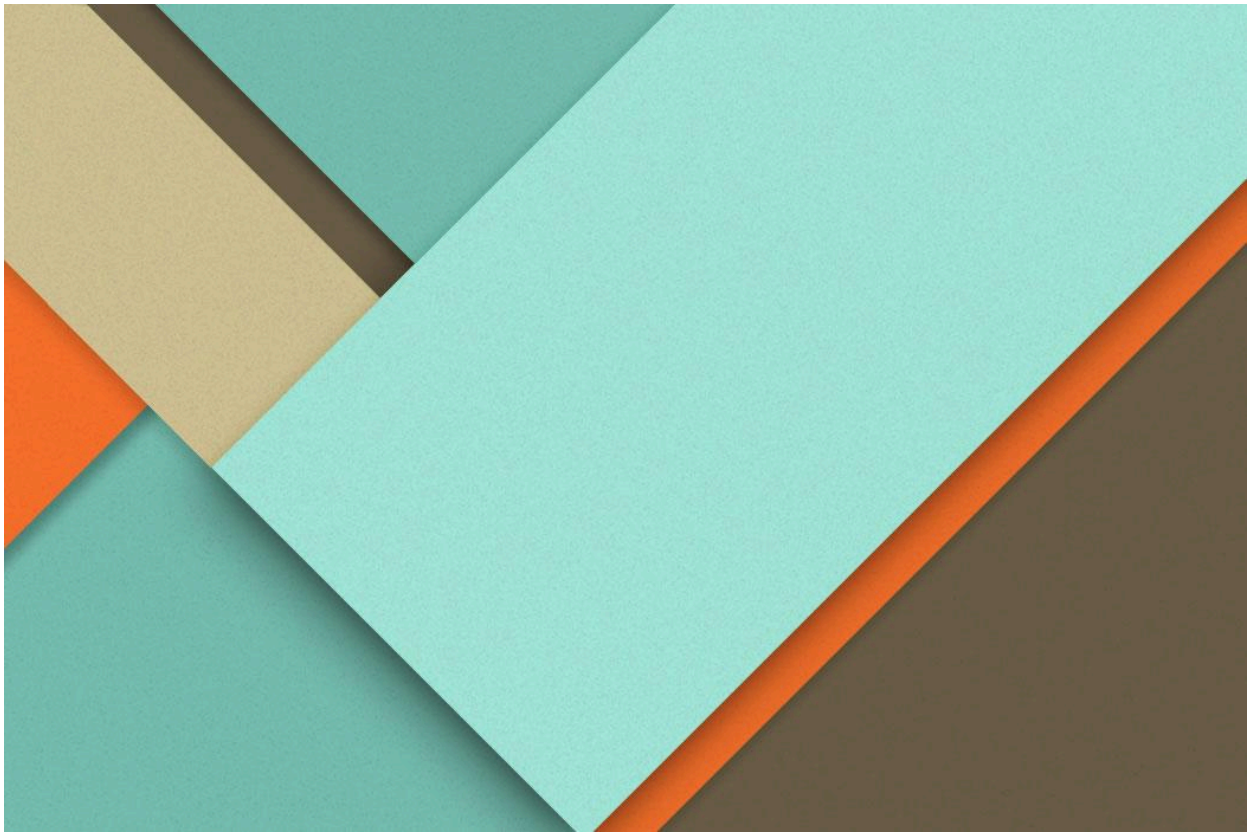


ESP8266 Traffic Density-Based Traffic Light Control System Manual

07.11.2024

Rishika Manocha
22UCS167

Sheetal Sharma
22UCS195

Acknowledgments

We would like to express our heartfelt gratitude to **Dr. Rajbir Kaur**, Head of the Department of Computer Science and Engineering at **The LNM Institute of Information Technology** and professor of the *Communication Technologies For Internet of Things (CTIoT)* course, under whose mentorship we undertook this project. Her invaluable guidance, expert insights, and continued support throughout the duration of this project were instrumental in shaping our approach and achieving our objectives. Dr. Kaur's mentorship has been a cornerstone in driving the project to completion, and we are sincerely grateful for her leadership.

We extend special thanks to **Mr. Shivam Maheshwari**, Technical Superintendent in the Computer Programming Lab, for his unwavering support in providing essential resources, critical technical assistance, and logistical aid throughout this project. His contributions were crucial to ensuring the project's successful implementation.

We are also grateful to **Mr. Udayveer Singh**, Junior Technical Superintendent of the Mechatronics Lab, for providing the IR sensors and additional equipment needed for our traffic modulation system.

