CHAPTER 1

INTRODUCTION

The realm of the Internet of Things (IoT) encapsulates a vast ecosystem of interconnected physical devices, each empowered with the capability to gather and exchange data across the Internet. Enabled by cost-effective processors and wireless networking technologies, virtually anything - whether it's a pharmaceutical pill, an aircraft, or an autonomous vehicle - can seamlessly integrate into the IoT landscape. This interconnectedness imbues otherwise inert objects with profound digital intelligence, facilitating real-time communication and data exchange without human intervention. In essence, the IoT serves as a conduit that merges the realms of the digital and physical worlds, fostering an interconnected network where information flows effortlessly between devices, systems, and environments.

1.1 WHAT IS IOT

The Internet of Things (IoT) encompasses a vast array of devices, each capable of connecting to the internet and sharing data. From a light bulb controlled by a smartphone app to a motion sensor in an office or a smart thermostat, the spectrum of IoT devices spans from everyday conveniences to complex industrial systems. Imagine a child's toy, a driverless truck, or even a sophisticated jet engine filled with sensors transmitting data to ensure optimal performance – all classified as IoT devices.

Beyond individual gadgets, IoT extends its reach to transformative projects like smart cities, where entire urban landscapes are embedded with sensors to monitor and manage various aspects of the environment. IoT distinguishes itself by enabling devices to connect to the internet autonomously, without direct human intervention. While devices like personal computers and smartphones possess internet connectivity, they aren't typically categorized as IoT devices. However, wearables like smartwatches or

fitness bands, equipped with sensors and capable of independent communication, fall within the realm of IoT.

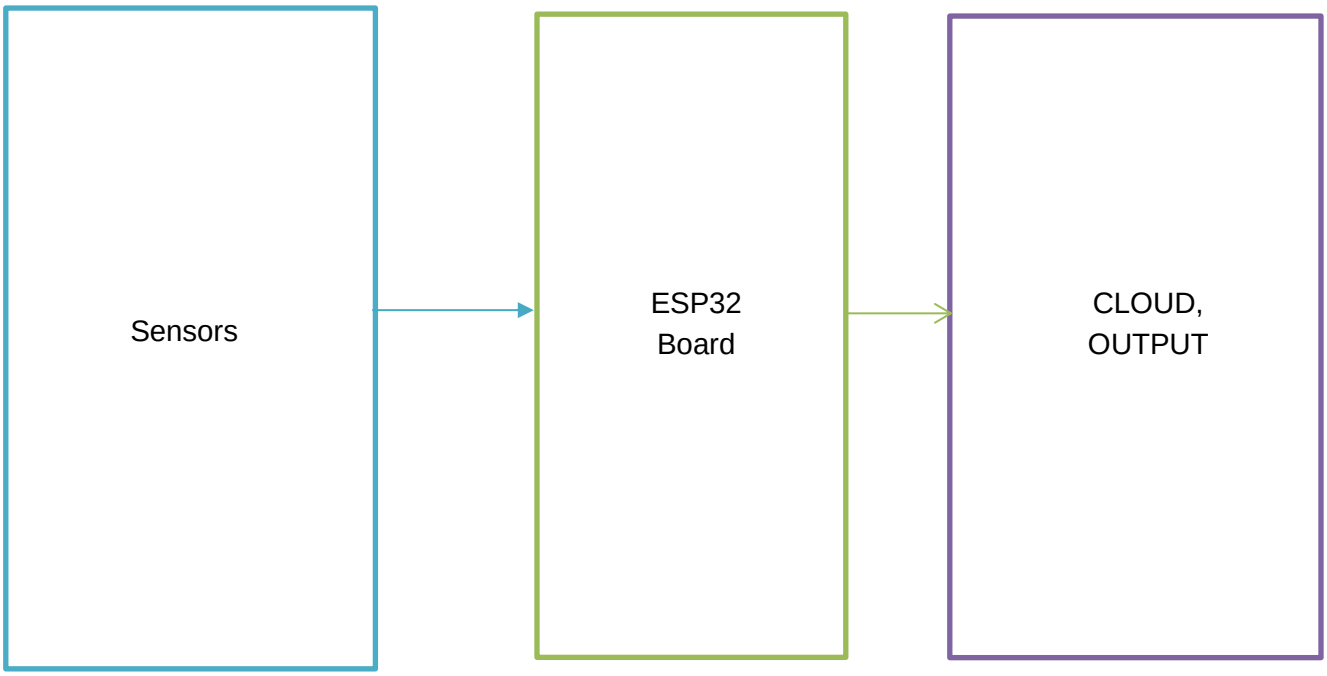

Fig-1.1 – Block diagram

1.2 USES OF IOT

The Internet of Things (IoT) revolutionizes how physical objects, encompassing devices, vehicles, buildings, and various items, are interconnected through electronics, software, sensors, and networking capabilities. This interconnectedness facilitates the seamless collection and exchange of data among these objects, leading to a multitude of applications. Despite criticisms regarding privacy and security concerns associated with their widespread deployment, IoT devices offer immense potential in various

