REAL-TIME VEHICLE DETECTION AND ADAPTIVE HEADLIGHT SYSTEM

A Project Report Submitted in Partial Fulfilment of Requirements to

Centre for Development of Advanced Computing Hyderabad

For the Award of the Degree of

Post Graduate Diploma

In

Embedded Systems and Design

Under the Esteemed Guidance of
Mr. Sarath V P

Submitted by

Priyanshu Singh (240850330041)
P.Saicharan (240850330042)
Siva kumar (240850330043)
Saikumar Boina (240850330044)

August 2024 - February 2025

Contents

1. Abstract

Real-Time Vehicle Detection and Adaptive Headlight System is an innovative solution designed to significantly enhance road safety by addressing visibility challenges during night-time driving and adverse weather conditions. The system utilizes an array of sensors, including a DHT11 to monitor temperature and humidity indicative of foggy or rainy conditions, an LDR to continuously gauge ambient light levels, and an ultrasonic sensor to detect oncoming vehicles or obstacles. Central to its operation is the ESP32 microcontroller, which leverages built-in Wi-Fi and Bluetooth capabilities to process sensor data in real time and transmit it to the cloud via HTTP for remote monitoring and proactive safety alerts. The project incorporates an LM393 module for precise control of LED brightness and an L298N motor driver that adjusts the servo motor to dynamically modify the headlight beam direction. This integration allows the system to automatically switch between high and low beams, effectively reducing glare and ensuring optimal illumination. Designed with durability and scalability in mind, the system not only improves driver visibility and comfort but also minimizes the risk of accidents by rapidly responding to sudden changes in environmental and traffic conditions. Extensive testing under varied scenarios confirms its reliability, making it a robust, energy-efficient solution that lays the foundation for next-generation automotive lighting systems and smart vehicular safety technologies.

2. ESP32 Microcontroller

2.1 Hardware Components

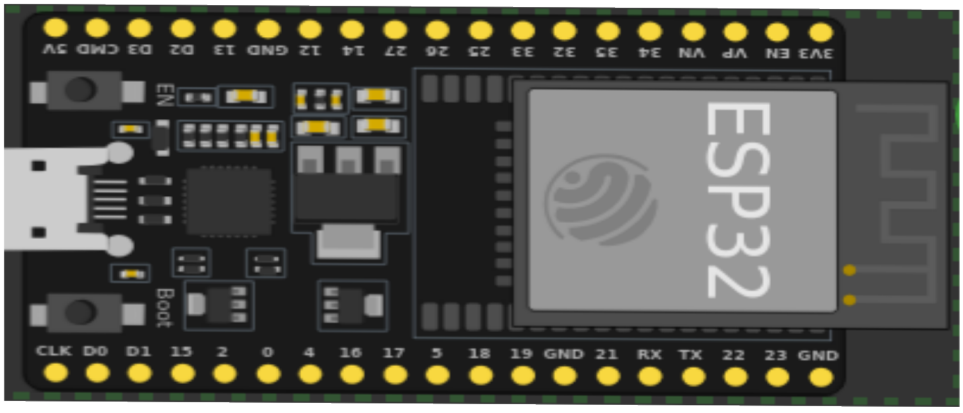


Figure 1: ESP32 Microcontroller

The ESP32 is a powerful, low-cost microcontroller developed by Espressif Systems that serves as the central processing unit in our project. It features dual-core Tensilica LX6 processors that can run at speeds up to 240 MHz, offering the high-performance computing needed for real-time sensor data processing and dynamic headlight control. The chip boasts a rich set of GPIO pins—typically over 30—which are highly configurable for various functions such as analog-to-digital conversion, PWM output, and digital I/O, making it extremely versatile for interfacing with sensors and actuators. In terms of memory, the ESP32 comes with up to 520 KB of SRAM and supports external flash memory, ensuring ample space for program code and runtime data. Additionally, its integrated Wi-Fi and Bluetooth capabilities allow seamless wireless connectivity, enabling remote monitoring and control via cloud services. The power management features of the ESP32 include multiple low-power modes, which make it energy efficient—a critical factor for battery-powered and real-time applications. Compared to other microcontrollers, such as the Arduino UNO or simpler 8-bit controllers, the ESP32 stands out due to its superior processing power, integrated connectivity, extensive I/O options, and power efficiency. These attributes make it the ideal choice for our adaptive headlight system, where rapid sensor response, robust real-time data processing, and reliable wireless communication are essential for ensuring enhanced road safety and dynamic performance.

2.2 Ultrasonic Sensor(HC-SR04)

The HC-SR04 is an ultrasonic distance sensor that measures how far an object is by emitting high-frequency sound waves and detecting the echo that bounces back. It operates by sending a 10-microsecond pulse to its Trigger pin, which prompts the sensor to emit an ultrasonic burst at 40 kHz. When this burst encounters an object, it reflects back to the sensor's Echo pin, where the duration of the returning pulse is measured. This time-of-flight measurement is then converted into a distance value using the speed of sound. The sensor typically measures distances ranging from 2 cm to 400 cm with an accuracy of around 3 mm. It is powered by a 5V DC supply and consists of four pins: VCC (power supply), Trigger (input pulse to start measurement), Echo (output pulse representing the time delay), and GND (ground). Unlike communication protocols such as I2C or SPI, the HC-SR04 uses simple digital pulse signals for its operation, making it straightforward to interface with microcontrollers like the ESP32. In our adaptive headlight project, this sensor is particularly beneficial as it accurately detects oncoming vehicles or obstacles. This real-time distance information is critical for dynamically adjusting the headlight beams, ensuring that the system can quickly switch between high and low beams to reduce glare and enhance road safety.

Figure 2: Ultrasonic Sensor (HC-SR04)

2.3 LDR(light dependent resistor)

The LDR sensor, or Light Dependent Resistor, is a passive component that exploits the principle of photoconductivity to sense ambient light intensity. Its core operating mechanism is based on a semiconductor material—commonly cadmium sulfide (CdS)—which exhibits high resistance in the dark and low resistance when illuminated. When light photons strike the semiconductor surface, they provide enough energy to free electrons, thereby increasing the material's conductivity. This change in resistance is inversely proportional to the intensity of the light: the brighter the light, the lower the resistance.

In practical applications, the LDR is typically integrated into a voltage divider circuit. In this configuration, the varying resistance of the LDR produces an output voltage that reflects the current light level, which can be read by the microcontroller's analog-to-digital converter (ADC). The sensor does not utilize digital communication protocols such as I2C or SPI; rather, it provides an analog signal that requires minimal additional circuitry, making it both cost-effective and easy to implement.

Within our adaptive headlight system, the LDR plays a critical role by continuously monitoring ambient light conditions. When the sensor detects low light levels—indicative of dusk, night, or inclement weather—it triggers the microcontroller to activate or adjust the headlights accordingly. This ensures that the vehicle’s illumination is optimized for safety, reducing the risk of accidents caused by insufficient lighting or unnecessary glare. The LDR’s rapid response to changes in environmental light conditions not only enhances driver visibility but also contributes to energy efficiency by ensuring that the headlights operate only when needed.

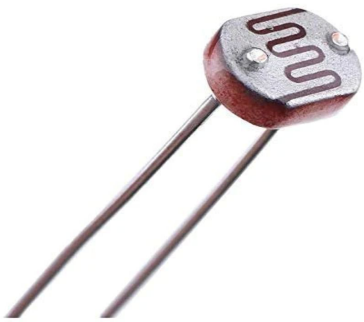


Figure 3: LDR Sensor

2.4 Fog Sensors(DHT11)

The DHT11 sensor is a compact, low-cost digital sensor that measures both temperature and relative humidity, making it an ideal component for monitoring environmental conditions in our adaptive headlight system. It operates on the principle of capacitive humidity measurement and uses a thermistor to gauge temperature. Internally, the sensor consists of a humidity sensing element and a temperature sensing element. When the sensor is activated, it sends out a single-wire digital signal that contains a 40-bit data sequence representing the humidity and temperature readings. This simple communication protocol eliminates the need for complex wiring or multiple communication channels like I2C or SPI, making integration with microcontrollers such as the ESP32 straightforward.

In our project, the DHT11 plays a pivotal role as a fog sensor. Fog typically forms under conditions of high humidity and moderate temperature, and by continuously monitoring these parameters, the sensor provides critical data that indicate the onset of foggy conditions. This allows the system to adjust the headlight brightness and beam direction accordingly to ensure optimal visibility and safety during adverse weather. The sensor's real-time feedback is essential for the adaptive control of headlights, helping to minimize glare and improve driver response time. Overall, the DHT11's blend of simplicity, efficiency, and reliable performance makes it a valuable component in our quest to enhance road safety through dynamic headlight adjustments.

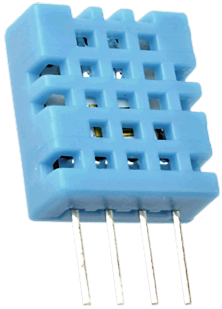


Figure 4: Ultrasonic Sensor

2.5 LM393 Sensors

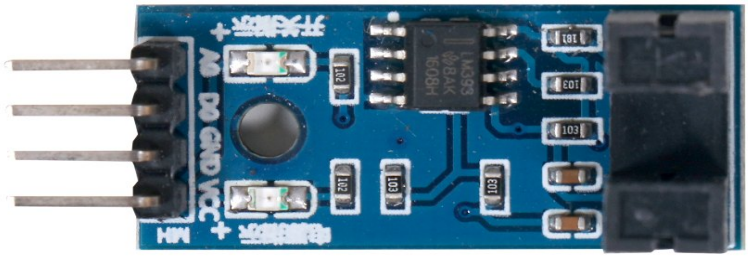


Figure 5: LM393 Sensor

The LM393 is a dual comparator module that plays a crucial role in our adaptive headlight system by ensuring precise brightness control of the LED headlights. At its core, the LM393 compares two voltage levels—the sensor's analog output (for instance, from an LDR) and a preset reference voltage. When the sensor voltage falls above or below this reference threshold, the LM393 outputs a corresponding digital signal. This digital output is then used by the system to modulate the LED brightness, ensuring that the headlights adapt seamlessly to varying ambient light conditions.

The working principle of the LM393 is straightforward yet effective. It contains two independent voltage comparators with open-collector outputs, meaning that each comparator can pull the output low when its input voltage condition is met, or remain in a high-impedance state otherwise. This design allows for easy integration into circuits where quick decision-making based on analog signals is essential. Notably, the LM393 does not require complex digital communication protocols such as I2C or SPI; instead, it uses basic analog-to-digital threshold detection, which simplifies the overall circuitry and enhances response speed.

In our project, the LM393 is particularly beneficial as it translates the continuously varying sensor data—such as the ambient light levels measured by the LDR—into precise control signals. These signals drive the LED brightness adjustments in real time, ensuring that the headlights are neither too bright nor too dim. This dynamic response not only improves visibility during challenging driving conditions but also contributes significantly to road safety by reducing glare and optimizing illumination based on immediate environmental conditions.

2.6 Fire Sensor

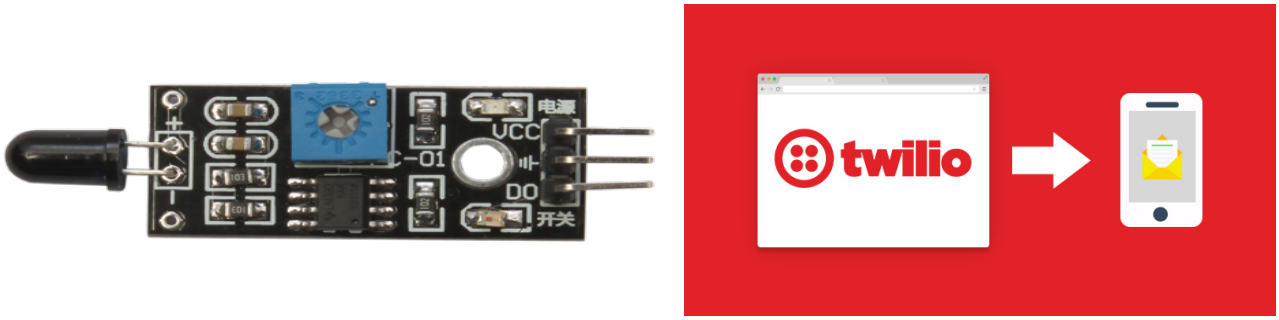


Figure 6: Fire Sensor

The IR-based flame sensor in our project is designed to detect the specific infrared radiation emitted by flames, making it an effective tool for early fire hazard detection. This sensor uses an IR photodiode and optical filters to accurately sense the unique wavelengths produced during combustion. When a flame is present, the sensor's output signal changes—typically shifting from a low to a high state—which the microcontroller interprets as an alert for an internal hazard. In our system, this sensor becomes particularly crucial during scenarios where the driver may not be present, ensuring that any potential fire or combustion incident is detected promptly. Once an anomaly is detected, the microcontroller leverages Twilio's communication API to send an SMS alert in real time. Twilio is renowned for its robust, transactional messaging services, which are essential for ensuring that critical alerts reach the intended recipients without delay. This integration not only helps in mitigating risks by triggering immediate response actions but also enhances the overall reliability and trustworthiness of our safety system, ensuring that no alert is missed or delayed.

2.7 Buzzer Alert

Figure 7: Active Buzzer

An active buzzer is an electromechanical device that produces sound when a DC voltage is applied, thanks to its built-in oscillator circuit. Unlike passive buzzers, which require an external driving signal, an active buzzer generates a consistent tone on its own, making it easier to interface with microcontrollers like the ESP32. The device typically operates within a voltage range of 3–5V and produces a sound output sufficient to alert users even in noisy environments. Internally, the active buzzer converts electrical energy into audible sound through a piezoelectric element that vibrates at a specific frequency—commonly around 2–4 kHz—ensuring the alert is both clear and attention-grabbing.

In our project, the active buzzer serves as an immediate audible warning system. When the system detects hazardous conditions—such as an internal fire through the IR-based flame sensor or other critical alerts—the buzzer is activated to promptly notify the driver or nearby personnel. This audible alert functions as a fail-safe mechanism, ensuring that even if visual alerts or SMS notifications (triggered through Twilio) are overlooked, the buzzer's sound can quickly draw attention to the emergency. The simplicity and reliability of the active buzzer, coupled with its ease of integration and efficient power consumption, make it an essential component in enhancing the overall safety and responsiveness of our adaptive headlight system.

2.8 L298N Motor Driver

The L298N Motor Driver is a dual H-bridge integrated circuit designed to control motors by allowing current to flow in either direction through the load. This capability makes it ideal for managing the operation of DC motors and stepper motors, enabling both forward and reverse motion. Internally, the L298N contains two H-bridges, each of which can independently drive a motor. It accepts low-power control signals from a microcontroller—such as the ESP32 in our project—and amplifies them to drive the higher currents required by the motors. The module also includes built-in flyback diodes to protect against voltage spikes generated by the inductive load of the motors during switching, enhancing its reliability and durability.

In our adaptive headlight system, the L298N is a critical component for controlling the servo motor responsible for adjusting the headlight beam's direction. When sensors, like the ultrasonic sensor, detect oncoming vehicles or obstacles, the ESP32 processes this information and sends corresponding control signals to the L298N. The motor driver then modulates the power supplied to the servo motor, enabling precise adjustments of the headlight angle to reduce glare and improve road safety. This seamless integration between sensor data, microcontroller processing, and motor control ensures that the headlight system responds dynamically to real-time environmental changes.

The L298N stands out in our project for its efficiency in handling the higher power demands of the servo motor while isolating the microcontroller from direct high-current exposure. Its versatility, ease of interfacing, and robust design make it a preferred choice over simpler or less capable motor drivers. Overall, the L298N not only enables smooth and responsive mechanical adjustments but also contributes significantly to the system’s reliability and safety, ensuring that the adaptive headlight system can perform optimally in a variety of driving conditions.

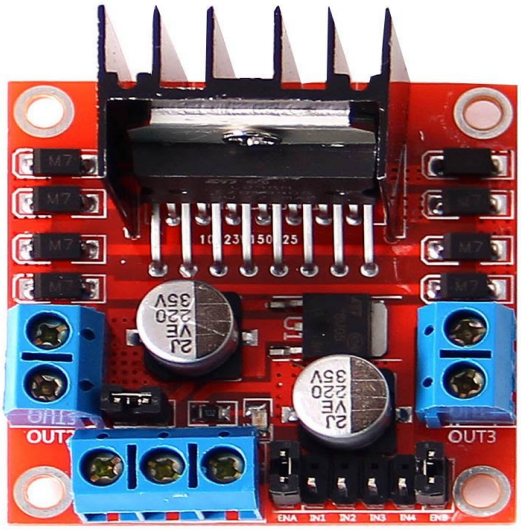


Figure 8: LM298

2.9 LED (Light Emmiting Diode)

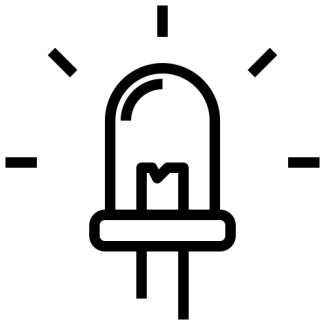


Figure 9: LED

LEDs, or Light Emitting Diodes, are semiconductor devices that convert electrical energy into light through a process known as electroluminescence. When a forward voltage is applied, electrons recombine with holes at the p-n junction within the LED, releasing energy in the form of photons. This process is highly efficient, allowing LEDs to produce bright light with minimal energy loss and heat generation. The specific wavelength—or color—of the light depends on the semiconductor material's bandgap, making LEDs versatile for various lighting applications. In our adaptive headlight system, LEDs serve as the primary source of illumination, offering rapid response times that are crucial for dynamically adjusting brightness in real-time based on sensor inputs. The brightness of the LED can be precisely controlled using pulse width modulation (PWM) signals generated by the ESP32 microcontroller, ensuring that the headlight intensity adapts smoothly to varying environmental conditions such as changes in ambient light, fog, or oncoming vehicle detection. Their energy efficiency, longevity, and ability to withstand frequent switching make LEDs a superior choice over traditional incandescent bulbs, directly contributing to improved road safety and driver comfort. By integrating LEDs with sensor feedback, our system is able to provide optimal lighting conditions that enhance visibility while reducing glare, making them indispensable to the overall functionality of the adaptive headlight system.

2.10 OLED (Organic Light Emmiting Diode)

OLED, which stands for Organic Light Emitting Diode, is a self-illuminating display technology that uses organic compounds to emit light when an electric current is applied. Unlike traditional LCDs, OLEDs don’t require a separate backlight, which makes them energy efficient and capable of producing deep blacks and high-contrast visuals. Typically, these displays are interfaced with microcontrollers like the ESP32 via communication protocols such as I2C or SPI, allowing for simple integration and real-time data display.

In our adaptive headlight system, the OLED display is used as a local user interface to present vital information such as sensor readings, system status, and alert messages. This real-time visualization helps in quickly verifying the performance of the system, troubleshooting issues, and confirming that sensor data (from the LDR, ultrasonic sensor, DHT11, and IR flame sensor) is being accurately captured and processed. The OLED’s clear and crisp display enhances the user experience by offering immediate feedback, which is essential in safety-critical applications. Additionally, its low power consumption and compact size make it an ideal choice for automotive applications where efficiency and space are at a premium.

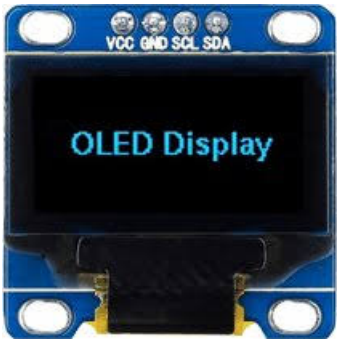


Figure 10: OLED

3. Communication and Software Components

3.1 Thingspeak

In our project, the ESP32's built-in Wi-Fi module enables it to collect sensor data and transmit it to ThingSpeak, a cloud-based IoT platform designed for data storage, visualization, and analysis. The ESP32 sends the sensor readings—such as ambient light levels, temperature, humidity, and distance measurements—using the HTTP protocol. Each sensor value is assigned to a specific field in a ThingSpeak channel, and data is sent as structured information that ThingSpeak processes and logs.

ThingSpeak's role is pivotal: it acts as a centralized repository that not only stores the real-time data but also provides a customizable dashboard for visualizing trends and monitoring system performance over time. While HTTP is commonly used because it is straightforward to implement and compatible with ThingSpeak's API, the platform also supports MQTT. MQTT is a lightweight, publish-subscribe messaging protocol that could be used to minimize overhead and ensure efficient, real-time communication, especially in scenarios where bandwidth and power are limited.

We are using ThingSpeak because it simplifies the process of remote monitoring and data analysis, offering an accessible way to track sensor data without having to build a complex back-end infrastructure. This cloud integration enhances our adaptive headlight system by providing real-time insights and enabling proactive adjustments, which are crucial for ensuring road safety and efficient performance.

3.2 I2C(Inter-Integrated Circuit)

I2C, or Inter-Integrated Circuit, is a synchronous, multi-master, multi-slave communication protocol introduced by Philips Semiconductors (now NXP Semiconductors) in the early 1980s. It was designed to facilitate communication between various integrated circuits using only two wires, which significantly reduces the complexity of wiring in embedded systems. The two fundamental lines used in I2C are:

SDA (Serial Data Line): This line carries the data between devices

SCL (Serial Clock Line): This line provides the clock signal used to synchronize data transfer.

The I2C protocol operates on a master-slave model where the master (in our project, the ESP32) initiates communication, generates the clock signal, and controls the data flow. Each device on the bus is assigned a unique address, which allows the master to communicate with multiple peripherals using just these two lines. The protocol supports various data transfer speeds such as standard mode (100 kbit/s), fast mode (400 kbit/s), and even higher speeds in some applications, making it versatile for different performance requirements.

In our adaptive headlight system, we use I2C primarily to interface the OLED display with the ESP32. The OLED provides a compact, efficient means of displaying real-time sensor data and system status, ensuring that users can quickly monitor and verify the performance of the system. On the ESP32, the I2C pins are typically configured as GPIO21 for SDA and GPIO22 for SCL, though these assignments can be adjusted based on the specific hardware setup. By using I2C, we can efficiently transmit display commands and data over a minimal number of wires, maintaining a clean and robust circuit design.

3.3. Twilio

Twilio is a robust cloud communications platform that provides APIs for sending SMS, making voice calls, and facilitating other communication channels. In our project, Twilio is integrated into the communication module to send immediate SMS alerts in critical situations. For instance, if the IR-based flame sensor detects a potential internal hazard when the driver is absent, the system triggers an alert via Twilio. The ESP32 sends the alert data to our backend, which then uses Twilio's API to dispatch an SMS notification to designated recipients, ensuring that the alert is delivered in real time.This integration is crucial because it ensures that emergency notifications are not solely dependent on local visual or audible alerts, but are also communicated directly to the concerned individuals. The reliability and speed of Twilio's messaging services help to minimize response times during emergencies. Additionally, Twilio's user-friendly API and comprehensive documentation make it easy to implement and maintain, providing a scalable solution for future enhancements in our communication system. Overall, Twilio adds an essential layer of connectivity, ensuring that our adaptive headlight system can effectively alert users and emergency responders, thereby enhancing overall safety and responsiveness.

4. System Working Principle

Initial Setup and Hardware Initialization:

The ESP32 initializes the required hardware components such as the OLED display, motors, LEDs, sensors, etc.

The Wi-Fi connection is established with provided credentials (SSID and password), which allows the ESP32 to communicate with the web interface for control and status updates.

Ultrasonic Distance Measurement:

The ultrasonic sensor (connected to TRIG_PIN and ECHO_PIN) is used to measure the distance from obstacles. The ESP32 sends a pulse to the ultrasonic sensor and measures the time taken for the echo to return.

Based on this time, the distance to the obstacle is calculated and used for various purposes, such as controlling the movement of the car (moving forward, stopping, etc.).

Web Server for Remote Control:

The ESP32 runs a web server that serves a web page (HTML interface) to control the car remotely. Users can issue commands like Move Forward, Move Backward, Move Left, Move Right, Stop, and toggle between Auto Mode and Manual Mode.

When a user clicks a button, the corresponding command is sent to the ESP32 via an HTTP request (/control?cmd=...), which processes the command and moves the motors accordingly.

Motor Control:

The car's motor driver (connected to pins ENA, IN1, IN2, ENB, IN3, and IN4) is used to control the movement of the car.

Functions like moveForward(), moveBackward(), moveLeft(), moveRight(), and stopCar() are defined to control the direction of the car.

In Auto Mode, the car will automatically move forward if the distance is clear (> 60 cm), slow down if the distance is between 30–60 cm, and stop if the distance is less than 15 cm.

LED Control Based on Distance and Light:

The LDR sensor (Light Dependent Resistor) is used to detect the ambient light. Based on the value of the LDR, the brightness of the LEDs is adjusted.

The car adjusts the LED brightness according to the distance from obstacles and the ambient light detected by the LDR.

Fire Detection and Emergency Response:

The flame sensor is used to detect the presence of fire. If fire is detected (i.e., the flame sensor reads LOW), the following actions are triggered:

Buzzer is activated (indicating fire).

An emergency SMS is sent via Twilio API to notify the recipient (a predefined phone number). The SMS contains a message: "Emergency! Fire detected by the sensor!".

The function sendEmergencySMS() constructs the request payload for Twilio and sends the message. The response from Twilio is checked to confirm if the SMS was successfully sent.

Temperature and Humidity Monitoring:

The DHT11 sensor is used to measure the temperature and humidity of the environment. These values are displayed on the OLED display, alongside the motor RPM and flame sensor status.

RPM (Rotations Per Minute) Monitoring:

An RPM sensor is used to count pulses generated by the motor. Each pulse represents a rotation, and the number of pulses counted within 1 second is used to calculate the RPM.

The RPM is displayed on the OLED display for monitoring the motor speed.

OLED Display:

The OLED display shows real-time status information such as:

Motor RPM

Temperature

Humidity

Flame Sensor Status (Fire or No Fire)

The display is updated regularly with these values to provide the user with the current status of the system.

Key Interactions:

Web Interface: A user can interact with the car remotely via a browser. The web interface allows the user to send commands to the ESP32 to control the car’s movements and toggle between auto and manual modes.

Auto Mode: In Auto Mode, the car automatically moves forward or stops depending on the distance detected by the ultrasonic sensor.

Fire Detection: If the flame sensor detects fire, it triggers an alarm (buzzer) and sends an emergency SMS via Twilio.

Sensor Feedback: The car continuously monitors the environment through sensors like the ultrasonic sensor (distance), LDR (light), DHT11 (temperature and humidity), and flame sensor (fire). This information is displayed on the OLED screen for the user to view.

5. Code

```
#include <Arduino.h>
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>
#include <Base64.h>

// OLED Display Settings
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C  // I2C address for the OLED display

// New OLED Pins
#define OLED_SDA 13  // GPIO 13 (D7) for SDA
#define OLED_SCL 17  // GPIO 17 (D9) for SCL

// Ultrasonic Sensor Pins
#define TRIG_PIN 5
#define ECHO_PIN 18

// LED Pins
#define LED1_PIN 4
#define LED2_PIN 2
#define LED3_PIN 15
#define LED4_PIN 16

// LDR Pin
#define LDR_PIN 32

// Motor Driver Pins
#define ENA 22
#define IN1 19
#define IN2 21
#define ENB 26
#define IN3 23
#define IN4 25

// Flame Sensor and Buzzer Pins
#define FLAME_SENSOR_DIGITAL_PIN 33
#define BUZZER_PIN 14

// DHT11 Sensor Pin
#define DHTPIN 27
#define DHTTYPE DHT11

// RPM Sensor Pin
#define SENSOR_PIN 34  // D0 connected to GPIO34

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Twilio credentials
const char* accountSID = "ACd72039eb7162144d15f9ab852dea04d0";  // Replace with your Twilio Account SID
const char* authToken = "c8adc8f9dd4225391aa0b56efbc5e4f2";    // Replace with your Twilio Auth Token
const char* fromNumber = "+13159083986";  // Replace with your Twilio phone number
const char* toNumber = "+919866844335"; // Replace with the recipient phone number

WiFiClient client ;  // Secure client for HTTPS
WebServer server(80);
bool autoMode = false;
```

```
DHT dht(DHTPIN, DHTTYPE);

// RPM Sensor Variables
volatile int pulseCount = 0;
unsigned long prevTime = 0;
float rpm = 0;

// OLED Display Object
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Function Declarations
void IRAM_ATTR countPulses();
void moveForward();
void moveBackward();
void moveLeft();
void moveRight();
void stopCar();
float getDistance();
void handleRoot();
void handleControl();
void updateOLED(float rpm, float temperature, float humidity, int flameDigital);
void sendEmergencySMS();

// Interrupt Service Routine (ISR) for RPM Sensor
void IRAM_ATTR countPulses() {
  pulseCount++;
}

void setup() {
  Serial.begin(115200);

  // Initialize I2C with new pins
  Wire.begin(OLED_SDA, OLED_SCL);

  // Initialize OLED Display
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {  // Corrected initialization
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Halt forever if OLED fails
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("Initializing...");
  display.display();

  // Ultrasonic Sensor
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // LED Pins
  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(LED3_PIN, OUTPUT);
  pinMode(LED4_PIN, OUTPUT);

  // LDR
  pinMode(LDR_PIN, INPUT);

  // Motor Pins
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
```

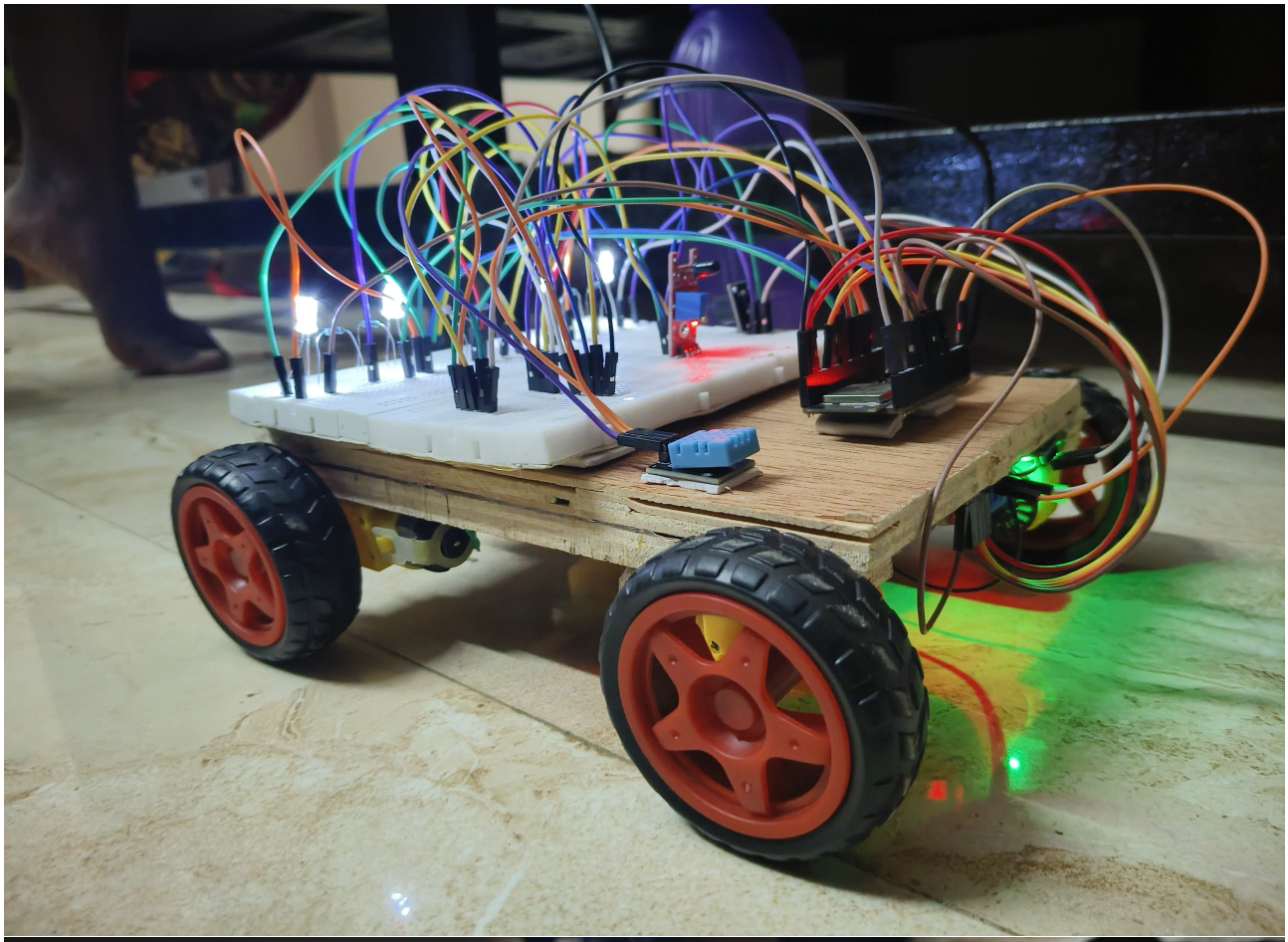
```
  // Flame Sensor and Buzzer
   pinMode(FLAME_SENSOR_DIGITAL_PIN, INPUT);
   pinMode(BUZZER_PIN, OUTPUT);

   dht.begin();    // DHT11 Sensor

   // RPM Sensor
   pinMode(SENSOR_PIN, INPUT);
   attachInterrupt(digitalPinToInterrupt(SENSOR_PIN), countPulses, RISING);

   stopCar(); // Ensure motors are stopped on startup

   // Connect to WiFi
   WiFi.begin(ssid, password);
   Serial.print("Connecting to WiFi...");
   while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
   }

   Serial.println("\nWiFi connected!");
   Serial.print("IP Address: ");
   Serial.println(WiFi.localIP());

   server.on("/", handleRoot);
   server.on("/control", handleControl);
   server.begin();
}

void moveForward() {
   analogWrite(ENA, 255);
   analogWrite(ENB, 255);
   digitalWrite(IN1, HIGH);
   digitalWrite(IN2, LOW);
   digitalWrite(IN3, HIGH);
   digitalWrite(IN4, LOW);
}

void moveBackward() {
   analogWrite(ENA, 255);
   analogWrite(ENB, 255);
   digitalWrite(IN1, LOW);
   digitalWrite(IN2, HIGH);
   digitalWrite(IN3, LOW);
   digitalWrite(IN4, HIGH);
}

void moveLeft() {
   analogWrite(ENA, 255);
   analogWrite(ENB, 255);
   digitalWrite(IN1, LOW);
   digitalWrite(IN2, HIGH);
   digitalWrite(IN3, HIGH);
   digitalWrite(IN4, LOW);
}

void moveRight() {
   analogWrite(ENA, 255);
   analogWrite(ENB, 255);
   digitalWrite(IN1, HIGH);
   digitalWrite(IN2, LOW);
   digitalWrite(IN3, LOW);
   digitalWrite(IN4, HIGH);
}

void stopCar() {
   digitalWrite(IN1, LOW);
   digitalWrite(IN2, LOW);
   digitalWrite(IN3, LOW);
   digitalWrite(IN4, LOW);
}
```

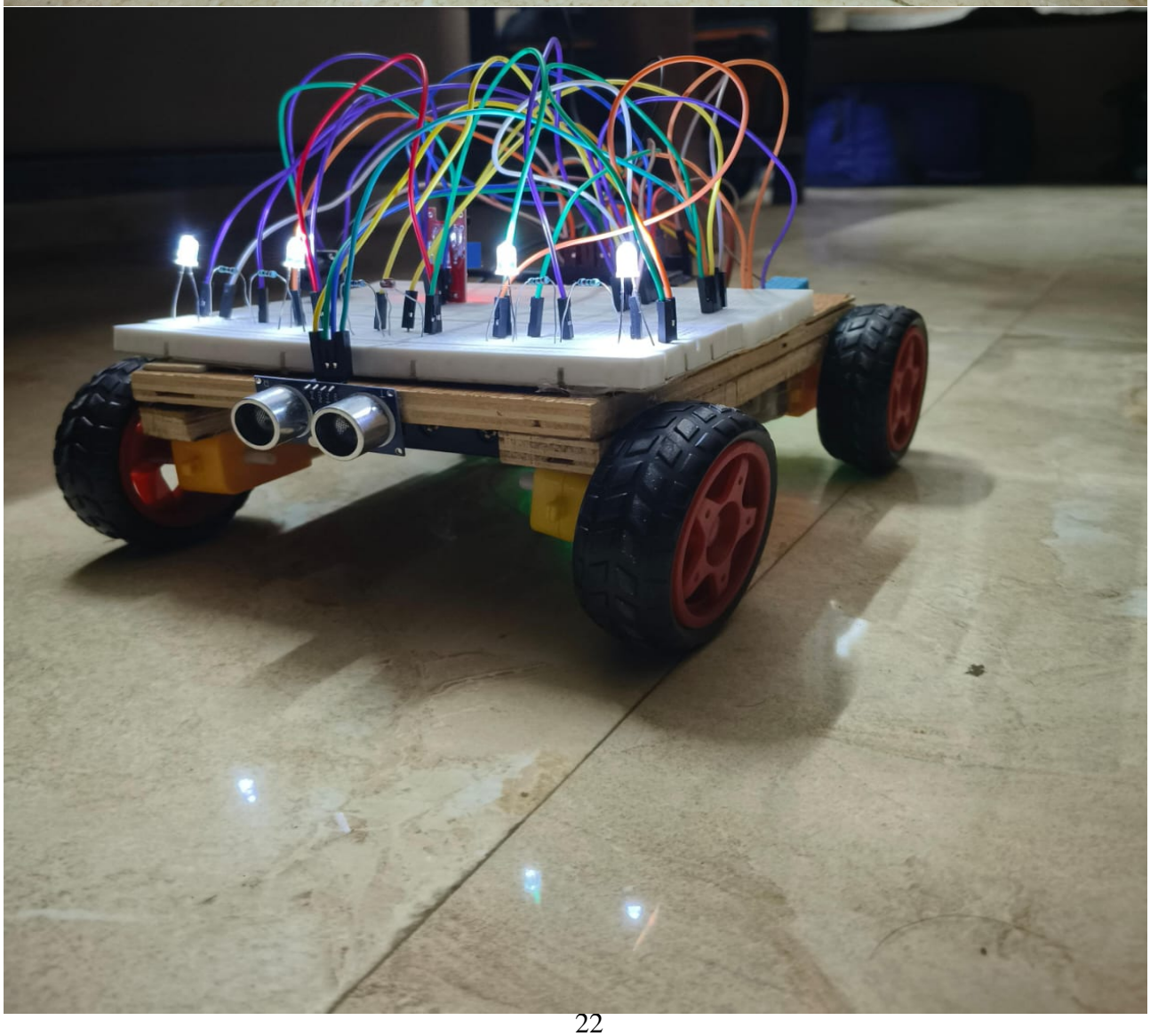
```
// Function to get stable ultrasonic distance
float getDistance() {
  float total = 0;
  int validReadings = 0;

  for (int i = 0; i < 3; i++) {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    long duration = pulseIn(ECHO_PIN, HIGH, 30000);
    if (duration > 0) {
      float distance = (duration * 0.0343) / 2;
      total += distance;
      validReadings++;
    }
    delay(50);
  }

  if (validReadings == 0) {
    Serial.println("No valid ultrasonic reading!");
    return -1;
  }

  float avgDistance = total / validReadings;
  Serial.print("Ultrasonic Distance: ");
  Serial.println(avgDistance);
  return avgDistance;
}

void handleRoot() {
  String html = R"rawliteral(
    <!DOCTYPE html>
    <html>
    <head>
      <title>Car Control</title>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <style>
        body { font-family: Arial, sans-serif; text-align: center; }
        .btn { font-size: 20px; padding: 15px; width: 120px; margin: 5px; }
        .stop { background-color: red; color: white; }
      </style>
    </head>
    <body>
      <h1>ESP32 Car Control</h1>
      <button class="btn" onclick="sendCommand('F')">Forward</button><br>
      <button class="btn" onclick="sendCommand('L')">Left</button>
      <button class="btn stop" onclick="sendCommand('S')">Stop</button>
      <button class="btn" onclick="sendCommand('R')">Right</button><br>
      <button class="btn" onclick="sendCommand('B')">Backward</button><br>
      <button class="btn" onclick="sendCommand('AUTO')">Auto Mode</button>
      <button class="btn" onclick="sendCommand('MANUAL')">Manual Mode</button>

      <script>
        function sendCommand(cmd) {
          fetch("/control?cmd=" + cmd);
        }
      </script>
    </body>
    </html>
  )rawliteral";

  server.send(200, "text/html", html);
}
```

```
void handleControl() {
  if (!server.hasArg("cmd")) {
    server.send(400, "text/plain", "Missing command");
    return;
  }

  String command = server.arg("cmd");
  Serial.println("Command received: " + command);

  if (command == "F") moveForward();
  else if (command == "B") moveBackward();
  else if (command == "L") moveLeft();
  else if (command == "R") moveRight();
  else if (command == "S") stopCar();
  else if (command == "AUTO") autoMode = true;
  else if (command == "MANUAL") autoMode = false;

  server.send(200, "text/plain", "OK");
}

// Function to update OLED display with only speed, temperature, humidity, and flame status
void updateOLED(float rpm, float temperature, float humidity, int flameDigital) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("ESP32 Car Status");
  display.println("-----------------");
  display.print("Speed: ");
  display.print(rpm);
  display.println(" RPM");
  display.print("Temp: ");
  display.print(temperature);
  display.println(" C");
  display.print("Humidity: ");
  display.print(humidity);
  display.println(" %");
  display.print("Flame: ");
  display.println(flameDigital ? "Fire" : " NO Fire!");
  display.display();
}

void sendEmergencySMS() {
  Serial.println("Sending SMS via Twilio...");

String payload = "To=" + String(toNumber) +
            "&MessagingServiceSid=" + "MG98417dc5337cde752549b983d822d61d" +  // Use the Message Service SID
            "&Body=" + "Emergency! Fire detected by the sensor!";

  Serial.println("Payload: " + payload);

  // Attempt to connect to Twilio's API
  if (client.connect("api.twilio.com", 443)) {
    Serial.println("Connected to Twilio API");

    // Create the authorization header with base64 encoding
    String authHeader = "Basic " + base64::encode(String(accountSID) + ":" + String(authToken));


    // Send the HTTP POST request to Twilio

    client.println("POST /2010-04-01/Accounts/" + String(accountSID) + "/Messages.json HTTP/1.1");
    client.println("Host: api.twilio.com");
    client.println("Authorization: " + authHeader);
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.println("Content-Length: " + String(payload.length()));
    client.println();  // Blank line between headers and body
    client.println(payload);  // The data (message to be sent)

    Serial.println("Request sent to Twilio");
```

```
        // Wait for response from Twilio
        delay(2000);  // Wait for response from Twilio (may take a second)

        // Check response from Twilio
        bool success = false;
        while (client.available()) {
            String line = client.readStringUntil('\n');
            Serial.println(line);  // Print the response from Twilio for debugging

            // Check if Twilio API responds with a success status code
            if (line.indexOf("HTTP/1.1 200 OK") >= 0) {
                success = true;
            }
        }

        if (success) {
            Serial.println("SMS sent successfully!");
        } else {
            Serial.println("Failed to send SMS. Check the Twilio API response.");
        }
    } else {
        Serial.println("Failed to connect to Twilio API. Check network connectivity or Twilio settings.");
    }
}

void loop() {
    server.handleClient(); // Handle web server requests

    int ldrValue = analogRead(LDR_PIN); // Read LDR value
    float distance = getDistance();

    if (distance > 0) {
        Serial.print("Distance: ");
        Serial.print(distance);
        Serial.println(" cm");

        // Map distance (0-100 cm) to LED brightness (0-255)
        int brightness = map(constrain(distance, 0, 100), 0, 100, 255, 50);
        int oppositeBrightness = 255 - brightness;

        // Adjust brightness based on LDR reading
        if (ldrValue > 2048) { // Bright environment
            brightness = 255 - brightness;
        }

        analogWrite(LED1_PIN, brightness);
        analogWrite(LED2_PIN, brightness);
        analogWrite(LED3_PIN, oppositeBrightness);
        analogWrite(LED4_PIN, oppositeBrightness);
    }

    if (autoMode) {
        if (distance > 60) {
            Serial.println("Moving Forward...");
            moveForward();
        } else if (distance > 30) {
            Serial.println("Obstacle detected, slowing down...");
            moveForward();
        } else if (distance > 15) {
            Serial.println("Obstacle detected! Stopping...");
            stopCar();
        }
    }

    // Flame Sensor and Buzzer Logic
    int flameDigital = digitalRead(FLAME_SENSOR_DIGITAL_PIN);
    Serial.print("Fire Status: ");
    Serial.println(flameDigital);  // 0 = Fire detected, 1 = No fire
```

```
  if (flameDigital == 1) {  // Fire detected (assuming LOW means fire detected)
    Serial.println("      Fire detected! Activating buzzer and sending SMS...");

    digitalWrite(BUZZER_PIN, HIGH);  // Turn on buzzer

    // Send an emergency SMS via Twilio
    sendEmergencySMS();
  } else {
    Serial.println("      No fire detected. Buzzer OFF.");
    digitalWrite(BUZZER_PIN, LOW);
  }

  // DHT11 Sensor Logic
  float temperature = dht.readTemperature();  // Read temperature (Celsius)
  float humidity = dht.readHumidity();       // Read humidity (%)

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("      Failed to read from DHT sensor!");
  } else {
    Serial.print("      Temperature: ");
    Serial.print(temperature);
    Serial.println("°C");

    Serial.print("      Humidity: ");
    Serial.print(humidity);
    Serial.println("%");
  }

  // RPM Sensor Logic
  unsigned long currentTime = millis();

  // Calculate RPM every second
  if (currentTime - prevTime >= 1000) {
    detachInterrupt(digitalPinToInterrupt(SENSOR_PIN));  // Temporarily disable interrupt

    rpm = (pulseCount * 60.0) / 20.0;  // Assuming 20 pulses per revolution

    Serial.print("Motor Speed: ");
    Serial.print(rpm);
    Serial.println(" RPM");

    pulseCount = 0;
    prevTime = currentTime;

    attachInterrupt(digitalPinToInterrupt(SENSOR_PIN), countPulses, RISING); // Re-enable interrupt
  }

  // Update OLED Display with only speed, temperature, humidity, and flame status
  updateOLED(rpm, temperature, humidity, flameDigital);

  delay(100);
}
```





Conclusion and Future Scope

Enhanced Road Safety:

The adaptive headlight system significantly reduces the risk of accidents by dynamically adjusting illumination based on real-time sensor data, ensuring improved visibility during adverse conditions.

Real-time Sensor Integration:

By integrating sensors like the ultrasonic, LDR, DHT11, and IR-based flame sensor, the system can quickly respond to changes in the environment, providing immediate adjustments for optimal safety.

Robust Hardware Architecture:

Leveraging the capabilities of the ESP32, L298N motor driver, and efficient LED arrays, the project establishes a reliable and responsive platform that can withstand diverse operational challenges.

Seamless Cloud Connectivity:

Utilizing ThingSpeak for data transmission, storage, and visualization enables remote monitoring and efficient data analysis, transforming raw sensor data into actionable insights.

Proactive Alerting Mechanism:

With integrated alert systems such as active buzzers and Twilio-based SMS notifications, the project ensures that potential hazards are communicated promptly to drivers or relevant authorities.

Energy Efficiency:

The use of low-power components and LED technology ensures that the system operates with minimal energy consumption, making it both sustainable and cost-effective over time.

Modular and Scalable Design:

The project's architecture allows for future expansion by easily incorporating additional sensors or advanced functionalities, ensuring adaptability to evolving technology requirements.

User-friendly Interface:

The ThingSpeak dashboard offers an intuitive platform for visualizing real-time data and monitoring system performance, enhancing user interaction and decision-making.

Advanced Algorithm Development:

Future enhancements could include integrating AI/ML algorithms for predictive analytics, which would enable more precise and proactive headlight adjustments based on traffic patterns and environmental conditions.

Integration with Smart City Infrastructure:

The system has the potential to be part of larger IoT networks and smart city frameworks, contributing to coordinated traffic management, enhanced public safety, and overall improved urban mobility.