SurveX:Environment Surveillance Bot

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology

by

Shrirang Rekhate (Reg. No.2022BEC066)

Department of Electronics and Telecommunication Engineering,
Shri Guru Gobind Singhji Institute of Engineering & Technology,
Vishnupuri, Nanded, Maharashtra, India, 431606.
2025-26

ABSTRACT

The "environment surveillance bot" project focuses on the design and implementation of an intelligent robotic system aimed at monitoring and analyzing environmental conditions in real-time. This bot utilizes a combination of sensors, such as temperature, humidity, air quality, and pollution detectors, to gather crucial data about the surrounding environment. The bot is equipped with advanced algorithms for data processing, pattern recognition, and anomaly detection, enabling it to track environmental changes and identify potential hazards, such as increased pollution levels or abnormal weather patterns. By leveraging iot (internet of things) technologies and cloud computing, the system is capable of transmitting collected data to a central platform for further analysis and decision-making. The project aims to enhance environmental monitoring, provide early warnings for natural disasters, and assist in urban planning and sustainable development efforts. Additionally, the bot's autonomous navigation system allows it to operate in diverse environments, including urban, rural, and remote locations, ensuring comprehensive surveillance and data collection. The overall goal of this project is to create an effective, scalable, and cost-efficient tool for real-time environmental monitoring, promoting a healthier and more sustainable planet.

Keywords: IoT, Sensor Fusion, Real-time Monitoring, WebSocket Communication, Environmental Data Logging.

CONTENTS

Chapter 6

Conclusions

References

Chapter 1

INTRODUCTION

In the face of growing environmental challenges such as climate change, air pollution, and natural disasters, the need for efficient and real-time environmental monitoring has become more critical than ever. The *Environment Surveillance Bot* project addresses this need through the development of an intelligent robotic system designed to autonomously monitor and analyze environmental conditions in real-time. By integrating advanced sensor technologies—including temperature, humidity, air quality, and pollution detectors—with IoT (Internet of Things) and cloud computing, this bot is capable of collecting, processing, and transmitting vital environmental data.

Equipped with intelligent algorithms for data analysis, pattern recognition, and anomaly detection, the bot can identify potential hazards and track environmental changes with high accuracy. Its autonomous navigation system enables it to operate effectively across a wide range of terrains, from densely populated urban areas to remote rural regions. This adaptability ensures comprehensive environmental surveillance and enhances early warning systems for natural disasters.

The overarching goal of this project is to create a scalable, cost-efficient, and intelligent solution for environmental monitoring that not only supports data-driven decision-making but also contributes to sustainable development and urban planning. By promoting a proactive approach to environmental management, the *Environment Surveillance Bot* serves as a step forward in building a healthier and more sustainable future for our planet.

1.1 PROJECT AIM

The aim of project is to design and construct a environment surveillance bot that will be operated by human to get environmental conditions such as temperature, gases etc using ESP 32 and various sensors.

1.2 PROJECT OBJECTIVE

The primary objective of the Environment Surveillance Bot project is to design and implement an intelligent, autonomous robotic system capable of real-time environmental monitoring and analysis. Continuously collect environmental data using integrated sensors for temperature, humidity, air quality, and pollution levels. Analyze and detect anomalies through advanced algorithms and pattern recognition techniques. Provide real-time data transmission via IoT and cloud-based platforms for remote monitoring and decision-making.

1.3 PROJECT SCOPE AND LIMITATIONS

The Environment Surveillance Bot is designed to autonomously monitor environmental conditions using sensors for temperature, humidity, air quality, and pollution. It processes data in real-time, detects anomalies, and transmits information via IoT and cloud platforms. The bot can navigate various terrains, supporting applications in urban planning and disaster management. However, it may face limitations such as sensor inaccuracies, limited battery life, connectivity issues in remote areas, and high deployment costs.

Chapter 2

LITERATURE SURVEY

Environmental monitoring has gained significant attention in recent years due to increasing concerns over climate change, air pollution, and the need for sustainable development. Various studies have explored the use of wireless sensor networks (WSNs), IoT, and autonomous systems for real-time environmental data collection and analysis.

According to research by Sharma et al. (2020), IoT-based environmental monitoring systems enable efficient and remote tracking of parameters like temperature, humidity, and air quality, improving decision-making in urban planning and disaster management. Similarly, Ahmed and Kumar (2019) highlighted the benefits of integrating cloud computing with sensor-based systems for scalable and centralized environmental data analysis.

Robotic systems have also been explored for mobile data collection. For instance, the work by Li and Chen (2018) demonstrates the use of autonomous drones equipped with environmental sensors for wide-area air quality monitoring. Their findings suggest that mobility significantly enhances the coverage and accuracy of data collection compared to static systems.

Furthermore, pattern recognition and anomaly detection algorithms have been used to identify abnormal environmental events. A study by Gupta et al. (2021) implemented machine learning techniques to detect pollution spikes and sudden weather changes, emphasizing the importance of real-time processing in predictive environmental monitoring.

While existing systems demonstrate strong potential, many face challenges related to power efficiency, sensor accuracy, terrain adaptability, and real-time data transmission—issues this project seeks to address through a cost-effective, autonomous, and IoT-enabled robotic platform capable of operating in diverse environments.

