Smart Home Monitoring and Safety System using ESP32

Wireless Sensor Networks (DA232)

Innehåll

Introduction....3
System Overview....3
System Design....4
Hardware design....4
Software Design....5
Communication Design....6
Implementation....6
Evaluation....8
Discussion....9
Conclusion....9
References....9
User Manual....10

1. Introduction

Smart home systems are becoming increasingly important for improving safety, comfort, and energy efficiency in modern homes. However, many commercial systems are expensive and difficult to customize for small-scale or educational projects. The purpose of this project was to design and implement a low-cost smart indoor monitoring and safety system using an ESP32 microcontroller, different sensors, and wireless communication technologies.

The developed system monitors environmental conditions inside a house and reacts automatically when dangerous situations occur. The system can detect gas leakage, flame/fire, motion, room brightness, temperature, and humidity. Sensor data is sent wirelessly to a cloud dashboard where the user can monitor the house remotely in real time.

The project also includes automation features such as turning on a room light when motion is detected in a dark room, opening a window during dangerous conditions, and controlling a relay according to the room temperature. The ESP32 communicates with a cloud dashboard through Wi-Fi using HTTPS requests.

The main goal of the project was to combine wireless sensor networks, cloud communication, and smart automation into a single integrated smart home system.

2. System Overview

The system is based on an ESP32 microcontroller connected to multiple sensors and actuator. The ESP32 continuously reads sensor data, processes the information locally, and sends the results to a cloud dashboard. The implemented system includes the following sensors and components:

- DHT11 temperature and humidity sensor
- MQ-series gas sensor
- PIR motion sensor
- Flame sensor
- LDR light sensor
- LED
- Buzzer
- Relay module
- Servo motors

The ESP32 connects to a Wi-Fi network and communicates with the cloud dashboard hosted on cloud. Sensor information is sent to the dashboard every two seconds using HTTPS POST requests in JSON format. The system can detect dangerous gas levels, detect flame/fire, detect room movement, measure temperature and humidity, automatically control room lighting, open a window during danger situations, control a relay depending on temperature, receive remote door commands from the cloud

dashboards. The cloud dashboard allows the user to monitor the house remotely and interact with the system.

3. System Design

3.1 Hardware Design

The hardware design of the system was based on the ESP32 microcontroller, which acted as the central processing and communication unit. Multiple sensors and actuators were connected to the ESP32 in order to monitor indoor environmental conditions and automatically react to dangerous situations. The hardware connections used in the system are shown in Figure 2. Different GPIO pins were assigned depending on whether the connected device required analog or digital communication. The implemented hardware components included a DHT11 sensor for temperature and humidity measurement, an MQ-series gas sensor for gas leakage detection, a PIR sensor for motion detection, a flame sensor for fire detection, and an LDR sensor for measuring room brightness. In addition, several actuators were connected to the ESP32, including a relay module, buzzer, RGB LED, room light, and two servo motors. The gas sensor was connected using its analog output in order to continuously monitor gas levels and detect dangerous values. The PIR sensor was used to detect movement inside the room, while the LDR sensor was combined with the motion sensor to create automatic room-light control. The room light only activates when both darkness and movement are detected. A relay module was used to simulate automatic heater control depending on the measured temperature. Two servo motors were implemented in the system. The first servo motor controlled a window that automatically opens during dangerous situations such as gas leakage or flame detection to improve ventilation and safety. The second servo motor was used for remote door control through the cloud dashboard.

Component	ESP32 GPIO Pin	Signal Type
DHT11 Sensor	GPIO 4	Digital
Gas Sensor (MQ series)	GPIO 35	Analog
PIR Motion Sensor	GPIO 27	Digital
Flame Sensor	GPIO 13	Digital
LDR Sensor	GPIO 34	Analog
Buzzer	GPIO 25	Digital Output
RGB LED – Red	GPIO 16	Digital Output
RGB LED – Green	GPIO 17	Digital Output
RGB LED – Blue	GPIO 18	Digital Output
Relay Module	GPIO 23	Digital Output
Window Servo Motor	GPIO 19	PWM
Door Servo Motor	GPIO 14	PWM
Room Light (LED)	GPIO 22	Digital Output

Figure 1:GPIO connections between the ESP32, sensors, and actuator used in the smart home monitoring and automation system.

Figure 2 illustrates the hardware connections between the ESP32, sensors, and actuators used in the smart home monitoring system.