We also acknowledge the collaboration and dedication within our team. Together, as **Rishika Manocha** and **Sheetal Sharma**, we divided responsibilities, tackled challenges, and developed a deeper understanding of IoT systems, traffic management, and embedded technologies.

We are sincerely grateful for the combined support from our professors, technical staff, and team members, whose contributions were fundamental to the project's success. In conclusion, we thank everyone who contributed their expertise and resources, making this project a rewarding and enriching experience.

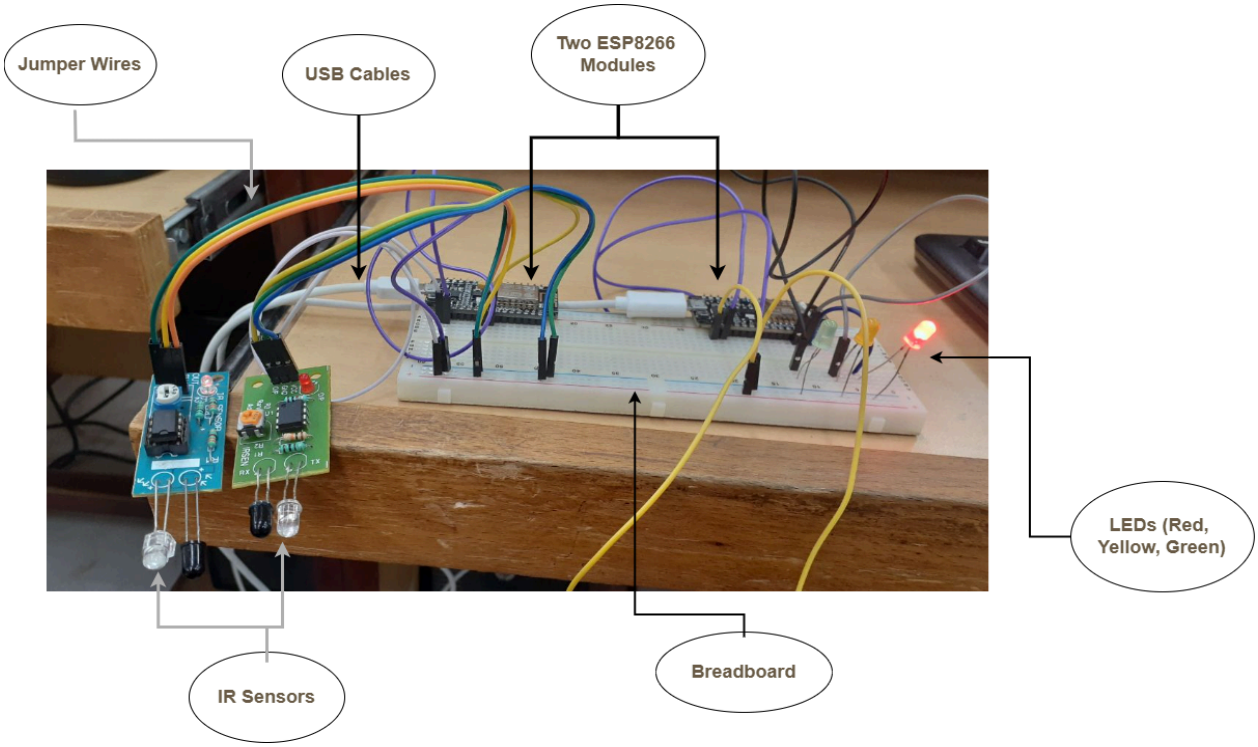
The LNM Institute of Information Technology
Deemed university in Jaipur, Rajasthan

Objective

This project simulates an IoT-based traffic control system using ESP8266 modules and IR sensors to detect vehicle density. Based on the traffic density (high, medium, or low), the duration of the green and red lights is modulated. This is a prototype for real-world applications in smart traffic management systems.

Hardware Requirements

1. **ESP8266 Modules (x2)** : One for sensing density and sending data; the other for controlling the traffic lights.
2. **USB Cables (x2)** – To power and program each ESP8266 module
3. **LEDs (Red, Yellow, Green)**: Represent traffic lights.
4. **IR Sensors (x2)**: Detects vehicle presence and density.
5. **Breadboard**: For component connections.
6. **Jumper Wires**: For wiring and connectivity.



Software Requirements

1. **Arduino IDE**: Used for programming the ESP8266 modules.
2. **Libraries:**
    a. **ESP8266WiFi.h –** to enable WiFi connectivity.
    b. **ESP8266HTTPClient.h –** to make HTTP requests.
    c. **ESP8266WebServer.h –** to set up a server on ESP8266.