Chapter 3

PRE-REQUIREMENTS OF PROJECT

HARDWARE REQUIREMENT

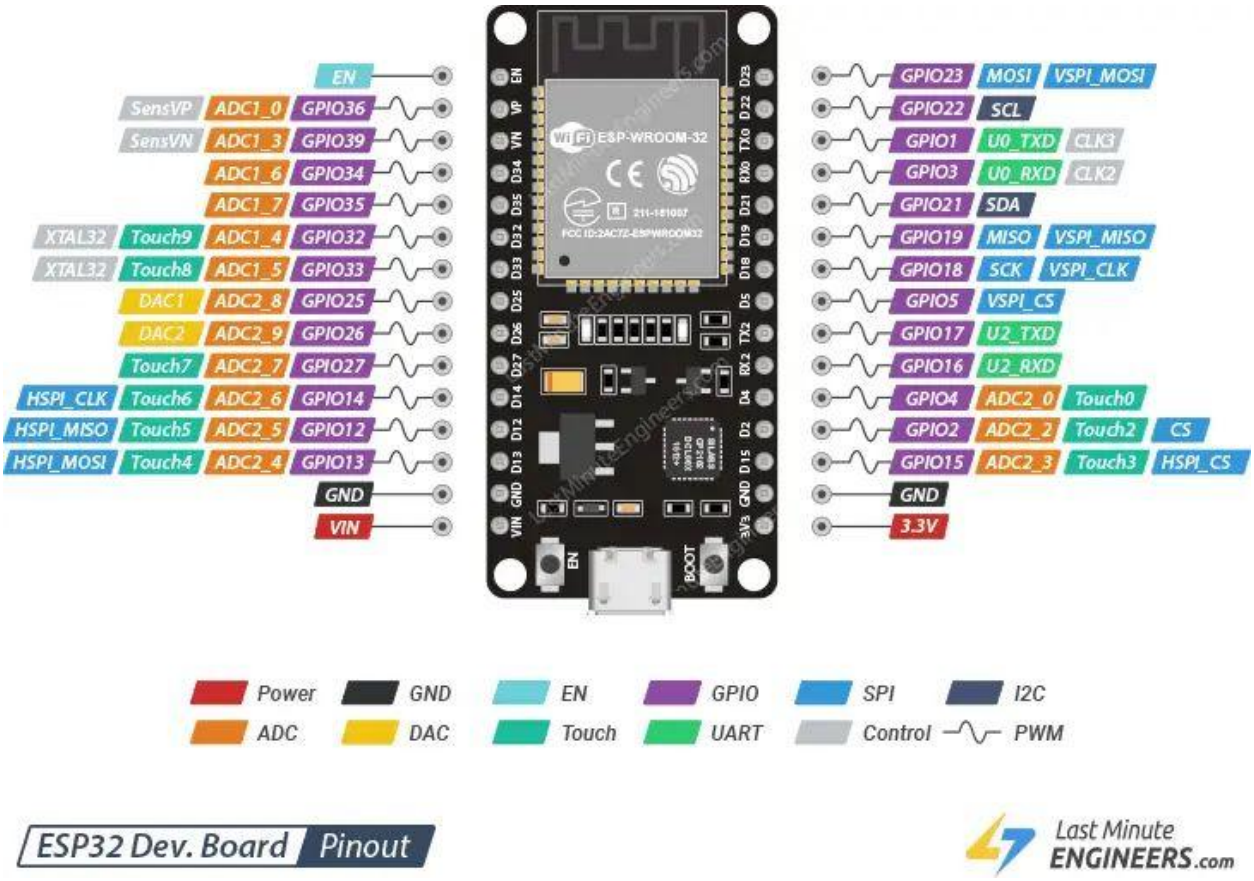
- ESP-32
- ESP CAM
- L289N Motor Driver
- DHT11 Temperature Sensor
- MQ2 Gas Sensor
- Ultrasonic Sensor

SOFTWARE REQUIREMENT

- HTML
- CSS
- Javascript
- Typescript

HARDWARE

3.1.A ESP-32



The ESP32 is a powerful, highly integrated SoC (System on Chip) developed by Espressif Systems, widely used in IoT, robotics, and embedded applications. It is an upgraded version of the earlier ESP8266 and offers significant improvements in terms of performance, connectivity, and functionality.

Key On-Chip Features of ESP32:

1] <u>Processor:</u>

Dual-core Tensilica Xtensa LX6 microprocessor.

Clock speed up to 240 MHz.

Ultra-low power co-processor for handling sensor readings in deep sleep mode.

2] <u>Memory:</u>

520 KB SRAM (internal).

Support for external flash memory via SPI (typically 4MB to 16MB in development boards).

3] Connectivity:

Wi-Fi (802.11 b/g/n) with support for both STA and AP modes.

Bluetooth 4.2 and Bluetooth Low Energy (BLE).

On-chip antenna switch, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules.

4] I/O and Peripheral Interfaces:

34 Programmable GPIO pins.

12-bit ADC (18 channels), 2 × 8-bit DACs.

SPI, I2C, I2S, UART – for interfacing with sensors, modules, and external peripherals.

PWM outputs for motor control, LEDs, etc.

Hall Sensor and Temperature Sensor built-in.

5] Timers and RTC:

Multiple general-purpose timers.

Real-Time Clock (RTC) for low-power operations and wake-up scheduling.

Security Features:

Hardware encryption (AES, SHA-2, RSA, ECC).

Secure boot and flash encryption for data protection.

