TECHNOLOGICAL UNIVERSITY OF THE PHILIPPINES
Ayala Boulevard, Ermita, Manila
CIT-ELECTRONICS DEPARTMENT

CPET11L-M – Microprocessor and Microcontroller Systems, Lab
1st Semester, SY 2-24-2025

"ATOM THE RECYCABOT"

A SUMO-BOT PROJECT

MEMBERS:

ARENAS, JOSEPH C,

ESTRADA, ADRIENE CYRUZ

GUTIERREZ, GEO KENTZER

PACIS, LIAN GIL B.

RECAÑA, JORDAN

NOVEMBER 2025

I. OBJECTIVES

- To design and build a single ESP32 microcontroller system that successfully integrates and calibrates four distinct detection capabilities such as earthquake (vibration), fire/smoke, hazardous gas, and motion.
- To develop a real-time hazard monitoring program that continuously reads sensor data, filters noise and interprets each sensor's threshold for accurate hazard detection.
- To implement Wi-Fi communication using the ESP32 to transmit alert signals immediately when any hazard is detected.
- To create a smartphone notification system (using Telegram Bot API) that provides instant alerts with hazard type, time, and severity level.
- To enhance skills in C++ programming for dual-core microcontrollers, mechanical assembly, and real-time logic.

II. EQUIPMENT AND MATERIALS

HARDWARE

- Cellphone
- ESP32
- ESP32 Shield Board
- Jumper Wires (Male to Male and Male to Female)
- Junction Box
- MQ137 – Ammonia Gas Sensor
- DHT11 Temperature and Humidity Sensor
- PIR Motion Sensor
- LED (Onboard)
- MPU6050 Accelerometer & Gyroscope Sensor
- I2C Liquid Crystal Display
- Laptop
- Screws and Nuts
- Philip's Screw Drivers