3.2 Software Design

The software of the smart home monitoring and safety system was developed using Arduino IDE in C++ for the ESP32 microcontroller. The program is responsible for reading sensor values, processing the data, controlling actuators, and communicating with the cloud dashboard through Wi-Fi. The flowchart for the code is showed in figure 2. When the system starts, the ESP32 initializes the sensors, GPIO pins, servo motors, and Wi-Fi connection. After connecting to the network, the ESP32 continuously reads data from the DHT11 sensor, gas sensor, PIR motion sensor, flame sensor, and LDR sensor. The sensor data is then processed to determine the current condition of the environment. The software uses conditional logic to control the system automatically. If the room is dark and motion is detected, the room light turns on automatically. If dangerous gas levels or flame are detected, the system activates the buzzer, changes the RGB LED status, and opens the window servo for safety purposes. The relay module is also controlled according to the measured temperature. The ESP32 sends sensor data and actuator states to the cloud dashboard every two seconds using HTTPS POST requests. The cloud dashboard can also send commands back to the ESP32, such as opening or closing the door servo remotely.

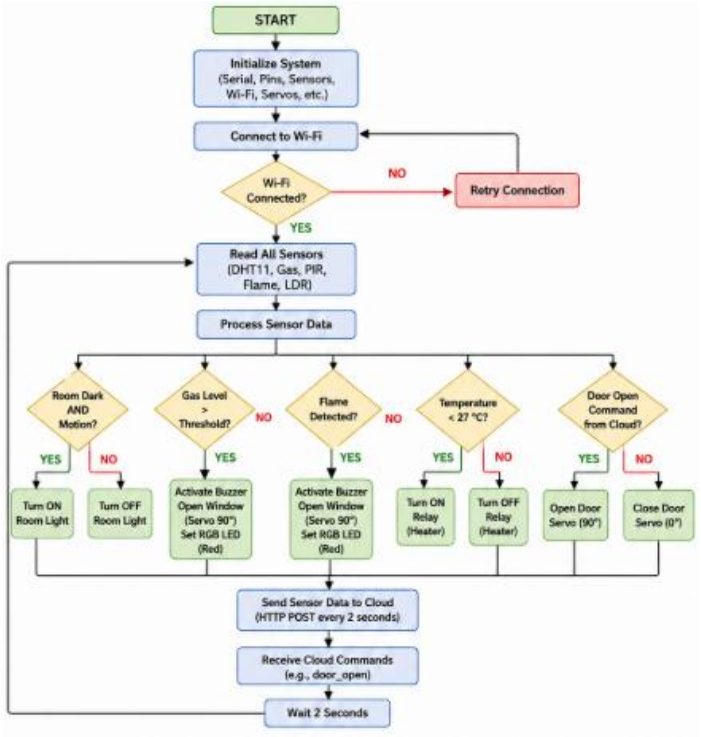

Figure 2: The flowchart

3.3 Communication design

The communication system of the project was implemented using Wi-Fi communication between the ESP32 microcontroller and the cloud dashboard. The ESP32 connects to the local Wi-Fi network and sends sensor data to the cloud through HTTPS requests. This allows the user to monitor the smart home system remotely using a mobile or web dashboard. The communication was implemented using the WiFi, HTTPClient, and WiFiClientSecure libraries in Arduino IDE. After connecting to the Wi-Fi network, the ESP32 continuously sends updated sensor values to the cloud every two seconds. The transmitted data includes temperature, humidity, gas level, motion status, flame detection, room brightness, relay status, and actuator states. Sensor data is formatted into a JSON payload before being transmitted to the cloud server using HTTP POST requests. This allows structured communication between the ESP32 and the dashboard. The system also receives commands from the cloud dashboard, such as controlling the door servo remotely. Wi-Fi communication was selected because it allows long-distance remote monitoring through the internet without requiring direct physical connection to the system. HTTPS communication was used to improve communication reliability and provide secure data transmission between the ESP32 and the cloud server.

4. Implementation

The implementation of the smart home monitoring and safety system was carried out incrementally by integrating and testing one component at a time. The ESP32

microcontroller was first connected to the Wi-Fi network to verify stable communication with the cloud dashboard. After successful communication was established, the sensors and actuators were connected and tested individually using the Arduino Serial Monitor.

The developed prototype of the system is shown in Figure 3. The prototype includes the ESP32, sensors, servo motors, relay module, RGB LED, and buzzer connected on a breadboard for testing and development purposes.