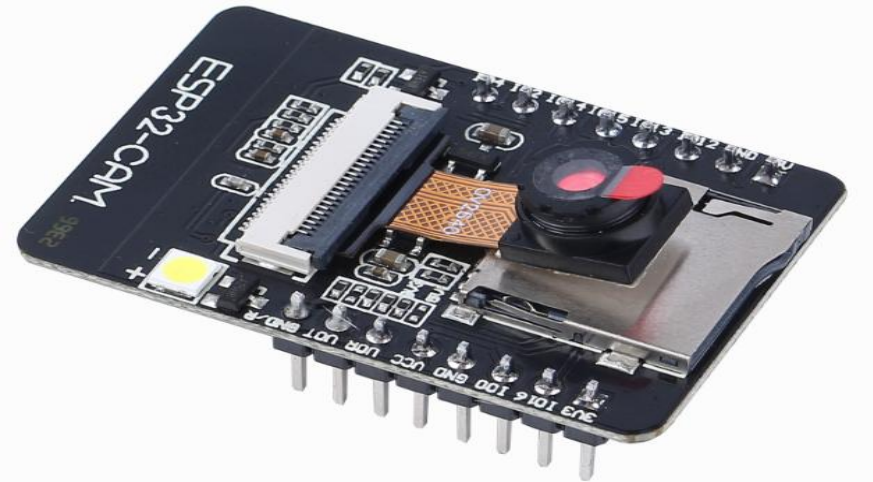
Random number generator for secure communications.

Power Management:

Multiple power modes: Active, Modem-sleep, Light-sleep, and Deep-sleep.

Ultra-low power consumption in sleep modes (ideal for battery-powered bots)

3.1.B ESP CAM



The ESP32-CAM is a low-cost development board based on the ESP32-S chip, integrated with a camera module and wireless communication capabilities. It combines powerful computing features with real-time image and video capture, making it ideal for applications in surveillance, remote monitoring, face recognition, and smart robotics.

Key Features of ESP32-CAM:

Microcontroller:

ESP32-S (Tensilica Xtensa dual-core 32-bit LX6 microprocessor).

Clock speed up to 240 MHz.

Built-in 520 KB SRAM and support for external PSRAM up to 4MB (often included on-board).

Camera Module:

Comes with an OV2640 camera sensor (2MP).

Supports image resolutions from 160x120 up to 1600x1200.

JPEG, BMP, and grayscale format support.

Can capture still images and short video streams.

Wireless Connectivity:

Wi-Fi 802.11 b/g/n for wireless data transfer.

Supports both Access Point (AP) and Station (STA) modes.

No built-in Bluetooth on most ESP32-CAM variants.

Storage and Expansion:

MicroSD card slot for local storage (e.g., saving images or logs).

Up to 4GB supported in FAT32 format.

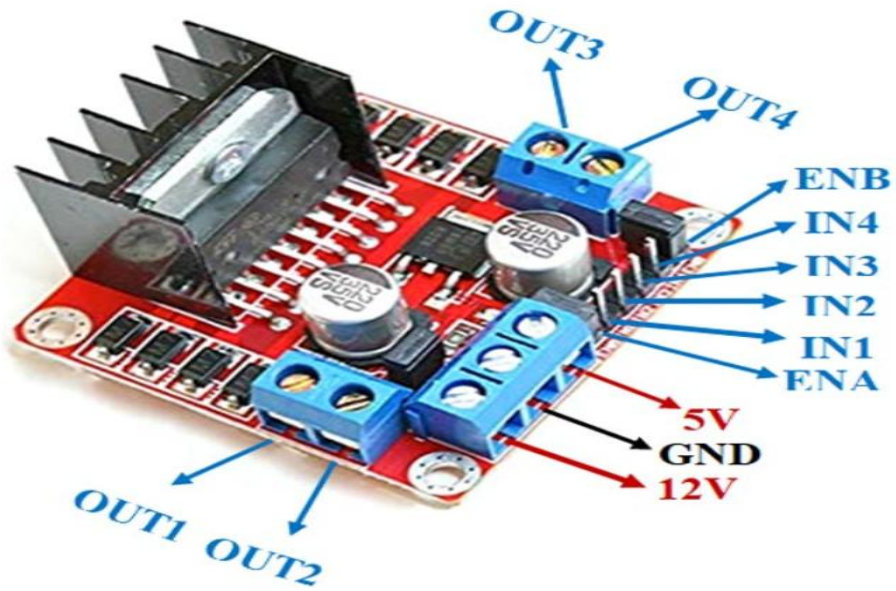
I/O Capabilities:

9 usable GPIO pins (limited due to camera and SD card use).

Support for UART, SPI, I2C, PWM.

Flash LED for night vision or low-light imaging.

3.1.C L289N MOTOR DRIVER



The L298N Motor Driver is a dual H-bridge motor driver module that allows control of the speed and direction of two DC motors or one stepper motor simultaneously. It is widely used in robotics and automation projects due to its reliability and ease of use.

Key Features:

Dual H-Bridge Configuration: Can control 2 DC motors independently or 1 stepper motor.

Voltage Range:

Motor supply (VMS): 5V to 35V.

Logic supply (VLS): 5V (can be powered from onboard 5V regulator).

Current Output: Up to 2A per channel (with proper heat dissipation).

Control Inputs: Takes TTL-level signals (from microcontrollers like ESP32 or Arduino) for controlling motor direction and speed.

PWM Support: Allows speed control using Pulse Width Modulation (PWM) signals.

Built-in 5V Regulator: Can provide power to the logic circuit if input voltage is above 7V (via jumper cap).

<u>Pin Description:</u>
IN1, IN2, IN3, IN4: Motor direction control inputs.
ENA, ENB: Enable pins for Motor A and Motor B (can also be used for PWM speed control).
OUT1, OUT2, OUT3, OUT4: Outputs to motors.
VMS (12V), GND, 5V: Power pins.

3.1.D DHT11 TEMPERATURE SENSOR

The DHT11 is a popular, low-cost sensor used for measuring temperature and humidity in various environmental monitoring applications. It is widely used in IoT projects, home automation systems, and robotics due to its simplicity and ease of integration with microcontrollers like ESP32 or Arduino.