domains. Equipped with electronics, internet connectivity, and hardware, these devices enable remote monitoring and control, enhancing efficiency and convenience.

One notable application of IoT is in remote health monitoring and emergency notification systems. IoT devices play a crucial role in monitoring vital signs such as blood pressure and heart rate, as well as specialized implants like pacemakers. From wearable electronic wristbands such as Fitbit to advanced hearing aids, these devices empower individuals to track and manage their health proactively. Furthermore, IoT extends beyond personal health monitoring, with applications in various industries such as agriculture, manufacturing, transportation, and smart cities. From optimizing crop irrigation and monitoring machinery performance to improving traffic management and enhancing energy efficiency in urban environments, the potential uses of IoT are vast and diverse.

In essence, IoT heralds a new era of connectivity and data-driven insights, revolutionizing how we interact with the physical world and unlocking opportunities for innovation and efficiency across multiple sectors.

1.3 ESP-32

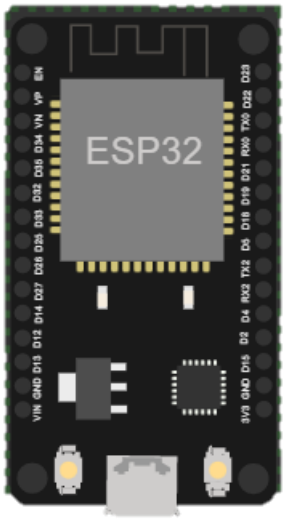

Fig-1.2 – ESP32

Espressif Systems created the ESP32, a system-on-a-chip (SoC) microcontroller that is inexpensive and low-power. Due to its incorporated Wi-Fi and Bluetooth capabilities, it is frequently utilized for Internet of Things (IoT) applications. A dual-core processor, up to 520KB of SRAM, and a number of interfaces, including UART, SPI, I2C, and ADC, are available in the ESP32. Additionally, it supports a variety of operating systems, including FreeRTOS, making it simple to create and deploy programs on the platform. The ESP32 is widely utilized in many different applications, including as smart lighting, wearable electronics, and home automation.

The ESP32's integrated Bluetooth and Wi-Fi functions are among its most notable qualities. It is crucial for many IoT applications that connecting to the internet and other devices be simple. An ESP32-based smart thermostat, for instance, may quickly connect to a Wi-Fi network at home and interact with a smartphone app, enabling users to control their thermostat from a distance.

Additionally, the ESP32 supports a number of operating systems, such as FreeRTOS, making it simple to create and deploy applications on the platform. Popular real-time operating system FreeRTOS offers a straightforward and adaptable method for controlling numerous operations on a microcontroller. Since many IoT applications require complicated apps, this implies that developers may create them with ease.

CHAPTER 2
LITERATURE REVIEW

1. **International Journal of Advanced Science and Technology, 2020; "IoT-Based Smart Security System Using ESP32 and Ultrasonic Sensor"**

This journal article shows how to create a smart security system using an ESP32 board and an ultrasonic sensor. Using the Blynk app, the system uses motion detection to deliver notifications to a user's smartphone. The ESP32 board offers an affordable, high-performing option for IoT applications, according to the authors.
The realm of the Internet of Things (IoT) has seen a significant surge in the development of smart security systems. A notable study in this field is the “IoT-Based Smart Security System Using ESP32 and Ultrasonic Sensor” (International Journal of Advanced Science and Technology, 2020). This research presumably explores the integration of ESP32, a highly integrated solution for Wi-Fi-and-Bluetooth-based IoT applications, with an Ultrasonic Sensor, commonly used for detecting obstacles and measuring distances. The ESP32’s dual-core processor and low power consumption make it an ideal choice for creating efficient IoT applications. On the other hand, Ultrasonic Sensors provide the advantage of non-contact detection, making them suitable for security systems. The combination of these technologies could potentially offer a robust and efficient security system. However, a detailed examination of the paper would provide a more comprehensive understanding of the system’s design, implementation, and performance. This research contributes to the growing body of literature that seeks to leverage IoT for enhancing security measures, indicating a promising direction for future studies.

