ABSTRACT

Water scarcity and efficient water resource management have become critical issues in modern society. This paper presents an Internet of Things (IoT)-based water level monitoring system designed to address these challenges by providing real-time water level data for reservoirs, tanks, and other storage systems. The proposed system utilizes IoT-enabled sensors to continuously monitor water levels and transmit the data to a central platform through wireless communication. The system employs microcontrollers integrated with ultrasonic or capacitive sensors to measure water levels accurately and efficiently. The collected data is sent to a cloud-based platform where it is processed, stored, and visualized through a user-friendly interface accessible via mobile or web applications. This real-time monitoring enables users to track water levels remotely, set threshold alerts, and optimize water usage. Additionally, predictive analytics can help in identifying trends and preventing water wastage or overflow.

SYNOPSIS

The IoT-based water level monitoring system is designed to address the challenges of water resource management by providing a smart, automated solution for real-time monitoring and efficient utilization of water. Traditional methods of tracking water levels are often manual, prone to errors, and inefficient. This system leverages IoT technology to provide accurate, remote monitoring of water levels in reservoirs, tanks, and other storage systems. It consists of sensors, such as ultrasonic or capacitive sensors, connected to a microcontroller (e.g., Arduino or ESP32), which processes the data and transmits it to a cloud-based platform via wireless communication. The system enables users to remotely access data through mobile or web applications, set threshold alerts, and visualize water level trends in real time. Its features include configurable notifications for critical water levels, predictive analytics for maintenance, and a scalable, energy-efficient design. This solution finds applications in households, agriculture, industrial setups, and municipal water supply systems. By promoting water conservation and efficient usage, the IoT-based water level monitoring system contributes to sustainable resource management and addresses both individual and societal needs.

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. IoT - Internet of Things
2. ESP32 - Espressif Systems 32-bit Microcontroller
3. Wi-Fi - Wireless Fidelity
4. GPIO - General Purpose Input/output
5. IDE - Integrated Development Environment
6. PWM - Pulse Width Modulation
7. RFID - Radio Frequency Identification
8. AC – Alternative Current
9. DC – Direct Current
10. I2C - Inter-Integrated Circuit
11. SPI - Serial Peripheral Interface
12. CPU - Central Processing Unit
13. UART - Universal Asynchronous Receiver-Transmitter