Key Features:

Temperature Range:

Measures temperature from 0°C to 50°C with an accuracy of ±2°C.

Humidity Range:

Measures relative humidity from 20% to 80% RH with an accuracy of ±5% RH.

Digital Output:

Provides a single-wire digital output, simplifying data acquisition (no need for analog-to-digital conversion).

Low Power Consumption:

Operates on a 3.3V to 5V power supply, making it compatible with most microcontrollers.

Sampling Rate:

Typically, outputs data every 1-2 seconds.

Pinout:

The DHT11 has 3 pins (sometimes 4 in some versions), typically labeled as:

VCC: Power supply (3.3V to 5V).

GND: Ground.

DATA: Digital signal output (used to send temperature and humidity data).

NC (optional): Not connected in some versions.

3.1.E MQ2 Gas Sensor

The MQ2 is a widely used gas sensor designed to detect a variety of gases, including methane (CH4), carbon monoxide (CO), liquefied petroleum gas (LPG), smoke, and other combustible gases. It is frequently used in environmental monitoring, safety systems, and home automation applications to ensure safe and healthy living conditions by detecting hazardous gases in the air.

Key Features:

Gas Detection Range:

Methane (CH4): 200 to 10000 ppm.

Carbon monoxide (CO): 20 to 2000 ppm.

LPG: 200 to 10000 ppm.

Smoke: 100 to 1000 ppm.

Heating Element:

The sensor uses a heating element and a sensitive layer to detect gas presence through changes in resistance.

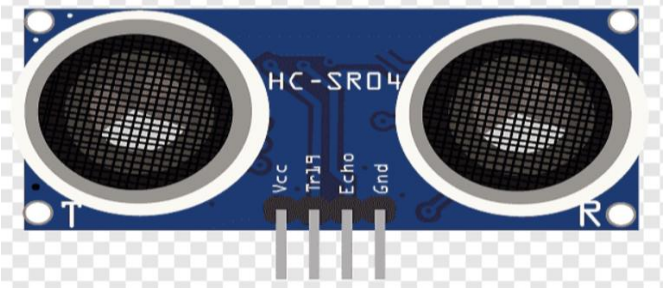
Operating Voltage: Typically operates at 5V but can be used at 3.3V with some adaptations.

Long Lifespan: The MQ2 sensor has a relatively long operational life and is durable with proper use.

Pinout:

The MQ2 sensor typically has 4 pins:
VCC: Power supply pin (typically 5V).
GND: Ground.
DO (Digital Output): Provides a HIGH or LOW signal when the gas concentration exceeds the set threshold.
AO (Analog Output): Provides an analog voltage proportional to the concentration of gases.

3.1.F ULTRASONIC SENSOR



The Ultrasonic Sensor is a widely used distance measurement sensor that utilizes ultrasonic waves to measure the distance between the sensor and an object. The most commonly used ultrasonic sensor is the HC-SR04, which is ideal for robotic applications, including environmental surveillance systems like our *Environment Surveillance Bot*.

Key Features:

Measurement Principle:
The sensor emits a high-frequency sound wave (ultrasonic wave) and measures the time it takes for the wave to reflect back from an object. The distance is then calculated based on the speed of sound.

Operating Voltage:
Typically operates at 5V with a current consumption of around 15-20 mA during operation.

Distance Range:
Can measure distances from 2 cm to 400 cm (or about 1 inch to 13 feet).

Accuracy:
The sensor can measure distances with an accuracy of about 3 mm under optimal conditions.

Resolution:
Provides fairly high resolution and can detect minute changes in the distance when used for proximity sensing.

Output:
Digital Output – The sensor provides a pulse width that can be measured by a microcontroller (such as ESP32) to calculate distance.

Typically, it uses two pins: Trigger (TRIG) and Echo (ECHO).
Pinout:

The HC-SR04 Ultrasonic Sensor has 4 pins:
VCC: Power supply (5V).
GND: Ground.
TRIG (Trigger): Input pin that starts the measurement by sending a 10µs pulse to the sensor.
ECHO (Echo): Output pin that provides a pulse width proportional to the time taken by the ultrasonic wave to return after hitting an object.

3.2 CODE

ESP Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <WebSocketsServer.h>
#include <Wire.h>
#include <MPU6050_tockn.h>
#include <MQUnifiedsensor.h>
#include <Bonezegei_DHT11.h>

// ==WiFi Credentials ==
#define WIFI_SSID  "$HREE"
#define WIFI_PASSWORD    "12345678900"

// ==Server Instances ==
WebServer httpServer(80);// HTTP server (optional)
WebSocketsServer webSocket(81); // WebSocket server on port 81

// ==Sensor & Module Instances==
MPU6050 mpu6050(Wire); // MPU6050 sensor via I2C
#define MQ2_PIN  32
#define MQ2_BOARD    "ESP32"
#define MQ2_TYPE        "MQ-2"
#define VOLTAGE_RESOLUTION   3.3
#define ADC_RESOLUTION       12
#define RATIO_CLEAN_AIR      9.83
MQUnifiedsensor MQ2(MQ2_BOARD, VOLTAGE_RESOLUTION, ADC_RESOLUTION, MQ2_PIN, MQ2_TYPE);
#define DHTPIN 33
Bonezegei_DHT11 dht(DHTPIN);
// == Ultrasonic Sensor Pins ==
const int trigFront = 12, echoFront = 13;
const int trigLeft = 14, echoLeft = 27;
const int trigRight = 26, echoRight = 25;
// == Motor Driver Pins (L298N)==
const int IN1 = 16, IN2 = 17, IN3 = 5, IN4 = 18;
const int ENA = 4, ENB = 19;
// PWM configuration
#define PWM_CHANNEL_A  0
#define PWM_CHANNEL_B  1
#define PWM_FREQ       1000
#define PWM_RES        8
int motorSpeed = 80; // Default speed
// == Timing ==
unsigned long prevSensorMillis = 0;
const unsigned long sensorInterval = 1000;  // 1 second interval