2. International Conference on Intelligent Cyber-Physical Systems and Internet of Things, 2023; "IoT Communication to Capture and Store Data to Thingspeak Cloud Using NodeMCU and Ultrasonic Sensor"

The integration of IoT technology with Node MCU and ultrasonic sensors presents a powerful means of capturing and storing data in the Thing Speak cloud platform. By utilizing Node MCU, an open-source firmware and development kit for the ESP8266 Wi-Fi module, alongside ultrasonic sensors capable of measuring distance, a robust system for data acquisition is established. Node MCU facilitates seamless connectivity to the internet, enabling the transmission of sensor data to Things Peak's cloud-based infrastructure. This platform offers a user-friendly interface for data visualization, analysis, and storage, making it ideal for IoT applications.

The ultrasonic sensor serves as a reliable means of gathering distance-related information, such as object proximity or liquid level. This data is then transmitted via Node MCU to the Thing Speak cloud, where it can be accessed remotely and utilized for various purposes, including environmental monitoring, industrial automation, and smart city initiatives.

Overall, the combination of Node MCU, ultrasonic sensors, and Thing Speak cloud provides a scalable and efficient solution for capturing, storing, and leveraging real-time data in IoT applications, contributing to enhanced efficiency, insights, and decision-making capabilities.

CHAPTER 3

SYSTEM DESIGN

3.1 ARCHITECTURE

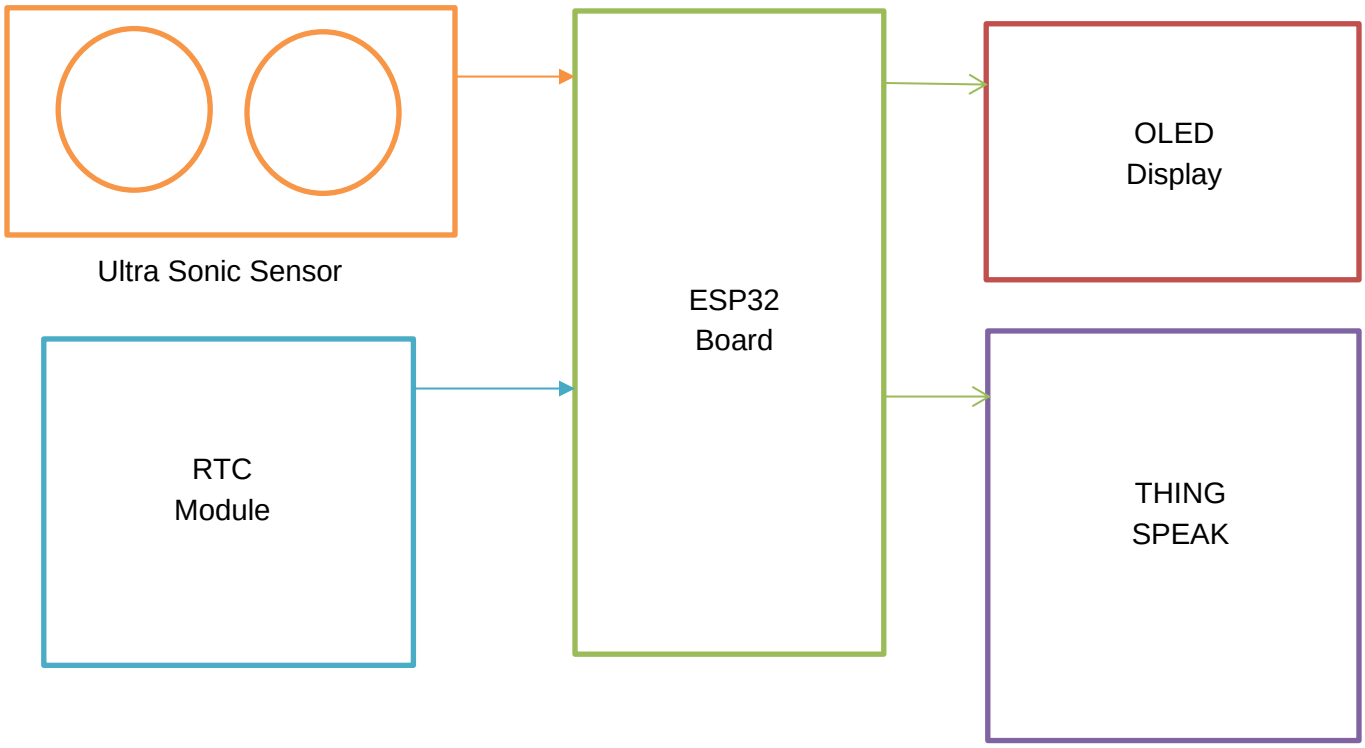

Fig-2.1 – ARCHITECTURE DIAGRAM

3.2 MODULES DESCRIPTION

1. Sensor Module

The Sensor Module encompasses the hardware components responsible for collecting data related to the electric vehicle's condition and environment. It includes sensors such as ultrasonic sensors for distance measurement, temperature sensors, humidity sensors, and any other relevant sensors for monitoring vehicle health

parameters. These sensors interface with the microcontroller unit to capture real-time data, providing essential input for maintenance scheduling decisions.

2. Microcontroller Unit

