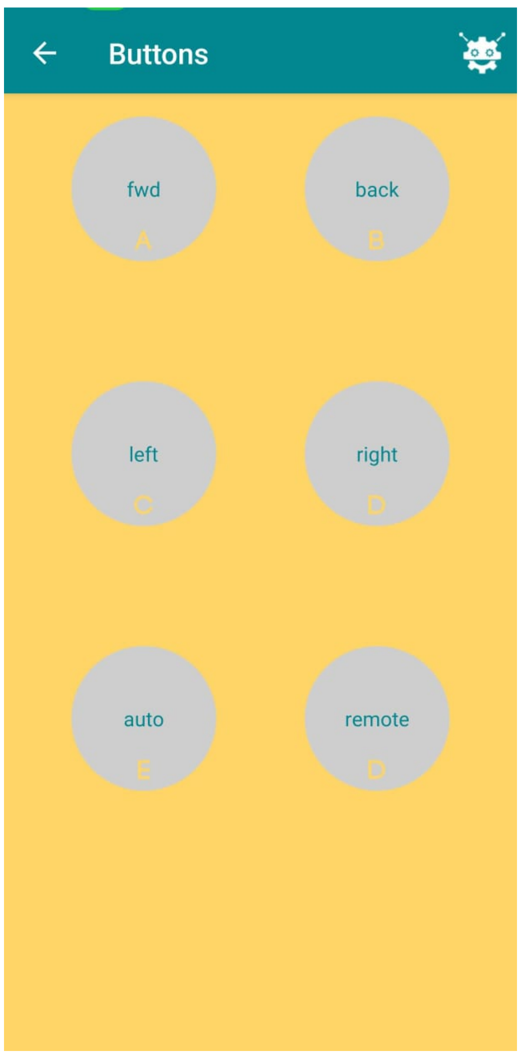
The Radar unit consists of an ultrasonic server mounted on a servo motor which sweeps periodically in an interval of 165 degrees. This functionality helps to visualise the space around the car on the monitor which can be beneficial if the car is not directly observable by the user to monitor the path around it.

The Ultrasonic sensor keeps reading data which is now recorded along with the current angle of the servo motor. This continuous data stream is processed using *processing.io* which takes data from the serial monitor and makes a visual plot of it , exactly how a radar should look like.The speed of the servo motor also depends on how far the object is from the current angle. If the object is close to the sensor the servo starts sweeping faster and vice-versa because the response time of the ultrasonic depends on the distance of the object.

This continous data-stream is published on OM2M server . The data from the OM2M server can also be picked up to plot the same radar as that from the serial monitor. This brings IoT and wireless communication into the picture as well.