// == Data Structures ==
struct MPUData {
  float angleX, angleY, angleZ;
};
struct DHTData {
  float tempC, tempF;
  int humidity;
};
struct MQ2Data {
  float lpg, co, smoke;
};
// == Function Prototypes ==
MPUData getMPUData();
DHTData getDHTData();
MQ2Data readMQ2Sensor();
long measureDistance(int trigPin, int echoPin);
void setupMotorPWM();
void motorForward();
void motorBackward();
void motorLeft();
void motorRight();
void motorStop();
void calibrateMQ2();

// == WebSocket Event Handler ==
void onWebSocketEvent(uint8_t client_num, WStype_t type,
uint8_t * payload, size_t length) {
  switch (type) {
    case WStype_CONNECTED:
      Serial.printf("[%u] Connected from %s\n", client_num,
webSocket.remoteIP(client_num).toString().c_str());
      webSocket.sendTXT(client_num, "Connected to ESP32!");
      break;
    case WStype_TEXT: {
        String cmd = String((char*)payload).substring(0, length);
        Serial.printf("[%u] Received command: %s\n", client_num,
cmd.c_str());
       if      (cmd == "forward")   motorForward();
       else if (cmd == "backward")  motorBackward();
       else if (cmd == "left")      motorLeft();
       else if (cmd == "right")     motorRight();
       else if (cmd == "stop")      motorStop();
       else if (cmd.startsWith("speed:")) {
         motorSpeed = cmd.substring(6).toInt();
         Serial.printf("Motor speed set to: %d\n", motorSpeed);
       }
       break;
      }

    case WStype_DISCONNECTED:
      Serial.printf("[%u] Disconnected\n", client_num);
      motorStop(); // Safety
      break;
  }
}

// ==HTTP Test Page ==
void handleRoot() {
  httpServer.send(200, "text/html", "<html><body><h2>ESP32 WebSocket Server
Running</h2></body></html>");
}
// ==Setup==
float errorX, errorY, errorZ;
void setup() {
  Serial.begin(115200);
  Wire.begin();
  // Initialize sensors
  mpu6050.begin();
  mpu6050.calcGyroOffsets(true);
  errorX = mpu6050.getAngleX();
  errorY = mpu6050.getAngleY();
  errorZ = mpu6050.getAngleZ();
  dht.begin();
  // Ultrasonic pins
  pinMode(trigFront, OUTPUT); pinMode(echoFront, INPUT);
  pinMode(trigLeft, OUTPUT);  pinMode(echoLeft, INPUT);
  pinMode(trigRight, OUTPUT); pinMode(echoRight, INPUT);
  // Motor driver pins
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  setupMotorPWM();
  motorStop();
  // WiFi
  Serial.print("Connecting to WiFi");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected! IP: ");
  Serial.println(WiFi.localIP());
```

```
  // HTTP & WebSocket server
  httpServer.on("/", handleRoot);
  httpServer.begin();
  webSocket.begin();
  webSocket.onEvent(onWebSocketEvent);

  // MQ-2 sensor setup
  Serial.println("MQ-2 Sensor Calibration");
  MQ2.setRegressionMetho);
  MQ2.setA(574.25); MQ2.setB(-2.222);
  MQ2.init();
  calibrateMQ2();
  MQ2.serialDebug(true);
}