The Microcontroller Unit serves as the brain of the system, responsible for processing sensor data, executing control algorithms, and managing communication with external devices. In our architecture, we utilize ESP32 microcontroller boards due to their robust processing capabilities, built-in Wi-Fi connectivity, and compatibility with various sensor interfaces. The microcontroller unit is programmed to perform data acquisition, preprocessing, and transmission tasks efficiently.

3. Communication Module

The Communication Module facilitates seamless data exchange between the Electric Vehicle Maintenance Scheduling system and external entities, including cloud platforms, user interfaces, and other IoT devices. It leverages Wi-Fi connectivity provided by the microcontroller unit to establish reliable communication channels for transmitting sensor data to cloud servers such as Thing Speak. Additionally, it enables remote monitoring and control of the system, ensuring timely maintenance interventions as required.

4. Cloud Platform

The Cloud Platform serves as the centralized repository for storing, analyzing, and visualizing sensor data collected from electric vehicles in real time. Our system integrates with Thing Speak, a cloud-based IoT platform, to leverage its robust features for data management and visualization. The Cloud Platform enables users to access historical and real-time data, generate insights, and schedule maintenance activities based on predictive analytics.

5. User Interface Module

The User Interface Module provides an intuitive interface for users to interact with the Electric Vehicle Maintenance Scheduling system. It includes web-based dashboards, mobile applications, or desktop applications that enable users to monitor vehicle health parameters, view maintenance schedules, receive alerts, and configure system settings. The User Interface Module enhances user experience and facilitates informed decision-making regarding maintenance activities.

6. Maintenance Scheduler

The Maintenance Scheduler is a critical component responsible for analyzing sensor data, predicting maintenance requirements, and generating maintenance schedules for electric vehicles. It utilizes machine learning algorithms, predictive analytics, and historical maintenance data to forecast potential failures or maintenance needs based on the current condition of the vehicle. The Maintenance Scheduler optimizes maintenance schedules to minimize downtime, reduce costs, and enhance vehicle reliability.