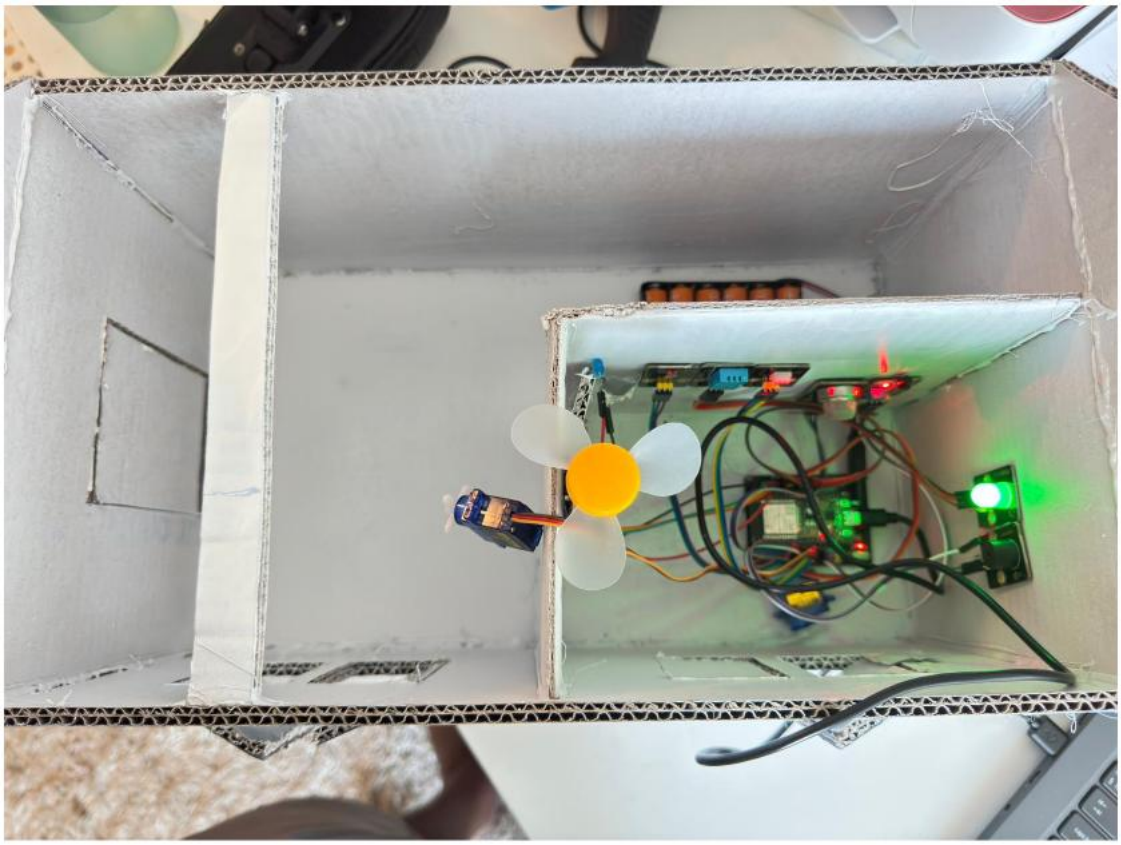

Figure 3:Prototype implementation.

The implementation started with sensor integration. The DHT11 sensor was used for measuring temperature and humidity. The corresponding code used to read the temperature and humidity values is shown in Figure 4.

```
float temperature = dht.readTemperature();
float humidity    = dht.readHumidity();
```

Figure 4:Code segment used for temperature and humidity measurement using the DHT11 sensor.

The PIR motion sensor and LDR sensor were combined to create automatic room-light control. The implemented logic ensures that the room light only turns on when the room is dark and movement is detected. The corresponding implementation is shown in Figure 5. This design helps reduce unnecessary energy consumption.

```
bool roomDark       = (ldrValue < 2500);
bool motionDetected = (motionValue == HIGH);
bool roomLightOn    = (roomDark && motionDetected);
```

Figure 5:Code segment implementing automatic room-light control using the PIR motion sensor and LDR sensor.

The gas sensor and flame sensor were integrated for safety monitoring. The system continuously checks whether dangerous conditions are detected. The gas and flame detection logic is shown in Figure 6. If either gas leakage or flame is detected, the system activates the danger mode. During danger mode, the buzzer turns on, the RGB LED changes state, and the window servo automatically opens to improve ventilation.

```
bool flameDetected  = (flameValue == LOW);
bool gasDetected    = (gasValue > 1300);
bool dangerDetected = gasDetected || flameDetected;
```

Figure 6:Code segment implementing gas and flame detection logic.

If either gas leakage or flame is detected, the system activates the danger mode. During danger mode, the buzzer turns on, the RGB LED changes state and an automatic window-opening mechanism which is controlled by a servo motor. The implementation used for opening the window during dangerous situations is shown in Figure 7. When the dangerous condition disappears, the window servo automatically closes again.

```
bool windowOpen;
if (dangerDetected) {
  digitalWrite(buzzerPin, HIGH);
  setRGB(LOW, HIGH, LOW);
  windowServo.write(90);
  windowOpen = true;
  Serial.println("Danger! Buzzer ON, Window OPEN");
```

Figure 7:Code segment controlling automatic window opening during danger conditions.

The relay module was implemented for automatic temperature-based fan control. The relay activates when the measured temperature becomes lower than 27°C. The corresponding implementation is shown in Figure 8. This simulates automatic fan control inside the smart home environment.

```
if (temperature < 27) {
  digitalWrite(relayPin, HIGH);
  relayOn = true;
  Serial.println("Temp < 27C -> Relay ON");
```

Figure 8:Code segment implementing temperature-based relay control.

Cloud communication was implemented using HTTPS POST requests. Sensor values and actuator states are converted into a JSON payload and transmitted to the cloud dashboard every two seconds. The communication logic is shown in figure 9.

```
const unsigned long SEND_INTERVAL_MS = 2000;
```

Figure 9:Code segment implementing cloud communication and data transmission.

The ESP32 also receives commands from the cloud dashboard, allowing remote control of the door servo motor. The system checks the server response and updates the door position automatically. Throughout the implementation process, the Serial Monitor was used extensively for debugging, monitoring sensor values, and verifying Wi-Fi communication. Each sensor and actuator was tested individually before integrating the complete system together.

5. Evaluation

The system was evaluated by testing each sensor, actuator, and communication function individually before testing the complete integrated system. The Arduino Serial Monitor was used to verify sensor readings and communication messages, while the cloud dashboard was used to monitor sensor values and actuator states remotely. The DHT11 sensor was tested by comparing the measured temperature and humidity values with the surrounding room conditions. The PIR motion sensor was tested by creating movement in front of the sensor and verifying that motion was detected correctly. The LDR sensor was tested under different lighting conditions to ensure that the room light only activated when the room was dark and motion was present. The gas sensor and flame sensor were tested by exposing them to gas and flame sources, respectively. During testing, the system successfully entered danger mode, activated the buzzer, changed the RGB LED status, and opened the window servo. The relay module connected to the fan was also tested by changing the temperature conditions and observing the relay response. Figure 10 and Figure 11 show the cloud dashboard used during testing and evaluation. The dashboard displayed real-time information including temperature, humidity, gas status, motion detection, relay status, door state, and window state, allowing the functionality of the complete system to be verified. The results showed that the system functioned as intended and successfully integrated sensing, communication, and automatic control. Sensor values were updated on the dashboard every two seconds, providing near real-time monitoring of the environment. Communication between the ESP32 and the cloud dashboard remained stable throughout testing, and all actuators responded correctly to both sensor inputs and cloud commands. The system demonstrated good reliability during normal operation, although small delays were occasionally observed due to Wi-Fi and cloud communication. These delays were typically only a few seconds and did not significantly affect the performance of the system. Overall, the update rate, reliability, and response time were considered suitable for a smart home monitoring and safety application.

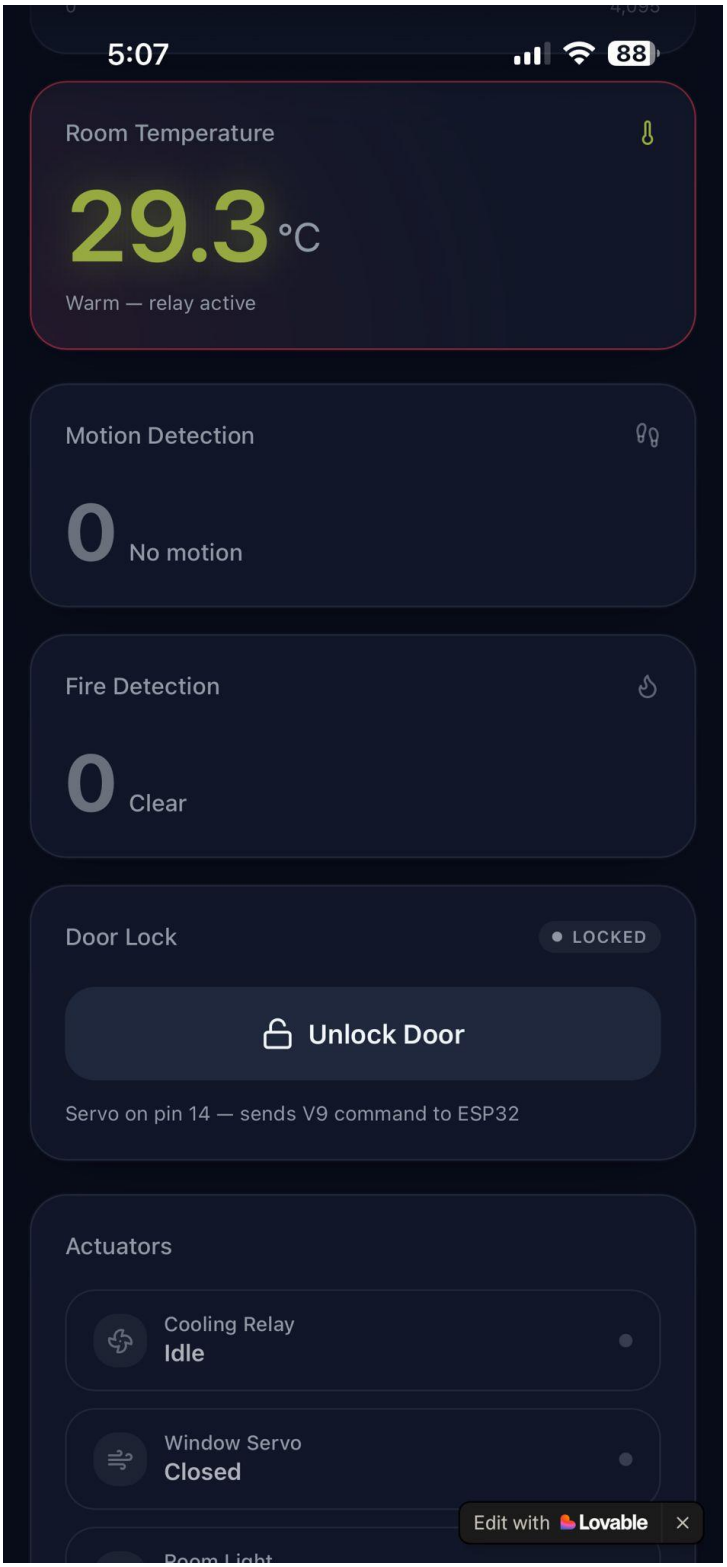

Figure 10: Cloud dashboard interface used for real-time monitoring and control of the smart home.



Figure 11: Cloud dashboard interface used for real-time monitoring and control of the smart home.

6. Discussion

The implemented smart home monitoring system successfully demonstrated wireless monitoring, automatic control, and cloud communication using the ESP32 microcontroller. The results showed that the system could reliably detect gas leakage, flame, motion, temperature, humidity, and room brightness while automatically controlling actuators such as the buzzer, relay, room light, and servo motors. One important observation was that combining the PIR sensor and the LDR sensor improved energy efficiency because the room light only activated when both darkness and movement were detected. The cloud communication also functioned reliably and allowed remote monitoring and control through the dashboard. Despite the successful implementation, the system has some limitations. The DHT11 sensor has limited measurement accuracy and relatively slow response time. The gas sensor also requires warm-up time before stable readings can be obtained and consumes relatively high power due to its internal heating element. In addition, the system depends on a stable Wi-Fi connection for cloud communication. Some possible sources of error include unstable sensor readings, Wi-Fi interruptions, sensor noise, and environmental conditions affecting the measurements. The gas sensor readings may also vary depending on temperature and air conditions. Possible future improvements include using more accurate sensors, adding battery operation with power-saving modes, supporting multiple temperature sensors, implementing mobile notifications, and improving the dashboard with more advanced monitoring features.

7. Conclusion

In this project, a smart indoor monitoring and safety system was successfully designed and implemented using ESP32 and IoT technologies. The system integrated multiple sensors and actuators to monitor environmental conditions and automatically react to dangerous situations. The implemented system successfully detected gas leakage, flame, motion, room brightness, temperature, and humidity while communicating with a cloud dashboard through Wi-Fi. Automatic functions such as room-light control, window opening during danger conditions, relay control, and remote door operation were also implemented successfully. The project demonstrated reliable wireless communication, real-time monitoring, and automatic smart home functionality using low-cost hardware components. Overall, the project achieved the intended goals of indoor safety monitoring, automation, and remote cloud-based control.

8. References

- ESP32 Documentation
- Arduino IDE Documentation
- DHT11 Sensor Documentation
- ESP32Servo Library Documentation
- WiFi.h Library Documentation
- HTTPClient Library Documentation
- Lovable Cloud Documentation
- Course Material

9. User Manual

9.1 System Overview

The smart home monitoring and safety system is based on an ESP32 microcontroller connected to multiple sensors and actuators. The system monitors environmental conditions such as temperature, humidity, gas leakage, flame detection, room brightness, and motion detection. Sensor data is sent wirelessly to a cloud dashboard through Wi-Fi, allowing the user to monitor and control the system remotely.

The system can also automatically react to dangerous situations by activating a buzzer, changing RGB LED status, opening a window servo, controlling room lighting, and operating a relay module.

9.2 Hardware and Software Requirements

Hardware Requirements

- ESP32 development board
- DHT11 temperature and humidity sensor
- MQ-series gas sensor
- PIR motion sensor
- Flame sensor
- LDR sensor

- Servo motors
- Relay module
- RGB LED
- Buzzer
- Breadboard and jumper wires
- Wi-Fi connection
- USB cable

Software Requirements

- Arduino IDE
- ESP32 board package
- WiFi library
- HTTPClient library
- WiFiClientSecure library
- DHT sensor library
- ESP32Servo library

9.3 Installation Instructions

1. Install Arduino IDE on the computer.
2. Install the ESP32 board package in Arduino IDE.
3. Install all required libraries.
4. Connect all sensors and actuators to the ESP32 according to the GPIO connection table shown in Figure 2.
5. Open the project code in Arduino IDE.
6. Enter the Wi-Fi name and password in the code.
7. Configure the cloud server information and device secret.
8. Connect the ESP32 to the computer using a USB cable.
9. Upload the code to the ESP32.

9.4 Running the System

1. Open the Arduino Serial Monitor and select a baud rate of 115200.
2. Power on the ESP32.
3. Wait for the ESP32 to connect to the Wi-Fi network.
4. After successful connection, the ESP32 starts reading sensor values and sending data to the cloud dashboard every two seconds.
5. Open the cloud dashboard to monitor sensor readings and system status.
6. Test the sensors by creating movement, changing light conditions, or exposing the gas/flame sensors carefully to verify the system response.

During danger situations:

- the buzzer activates,
- the RGB LED changes status,
- and the window servo opens automatically.

The cloud dashboard can also remotely control the door servo.

9.5 Troubleshooting

In figure 12 is it possible to find the solutions for the most common problems.

Problem	Possible Cause	Solution
ESP32 does not connect to Wi-Fi	• Wrong Wi-Fi name or password • Weak Wi-Fi signal • Router issues	• Check Wi-Fi name and password in the code • Move ESP32 closer to the router • Restart the router and try again
No sensor values on dashboard	• Internet connection problem • Wrong server URL or device secret • Firewall blocking connection	• Check internet connection • Verify server URL and device secret in the code • Disable firewall temporarily and try again
Gas sensor readings unstable or high	• Sensor warm-up not completed • Sensor exposed to strong gas or air flow • Sensor aging	• Allow the gas sensor to warm up (2–5 minutes) • Test in normal room air • Replace sensor if readings remain inaccurate
DHT11 sensor not responding or wrong values	• Wrong wiring • Loose connection • Sensor damaged	• Check VCC, GND and DATA connections • Use a pull-up resistor (10kΩ) if needed • Replace the DHT11 sensor
Servo motor not moving	• Wrong GPIO connection • Insufficient power supply • Servo horn not attached properly	• Check servo signal wire connection • Use external 5V power supply if needed • Reattach servo horn and try again
Room light not turning on automatically	• PIR or LDR not positioned correctly • Wrong threshold value • Wiring or relay connection issue	• Adjust sensor position and test again • Check and adjust threshold value in the code • Verify relay wiring and power supply
Upload failed in Arduino IDE	• Wrong COM port selected • USB cable issue • Board driver not installed	• Select the correct COM port • Use a data USB cable • Install ESP32 board package and drivers

Figure 12:Troubleshooting.