//==Main ==
void loop() {
  httpServer.handleClient();
  webSocket.loop();
  if (millis() - prevSensorMillis >= sensorInterval) {
    prevSensorMillis = millis();
    MPUData mpu = getMPUData();
    MQ2Data mq2 = readMQ2Sensor();
    DHTData dhtData = getDHTData();
    long distFront = measureDistance(trigFront, echoFront);
    long distLeft  = measureDistance(trigLeft, echoLeft);
    long distRight = measureDistance(trigRight, echoRight);
    String json = "{";
    json += "\"mpu\":{\"angleX\":" + String(mpu.angleX, 2) +
            ",\"angleY\":" + String(mpu.angleY, 2) +
            ",\"angleZ\":" + String(0, 2) + "},";
    json += "\"mq2\":{\"lpg\":" + String(mq2.lpg, 2) +
            ",\"co\":" + String(mq2.co, 2) +
            ",\"smoke\":" + String(mq2.smoke, 2) + "},";
    json += "\"dht\":{\"tempC\":" + String(dhtData.tempC, 2) +
            ",\"tempF\":" + String(dhtData.tempF, 2) +
            ",\"humidity\":" + String(dhtData.humidity) + "},";
    json += "\"ultrasonic\":{\"front\":" + String(distFront) +
            ",\"left\":" + String(distLeft) +
            ",\"right\":" + String(distRight) + "}";
    json += "}";
    webSocket.broadcastTXT(json);
    if(distFront<10){motorStop();+}
  }
}
// ==Sensor Functions ==
MPUData getMPUData() {
  mpu6050.update();
  return {
    mpu6050.getAngleX() - errorX,
    mpu6050.getAngleY() - errorY,
    mpu6050.getAngleZ() - errorZ
  };
}
DHTData getDHTData() {
  if (dht.getData()) {
    return {
      dht.getTemperature(),
      dht.getTemperature(true),
      dht.getHumidity()
    };
  }
  return { 0, 0, 0 };
}
MQ2Data readMQ2Sensor() {
  MQ2.update();
  return { MQ2.readSensor(), 0, 0 }; // Only LPG currently
}
long measureDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW); delayMicroseconds(2);
  digitalWrite(trigPin, HIGH); delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  return duration * 0.034 / 2;
// ==Motor Control ==
void setupMotorPWM() {
  ledcSetup(PWM_CHANNEL_A, PWM_FREQ, PWM_RES);
  ledcSetup(PWM_CHANNEL_B, PWM_FREQ, PWM_RES);
  ledcAttachPin(ENA, PWM_CHANNEL_A);
  ledcAttachPin(ENB, PWM_CHANNEL_B);
  ledcWrite(PWM_CHANNEL_A, 0);
  ledcWrite(PWM_CHANNEL_B, 0);
  Serial.println("PWM channels initialized");
}
void motorForward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  ledcWrite(PWM_CHANNEL_A, motorSpeed);
  ledcWrite(PWM_CHANNEL_B, motorSpeed);
  //Serial.println("MOTOR: FORWARD");
}
void motorBackward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  ledcWrite(PWM_CHANNEL_A, motorSpeed);
  ledcWrite(PWM_CHANNEL_B, motorSpeed);
  //Serial.println("MOTOR: BACKWARD");
}
void motorLeft() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  ledcWrite(PWM_CHANNEL_A, motorSpeed);
  ledcWrite(PWM_CHANNEL_B, motorSpeed);
  //Serial.println("MOTOR: LEFT");
}
void motorRight() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  ledcWrite(PWM_CHANNEL_A, motorSpeed);
  ledcWrite(PWM_CHANNEL_B, motorSpeed);
  //Serial.println("MOTOR: RIGHT");
}
void motorStop() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  ledcWrite(PWM_CHANNEL_A, 0);
  ledcWrite(PWM_CHANNEL_B, 0);
  // Serial.println("MOTOR: STOP");
}
//== MQ-2 Calibration ==
void calibrateMQ2() {
  Serial.print("Calibrating MQ-2 sensor, please wait");
  float calcR0 = 0;
  for (int i = 0; i < 10; i++) {
    MQ2.update();
    calcR0 += MQ2.calibrate(RATIO_CLEAN_AIR);
    Serial.print(".");
    delay(1000);
  }
  calcR0 /= 10;
  MQ2.setR0(calcR0);
  Serial.println(" Calibration done!");