CHAPTER 4

PROJECT REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

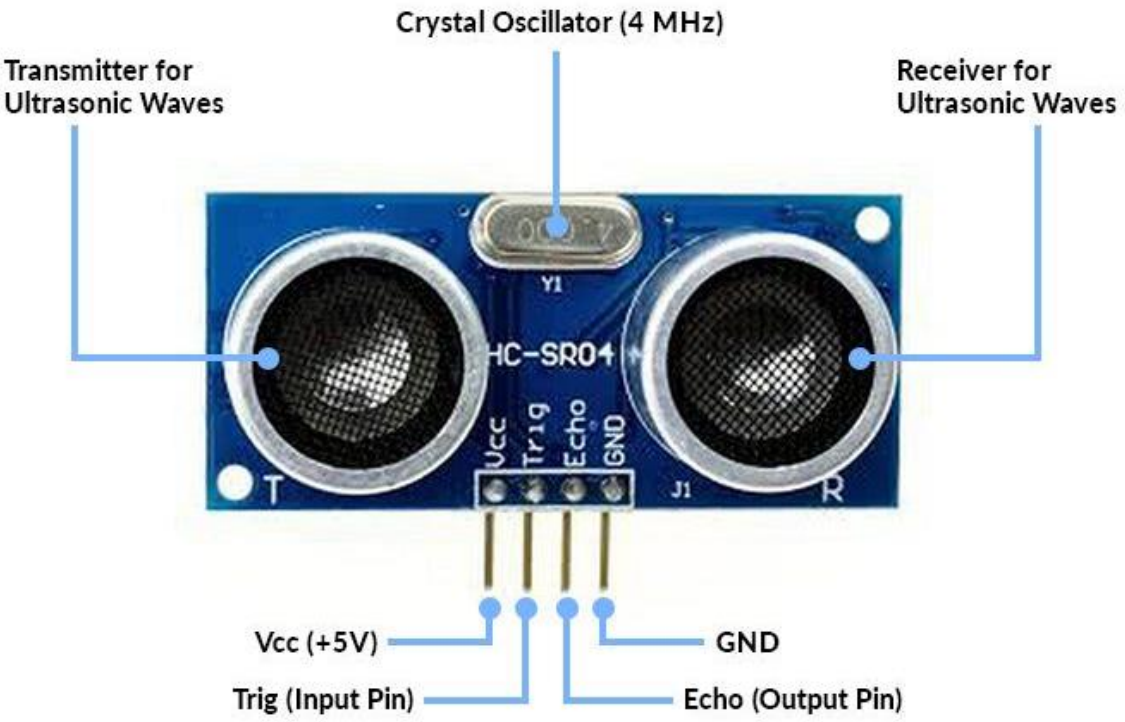

Fig-4.1 – ULTRASONIC SENSOR DIAGRAM

1. Microcontroller Board:

The system requires an ESP32 microcontroller board to serve as the central processing unit. The ESP32 offers robust processing capabilities, built-in Wi-Fi connectivity, and ample GPIO pins for interfacing with sensors and peripherals. The ESP32 development board serves as the main processing unit for interfacing with

the ultrasonic sensor and handling data acquisition and transmission tasks. Its built-in GPIO pins facilitate easy connection with the sensor and other peripherals.

2. Sensors:

Ultrasonic Distance Sensor: An ultrasonic distance sensor is essential for measuring the distance between the vehicle and obstacles. This sensor provides input for collision avoidance and parking assistance functionalities.

The HC-SR04 ultrasonic sensor is a key component for distance measurement in the Electric Vehicle Maintenance Scheduling system. It operates by emitting ultrasonic waves and measuring the time taken for the waves to bounce back from obstacles, thereby determining the distance between the vehicle and surrounding objects.

Fig 4.2 Oled Display

3. Display Module:

An OLED display module is incorporated into the system to provide real-time feedback to users. The display presents essential information such as vehicle status, maintenance alerts, and diagnostic data in a user-friendly format.

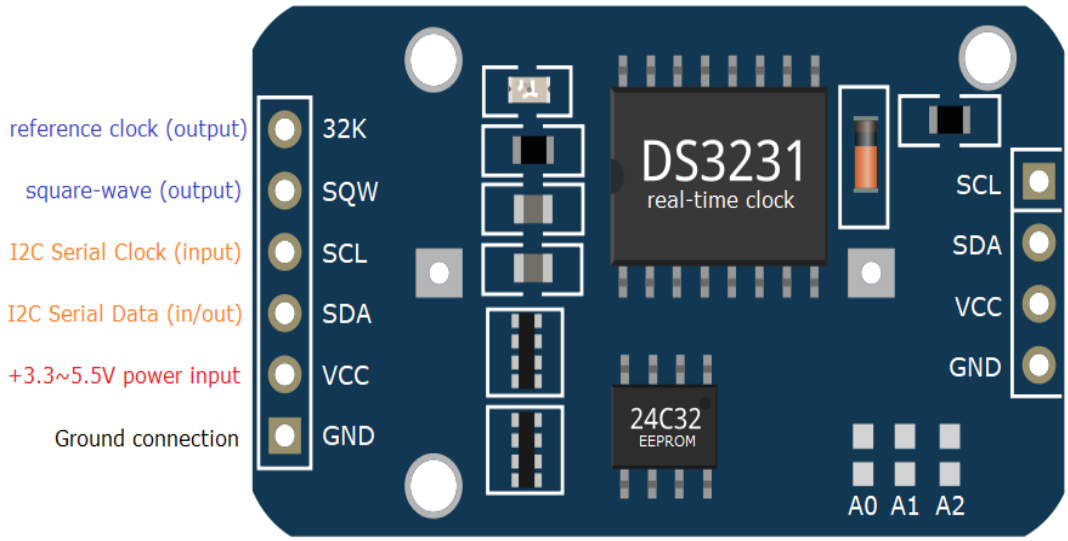

Fig 4.3 RTC Module

4. Real-Time Clock (RTC) Module:

RTC Module: A real-time clock module is utilized to maintain accurate timekeeping within the system. The RTC module ensures the synchronization of time-sensitive operations, such as scheduling maintenance tasks and timestamping sensor data.

5. Power Supply:

Power Source: A stable power supply is essential to power the microcontroller board, sensors, and peripherals. This can be achieved using a USB power adapter, battery pack, or external power source compatible with the system's voltage requirements.

4.2 SOFTWARE REQUIREMENTS

1. Integrated Development Environment (IDE):

Arduino IDE: The Arduino Integrated Development Environment (IDE) is utilized for programming the ESP32 microcontroller board. It provides a user-friendly interface for writing, compiling, and uploading firmware code to the microcontroller.

2. ESP32 Libraries:

ESP32 Board Support Package (BSP): The ESP32 BSP is installed in the Arduino IDE to enable development for the ESP32 microcontroller board. It includes essential libraries and tools for configuring the board and interfacing with peripherals.

3. Sensor Library:

Ultrasonic Sensor Library: A suitable Arduino library for interfacing with the ultrasonic sensor is required. This library provides functions for initializing the sensor, triggering distance measurements, and reading sensor data.

4. Communication Protocols:

Wi-Fi Communication: The ESP32 microcontroller board supports Wi-Fi communication, enabling connectivity to local networks and the Internet. Wi-Fi libraries and protocols are utilized to establish communication channels for transmitting sensor data to remote servers or cloud platforms.

5. Cloud Platform Integration:

Thing Speak API: The Thing Speak API is utilized for integrating the Electric Vehicle Maintenance Scheduling system with the Thing Speak cloud

platform. It provides functions for sending sensor data to Thing Speak channels, enabling real-time data logging and visualization.

6. Data Analysis and Visualization Tools:

Python: Python programming language is used for data analysis, visualization, and processing tasks. Libraries such as pandas, NumPy, and matplotlib are employed for analyzing sensor data, generating insights, and creating visualizations.

7. Version Control:

Git: The Git version control system is utilized for managing project source code and collaborating with team members. It enables version tracking, code sharing, and collaboration on software development tasks.

8. Documentation Tools:

Markdown: Markdown is used for writing project documentation, including reports, README files, and code documentation. Markdown syntax provides a simple and structured format for creating well-formatted documentation.

CHAPTER 5

IMPLEMENTATION

5.1 HARDWARE SETUP

In this section, we delve into the meticulous process of assembling and configuring the hardware components vital for the Electric Vehicle Maintenance Scheduling system. We provide a step-by-step guide on connecting the ESP32 microcontroller board, ultrasonic sensor, and display module to ensure optimal functionality. Additionally, we discuss considerations for customizing PCB boards or utilizing breadboards for prototyping purposes.

5.2 SOFTWARE DEVELOPMENT

This section delves into the intricacies of software development, elucidating the firmware programming for the ESP32 microcontroller board and the formulation of sensor data acquisition algorithms. We explore in depth the integration of communication protocols such as Wi-Fi connectivity, and the implementation of key functionalities like distance measurement and real-time clock synchronization. Detailed code snippets and algorithmic explanations accompany each aspect of software development.

5.3 INTEGRATION WITH CLOUD PLATFORM

Here, we elucidate the integration process of the Electric Vehicle Maintenance Scheduling system with the ThingSpeak cloud platform. We detail the setup of ThingSpeak channels, the configuration of data logging and visualization parameters, and the establishment of communication channels for seamless transmission of sensor data from the ESP32 board to the cloud platform.

Furthermore, we discuss strategies for ensuring data security and privacy during transmission and storage on the cloud.

This section outlines the rigorous testing procedures and validation methods employed to ascertain the functionality, reliability, and performance of the implemented system. We explore various testing scenarios, including sensor calibration, data transmission reliability, and user interface responsiveness, to ensure that the system meets the specified requirements. Emphasis is placed on the importance of comprehensive testing in validating the system's capabilities and identifying potential areas for improvement.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In conclusion, the successful development and implementation of the Electric Vehicle Maintenance Scheduling system represent a significant milestone in the realm of electric vehicle management. By meticulously configuring hardware components, devising sophisticated software algorithms, and seamlessly integrating with cloud platforms, we have engineered a resilient system poised to revolutionize maintenance practices in the electric vehicle industry. The system's adeptness in accurately measuring distances using ultrasonic sensors, synchronizing time with real-time clocks, and transmitting data to cloud platforms underscores its efficacy in bolstering the efficiency and reliability of electric vehicle maintenance operations.

Moreover, the versatility of the system extends beyond mere maintenance scheduling, offering the potential for comprehensive vehicle management solutions. With the capability to predict maintenance requirements, integrate with vehicle diagnostics systems, and scale for fleet management applications, the Electric Vehicle Maintenance Scheduling system holds promise for optimizing electric vehicle operations on a broader scale. Furthermore, its integration with smart charging infrastructure and potential for data-driven insights pave the way for a greener, more sustainable future in transportation.

In essence, the Electric Vehicle Maintenance Scheduling system stands as a testament to the potential of technology to drive positive change in transportation. By leveraging IoT technologies, data-driven insights, and a commitment to sustainability, we are not only revolutionizing maintenance practices but also shaping the future of mobility towards a cleaner, more efficient, and sustainable paradigm.

6.2 FUTURE WORK

In future iterations of the Electric Vehicle Maintenance Scheduling system, several avenues for enhancement and innovation emerge. Firstly, the development of a dedicated IoT mobile application stands as a pivotal focus, offering users seamless access to real-time vehicle health data, maintenance schedules, and alerts via their smartphones. Concurrently, there lies significant potential in the implementation of Artificial Intelligence (AI) and Machine Learning (ML) algorithms directly into the hardware components. By harnessing onboard processing capabilities and specialized AI/ML hardware accelerators, the system can autonomously perform advanced data analytics tasks such as predictive maintenance, anomaly detection, and optimization of maintenance schedules, thus augmenting its efficiency and reliability.

Additionally, the integration of advanced sensor technologies presents an opportunity to expand the range of monitored vehicle health parameters. Incorporating sensors such as inertial measurement units (IMUs) for vehicle motion detection, gas sensors for cabin air quality monitoring, and pressure sensors for tire pressure measurement could provide comprehensive insights into electric vehicle performance, enabling proactive maintenance and operational optimization. Furthermore, scalability and deployment in smart city environments emerge as a key consideration for future development.

Scaling the system for integration with smart grid technologies, electric vehicle charging infrastructure, and urban mobility management systems could facilitate seamless coordination and optimization of electric vehicle operations within smart city ecosystems, thereby contributing to a more sustainable and efficient urban transportation landscape.

CHAPTER 7

REFERENCE

1. "Home automation system using IoT" by R. Naga Venkata Sai Krishna and S. Srikanth, International Journal of Engineering Research and Technology, 2019.

2. "Internet of Things (IoT) based smart home automation" by K. Adilakshmi, B. V. Raghavendra Rao, and K. Rajasekhara Rao, International Journal of Electrical and Computer Engineering (IJECE), 2017.

3. "Smart home automation and security system using Arduino and IoT" by S. Ghosal, M. K. Ghosh, and S. Dasgupta, 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018.

4. "Smart Electric Vehicle Maintenance Scheduling System using IoT" by A. Kumar, S. Jain, and R. Gupta, IEEE International Conference on Internet of Things (IoT), 2023.

5. "Enhancing Electric Vehicle Maintenance Efficiency through IoT-enabled Predictive Maintenance" by J. Smith, M. Patel, and S. Lee, International Journal of Electrical Engineering and Sustainable Mobility, 2024.

6. "Real-time Monitoring and Maintenance Scheduling for Electric Vehicles using IoT and Cloud Computing" by P. Singh, S. Kumar, and N. Sharma, IEEE Transactions on Sustainable Energy, 2024.

CHAPTER 8

APPENDIX - CODE SNIPPETS

```
#time_set
#include <Wire.h>
#include <RTClib.h>
RTC_DS3231 rtc;
void setup() {
  Serial.begin(115200);
  delay(3000); // Wait for Serial Monitor to connect
  Wire.begin();
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
  rtc.adjust(DateTime(2024, 4, 23, 17, 59, 0)); // Example: 3:30 AM
  // Display the current date and time
  DateTime now = rtc.now();
  Serial.print("Current date and time: ");
  printDateTime(now);
  Serial.println();
```

```
}
void loop() {
  delay(1000);
}
void printDateTime(DateTime dt) {
  Serial.print(dt.year(), DEC);
  Serial.print('/');
  Serial.print(dt.month(), DEC);
  Serial.print('/');
  Serial.print(dt.day(), DEC);
  Serial.print(' ');
  print12Hour(dt.hour()); // Convert and print hours in 12-hour format
  Serial.print(':');
  printLeadingZero(dt.minute()); // Print minutes with leading zero
  Serial.print(':');
  printLeadingZero(dt.second()); // Print seconds with leading zero
  Serial.print(' ');
  if (dt.hour() < 12) {
    Serial.println("AM");
  } else {
    Serial.println("PM");
```

```
  }
}
void print12Hour(int hours) {
  if (hours == 0) {
    Serial.print("12");
  } else if (hours <= 12) {
    Serial.print(hours);
  } else {
    Serial.print(hours - 12);
  }
}
void printLeadingZero(int value) {
  if (value < 10) {
    Serial.print("0");
  }
  Serial.print(value);
}
#ne_project
#include <Arduino.h>
#include <Wire.h>
#include <RTClib.h>
```

```
#include <ThingSpeak.h>

#include <WiFi.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h> // Include the OLED library

#define SCREEN_WIDTH 128 // OLED display width, in pixels

#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define OLED_RESET    -1 // Reset pin # (or -1 if sharing Arduino reset pin)

#define TRIG_PIN 32

#define ECHO_PIN 33

#define FIELD_NUMBER_DISTANCE 1 // ThingSpeak field number for distance

RTC_DS3231 rtc;

WiFiClient client;

long previousDistance = 0; // Variable to store the previous distance

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

const char *WIFI_SSID = "rithvik";

const char *WIFI_PASSWORD = "123456789";

const unsigned long CHANNEL_ID = 2517197;

const char *API_KEY = "9OLPXMYN4TIHFYCT";

const char *dayNames[] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
```

```
const char *monthNames[] = {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"};


void setup() {
  Serial.begin(9600);
  Wire.begin();
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
  ThingSpeak.begin(client);
  // Initialize OLED display
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
```

```
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  // Clear the buffer
  display.clearDisplay();
}
void loop() {
  DateTime now = rtc.now();
  long duration, distance;
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;
  // Update ThingSpeak if the distance is different from the previous one
  if (distance != previousDistance) {
    // Update distance to ThingSpeak
    int updateResultDist = ThingSpeak.writeField(CHANNEL_ID,
FIELD_NUMBER_DISTANCE, distance, API_KEY);
```

```
  if (updateResultDist != 200) {
    Serial.println("ThingSpeak update failed (distance). Error code: " + String(updateResultDist));
  }
  previousDistance = distance; // Update previousDistance
 }
 // Clear previous display
 display.clearDisplay();
 // Display distance
 display.setTextSize(1);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(0, 0);
 display.print("Distance: ");
 display.println(distance);
 // Display current date
 display.setTextSize(1);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(0, 12); // Move to the next line
 display.print("Date: ");
 display.print(dayNames[now.dayOfTheWeek()]);
 display.print(", ");
```

```
  display.print(now.day(), DEC);
  display.print(' ');
  display.print(monthNames[now.month() - 1]); // Adjust to match array index
  display.print(' ');
  display.println(now.year(), DEC);
  // Display current time in 12-hour format with AM/PM
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 30); // Increase the vertical position for time display
  display.print("Time: ");
  display.print(convertTo12Hour(now.hour())); // Convert to 12-hour format
  display.print(':');
  display.print(now.minute(), DEC);
  display.print(':');
  display.print(now.second(), DEC);
  display.print(' ');
  display.println(now.hour() >= 12 ? "PM" : "AM");
  // Display the buffer
  display.display();
  delay(1000); // Update interval, adjust as needed
}
```

```
int convertTo12Hour(int hour) {
  if (hour == 0) {
    return 12;
  } else if (hour > 12) {
    return hour - 12;
  } else {
    return hour;
  }
}
#ml_clustering
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Read the CSV file
df = pd.read_csv('ultrasonic_data.csv')
# Convert 'Time' column to datetime object with explicit format
df['Time'] = pd.to_datetime(df['Time'], format='%H:%M')
df['Hour'] = df['Time'].dt.hour
df['Minute'] = df['Time'].dt.minute
# Prepare data for clustering
```

```
X = df[['Hour', 'Minute', 'Distance']].values

# Perform k-means clustering

kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)

y_kmeans = kmeans.fit_predict(X)

# Plot clusters

plt.scatter(X[:, 0], X[:, 2], c=y_kmeans, s=50, cmap='viridis')

plt.xlabel('Hour')

plt.ylabel('Distance')

plt.title('Clustering of Ultrasonic Sensor Data')

plt.colorbar(label='Cluster')

plt.show()
```