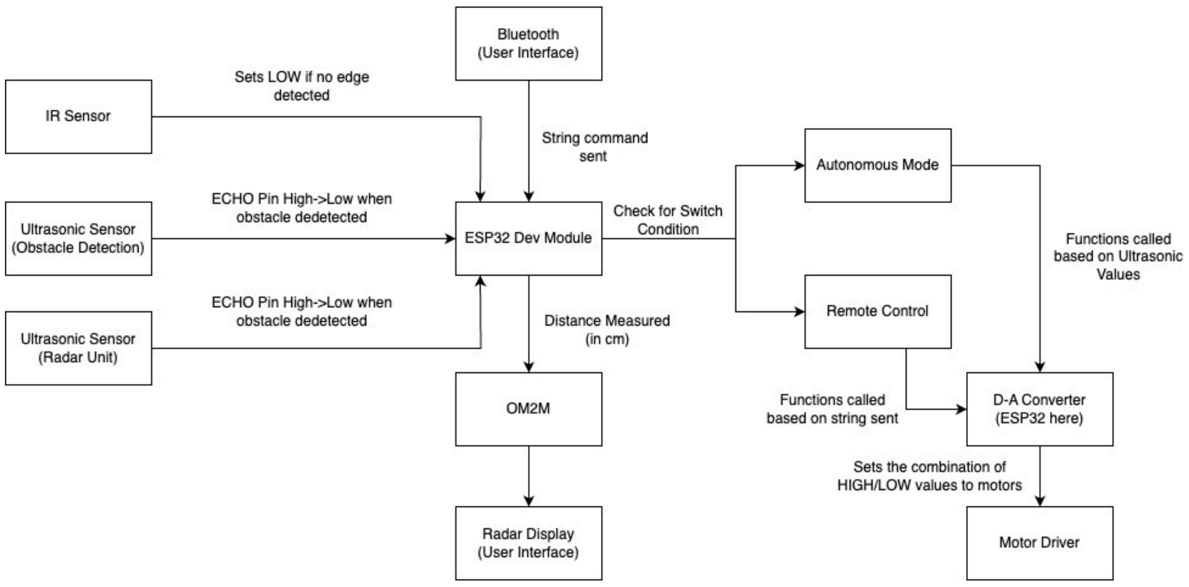
4.4 Remote Control

The Remote Control Mode of the car was facilitated by the Bluetooth Module of the ESP32 and an android Application *insert name.* The interface has four buttons for forward , backward , left and right and two buttons for mode auto and remote. The mode button sets the current mode of the car. In remote control mode , when you press forward button , it sends the forward signal to motors through Bluetooth-ESP communication. Till you don’t press any other button , the car will be in forward state only. There is a small delay in detecting a button press to ensure no conflicts in commands.

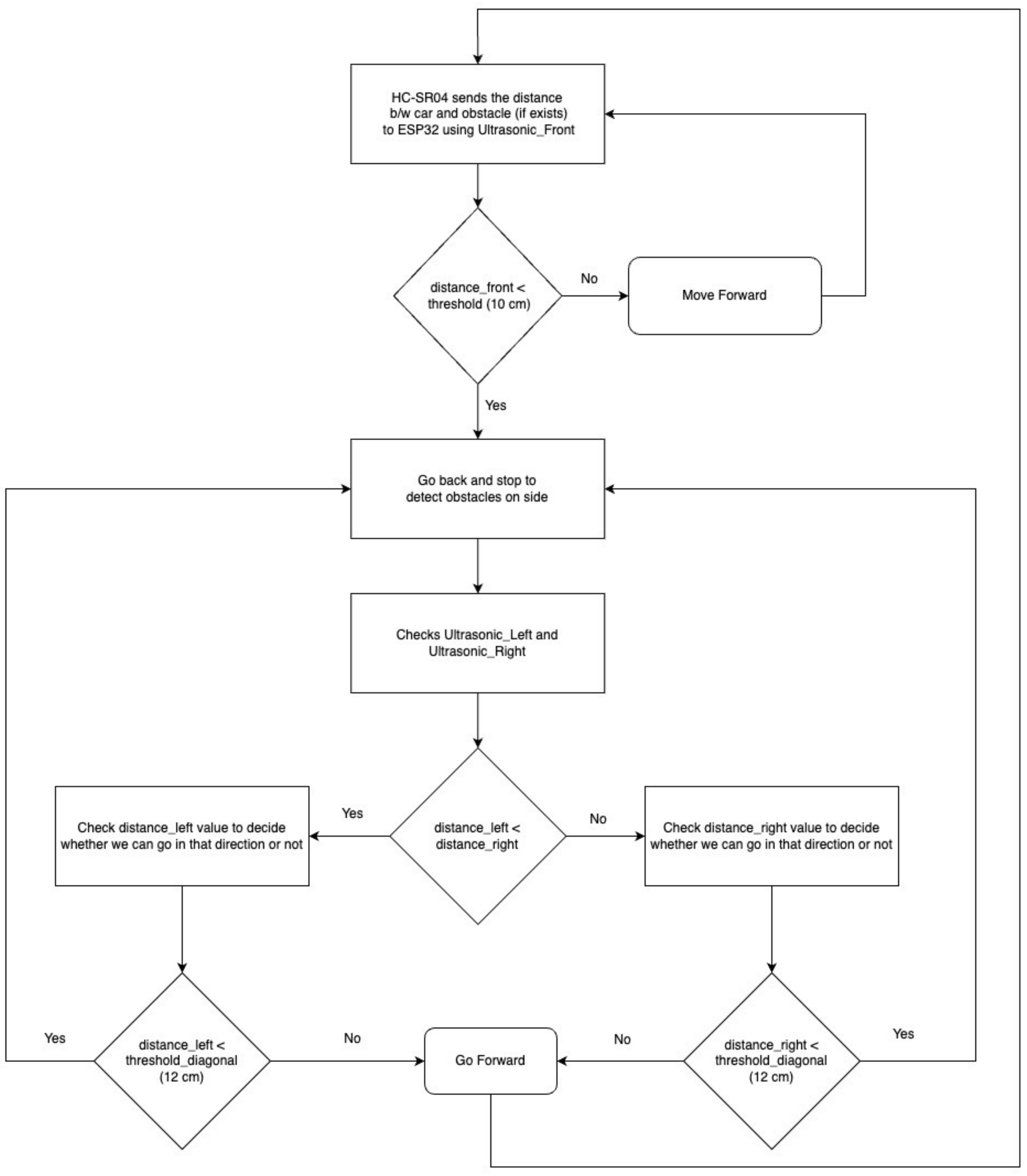


5. Flowchart

5.1 Logical Flowchart of Model


Bluetooth (User Interface)
IR Sensor
Sets LOW if no edge detected
String command sent
Ultrasonic Sensor (Obstacle Detection)
ECHO Pin High->Low when obstacle dedetected
ESP32 Dev Module
Check for Switch Condition
Autonomous Mode
Remote Control
Functions called based on Ultrasonic Values
Ultrasonic Sensor (Radar Unit)
ECHO Pin High->Low when obstacle dedetected
Distance Measured (in cm)
OM2M
Functions called based on string sent
D-A Converter (ESP32 here)
Sets the combination of HIGH/LOW values to motors
Radar Display (User Interface)
Motor Driver

5.2 Logical Flow of Code



HC-SR04 sends the distance b/w car and obstacle (if exists) to ESP32 using Ultrasonic_Front
distance_front < threshold (10 cm)
No
Move Forward
Yes
Go back and stop to detect obstacles on side
Checks Ultrasonic_Left and Ultrasonic_Right
distance_left < distance_right
Yes
No
Check distance_left value to decide whether we can go in that direction or not
Check distance_right value to decide whether we can go in that direction or not
distance_left < threshold_diagonal (12 cm)
distance_right < threshold_diagonal (12 cm)
Yes
No
Go Forward
No
Yes

6. Error Analysis

6.1 Excess Current to ESP

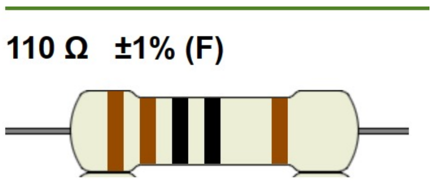
- When the 5 V outlet in the motor driver is used to power the ESP , the ESP is supplied with an excess current which is around $0.2A$ however the current we require is around the range $(0.2A, 0.4A)$. This leads to an overload and may lead to fatal errors in the ESP and the corresponding circuit and sensors connected to the ESP
- As a solution to this , we connect a resistor in parallel to the outlet of 5V and let the current flow to the ground . This limits the current flowing in the ESP. The calculations are shown below:

  Sample resistor value (in lab) = $110 ohms$

  Required current through ESP32 = $(0.02, 0.04)A$
  Actual current flow through ESP32 = $0.2A$

$$V = I.R$$
$$\frac{1}{I_{total}} = \frac{1}{I_1} + \frac{1}{I_2}$$
$$R = \frac{V}{I_{resistor}} = \frac{V}{I_{req}} - \frac{V}{I_{obs}}$$

Required Resistor value in range = $(108.5, 111.5) ohms$



6.2 Delay Due to OM2M

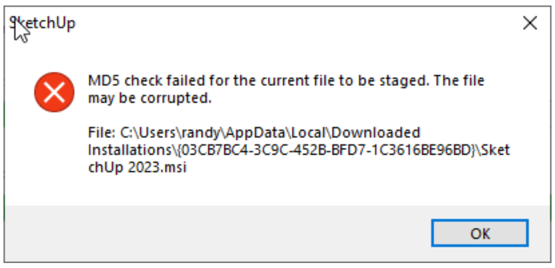
- The Radar Unit was initially implemented along with the OM2M where the data was being sent and received. Simultaneously the data was used to plot the realtime graph using the Processing.io application.
- But Uploading and retrieving data from the OM2M incurs a latency of about 600 ms which causes the Radar Ultrasonic Sensor to wait until the data transmission is completed leading problems in the smoothness of the graph

6.3 Avoiding Infinite Loop

- When the ground robot detects an object , it considers the left and thr right ultrasonic sensor values and determines which direction to rotate accordingly
- However , this may lead to an infinite loop as the robot may rotate in the left and then the right direction by same amounts and continue looping in this procedure.
- To avoid such an error , the amount by which the ground robot moves to left and right should be different as on infinite loop this will ensure that the robot always has some displacement angle after every cycle of left and right rotation.
- The delay values for the left and right are set to 300 and 350 milliseconds accordingly.

6.4 MD5 File Error

- After esptool.py wrote the new binary to the flash it read back the contents and it didn't match. File corruption, or write protection prevented the flash from being updated. This indicates that the flash isn't properly responding to commands at all (FlashID and SizeID should be non-zero values.)
- Also, while drawing current for ESP from Motor Driver , such an error occured as powering the ESP directly may lead to internal damages


SketchUp

MD5 check failed for the current file to be staged. The file may be corrupted.

File: C:\Users\randy\AppData\Local\Downloaded Installations\{03CB7BC4-3C9C-452B-BFD7-1C3616BE96BD}\SketchUp 2023.msi

6.5 Adjusting COM

⇒ One side of the robot car was experiencing insufficient contact with the ground due to an uneven weight distribution, leading to instability during movement

Identify Unstable Side: Determine the side of the car where instability is observed.

Calculate Torque: Assess the torque (τ) causing the instability due to uneven weight distribution.

Determine Counter Torque: Calculate the counter torque (τ_counter) required to stabilize the car.

Add a small weight (ΔW) to the lighter side.

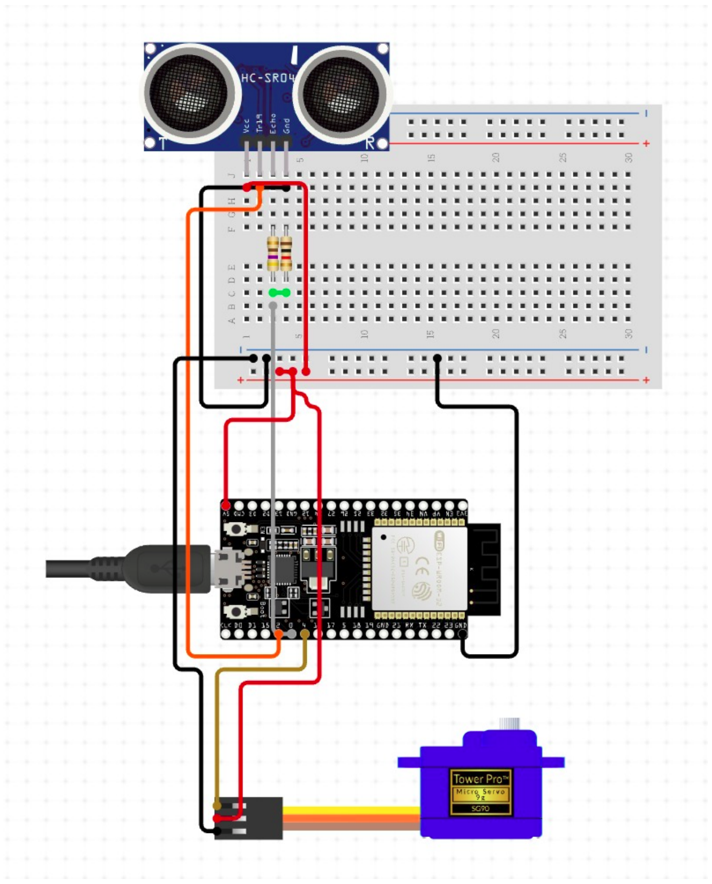
Testing: Assess the stability of the robot car after each weight addition.

Results:The addition of a small weight on the lighter side successfully improved the stability of the robot car. The weight was carefully selected to provide the necessary counter torque, ensuring better contact of all wheels with the ground