  if (isinf(calcR0)) {
    Serial.println("Warning: R0 is infinite. Check wiring.");
    while (true);
  }
  if (calcR0 == 0) {
    Serial.println("Warning: R0 is zero. Check wiring.");
    while (true);
  }
}
```

ESP CAM code

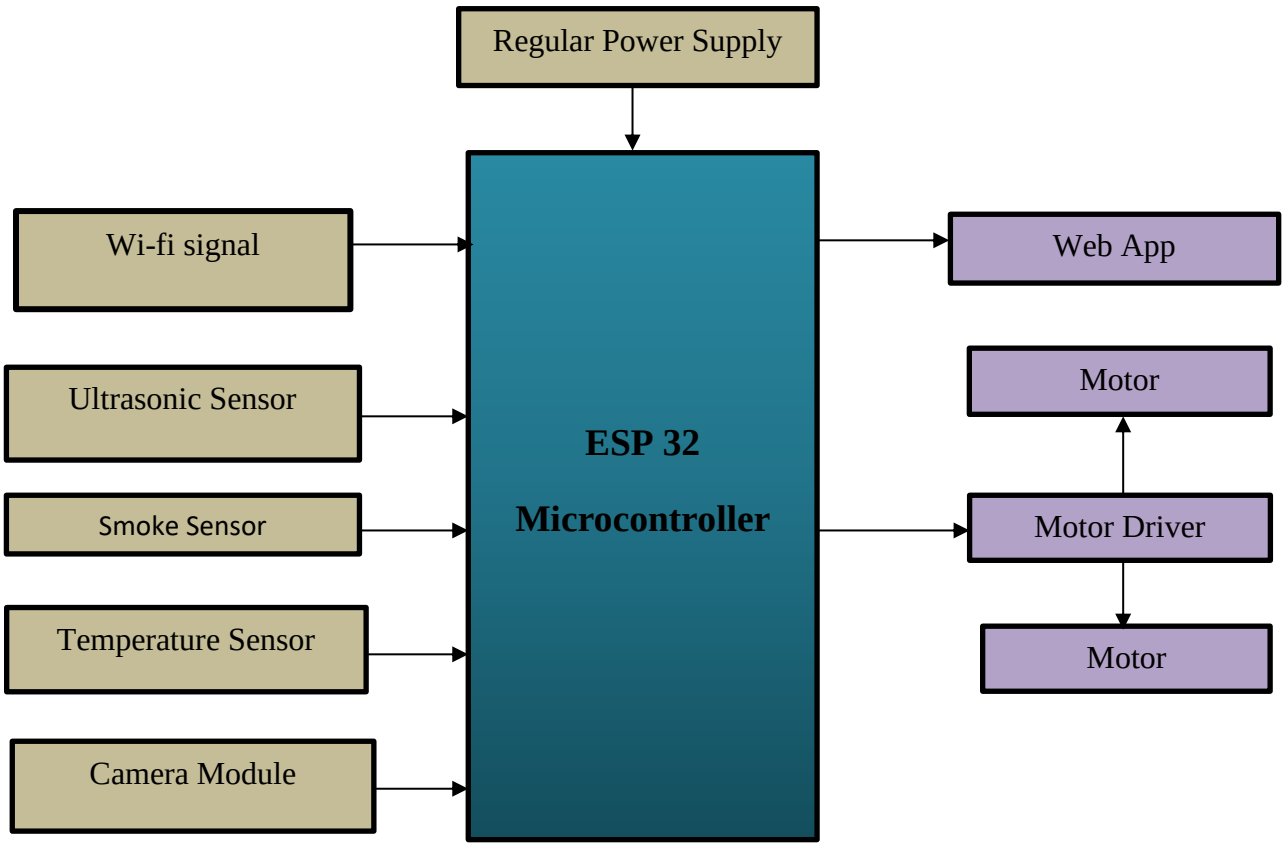
```
#include "esp_camera.h"
#include <WiFi.h>
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"
//CAM-IP--192.168.117.168
const char* ssid = "$HREE";
const char* password = "12345678900";
void startCameraServer();
void setupLedFlash(int pin);
void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG; // for streaming
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  //  for larger pre-allocated frame buffer.
  if(config.pixel_format == PIXFORMAT_JPEG){
    if(psramFound()){
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }
#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif
  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  sensor_t * s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit
saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1); // flip it back
    s->set_brightness(s, 1); // up the brightness just a bit
    s->set_saturation(s, -2); // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if(config.pixel_format == PIXFORMAT_JPEG){
    s->set_framesize(s, FRAMESIZE_QVGA);
  }
#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif
#if defined(CAMERA_MODEL_ESP32S3_EYE)
  s->set_vflip(s, 1);
#endif
// Setup LED FLash if LED pin is defined in camera_pins.h
#if defined(LED_GPIO_NUM)
  setupLedFlash(LED_GPIO_NUM);
#endif
  WiFi.begin(ssid, password);
  WiFi.setSleep(false);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  startCameraServer();
  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}
void loop() {
  // Do nothing. Everything is done in another task by the web
server
  delay(1000);
}
```

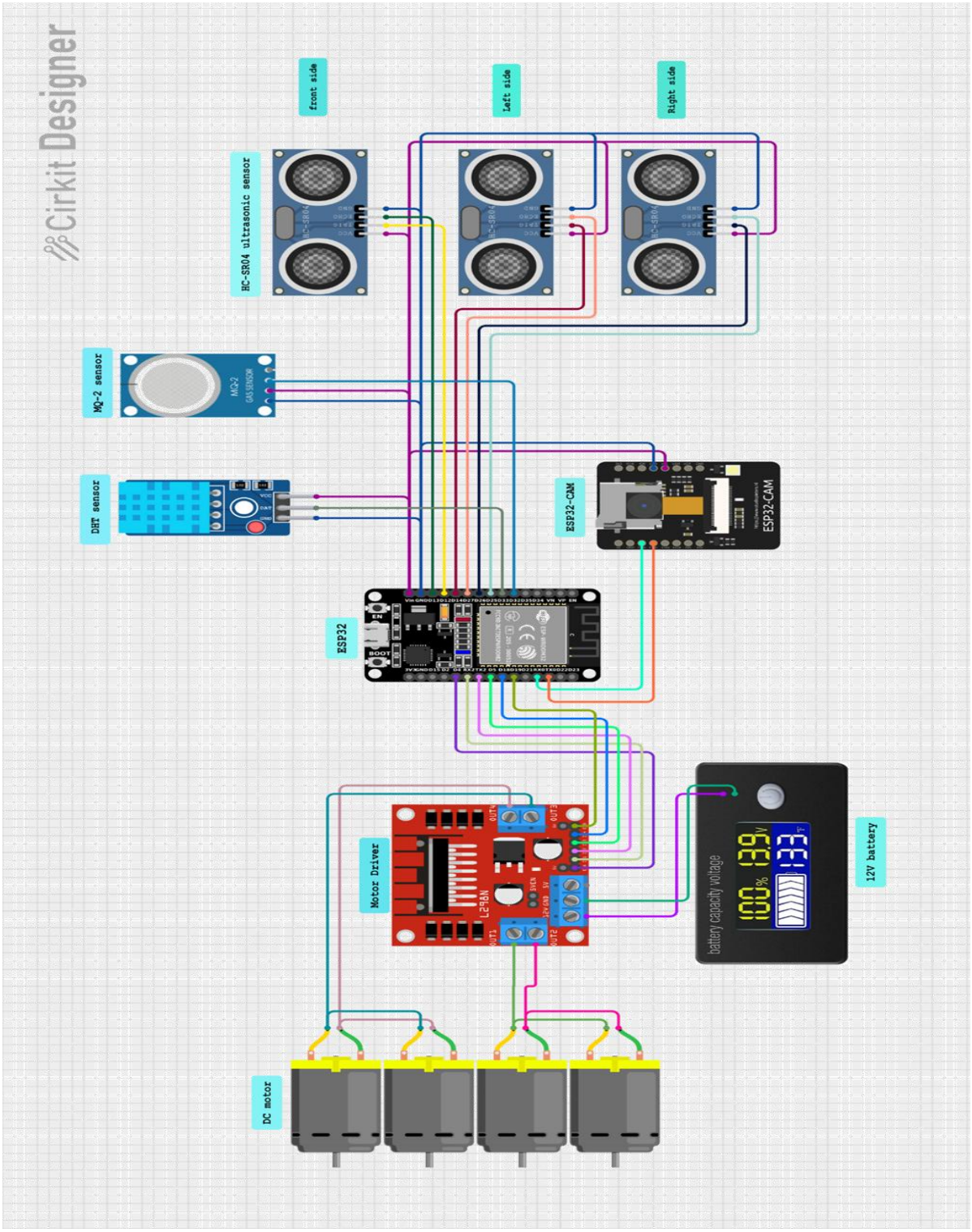
Chapter 4

WORKING:

4.1 Block Diagram:



4.2 SCHEMATIC DIAGRAM:



4.3 WORKING

1. Sensor Input and Data Acquisition

Sensors Involved:

- Environmental Sensors: DHT11 (temperature/humidity), MQ2 (gas levels)
- Position/Movement Sensors: MPU6050 (orientation, acceleration)
- Obstacle Detection: Ultrasonic sensors (front, left, right)

Operation:

Each sensor continuously monitors its respective parameter and sends real-time analog or digital signals to the microcontroller.

2. Microcontroller Working (ESP32)

Data Processing:

- The ESP32 reads the sensor values periodically.
- It filters the data to remove noise and performs simple computations.

Decision Making:

- It decides which movement commands to send based on this processed data.

3. Navigation Process

- Based on readings from ultrasonic sensors, the bot checks for obstacles.
- If an obstacle is detected too close, the controller triggers an immediate stop command.

Movement Commands:

- Commands such as “forward,” “backward,” “left,” or “right” are given by arrow keys from the device to the ESP32.
- These commands control the motor driver (L298N), which then drives the motors accordingly.

4. Output Command Execution

Motor Actions:

- The digital and PWM signals sent from the webserver to the motor driver determine the speed and direction of the motors.
- As a result, the bot moves in the desired direction while dynamically responding to obstacles.

5. Display and Data Transmission via Web Server

Real-Time Communication:

- The ESP32 connects to a WiFi network and runs an embedded HTTP server.
- Sensor data is formatted into a JSON object and transmitted via a WebSocket connection.

6.Web Interface:

- Users can access a simple web page to view live data, which includes sensor readings, the current status of motor actions, and optionally a live video feed from the ESP32-CAM.
- This interface allows remote monitoring and manual override if needed.

Chapter 6

EXPERIMENTAL RESULTS

5.1 ADVANTAGES

1] Real-Time Monitoring:
Continuously collects environmental data such as temperature, humidity, air quality, and gas levels for timely analysis and response.

2] Early Hazard Detection:
Detects pollution spikes, gas leaks, or unusual weather conditions, enabling early warnings and preventive actions.