TABLE OF CONTENTS

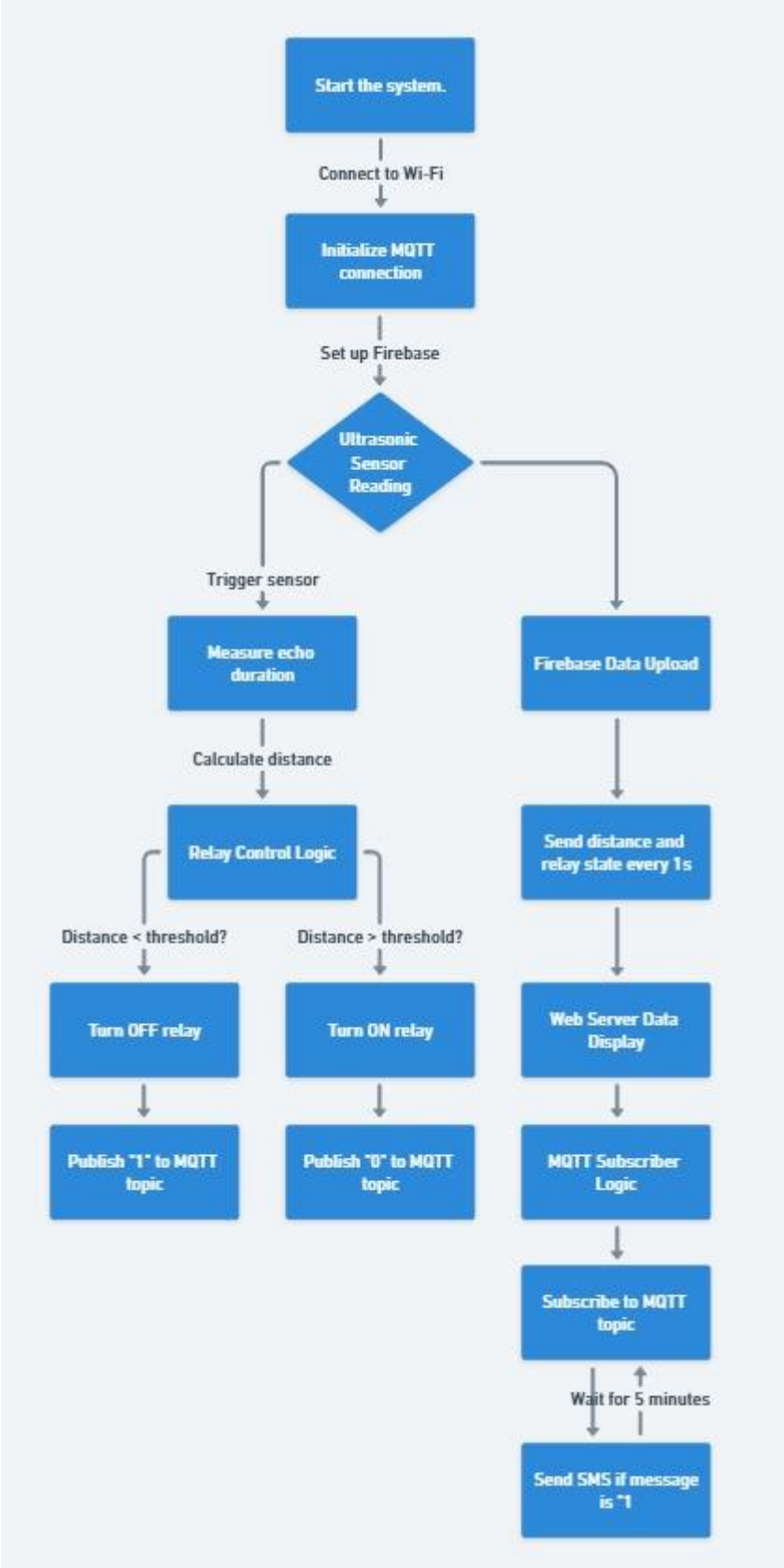
CHAPTER – 1

INTRODUCTION

1.1 INTRODUCTION

Water is a critical resource for life, yet its scarcity and inefficient management remain global challenges. With growing populations and increased demand for water in households, agriculture, and industries, the need for innovative solutions to optimize water usage has never been greater. Traditional methods of water level monitoring, often manual and error-prone, fail to provide timely and accurate data necessary for effective resource management. An Internet of Things (IoT)-based water level monitoring system offers a modern, automated solution to this problem. By integrating advanced sensors, microcontrollers, and cloud-based platforms, the system enables real-time monitoring and remote access to water level data. Users can receive alerts when critical water levels are reached, visualize usage trends, and make informed decisions to prevent wastage and ensure continuous availability.

1.2 BLOCK DIAGRAM



1.3 CONCLUSION

In conclusion, an IoT-based water level monitoring system presents a transformative approach to addressing water management challenges. By enabling real-time monitoring, remote access, and timely alerts, the system ensures efficient utilization of water resources while reducing wastage and preventing overflows. Its adaptability to various applications—ranging from households to large-scale industrial and municipal systems—demonstrates its versatility and importance in promoting sustainable practices. With the integration of IoT technology, this system not only simplifies water level monitoring but also contributes to a more efficient, data-driven approach to water conservation and resource management.

CHAPTER – 2

2.1 INTRODUCTION

Over the years, various approaches have been developed to monitor water levels and manage water resources effectively. Traditional methods often relied on manual inspection and mechanical systems, which were labor-intensive, prone to errors, and lacked real-time monitoring capabilities. As technology advanced, automated systems incorporating electronic sensors and basic data logging were introduced, offering improved accuracy and reduced manual effort. However, these systems were often limited to local monitoring and lacked remote accessibility. With the advent of the Internet of Things (IoT), water level monitoring has undergone significant transformation. Researchers and developers have explored IoT-enabled solutions that integrate sensors, microcontrollers, and cloud computing to provide real-time data access and remote monitoring. Many of these systems use wireless communication protocols like Wi-Fi, LoRa, or GSM to transmit data, coupled with mobile and web applications for user interaction. These solutions have demonstrated enhanced efficiency and scalability, particularly for large-scale applications in agriculture, industries, and municipal systems..

2.2.1 PREVIOUS WORKS

1. Akhil, R., & Kumar, S. (2020)

Automated Water Level Control Using IoT, published in the International Journal of Engineering and Advanced Technology (IJEAT). This study focuses on IoT-based water level monitoring systems to automate water pump operations and ensure efficient water management.

2. Gupta, R., & Jain, A. (2022)

Design and Implementation of a Low-Cost Water Level Monitoring System, published in the International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE). The authors propose a low-cost water level monitoring system using microcontrollers and sensors, emphasizing affordability and reliability.

3. Desai, P., & Sharma, N. (2021)

Real-Time Water Level Monitoring Using Sensors, published in the Journal of Sensor Applications. This paper discusses the integration of sensors for real-time monitoring and provides a framework for automation.

4. Patel, V., & Mehta, S. (2019)

Smart Water Management System Using IoT, published in the International Journal of Smart Innovations. The study explores IoT-enabled water management systems with features like remote monitoring and predictive analytics.

5. Thomas, J., & Roy, A. (2020)

A Sensor-Based Approach to Automated Water Level Detection, published in the Journal of Automation and Control Systems. This work highlights the use of ultrasonic sensors for precise water level measurement and automation in water tanks. These works provide a foundation for understanding the advancements and methodologies in automated water level monitoring systems.

2.2 SUMMARY

The previous work in the field of water level monitoring systems highlights various approaches to automation and IoT integration for efficient water management. Akhil and Kumar (2020) focused on the use of IoT to automate water pump control, ensuring optimized water usage and preventing overflow. Gupta and Jain (2022) proposed a cost-effective water level monitoring system, emphasizing the affordability and reliability of using microcontrollers and sensors. Desai and Sharma (2021) explored real-time water level monitoring through sensor-based solutions, while Patel and Mehta (2019) expanded on the use of IoT for smart water management, incorporating remote monitoring and predictive capabilities. Additionally, Thomas and Roy (2020) demonstrated the application of ultrasonic sensors for precise and automated water level detection. These studies collectively contribute to the development of efficient, automated systems for water management, laying the groundwork for the proposed water level monitoring system.

CHAPTER – 3

PROPOSED WORK

3.1 INTRODUCTION

Water is an essential resource, and efficient management of water levels in tanks is crucial for both residential and industrial applications. A reliable water level monitoring system can help automate water management, reduce human intervention, prevent overflow, and avoid dry run situations for pumps. The proposed work involves designing a Water Level Monitoring System using an ESP32 microcontroller, an ultrasonic sensor, and a relay module. The system will utilize the ultrasonic sensor to continuously monitor the water level in a tank. The ESP32, with its built-in Wi-Fi capability, will process the sensor data and control a relay module connected to a water pump. This will allow the system to automatically fill or stop water flow based on the detected water level. The system can be used in various applications, such as household water tanks, industrial water reservoirs, or agricultural water management systems.

3.2 METHODOLOGY

The methodology for the proposed Water Level Monitoring System using ESP32, an ultrasonic sensor, and a relay module involves several key stages to ensure the system operates effectively. The first step is requirement analysis and design, where the necessary components are selected based on the system's needs, including the ESP32 microcontroller, HC-SR04 ultrasonic sensor, relay module, and a suitable water pump. The circuit design follows, where the components are connected: the ultrasonic sensor to the ESP32 for water level measurement, the relay module to control the pump, and power considerations for all components. Next, software development involves writing code for the ESP32 to handle sensor data processing, water level calculations, and relay control. The ESP32 reads the sensor distance, calculates the water level, and compares it with predefined thresholds to control the pump. The system is then integrated, where all hardware components are connected, and testing is conducted to ensure correct operation. The system is evaluated under various conditions, including low and high water levels, to ensure the pump activates and deactivates appropriately. Finally, the system undergoes optimization for performance and power efficiency, and the code is refined to ensure smooth operation. If necessary, a remote monitoring interface, such as a web or mobile app, can be integrated, leveraging the ESP32's Wi-Fi capabilities. The methodology concludes with final testing and debugging, where the system's reliability and efficiency are verified before deployment.

3.3 CIRCUIT DIAGRAM AND EXPLANATION

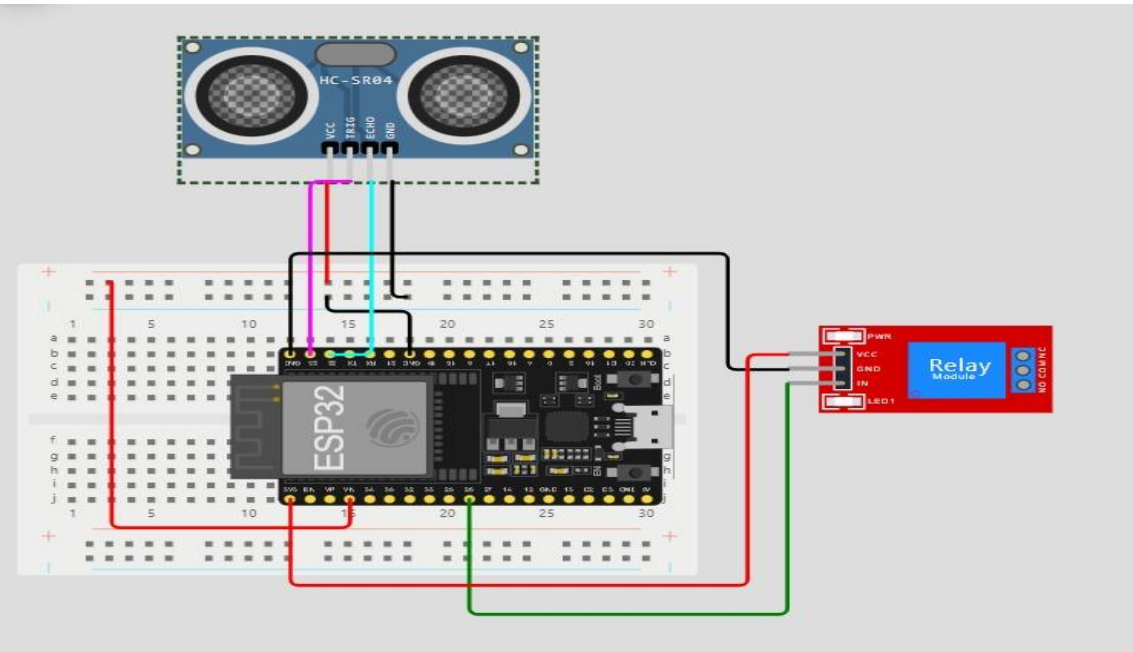


Fig 3.3 Circuit diagram

In fig 3.3 the circuit diagram of the proposed Water Level Monitoring System consists of key components: the ESP32 microcontroller, the HC-SR04 ultrasonic sensor, the relay module, and the water pump. The ESP32 is the central control unit, responsible for processing the data from the ultrasonic sensor and controlling the relay module to manage the water pump. The ultrasonic sensor is connected to the ESP32, with the Trig pin connected to a GPIO pin (e.g., GPIO 5) and the Echo pin connected to another GPIO pin (e.g., GPIO 18). The sensor measures the distance to the water surface and provides the data to the ESP32, which calculates the water level. Based on the measured water level, the ESP32 sends a control signal to the relay module connected to GPIO 23. The relay acts as an electronic switch, turning the water pump on or off by controlling the power supply to the pump. The pump is connected to the relay's normally open (NO) and common (COM) terminals, which allow current to flow when the relay is triggered. The system is powered by two separate power supplies: a 5V supply

for the ESP32 and the sensor, and a higher voltage supply (e.g., 12V) for the water pump. This circuit setup allows the ESP32 to continuously monitor and control the water level, automating the operation of the water pump based on the threshold conditions.

3.4 CODING

3.4.1 ARDUINO CODE

```
#include <Arduino.h>

#include <WiFi.h>

#include <Firebase_ESP_Client.h>

#include <WebServer.h>

#include <PubSubClient.h>


#define TRIG_PIN 4

#define ECHO_PIN 5

#define RELAY_PIN 26

#define CLOSE_DISTANCE 5  // Distance in cm to trigger relay OFF


WebServer server(80);


// MQTT Configuration

const char* mqtt_broker = "broker.hivemq.com";

const int mqtt_port = 1883;
```

```
const char* mqtt_topic = "esp32/relay_status";

WiFiClient espClient;

PubSubClient mqtt_client(espClient);


#include "addons/TokenHelper.h"

#include "addons/RTDBHelper.h"


#define WIFI_SSID "Aswin"

#define WIFI_PASSWORD "AK04@011"


#define API_KEY "AIzaSyAZiwEWb7rEHtml1KqOV6iapEQ8Yorc2c8"

#define DATABASE_URL "https://iotpro-ea2f4-default-rtdb.asia-southeast1.firebasedatabase.app/"


FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;


unsigned long sendDataPrevMillis = 0;

bool signupOK = false;

bool lastRelayState = true;  // To track relay state changes
```

```
float distance = 0.0;
bool relayState = true;      // Starting with relay ON

void reconnectMQTT() {
  while (!mqtt_client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client-" + String(random(0xffff), HEX);
    if (mqtt_client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Publish initial state after reconnecting
      mqtt_client.publish(mqtt_topic, "0");
    } else {
      Serial.print("failed, rc=");
      Serial.print(mqtt_client.state());
      Serial.println(" retrying in 5 seconds");
      delay(5000);
    }
  }
}

float measureDistance() {
```

```
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  return duration * 0.034 / 2;
}

void sendSensorData() {
  String json = "{";
  json += "\"distance\":" + String(distance) + ",";
  json += "\"relayState\":" + String(relayState ? "true" : "false");
  json += "}";
  server.send(200, "application/json", json);
}
void handleRoot() {
  String html = R"====(
    <!DOCTYPE html>
    <html>
```

```
<head>
  <title>Ultrasonic Sensor Dashboard</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      background: linear-gradient(to bottom, #1abc9c, #16a085);
      color: white;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 600px;
      margin: auto;
      padding: 20px;
      background: #ffffff10;
      border-radius: 15px;
      box-shadow: 0px 4px 20px rgba(0,0,0,0.3);
      margin-top: 50px;
    }
    .heading {
```

```
  font-size: 36px;
  margin-bottom: 20px;
  color: #f1c40f;
  text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.7);
}
.circle {
  width: 150px;
  height: 150px;
  border-radius: 50%;
  background: #3498db;
  color: #fff;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 32px;
  font-weight: bold;
  margin: 20px auto;
  box-shadow: 0px 4px 15px rgba(0,0,0,0.4);
  border: 4px solid #ffffff80;
}
.data {
```

```
  font-size: 24px;
  margin: 15px 0;
}
.relay {
  font-size: 28px;
  font-weight: bold;
  margin-top: 20px;
}
.relay.on {
  color: #2ecc71;
}
.relay.off {
  color: #e74c3c;
}
.relay.on::after {
  content: " ✅";
}
.relay.off::after {
  content: " ❌";
}
.footer {
```

```
      margin-top: 20px;
      font-size: 14px;
      color: #ecf0f1;
    }
  </style>
  <script>
    async function fetchSensorData() {
      const response = await fetch('/sensor');
      const data = await response.json();

      document.getElementById('distance').innerText = data.distance.toFixed(2) + " cm";
      const relayElement = document.getElementById('relayState');
      relayElement.innerText = data.relayState ? "Relay ON" : "Relay OFF";
      relayElement.className = "relay " + (data.relayState ? "on" : "off");
    }

    setInterval(fetchSensorData, 1000);
    window.onload = fetchSensorData;
  </script>
</head>
```

```
    <body>
      <div class="container">
        <h1 class="heading">Ultrasonic Sensor Dashboard</h1>
        <div class="circle" id="distance">-- cm</div>
        <p id="relayState" class="relay">--</p>
        <p class="footer">Real-time Monitoring</p>
      </div>
    </body>
    </html>
  )====";

  server.send(200, "text/html", html);
}
void setup() {
  Serial.begin(115200);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(RELAY_PIN, OUTPUT);

  // Initialize relay to ON state
```

```
digitalWrite(RELAY_PIN, LOW);
relayState = true;

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());

// Initialize MQTT and publish initial state
mqtt_client.setServer(mqtt_broker, mqtt_port);
reconnectMQTT();
mqtt_client.publish(mqtt_topic, "0");  // Initial state is ON (0)
Serial.println("Published initial state: 0 (ON)");

server.on("/", handleRoot);
server.on("/sensor", sendSensorData);
```

```
  server.begin();
  Serial.println("Web server started");

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;

  if (Firebase.signUp(&config, &auth, "", "")) {
    Serial.println("Sign up successful!");
    signupOK = true;
  } else {
    Serial.printf("Sign up error: %s\n", config.signer.signupError.message.c_str());
  }

  config.token_status_callback = tokenStatusCallback;
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

void loop() {
  server.handleClient();
```

```
  if (!mqtt_client.connected()) {
    reconnectMQTT();
  }
  mqtt_client.loop();

  distance = measureDistance();

  // Update relay state based on distance
  if (distance < CLOSE_DISTANCE) {  // Object is too close
    if (relayState) {  // Only publish and change state if it was previously ON
      relayState = false;
      digitalWrite(RELAY_PIN, HIGH);
      mqtt_client.publish(mqtt_topic, "1");
      Serial.println("Object too close! Relay OFF - Published '1'");
    }
  } else {  // Normal distance
    if (!relayState) {  // Only publish and change state if it was previously OFF
      relayState = true;
      digitalWrite(RELAY_PIN, LOW);
      mqtt_client.publish(mqtt_topic, "0");
      Serial.println("Normal distance - Relay ON - Published '0'");
```

```
    }
  }

  if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 1000 || sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();
    Firebase.RTDB.setFloat(&fbdo, "Ultrasonic/Distance", distance);
    Firebase.RTDB.setBool(&fbdo, "Relay/State", relayState);
  }
}
```

3.4.2 SERVER CODE FOR SMS (PYTHON)

```
import paho.mqtt.client as mqtt
import time
from twilio.rest import Client

# MQTT callback functions
def on_connect(client, userdata, flags, rc):
```

```
    if rc == 0:
        print("Connected to MQTT broker successfully")
        # Subscribe to the desired topic
        client.subscribe("esp32/relay_status")
    else:
        print(f"Failed to connect, return code {rc}")

def send_sms_twilio(account_sid, auth_token, from_number, to_number, message):
    client = Client(account_sid, auth_token)

    try:
        message = client.messages.create(
            body=message,
            from_=from_number,
            to=to_number
        )
        print(f"Message sent successfully: {message.sid}")
    except Exception as e:
        print(f"Failed to send message: {e}")

# Example usage
```

```
    account_sid = 'ACcdcf5eb7a1ec1782b74663229403cd13'  # Replace with your Account SID

    auth_token = '9d8bf016b4d0203ca865bf5a533bae83'      # Replace with your Auth Token

    from_number = '+12245400027'         # Replace with your Twilio phone number

    to_number = '+919943413269'          # Replace with the recipient's phone number

    message = 'Water Tank is full...!!! Please Turn of the Motor'


    def on_message(client, userdata, msg):

       try:

          # Decode the received MQTT message

          data = msg.payload.decode()

          print(f"Subscribed Data: {data}")


          if data == "1":

             send_sms_twilio(account_sid, auth_token, from_number, to_number,message)

             print("Data is 1, leaving loop for 5 minutes...")

             # Stop listening and pause for 5 minutes

             client.loop_stop()

             time.sleep(300)  # Pause for 300 seconds (5 minutes)

             print("Re-entering the loop...")
```

```
            client.loop_start()  # Restart listening
    except Exception as e:
        print(f"Error processing message: {e}")

# Main MQTT setup
broker = "broker.hivemq.com"  # Replace with your MQTT broker address
port = 1883               # Replace with your broker's port if needed

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

try:
    # Connect to the MQTT broker
    client.connect(broker, port, 60)
    print("Connecting to broker...")
    # Start the MQTT loop to listen for messages
    client.loop_start()  # Start the loop in the background
    while True:
        time.sleep(1)  # Keep the script alive
except Exception as e:
```

```
        print(f"Failed to connect to MQTT broker: {e}")
    finally:
        client.loop_stop()
        client.disconnect()
```

3.5 CONCLUSION

`In conclusion, the proposed Water Level Monitoring System using ESP32, an ultrasonic sensor, and a relay module provides an efficient and automated solution for maintaining optimal water levels in a tank. The system continuously monitors the water level through the ultrasonic sensor and uses the ESP32 to process the data, activating or deactivating the water pump as needed based on predefined threshold levels. This approach eliminates the need for manual intervention, ensuring that the tank is refilled when the water level is low and preventing overflow when the level is too high. The integration of the relay module allows precise control over the pump, making the system both reliable and energy-efficient. Additionally, the use of the ESP32 microcontroller enables the possibility of future enhancements, such as remote monitoring and control, using Wi-Fi connectivity. Overall, this system provides an effective, cost-efficient, and scalable solution for water level management in various applications.

CHAPTER – 4

HARDWARE COMPONENTS

4.1 ESP32 MICROCONTROLLER

Fig 4.1.1 ESP32 MICROCONTROLLER

The ***ESP32*** is a dual-core, low-power system-on-chip (SoC) microcontroller that combines powerful processing capabilities with multiple communication options. It includes built-in ***Wi-Fi*** and ***Bluetooth 4.2/Bluetooth LE***, making it well-suited for Internet of Things (IoT) applications. The chip's architecture allows it to handle complex tasks, such as data processing and multi-protocol communications, in real-time. The ESP32 is often selected for projects that require a balance of performance, power efficiency, and connectivity.

Specifications

- Wi-Fi and Bluetooth: 2.4 GHz 802.11 b/g/n Wi-Fi and Bluetooth 4.2 (BLE and Classic), supporting WPA/WPA2 with an on-board antenna.

- GPIO Pins: 34 GPIO pins, of which many support PWM, I2C, SPI, and UART protocols, making them versatile for various sensor and peripheral connections.

- Processor: Dual-core Xtensa LX6 CPU, up to 240 MHz, providing 600 DMIPS for intensive processing tasks.
- Memory: 520 KB SRAM and typically 4 MB Flash memory for program and data storage.
- It has 17 GPIO, 11 are usable(6 are used for communication with flash)

ESP32 PIN DIAGRAM

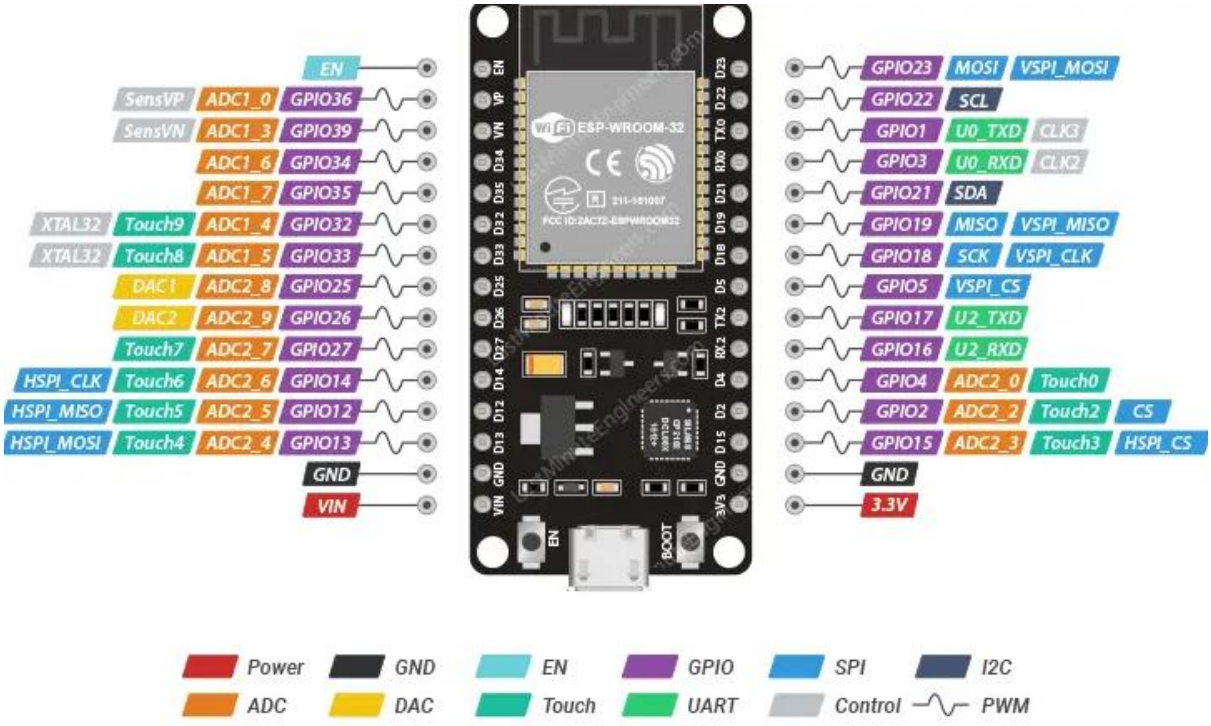


Fig 4.1.2 ESP32 Pin Diagram

4.2 RELAY MODULE

Fig 4.2 Relay Module

A ***relay module*** is an electrical device used to control high-voltage circuits with a low-voltage signal. It acts as an electrically operated switch, allowing a microcontroller (such as an ESP32 or Arduino) to control devices like motors, fans, or lights that require higher voltages than the microcontroller can safely handle.

Specifications

- Control Voltage: Typically 5V or 12V (depending on the module).
- Switching Voltage: Can handle AC or DC voltages, usually up to 250V AC or 30V DC.
- Switching Current: Can typically control up to 10A at 120V AC

4.3 ULTRASONI SOUND SENSOR (HC-SR04)

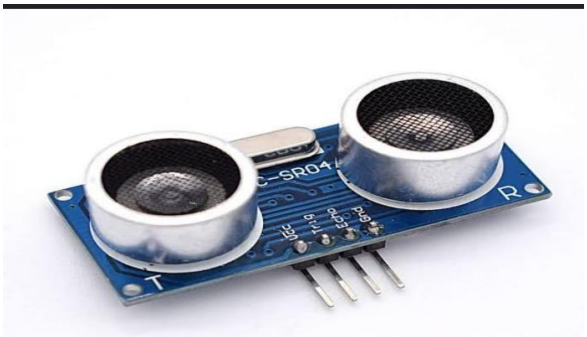


Fig 4.3 Ultrasonic sound sensor

The ultrasonic sensor (HC-SR04) is a critical component in the proposed water level monitoring system, designed to measure the distance between the sensor and the water surface accurately. It operates using ultrasonic sound waves, which are inaudible to the human ear. The sensor has two main elements: a transmitter and a receiver. When the Trig pin is triggered by the microcontroller, the sensor emits a short burst of ultrasonic waves from the transmitter. These waves travel through the air and reflect back upon hitting the water surface. The reflected waves are captured by the receiver, and the sensor calculates the time taken for the sound to travel to the surface and back. This time is used to compute the distance using the speed of sound formula. The measured distance is then sent to the microcontroller (ESP32) via the Echo pin, where it is processed to determine the water level in the tank. The sensor provides a reliable, non-contact method for level measurement, making it ideal for applications where physical sensors might face wear and tear or contamination**.**

4.3 CLOUD PLATFORM

In this project, the cloud platform refers to Firebase, which is used to store and manage the facial recognition data and facilitate communication between the camera module and the ESP32 microcontroller. Firebase is a cloud-based platform that provides real-time database services, authentication, and other backend functionalities, making it ideal for IoT-based applications like this one. When the camera module captures an image and processes the facial recognition, the results are uploaded to Firebase.

4.4 FIRE BASE CLOUD

Firebase Cloud serves as the backend infrastructure, enabling real-time data storage and communication between the system's components. The Real-Time Database stores facial recognition results, such as whether a detected face matches an authorized user, and provides instant updates to the ESP32 microcontroller, which uses this data to control the door lock via the relay module. Firebase also handles authentication to ensure secure access to the stored data, allowing only authorized devices to interact with the system. With its real-time capabilities, Firebase ensures seamless synchronization between the camera module, the Raspberry Pi or laptop processing the face recognition, and the ESP32, enabling immediate action based on the recognition results. This cloud platform streamlines the system's operation, reducing the need for local storage and processing while ensuring efficient, scalable, and secure door access control.

4.5 CONCLUSION

The ESP32 microcontroller and relay module play crucial roles in the Face Recognition-Based Door Access Control System. The ESP32 acts as the system's brain, managing communication with the cloud and processing the facial recognition data to determine whether access should be granted. It effectively controls the relay module, which serves as the electronic switch to unlock or lock the door based on the recognition results. The combination of the ESP32's connectivity and processing power with the relay's ability to control the door's locking mechanism ensures that the system operates securely and efficiently, offering both automation and manual control over door access.

CHAPTER – 5

EXPERIMENT RESULT AND ANALYSIS

5.1 INTRODUCTION

The experimental results of the proposed Water Level Monitoring System demonstrate its effectiveness in maintaining the water level within the predefined thresholds. During testing, the system successfully measured the water level using the ultrasonic sensor, and the ESP32 microcontroller processed the data to control the relay module, which operated the water pump as required. The results showed that the system responded accurately to changes in water levels, activating the pump when the level dropped below the minimum threshold and deactivating it when the maximum level was reached. The system's performance was consistent under varying environmental conditions, with reliable sensor readings and timely relay operation. These outcomes validate the system's reliability, efficiency, and potential for real-world applications, offering a cost-effective and automated solution for water management.

5.2 INITIAL SETUP

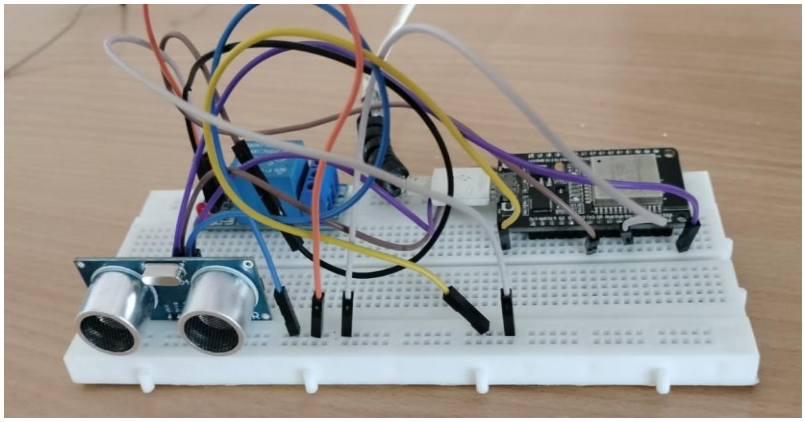


Fig 5.2

In fig 5.2. The initial setup of the Water Level Monitoring System involves assembling and connecting all hardware components, including the ESP32 microcontroller, HC-SR04 ultrasonic sensor, relay module, and water pump. The ultrasonic sensor is mounted at a fixed position above the water tank, and its Trig and Echo pins are connected to the designated GPIO pins of the ESP32. The relay module is connected to the ESP32 and linked to the water pump to control its operation, while a stable power supply is provided to ensure seamless functionality.

5.3 FINAL SETUP

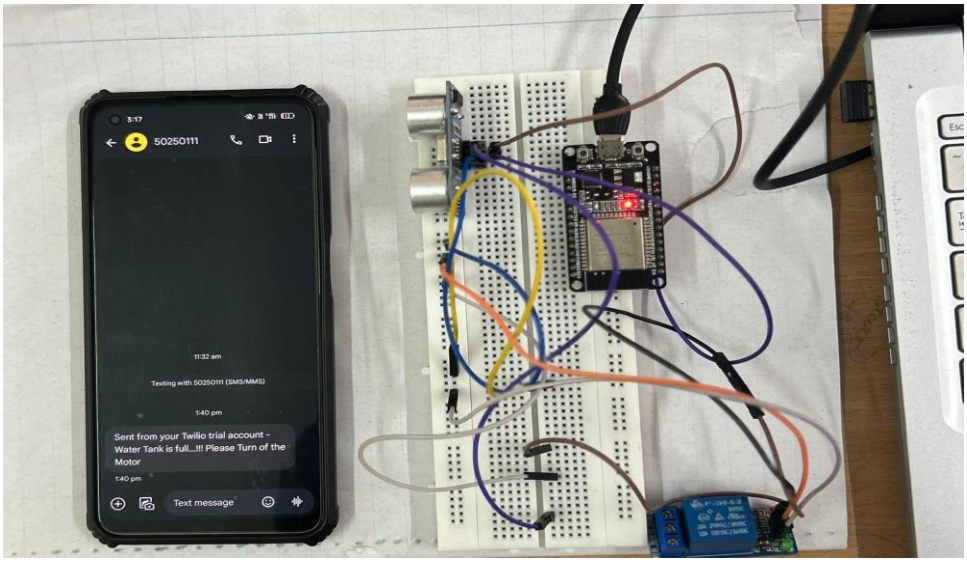


Fig 5.3

In fig 5.3. The final setup of the Water Level Monitoring System integrates all components into a fully functional system. The ultrasonic sensor is securely positioned above the tank, and the ESP32 is programmed to process sensor data and control the relay module, which operates the water pump. The system is tested to ensure accurate water level monitoring, proper relay operation, and seamless automation of the pump, ready for deployment.

CHAPTER – 6

CONCLUSION

6.1 CONCLUSION

In conclusion, the proposed Water Level Monitoring System effectively automates the process of maintaining water levels in a tank using an ESP32 microcontroller, ultrasonic sensor, and relay module. The system ensures accurate monitoring and reliable control of the water pump, eliminating the need for manual intervention. By maintaining water levels within predefined thresholds, it prevents overflow and dry running, conserving water and protecting equipment. The modular and scalable design allows for easy integration with IoT platforms for remote monitoring, enhancing its versatility. Overall, the system is a cost-effective, efficient, and sustainable solution for water management in residential, agricultural, and industrial applications.

6.2 FUTURE WORKS

Future work on the proposed Water Level Monitoring System could focus on enhancing its functionality and adaptability. Integration with IoT platforms can enable remote monitoring and control of water levels through mobile apps or web interfaces. Advanced features, such as real-time data logging and predictive analytics, can be implemented to optimize water usage and detect anomalies. Incorporating additional sensors, such as temperature or flow sensors, could provide more comprehensive water management solutions. The system could also be adapted for solar-powered operation to improve energy efficiency and environmental sustainability. Furthermore, scalability options can be explored to apply the system to larger or more complex water management setups, such as multi-tank systems or industrial applications.

CHAPTER – 7

REFERENCES

1. **Kumar, R., & Sharma, A. (2020). IoT-Based Automated Water Management Systems. Springer Publications.**
2. **Desai, P., & Jain, N. (2018). Sensor Applications in Real-Time Monitoring. McGraw Hill Education.**
3. **Gupta, S. (2019). Microcontroller-Based Automation: Applications and Principles. Wiley India**
4. **Brown, J. & Miller, T. (2021). Embedded Systems Design with ESP32. Packt Publishing.**
5. **Wilson, H. (2020). Practical Arduino and Sensor Projects. Apress.**
6. **Singh, R., & Verma, K. (2021). Digital Electronics and Control Systems. Pearson Education.**