SOFTWARE

- Arduino IDE
- Telegram Bot API

III. DIAGRAM

A. Block Diagram

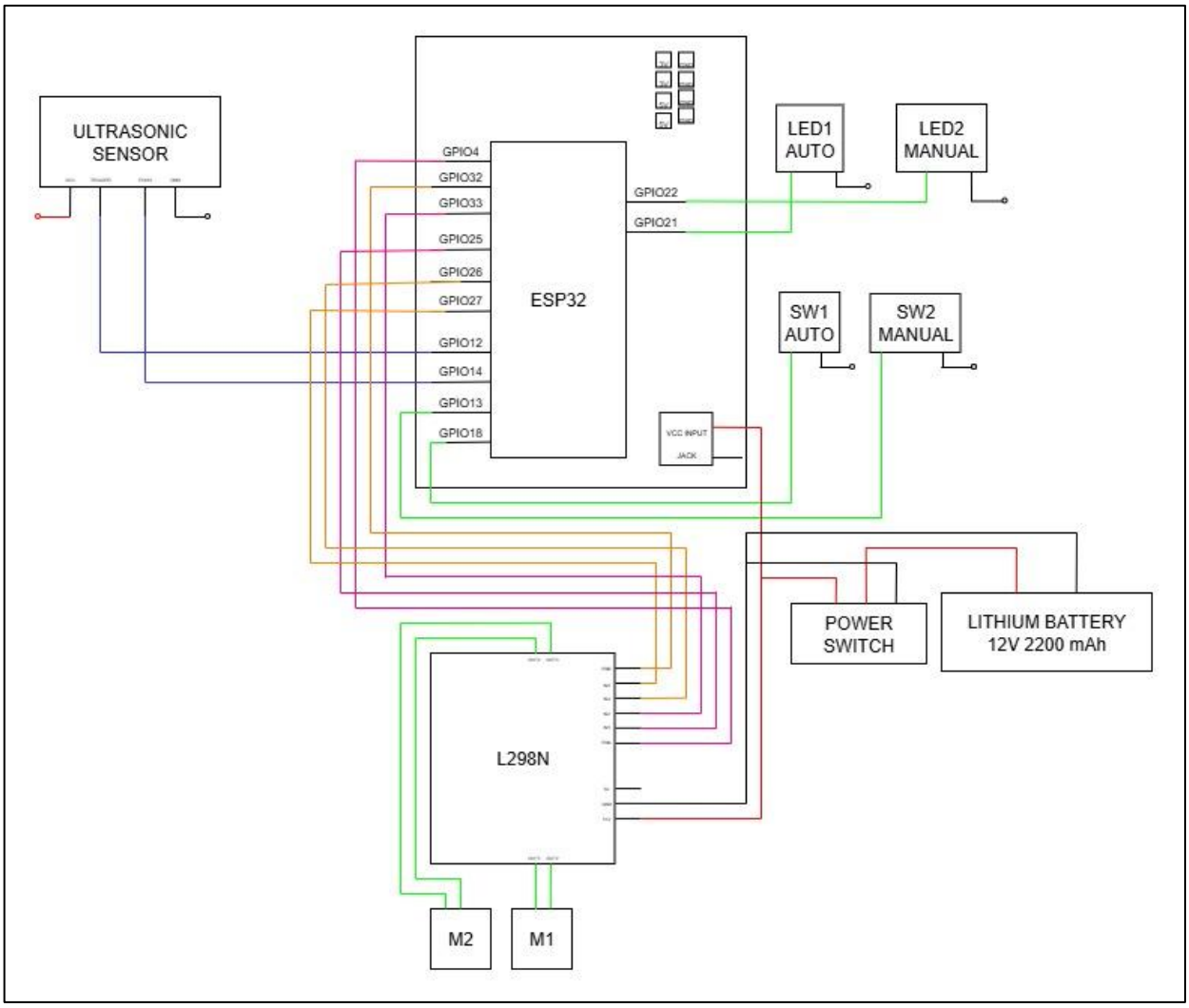

FIGURE 1

B. Pictorial Diagram

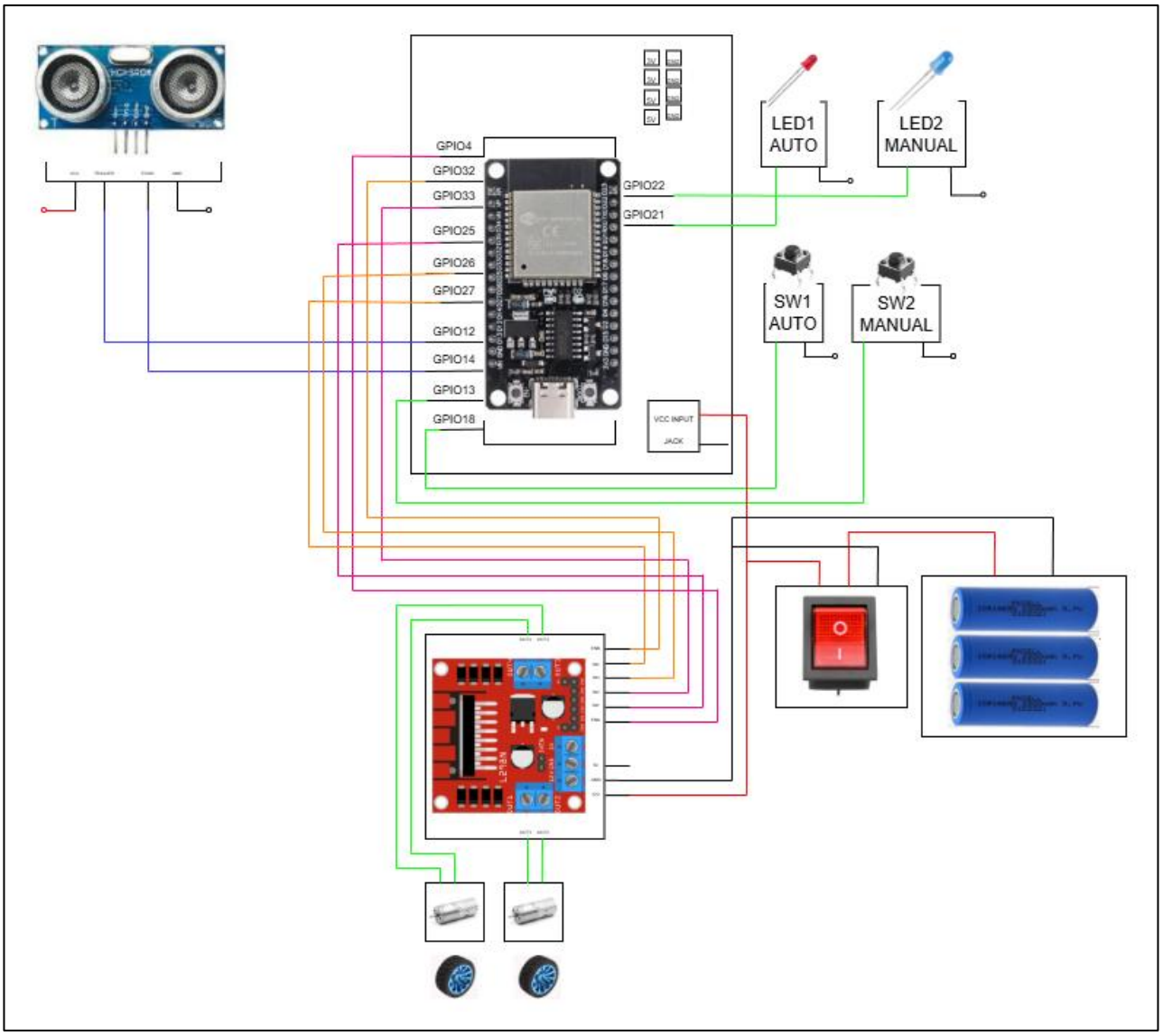

FIGURE 2

C. Actual Sumo Bot

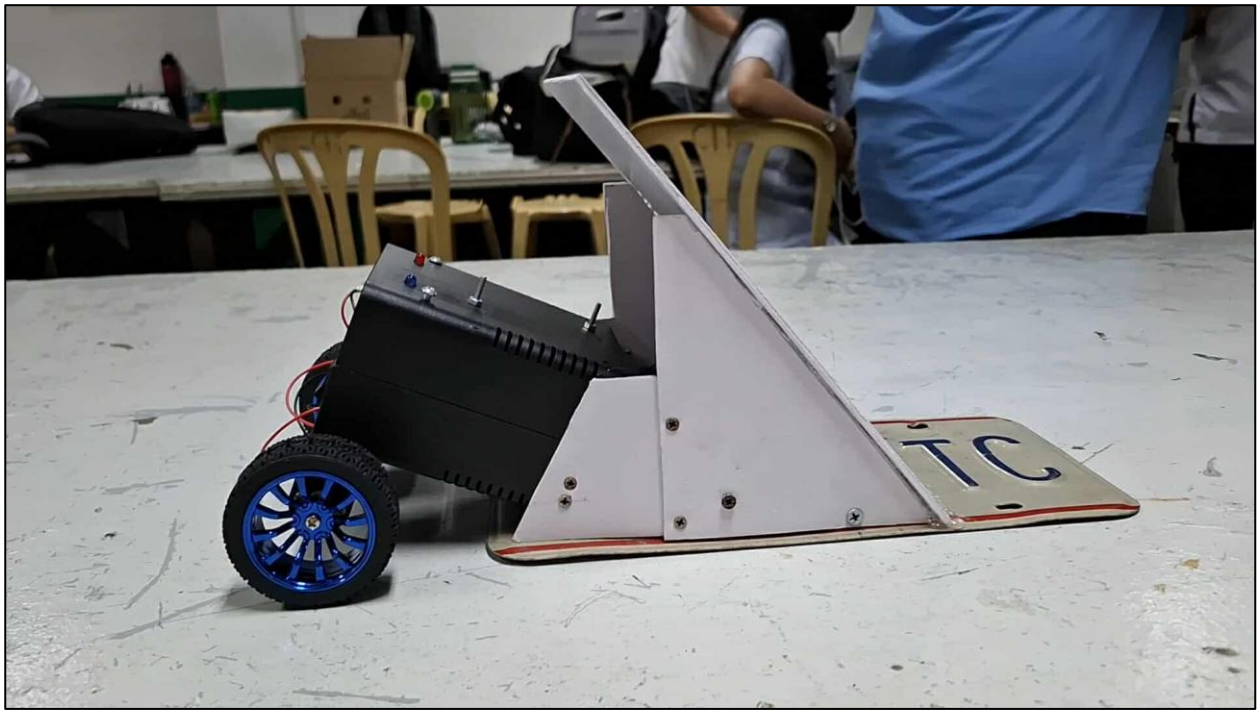

FIGURE 3

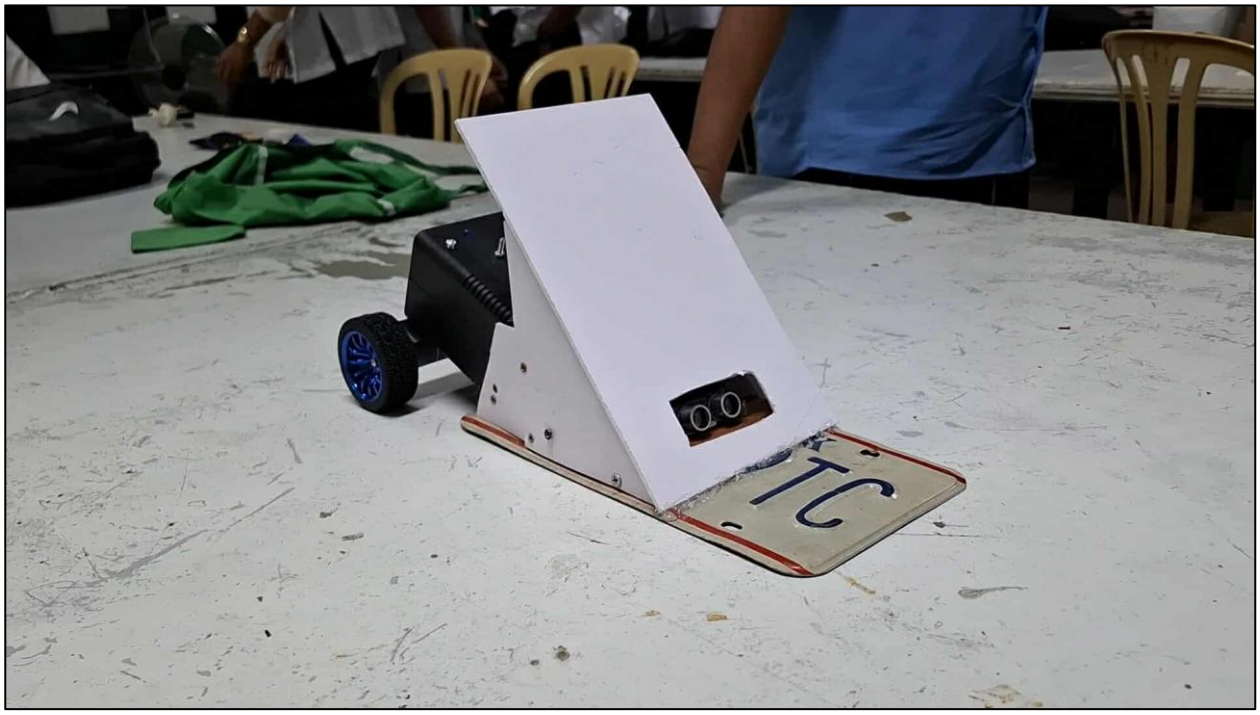

FIGURE 3.1

D. Source Code

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#include <DabbleESP32.h>
#include <BluetoothSerial.h>

// Ultrasonic
#define trigPin 12
#define echoPin 14

// Right Motor
#define enableRightMotor 32
#define rightMotorPin1 26
#define rightMotorPin2 27

// Left Motor
#define enableLeftMotor 4
#define leftMotorPin1 25
#define leftMotorPin2 33

// Buttons
#define buttonAuto 18
#define buttonManual 13

// LEDs
#define ledAuto 21
#define ledManual 22

bool autoMode = false;
bool manualMode = false;

long duration;
int distance;

void setup() {
 Serial.begin(115200);

 // Motor pins
 pinMode(enableRightMotor, OUTPUT);
 pinMode(rightMotorPin1, OUTPUT);
 pinMode(rightMotorPin2, OUTPUT);
 pinMode(enableLeftMotor, OUTPUT);
 pinMode(leftMotorPin1, OUTPUT);
 pinMode(leftMotorPin2, OUTPUT);

 // Ultrasonic pins
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);

 // Buttons
 pinMode(buttonAuto, INPUT_PULLUP);
 pinMode(buttonManual,
INPUT_PULLUP);

 // LEDs
 pinMode(ledAuto, OUTPUT);
 pinMode(ledManual, OUTPUT);

 // Bluetooth
 Dabble.begin("SumoBot_Manual");

 Serial.println(" SumoBot Ready! Use
buttons to select mode.");
}

void rotateMotor(int rightMotorSpeed, int
leftMotorSpeed) {
 digitalWrite(rightMotorPin1,
rightMotorSpeed > 0);
 digitalWrite(rightMotorPin2,
rightMotorSpeed < 0);
 digitalWrite(leftMotorPin1,
leftMotorSpeed > 0);
 digitalWrite(leftMotorPin2,
leftMotorSpeed < 0);

 analogWrite(enableRightMotor,
abs(rightMotorSpeed));
 analogWrite(enableLeftMotor,
abs(leftMotorSpeed));
}

int getDistance() {
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
```

```
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  return distance;
}

void runAutomaticMode() {
  int dist = getDistance();
  Serial.print("Distance: ");
  Serial.println(dist);

  if (dist <= 62 && dist > 0) {
    rotateMotor(200, 200); // Move forward
  } else {
    rotateMotor(0, 0); // Stop
  }
}

void runManualMode() {
  Dabble.processInput();

  int rightMotorSpeed = 0;
  int leftMotorSpeed = 0;

  if (GamePad.isUpPressed()) {
    rightMotorSpeed = 255;
    leftMotorSpeed = 255;
  } else if (GamePad.isDownPressed()) {
    rightMotorSpeed = -255;
    leftMotorSpeed = -255;
  } else if (GamePad.isLeftPressed()) {
    rightMotorSpeed = 255;
    leftMotorSpeed = -255;
  } else if (GamePad.isRightPressed()) {
    rightMotorSpeed = -255;
    leftMotorSpeed = 255;
  } else {
    rightMotorSpeed = 0;
    leftMotorSpeed = 0;
  }
  rotateMotor(rightMotorSpeed,
leftMotorSpeed);
}

void loop() {
   Dabble.processInput();
  // Read buttons (LOW = pressed)
  bool buttonAutoState =
!digitalRead(buttonAuto);
  bool buttonManualState =
!digitalRead(buttonManual);

  // Mode switching
  if (buttonAutoState) {
    autoMode = true;
    manualMode = false;
    digitalWrite(ledAuto, HIGH);
    digitalWrite(ledManual, LOW);
    Serial.println("Auto Mode Activated");
    delay(500);
  } else if (buttonManualState) {
    manualMode = true;
    autoMode = false;
    digitalWrite(ledAuto, LOW);
    digitalWrite(ledManual, HIGH);
    Serial.println("Manual Mode Activated");
    delay(500);
  }

  // Run active mode
  if (autoMode && !manualMode) {
    runAutomaticMode();
  } else if (manualMode && !autoMode) {
    runManualMode();
  } else {
    rotateMotor(0, 0);
  }
}
```

E. Dabble Application

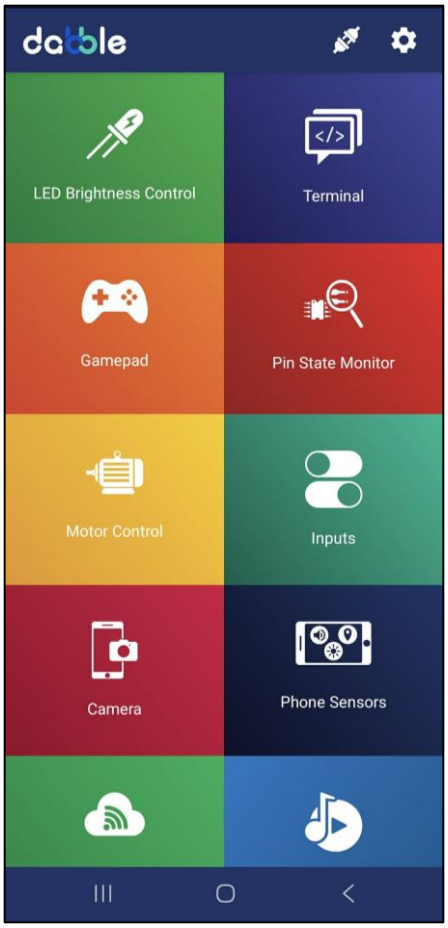

FIGURE 4

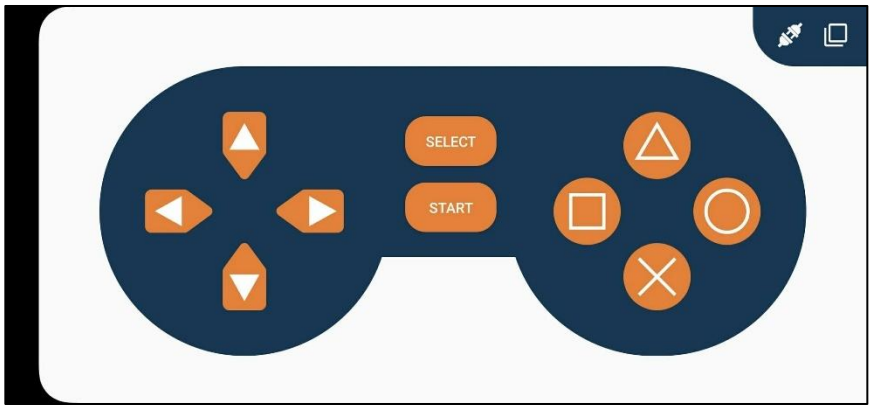

FIGURE 4.1

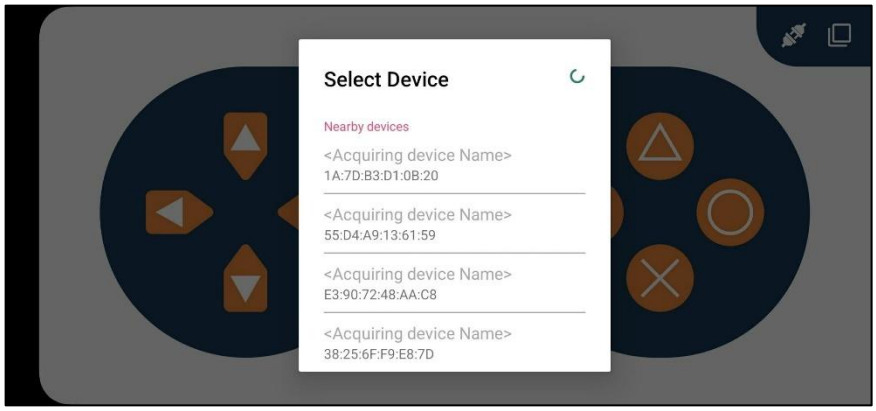

FIGURE 4.2

IV. PROCEDURE

A. Preparation

1. Plan Your Multi-Hazard Detector System

- Decide what hazards your system will monitor. For this project, the system will detect:
  – Earthquake/Vibration (MPU6050)
  – Fire/Temperature (DHT11 or flame sensor if included)
  – Gas Leak (MQ2 or MQ137)
  – Motion/Intrusion (PIR Sensor)
- Identify the main controller you will use.
  - This project will use the ESP32 microcontroller because it supports Wi-Fi for Telegram notifications.
- Decide how the system will notify the user.
  – In this project, the alert system will use Telegram Bot API to send real-time hazard alerts to a smartphone.
- Make a simple sketch or block diagram showing how the sensors connect to the ESP32 (MPU6050 via I2C, MQ2 via analog input, PIR via digital, etc.).

2. Gather All the Needed Parts

- Prepare all electronic components listed in Section 2 of this documentation.
  Typical components include:
  – ESP32 Board
  – MPU6050 Accelerometer/Gyroscope
  – MQ137 Gas Sensor
  – DHT11 Temperature & Humidity Sensor
  – PIR Motion Sensor
  – Jumper Wires
  – Breadboard
  – USB Cable

– Smartphone with Telegram app

– Power supply (5V)

3. Check Each Part Before Building

Before putting everything together, test each component one by one:

ESP32 Microcontroller

- Connect the ESP32 to your PC using the USB cable.
- Upload a simple “Blink” test to verify that the board is working.
- The onboard LED at GPIO 2 should blink.

MPU6050 (Earthquake/Vibration Sensor)

Connections:
- VCC → 3.3V
- GND → GND
- SDA → GPIO 21
- SCL → GPIO 22

How to test:
- Upload an MPU6050 test code (available in the MPU6050 library examples).
- Open Serial Monitor.
- Move or shake the sensor lightly.
- X, Y, Z values should change smoothly.

MQ2 / MQ137 Gas Sensor

Connections:
- VCC → 5V
- GND → GND
- A0 → GPIO 34 (Analog Input)

How to test:
- Upload a simple analog read test code.
- Open Serial Monitor.
- Bring a small amount of smoke or alcohol vapor near the sensor.
- The analog value should increase.

DHT11 Temperature & Humidity Sensor

Connections:
• VCC → 3.3V
• GND → GND
• Data → GPIO 4

How to test:
• Use the DHT library example “DHTtester”.
• Open Serial Monitor.
• Temperature and humidity values should appear and update.

PIR Motion Sensor

Connections:
• VCC → 5V
• GND → GND
• OUT → GPIO 33

How to test:
• Upload a simple digital read program.
• Move your hand in front of the sensor.
• Output should switch from LOW to HIGH (motion detected).

ESP32 WiFi & Telegram Test

• Connect ESP32 to Wi-Fi using a simple WiFi example sketch.
• Create a Telegram bot using BotFather.
• Send a test message from the ESP32 to confirm communication.

Wires and Connections

• Check for broken or loose wires.
• Choose correct lengths for neat wiring.
• Label each wire (SDA, SCL, A0, GPIO pins) for easier final assembly.

4. Final Check Before Assembly

• Ensure all sensors respond correctly during testing.
• Verify the ESP32 successfully connects to Wi-Fi.

• Confirm Telegram bot is working and receives messages.

• Review your wiring plan or design sketch before beginning the final assembly.

• Prepare a safe working space for connecting and assembling the components.

B. Actual

Step 1: Layout all the components as shown on Section III and connect them according to the pictorial diagram/block or one could refer to the following table for the pin connections.

ESP32 Connections

Component Pins	ESP32 GPIO Pins
MQ137 – DATA OUT	GPIO 34
MQ137 - VCC	5v VCC Out
MQ137 - GND	Ground (GND) Pin
MPU6050 - SCL	SCL 22
MPU6050 - SDA	SDA 21
MPU6050 - VCC	5v VCC Out
MPU6050 - GND	Ground (GND) Pin
DHT 11 – DATA OUT	GPIO 4
DHT 11 – VCC	5v VCC Out
DHT 11 – GND	Ground (GND) Pin
PIR Sensor – DATA OUT	GPIO 33
PIR Sensor – VCC	5v VCC Out
PIR Sensor – GND	Ground (GND) Pin
LED - Onboard	GPIO 2
I2C LCD - SDA	SDA 21
I2C LCD - SCL	SCL 22
I2C LCD – VCC	5v VCC Out
I2C LCD – GND	Ground (GND) Pin

Step 2: Upload the program to the ESP32. To check if both the program and hardware is working, please refer to the following section.

To Enable Telegram Notification

Step 3: Download the Telegram Application

Step 4: Create the Bot with BotFather

Step 5: Open Telegram and search for @BotFather. Make sure it's the official, verified bot.

Step 6: Start a chat and send the command /newbot.

Step 7: Choose a Name for your bot (e.g., "My Blog Bot").

Step 8: Choose a Username (e.g., "my_blog_bot") – it must be unique and end with bot.

Step 9: Get the Token: BotFather will give you a long API token. Copy this token and keep it safe; you'll need it for the next steps.

Step 10: Get ID from IDBOT Search for @IDBot.

Step 11: After IDBot responds, type in the command /getid. IDBot will then give you your ID

Step 12: Plot the given token from BotFather and ID from IDBot to your code and use functions to operate with the bot.

Step 13: Re-upload your code in ESP 32 after integrating the token of Telegram Bot API.

C. Checking

After connecting all necessary components to the microcontroller and uploading the program, verify that each component functions correctly by following these steps:

MPU6050 (Earthquake / Vibration Sensor):

- Slightly shake or tap the sensor.
- Confirm that the X, Y, and Z acceleration values change.
- If vibration exceeds the threshold, the ESP32 should print “Earthquake Detected” with its corresponding magnitude level and if enabled, send a test alert to Telegram.

MQ137 (Gas Sensor):

• Bring a small controlled source near it (lighter without flame).
• The reading should increase when gas concentration is detected.
• The LCD should show “Gas Leak Detected” when above the threshold and if enabled, send a test alert to Telegram.

DHT11 (Fire / Temperature Detector):

• Apply gentle heat nearby (lighter, hairdryer at low heat).
• Temperature should rise accordingly.
• If temperature passes the fire threshold, the ESP32 should identify “High Temperature / Fire Risk” and send a test alert to Telegram..

Buttons (Modes):

- Move your hand or body in front of the PIR sensor.
- The output should switch from LOW to HIGH.
- Telegram should send “Motion Detected” alert and optionally activate the onboard LED during detection.

D. Uploading

Upload the final code. Once uploaded, check the behavior of the overall Multi-Hazard Detector System if its functionality correlates with the algorithm of the program. Make sure to test all of the sensor to properly examine its performance, calibrate its sensitivity, and monitor the alert signals and messages that will be received by the user. If the output is emitting unexpected results, it is suggested to initially check the hardware components—mainly for loose wires or improper soldered components before heading towards the software to check for bugs in the program.

V. CONCLUSION

The project, Atom the recycabot, was a complete success. It not only fulfilled its purpose as an effective sumo bot but also achieved this by being constructed primarily from recycled materials. This approach demonstrated great resourcefulness, compromising with available components to avoid excessive spending. The ESP32 successfully demonstrated its robust capability to manage the complex, real-time autonomous robot. Each subsystem showcased a critical robotics concept: detecting opponents with ultrasonic sensors, avoiding boundaries with reflectance sensors, and implementing precise movement with a motor driver. The integration of these components, powered by the ESP32's fast processing, highlighted how to fuse sensor data into an effective, autonomous combat strategy. This project deepened our understanding of sensor interfacing, motor control, and the development of cost-effective algorithms for competitive robotics.

VI. PHOTO DOCUMENTATION