3] IoT and Cloud Integration:
Transmits collected data to the cloud for real-time access, remote monitoring, and data logging, aiding decision-makers and researchers.

4] Scalability and Flexibility:
The modular design allows easy integration of additional sensors or components based on specific environmental needs.

5] Cost-Effective Solution:
Uses affordable hardware like ESP32, making it an economical option for large-scale environmental monitoring.

6] Support for Smart Cities and Sustainability:
Provides data that can be used for urban planning, environmental conservation, and creating smarter, greener cities.

7] Mobile and Versatile:
Can be deployed in various environments—urban, rural, or remote—making it adaptable for different use cases.

5.2 DISADVANTAGES

1] Limited Sensor Accuracy:
Low-cost sensors like DHT11 and MQ2 may not provide highly precise or professional-grade readings.

2] Power Dependency:
The bot's operation is limited by battery life, especially in remote areas without access to charging facilities.

3] Connectivity Issues:
Real-time data transmission relies on stable internet; poor network coverage can affect performance and cloud updates.

4] Navigation Challenges:
The bot may struggle in complex or uneven terrains without advanced path-planning or obstacle-avoidance systems.

5] Weather Sensitivity:
Harsh weather conditions like rain or extreme heat can damage sensors or affect accuracy and mobility.

6] Maintenance Requirements:
Regular maintenance may be needed to clean sensors, update software, or recharge batteries, especially for long-term use.

7] Limited Processing Power:
Microcontrollers like ESP32 have limited resources for complex data processing or advanced machine learning tasks.

5.3 APPLICATIONS

1] Air Quality Monitoring:
Measures pollution levels (e.g., CO, smoke, harmful gases) in urban or industrial areas to assess and improve air quality.

2] Disaster Management:
Detects hazardous gas leaks, fires, or abnormal weather changes to provide early warnings during natural or industrial disasters.

3] Smart Agriculture:
Monitors environmental conditions like temperature and humidity to support precision farming and crop health management.

4] Urban Planning and Smart Cities:
Provides real-time environmental data to city planners for managing pollution, traffic emissions, and green zones.

5] Remote Area Surveillance:
Useful in monitoring hard-to-reach or dangerous locations, such as forests, mines, or remote villages.

6] Industrial Safety Monitoring:
Detects gas leaks and unsafe environmental conditions in factories or chemical plants, improving worker safety.

7] Environmental Research:
Assists researchers in collecting real-time field data for climate studies, pollution tracking, and ecological analysis.

8] School and College Projects:
Serves as a hands-on learning tool for students working on IoT, robotics, and environmental science.

Chapter 6

CONCLUSIONS

The Environment Surveillance Bot project demonstrates an effective solution for real-time environmental monitoring using a combination of sensors, autonomous mobility, and IoT technology. It enables the detection of environmental changes and hazards such as pollution, gas leaks, and abnormal weather conditions, supporting safer living conditions and sustainable development. Its low-cost, scalable design makes it suitable for diverse applications, including smart cities, agriculture, disaster management, and research.

Looking ahead, the project can be enhanced with high-precision sensors, AI-based data analysis, and renewable energy integration like solar power. Additional features such as image recognition, mobile app control, and multi-bot networks can further expand its capabilities. These improvements will make the system more intelligent, energy-efficient, and adaptable to complex, real-world environments—ultimately contributing to a smarter and more sustainable planet.

REFERENCES

- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9522383
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7791255
- https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf