PHASE 3: IMPLEMENTATION OF PROJECT TITLE :

Smart Fleet Management and Telematics System Using IOT

Objective

The goal of Phase 3 is to implement the core components of the Smart Fleet Management and Telematics System based on the design and planning carried out in Phase 2. This includes the development of the telematics AI engine, dashboard interface, optional IoT hardware integration, and the implementation of data security protocols.

1. Telematics AI Engine Development

Overview

The core of the Smart Fleet Management System is the AI-based telematics engine, which collects and analyzes vehicle and driver data to provide insights and recommendations.

Implementation

- Sensor Data Processing: The AI engine interprets GPS, speed, fuel usage, and maintenance sensor inputsto generate actionable reports.

- Predictive Analytics: Uses machine learning to predict maintenance needs and optimize route planning.- Data Source: Real-time data from vehicles and cloud-connected databases are used for training and inference.

Outcome

By the end of this phase, the AI engine will identify inefficiencies, recommend maintenance actions, and monitor compliance with driving standards.

2. Dashboard & User Interface Development

Overview
The system will include a centralized web-based dashboard for fleet managers to monitor all fleet-related metrics in real time.

Implementation

- Fleet Overview: Live data visualization of all active vehicles, showing location, status, and driverinformation.

- Alerts & Reports: Custom notifications for speeding, engine issues, and maintenance schedules.

- Multilingual UI: Currently supports English; other languages to be added in future phases.

Outcome

The dashboard will allow fleet operators to manage operations effectively through a user-friendly and data-rich interface.

3. IoT Device Integration (Optional)

Overview

While optional in this phase, we aim to connect the system to in-vehicle IoT devices for enhanced real-time data capture.

Implementation

- Hardware Integration: Devices such as OBD-II dongles, GPS trackers, and fuel sensors will be integrated.

- API Use: Standardized APIs from IoT platforms will be used for data ingestion.

Outcome

If hardware is available, the system will capture live vehicle diagnostics and driving behavior. Otherwise, mock data will be used for simulation.

4. Data Security Implementation

Overview

Data security is a high priority given the sensitive nature of location and operational data in fleet management.

Implementation

- Encryption: All fleet data (vehicle location, driver behavior, trip history) will be encrypted both in transit andat rest.

- Secure Access: Role-based access controls and secure cloud storage will be implemented to comply withindustry regulations.

Outcome

All data processed and stored by the system will be protected by enterprise-grade security protocols.

5. Testing and Feedback Collection

Overview

This phase includes the initial rollout of the system for testing with a small fleet segment.

Implementation

- Pilot Testing: Select vehicles and drivers will be enrolled for testing various system modules.
- Feedback Loop: Input from fleet managers and drivers will be collected on usability, accuracy, andperformance.

Outcome

Feedback will be analyzed to refine AI models and enhance UI/UX in Phase 4.

Challenges and Solutions

Challenge: AI Prediction Accuracy

Solution: Continuous model training with real data and expert feedback.

Challenge: User Adoption

Solution: Training sessions and UI enhancements based on early feedback.

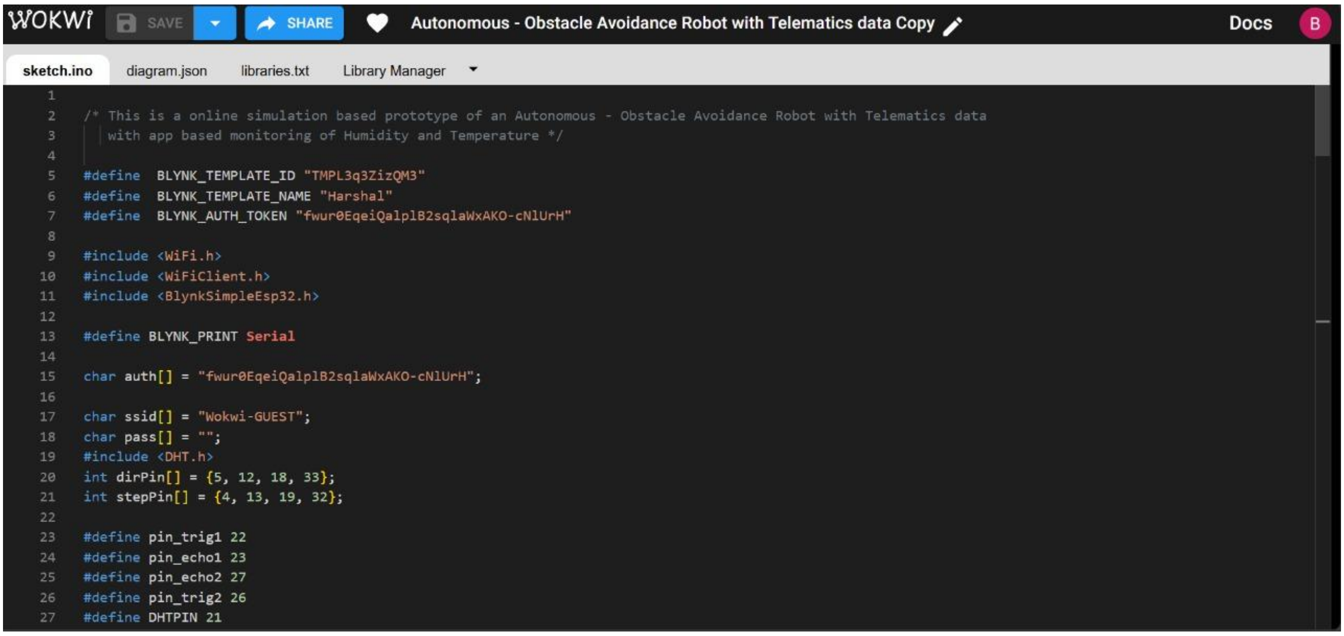
Challenge: Hardware Compatibility

Solution: Use of open-standard APIs and modular system design.

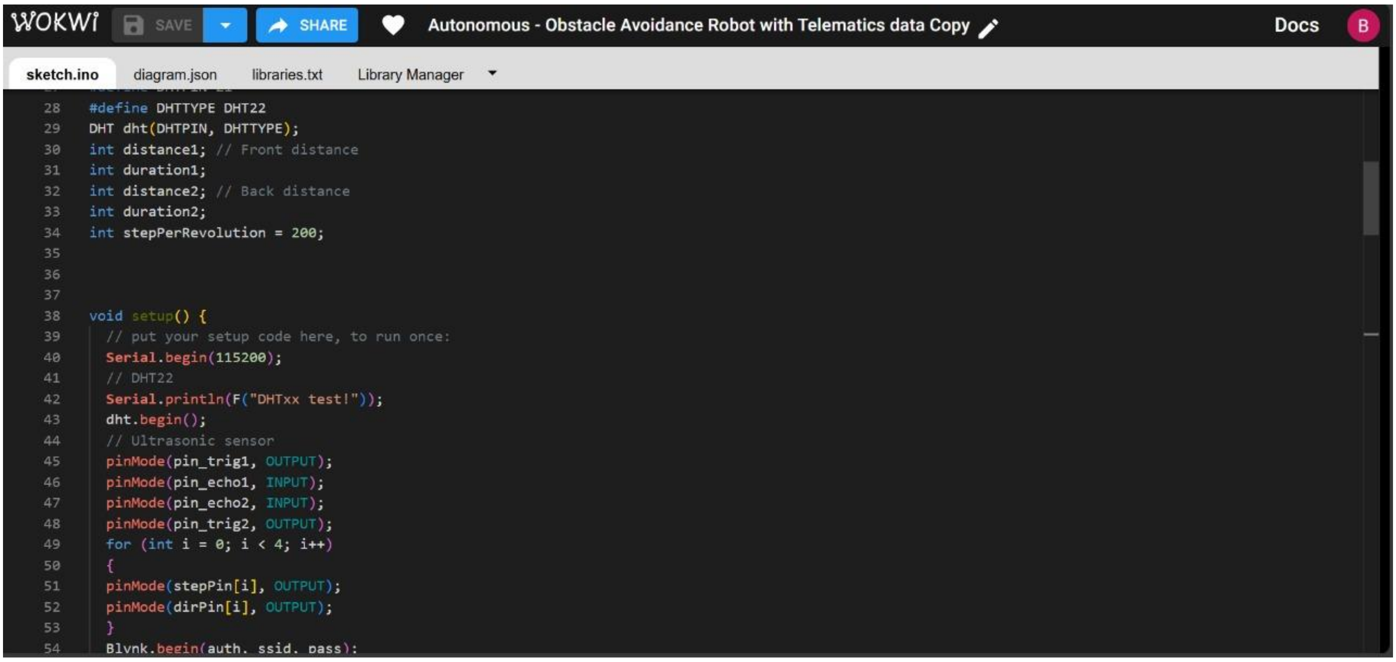
Outcomes of Phase 3

1. Functional AI Engine: Able to analyze fleet data and offer insights.
2. Operational Dashboard: Real-time fleet management capabilities.
3. IoT Integration (Optional): Basic connectivity with vehicle hardware.
4. Secure Data Framework: Encrypted and access-controlled data environment.
5. Initial User Testing: Feedback collected to guide future improvements.

Screenshots of Code and System in Use




```
/* This is a online simulation based prototype of an Autonomous - Obstacle Avoidance Robot with Telematics data
   with app based monitoring of Humidity and Temperature */

#define  BLYNK_TEMPLATE_ID "TMPL3q3ZizQM3"
#define  BLYNK_TEMPLATE_NAME "Harshal"
#define  BLYNK_AUTH_TOKEN "fwur0EqeiQalplB2sqlaWxAKO-cNlUrH"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

#define BLYNK_PRINT Serial

char auth[] = "fwur0EqeiQalplB2sqlaWxAKO-cNlUrH";

char ssid[] = "Wokwi-GUEST";
char pass[] = "";
#include <DHT.h>
int dirPin[] = {5, 12, 18, 33};
int stepPin[] = {4, 13, 19, 32};

#define pin_trig1 22
#define pin_echo1 23
#define pin_echo2 27
#define pin_trig2 26
#define DHTPIN 21
```




```
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
int distance1; // Front distance
int duration1;
int distance2; // Back distance
int duration2;
int stepPerRevolution = 200;



void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  // DHT22
  Serial.println(F("DHTxx test!"));
  dht.begin();
  // Ultrasonic sensor
  pinMode(pin_trig1, OUTPUT);
  pinMode(pin_echo1, INPUT);
  pinMode(pin_echo2, INPUT);
  pinMode(pin_trig2, OUTPUT);
  for (int i = 0; i < 4; i++)
  {
  pinMode(stepPin[i], OUTPUT);
  pinMode(dirPin[i], OUTPUT);
  }
  Blynk.begin(auth, ssid, pass);
```

WOKWI SAVE SHARE Autonomous - Obstacle Avoidance Robot with Telematics data Copy Docs B

sketch.ino diagram.json libraries.txt Library Manager

```
}

void loop()
{
  Blynk.run();
// Front ultrasonic sensor
  digitalWrite(pin_trig1, LOW);
  delayMicroseconds(2);
  digitalWrite(pin_trig1, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin_trig1, LOW);
  duration1 = pulseIn(pin_echo1, HIGH);
  distance1 = duration1 *0.034/2;
  Serial.print("FRONT-DISTANCE = ");
  Serial.print(distance1);

  delay(10); // this speeds up the simulation

// Second ultrasonic sensor
  digitalWrite(pin_trig2, LOW);
  delayMicroseconds(2);
  digitalWrite(pin_trig2, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin_trig2, LOW);
  duration2 = pulseIn(pin_echo2, HIGH);
  distance2 = duration2 *0.034/2;
  Serial.print(" BACK-DISTANCE = ");
```

WOKWI SAVE SHARE Autonomous - Obstacle Avoidance Robot with Telematics data Copy Docs B

sketch.ino diagram.json libraries.txt Library Manager

```
// Second ultrasonic sensor
  Serial.print(" BACK-DISTANCE = ");
  Serial.println(distance2);

  delay(100);

  // DHT22 temperature and humidity reading
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  Serial.print(F("Humidity: "));
  Serial.print(humidity);
  Serial.print(F("%  Temperature: "));
  Serial.print(temperature);
  Serial.println(F("°C "));

  delay(2000);

 // if in front the distance of ultrasonic sensor and obstacle is less than 20cm
  if (distance1 <= 5 )
  {
    for (int  i = 1; i < 4; i += 2) // upside motors
     {
      digitalWrite(dirPin[i], LOW);
     }
    for (int  i = 1; i < 4; i += 2)
    {
     digitalWrite(stepPin[i], HIGH);
```

WOKWI SAVE SHARE Autonomous - Obstacle Avoidance Robot with Telematics data Copy Docs B

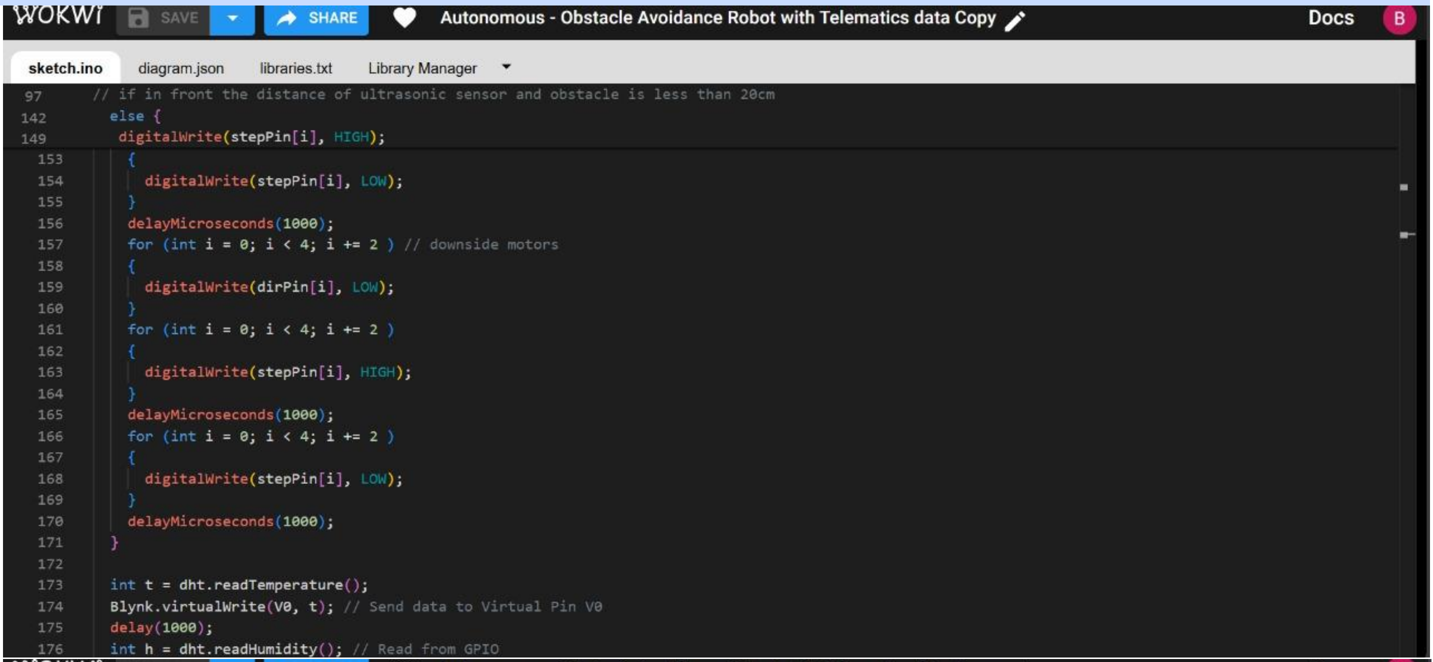
sketch.ino diagram.json libraries.txt Library Manager

```
  // if in front the distance of ultrasonic sensor and obstacle is less than 20cm
    {
      {
       digitalWrite(stepPin[i], HIGH);
      }
      delayMicroseconds(1000);
      for (int i = 1; i < 4; i += 2)
      {
        digitalWrite(stepPin[i], LOW);
      }
      delayMicroseconds(1000);

      for (int i = 0; i < 4; i += 2) // downside motors
      {
        digitalWrite(dirPin[i], HIGH);
      }
      for (int i = 0; i < 4; i += 2 )
      {
        digitalWrite(stepPin[i], HIGH);
      }
      delayMicroseconds(1000);
      for (int i = 0; i < 4; i += 2 )
      {
        digitalWrite(stepPin[i], LOW);
      }
      delayMicroseconds(1000);
      /*if (distance1 <= 20)
```

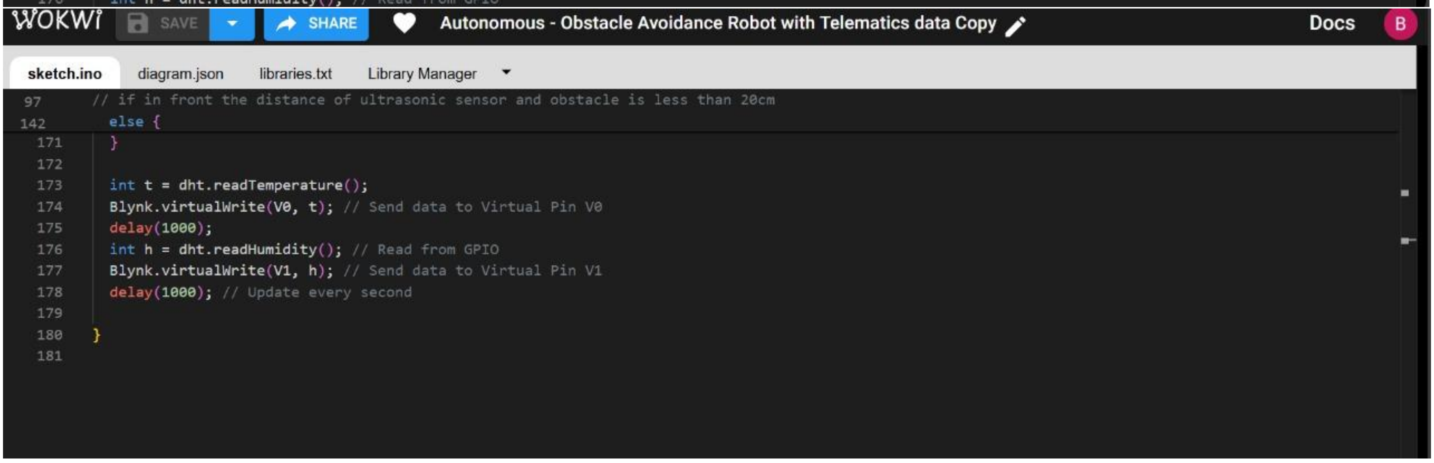
WOKWI SAVE SHARE Autonomous - Obstacle Avoidance Robot with Telematics data Copy Docs B

sketch.ino diagram.json libraries.txt Library Manager

```
  // if in front the distance of ultrasonic sensor and obstacle is less than 20cm
    {
      /*if (distance1 <= 20)
      {
        for (int  i = 1; i < 4; i += 2)
        {
        digitalWrite(stepPin[i], LOW);
        }
        for (int i = 0; i < 4; i += 2) // downside motors
        {
        digitalWrite(stepPin[i], LOW);
        }
      }*/
    }
    // if the front distance between ultrasonic sensor and obstacle is greater than 20 cm
    else {
     for (int  i = 1; i < 4; i += 2) // upwardside motors
       {
        digitalWrite(dirPin[i], HIGH);
       }
      for (int  i = 1; i < 4; i += 2)
      {
     digitalWrite(stepPin[i], HIGH);
      }
      delayMicroseconds(1000);
      for (int i = 1; i < 4; i += 2)
      {
```




```
// if in front the distance of ultrasonic sensor and obstacle is less than 20cm
  else {
   digitalWrite(stepPin[i], HIGH);
    {
      digitalWrite(stepPin[i], LOW);
    }
    delayMicroseconds(1000);
    for (int i = 0; i < 4; i += 2 ) // downside motors
    {
      digitalWrite(dirPin[i], LOW);
    }
    for (int i = 0; i < 4; i += 2 )
    {
      digitalWrite(stepPin[i], HIGH);
    }
    delayMicroseconds(1000);
    for (int i = 0; i < 4; i += 2 )
    {
      digitalWrite(stepPin[i], LOW);
    }
    delayMicroseconds(1000);
  }

  int t = dht.readTemperature();
  Blynk.virtualWrite(V0, t); // Send data to Virtual Pin V0
  delay(1000);
  int h = dht.readHumidity(); // Read from GPIO
```




```
// if in front the distance of ultrasonic sensor and obstacle is less than 20cm
  else {
  }

  int t = dht.readTemperature();
  Blynk.virtualWrite(V0, t); // Send data to Virtual Pin V0
  delay(1000);
  int h = dht.readHumidity(); // Read from GPIO
  Blynk.virtualWrite(V1, h); // Send data to Virtual Pin V1
  delay(1000); // Update every second

}
```