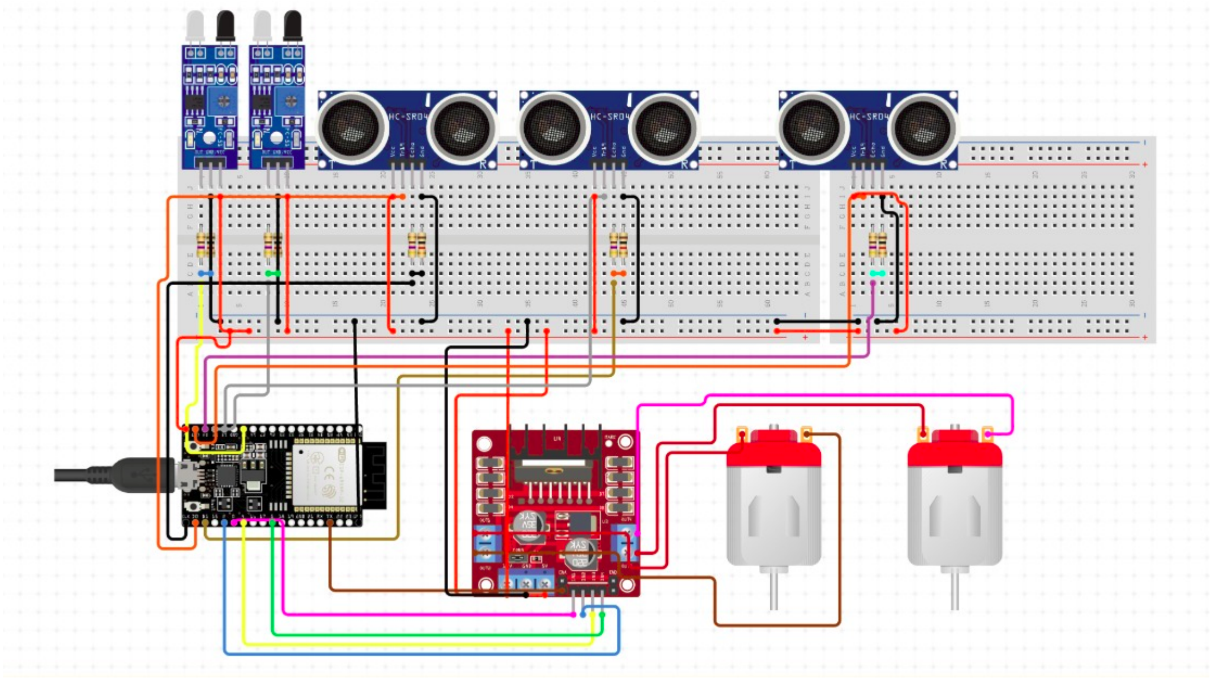
To rectify the instability, a small weight (ΔW) was added to the lighter side. The goal was to introduce a counter torque that would balance the torque causing the instability.

7. Circuit Diagram

7.1 Radar Unit



7.2 Motor Driver and Sensors



8. Appendix

8.1 Motor Driver and Integrated sensors

```
const int trigPin_1 = 13;
const int echoPin_1 = 12 ;
const int trigPin_2 = 32;
const int echoPin_2 = 33;
const int trigPin_3 = 25;
const int echoPin_3 = 35;
const float threshold = 22;
const float threshold_diagonal=18;
const int irSensorPin1 = 2;
const int irSensorPin2 = 21;
const int inp1=27;
const int inp2=26;
const int inp3=18;
const int inp4=19;
const int en1=14;
const int en2=15;
int turnflag=0;
void goback(){

  digitalWrite(inp1,HIGH);
  digitalWrite(inp2,LOW);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(500);
}
void goback_RR(){
  analogWrite(en1,255);
  analogWrite(en2,255);
  digitalWrite(inp1,HIGH);
  digitalWrite(inp2,LOW);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(1000);
}
void goback_r(){
  digitalWrite(inp1,HIGH);
  digitalWrite(inp2,LOW);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(500);
}

void gorev(){

  digitalWrite(inp1,LOW);
  digitalWrite(inp2,HIGH);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(1000);

}

void goleft(){
  digitalWrite(inp1,HIGH);
  digitalWrite(inp2,LOW);
  digitalWrite(inp4,HIGH);
  digitalWrite(inp3,LOW);
  delay(350);
}

void goleft_RR(){
  digitalWrite(inp1,HIGH);
  digitalWrite(inp2,LOW);
  digitalWrite(inp4,HIGH);
  digitalWrite(inp3,LOW);
  delay(350);
}

void goright_RR(){
  digitalWrite(inp1,LOW);
  digitalWrite(inp2,HIGH);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(300);
}

void goright(){
  digitalWrite(inp1,LOW);
  digitalWrite(inp2,HIGH);
  digitalWrite(inp4,LOW);
  digitalWrite(inp3,HIGH);
  delay(390);
}

void gofwd(){
  digitalWrite(inp1,LOW);
  digitalWrite(inp2,HIGH);
  digitalWrite(inp4,HIGH);
  digitalWrite(inp3,LOW);
  delay(15);
}

void gofwdsmall(){

  digitalWrite(inp1,LOW);
  digitalWrite(inp2,HIGH);
  digitalWrite(inp4,HIGH);
  digitalWrite(inp3,LOW);
  delay(100);
}

void setup() {

  Serial.begin(115200);
  pinMode(trigPin_1, OUTPUT);
  pinMode(echoPin_1, INPUT);
  pinMode(trigPin_2, OUTPUT);
  pinMode(echoPin_2, INPUT);
  pinMode(trigPin_3, OUTPUT);
  pinMode(echoPin_3, INPUT);
  pinMode(irSensorPin1, INPUT);
  pinMode(irSensorPin2, INPUT);
  pinMode(inp1,OUTPUT);
  pinMode(inp2,OUTPUT);
  pinMode(inp3,OUTPUT);
  pinMode(inp4,OUTPUT);
  pinMode(en1,OUTPUT);
  pinMode(en2,OUTPUT);

}

void loop() {
  analogWrite(en1,255);
  analogWrite(en2,255);
  // gofwd();
  int flag_ir = 0;
  int flag_us = 0;

  int irSensorState1 = digitalRead(irSensorPin1);
  int irSensorState2 = digitalRead(irSensorPin2);

  long duration;
  digitalWrite(trigPin_1, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin_1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_1, LOW);

  duration = pulseIn(echoPin_1, HIGH);
  float distance1 = duration * 0.034 / 2;
  // Serial.println(distance1);
  digitalWrite(trigPin_2, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin_2, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_2, LOW);

  duration = pulseIn(echoPin_2, HIGH);
  float distance2 = duration * 0.034 / 2;
  // Serial.println(distance2);
  // float distance2 = 15;

  digitalWrite(trigPin_3, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin_3, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_3, LOW);

  duration = pulseIn(echoPin_3, HIGH);

  float distance3 = duration * 0.034 / 2;
  // Serial.println(distance3);// float distance3=15;

  if (irSensorState1 == HIGH || irSensorState2 == HIGH) {
    Serial.print(irSensorState1);
    Serial.println(irSensorState2);
    flag_ir = 1;
    Serial.println("IR alert!");
  }
  // flag_ir=0;
  flag_us=US_combined(distance1,distance2,distance3);
  // Serial.println(irSensorState1,irSensorState2,distance1,distance2,distance3);
  // Serial.print(distance1);
  // Serial.print(distance2);
  // Serial.println(distance3);
  if (flag_ir == 1 || flag_us == 1) {
    // Serial.println("Topale alert!");
    // Serial.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");

    //Step 1 simon go back
    Serial.print("goback");
    // goback();

    //Step 2 check if IR alert
    //If ir alert simon turn back

    if(flag_ir == 1){
      Serial.println("gorev");
      goback_RR();
      if(turnflag==0){
        goleft_RR();
        turnflag=1;
      }
      else{
        goright_RR();
        turnflag=0;
      }
      // goleft_RR();
    }

    //else if US_1 has an alert , check dist2 and dist 3
    // turn to higher dist side
    //small move fwd
    else{
          goback();
      if(distance1<threshold){

          if(distance2>distance3){
            Serial.println("Goleft");
            goleft();
          }
          else{
            Serial.println("Goright");
            goright();
          }
          Serial.println("Gofwdsmall");
          gofwdsmall();
      }
      else if(distance2<threshold_diagonal){
        Serial.println("goleft");
        // goback_r();
        goleft();
      }
      else if(distance3<threshold_diagonal){
        Serial.println("goright");
        // goback_r();
        goright();
      }
}
     }
     else{
      // Serial.println("GOfwd");
      gofwd();

     }

  delay(10);
}


int US_combined(int distance1 , int distance2 , int distance3)
{
  int flag_ret=0;

  if (distance1 < threshold) {
    flag_ret = 1;
    // Serial.println(distance1);
    // Serial.println("US_1 Alert!");
  }
  if (distance2 < threshold_diagonal) {
    flag_ret = 1;
    // Serial.println("US_2 Alert!");
  }

  if (distance3 < threshold_diagonal) {
    flag_ret = 1;
    // Serial.println("US_3 Alert!");
  }

  return flag_ret;

}
```

8.2 Radar Unit Arduino code

```
#include <ESP32Servo.h>
#include <WiFi.h>
#include <HTTPClient.h>
Servo myServo; // Create a servo object
int pos = 15;  // Current servo position
int servoDirection = 1; // 1 for clockwise, -1 for counterclockwise
HTTPClient http;
const int trigPin = 5;
const int echoPin = 18;
const int servoPin = 15; // Define the GPIO pin connected to the servo
const char* ssid = "Divyansh";
const char* password = "dpkahotspot";
#define CSE_IP "192.168.154.90"
#define CSE_PORT 5089
#define OM2M_ORGIN "admin:admin"
#define OM2M_AE "RADAR"
#define OM2M_DATA_CONT "Node-1"
#define OM2M_MN "/~/in-cse/in-name/"
long long int count=0;
long duration;
int distance;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
  myServo.attach(servoPin); // Attach the servo to the specified pin

  WiFi.begin(ssid, password);
```

```
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }

    Serial.println("Connected to WiFi");


}

void loop() {
  // Read the distance asynchronously without blocking
  readDistanceAsync();

  myServo.write(pos);
  delay(10); // Adjust this delay for the desired speed of rotation

  // Increment or decrement the servo position based on direction
  pos += servoDirection;



  Serial.print(pos);
  Serial.print(",");

  // Send data to OM2M
    // HTTPClient http;
    // http.begin(String(om2mServer));
    // http.addHeader("Content-Type", "application/json;ty=4");
    // http.addHeader("X-M2M-Origin", "admin:admin");
    // http.addHeader("Content-Length", "100");
    // String dataPayload = "{\"m2m:cin\":{\"con\":\"" + String(pos) + "\"}}";


    // int httpResponseCode = http.POST(dataPayload);
    // if (httpResponseCode > 0) {
    //     Serial.println("Data sent successfully to OM2M");
    // } else {
    //     Serial.println(http.errorToString(httpResponseCode).c_str());
    // }

    // http.end();

    om2mPublish(distance,pos);




  // Check if the servo has reached the limit
  if (pos >= 165 || pos <= 15) {
    // Change direction when the servo reaches the limit
    servoDirection *= -1;
  }
}

void om2mPublish(int fillpercent, int aqivalue) {
  unsigned long int epochTime = count++;
  String data = "[" + String(epochTime) + ", " + String(fillpercent) + " , " + String(aqivalue) + "]";

  String server = "http://" + String() + CSE_IP + ":" + String() + CSE_PORT + String() + OM2M_MN;

  // Serial.println(data);
  http.begin(server + String() + OM2M_AE + "/" + OM2M_DATA_CONT + "/");

  http.addHeader("X-M2M-Origin", OM2M_ORGIN);
  http.addHeader("Content-Type", "application/json;ty=4");
  http.addHeader("Content-Length", "100");

  String label = "Bin-1";

  String req_data = String() + "{\"m2m:cin\": {"
                  + "\"con\": \"" + data + "\","
                  + "\"rn\": \"" + "cin_" + String(epochTime) + "\","
                  + "\"lbl\": \"" + label + "\","
                  + "\"cnf\": \"text\""
                  + "}}";
  int code = http.POST(req_data);
  // Serial.println(http.errorToString(code).c_str());
  http.end();
}

void readDistanceAsync() {
  static unsigned long previousMillis = 0;
  const long interval = 100; // Adjust the interval as needed

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    // Trigger the ultrasonic sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Read the duration of the pulse
    duration = pulseIn(echoPin, HIGH);

    // Calculate the distance
    distance = duration * 0.034 / 2;

    // Print the distance
  }
    Serial.print(distance);
    Serial.print(".");
}
```

8.3 Linear Regression code

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

data = [(1.0, 2.1), (2.0, 3.9), (3.0, 6.0), (4.0, 8.1), (5.0, 9.8)]
# Random Data

x_data = np.array([x for x, y in data]).reshape(-1, 1)
y_data = np.array([y for x, y in data])

model = LinearRegression()
model.fit(x_data, y_data)

y_pred = model.predict(x_data)

plt.scatter(x_data, y_data, label="Data Points")

plt.plot(x_data, y_pred, color='red', label="Linear Regression Line")

plt.xlabel("Actual Distance")
plt.ylabel("Actual Distance Measured by Sensor")
plt.legend()

plt.title("Linear Regression")
plt.show()

print(f"Linear Relation: y = {model.coef_[0]:.2f}x + {model.intercept_:.2f}")
```

8.4 Processing code

```
import processing.serial.*;
import java.awt.event.KeyEvent;
import java.io.IOException;
Serial myPort;
// by www.andprof.com
// defubes variables
String angle="";
String distance="";
String data="";
String noObject;
float pixsDistance;
int iAngle, iDistance;
int index1=0;
int index2=0;
PFont orcFont;
void setup() {

 size (1200, 700);
 smooth();
 myPort = new Serial(this,"COM3", 115200);
 myPort.bufferUntil('.');
}
void draw() {

  fill(98,245,31);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0,4);
  rect(0, 0, width, height-height*0.065);

  fill(98,245,31); // green color
  // calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}
void serialEvent (Serial myPort) { // starts reading data from the Serial Port
  // reads the data from the Serial Port up to the character '.' and puts it into the String variable "data".
  data = myPort.readStringUntil('.');
  data = data.substring(0,data.length()-1);

  index1 = data.indexOf(",");
  angle= data.substring(0, index1);
  distance= data.substring(index1+1, data.length());

  // converts the String variables into Integer
  iAngle = int(angle);
  iDistance = int(distance);
}
void drawRadar() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  noFill();
  strokeWeight(2);
  stroke(98,245,31);
  // draws the arc lines
  arc(0,0,(width-width*0.0625),(width-width*0.0625),PI,TWO_PI);
  arc(0,0,(width-width*0.27),(width-width*0.27),PI,TWO_PI);
  arc(0,0,(width-width*0.479),(width-width*0.479),PI,TWO_PI);
  arc(0,0,(width-width*0.687),(width-width*0.687),PI,TWO_PI);
  // draws the angle lines
  line(-width/2,0,width/2,0);
  line(0,0,(-width/2)*cos(radians(30)),(-width/2)*sin(radians(30)));
  line(0,0,(-width/2)*cos(radians(60)),(-width/2)*sin(radians(60)));
  line(0,0,(-width/2)*cos(radians(90)),(-width/2)*sin(radians(90)));
  line(0,0,(-width/2)*cos(radians(120)),(-width/2)*sin(radians(120)));
  line(0,0,(-width/2)*cos(radians(150)),(-width/2)*sin(radians(150)));
  line((-width/2)*cos(radians(30)),0,width/2,0);
  popMatrix();
}
void drawObject() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  strokeWeight(9);
  stroke(255,10,10); // red color
  pixsDistance = iDistance*((height-height*0.1666)*0.025); // covers the distance from the sensor from cm to pixels
  // limiting the range to 40 cms
  if(iDistance<40){
    // draws the object according to the angle and the distance
  line(pixsDistance*cos(radians(iAngle)),-pixsDistance*sin(radians(iAngle)),(width-width*0.505)*cos(radians(iAngle)),-(width-width*0.5
  }
  popMatrix();
}
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30,250,60);
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  line(0,0,(height-height*0.12)*cos(radians(iAngle)),-(height-height*0.12)*sin(radians(iAngle))); // draws the line according to the a
  popMatrix();
}
void drawText() { // draws the texts on the screen

  pushMatrix();
  if(iDistance>40) {
  noObject = "Out of Range";
  }
  else {
  noObject = "In Range";
  }
  fill(0,0,0);
  noStroke();
  rect(0, height-height*0.0648, width, height);
```

```
    fill(98,245,31);
    textSize(25);

    text("10cm",width-width*0.3854,height-height*0.0833);
    text("20cm",width-width*0.281,height-height*0.0833);
    text("30cm",width-width*0.177,height-height*0.0833);
    text("40cm",width-width*0.0729,height-height*0.0833);
    textSize(40);
    text("Hani's Experiments", width-width*0.875, height-height*0.0277);
    text("Angle: " + iAngle +" °", width-width*0.48, height-height*0.0277);
    text("Distance: ", width-width*0.26, height-height*0.0277);
    if(iDistance<40) {
    text("        " + iDistance +" cm", width-width*0.225, height-height*0.0277);
    }
    textSize(25);
    fill(98,245,60);
    translate((width-width*0.4994)+width/2*cos(radians(30)),(height-height*0.0907)-width/2*sin(radians(30)));
    rotate(-radians(-60));
    text("30°",0,0);
    resetMatrix();
    translate((width-width*0.503)+width/2*cos(radians(60)),(height-height*0.0888)-width/2*sin(radians(60)));
    rotate(-radians(-30));
    text("60°",0,0);
    resetMatrix();
    translate((width-width*0.507)+width/2*cos(radians(90)),(height-height*0.0833)-width/2*sin(radians(90)));
    rotate(radians(0));
    text("90°",0,0);
    resetMatrix();
    translate(width-width*0.513+width/2*cos(radians(120)),(height-height*0.07129)-width/2*sin(radians(120)));
    rotate(radians(-30));
    text("120°",0,0);
    resetMatrix();
    translate((width-width*0.5104)+width/2*cos(radians(150)),(height-height*0.0574)-width/2*sin(radians(150)));
    rotate(radians(-60));
    text("150°",0,0);
    popMatrix();
  }
```

9. References

- Obstacle avoidance reference
- Ultrasonic sensor datasheet
- IR datasheet
- Motor driver
- Processing.io
- ESP32 pinout