Connection Procedure

1. **Connect the ESP8266 Modules**:
    - Designate one ESP8266 module as the *client* (ESP2) and the other as the *server* (ESP1).
    - Power both modules using USB cables connected to your computer.
2. **IR Sensors to ESP2 (Client)**:
    - Connect the *VCC* pin of each IR sensor to the *3.3V* pin on ESP2.
    - Connect the *GND* pin of each IR sensor to a *GND* pin on ESP2.
    - Attach the *OUT* pin of each IR sensor to two separate GPIO pins on ESP2 (for example in our case, D5 and D6).
3. **Traffic Light LEDs to ESP1 (Server)**:
    - Connect each LED (Red, Yellow, Green) to the *D1, D2, and D3* GPIO pins on ESP1, respectively.
    - Connect the *negative* (cathode) leg of each LED to the *GND* pin on ESP1.
4. **Wi-Fi Configuration**:
    - Ensure both ESP8266 modules are connected to the same Wi-Fi network.
    - Set a *static IP address* for each module in the code to ensure stable communication between the server and client.
5. **Testing Connections**:
    - After setting up, upload the client and server codes to ESP2 and ESP1, respectively.
    - Use Serial Monitors to verify connection status and troubleshoot any connectivity issues.

Code

I. ESP1 Code (Traffic Light Controller)

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

// Wi-Fi Credentials
const char* ssid = "my_network";
const char* password = "secret";

// Traffic Light Pins
const int redPin = D1;
const int yellowPin = D2;
const int greenPin = D3;
ESP8266WebServer server(80); // HTTP server on port 80
String density = "low"; // Default traffic density
void setup() {
Serial.begin(9600);

// Set LED pins as OUTPUT
pinMode(redPin, OUTPUT);
pinMode(yellowPin, OUTPUT);
pinMode(greenPin, OUTPUT);

// Static IP Configuration for ESP1
IPAddress local_IP(192, 168, 43, 99);
IPAddress gateway(192, 168, 43, 1);
IPAddress subnet(255, 255, 255, 0);
WiFi.config(local_IP, gateway, subnet);
WiFi.begin(ssid, password);

// Connect to Wi-Fi
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nConnected! IP Address: " +
WiFi.localIP().toString());

// Define route to receive density data
server.on("/density", HTTP_POST, []() {
if (server.hasArg("density")) {
density = server.arg("density");
```

```
Serial.println("Received density: " + density);
4
server.send(200, "text/plain", "Density updated");
} else {
server.send(400, "text/plain", "Bad Request: 'density' parameter
missing");
}
});
server.begin(); // Start the server
Serial.println("Server started");
}

void loop() {
server.handleClient(); // Handle HTTP requests
if (density == "high") {
runTrafficCycle(12000, 5000); // High traffic: Long green, short
red
} else if (density == "medium") {
runTrafficCycle(8000, 8000); // Medium traffic: Balanced green and
red
} else {
runTrafficCycle(5000, 12000); // Low traffic: Short green, long
red
}
}

// Traffic light cycle function
void runTrafficCycle(int greenDuration, int redDuration) {
greenLight(greenDuration);
yellowLight(2000); // Yellow light for 2s
redLight(redDuration);
}
void greenLight(int duration) {
digitalWrite(greenPin, HIGH);
delay(duration);
digitalWrite(greenPin, LOW);
}
void yellowLight(int duration) {
digitalWrite(yellowPin, HIGH);
delay(duration);
digitalWrite(yellowPin, LOW);
}
void redLight(int duration) {
digitalWrite(redPin, HIGH);
delay(duration);
digitalWrite(redPin, LOW);
}
```

II. ESP2 Code (Density Sensor and Transmitter)

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// Wi-Fi Credentials
const char* ssid = "my_network";
const char* password = "secret";

// Server IP and Static IP Configuration
IPAddress serverIP(192, 168, 43, 99); // ESP1 IP Address
IPAddress local_IP(192, 168, 43, 100); // ESP2 Static IP
IPAddress gateway(192, 168, 43, 1);
IPAddress subnet(255, 255, 255, 0);

// IR Sensor Pins
const int irSensor1 = D5;
const int irSensor2 = D6;

void setup() {
Serial.begin(9600);
pinMode(irSensor1, INPUT);
pinMode(irSensor2, INPUT);

// Configure Static IP
WiFi.config(local_IP, gateway, subnet);
WiFi.begin(ssid, password);

// Connect to Wi-Fi
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nConnected! Client IP Address: " +
WiFi.localIP().toString());
}

void loop() {
int vehicleCount = getVehicleCount();
String density = getDensity(vehicleCount);
if (WiFi.status() == WL_CONNECTED) {
WiFiClient client;
HTTPClient http;
String url = "http://" + serverIP.toString() + "/density";
```

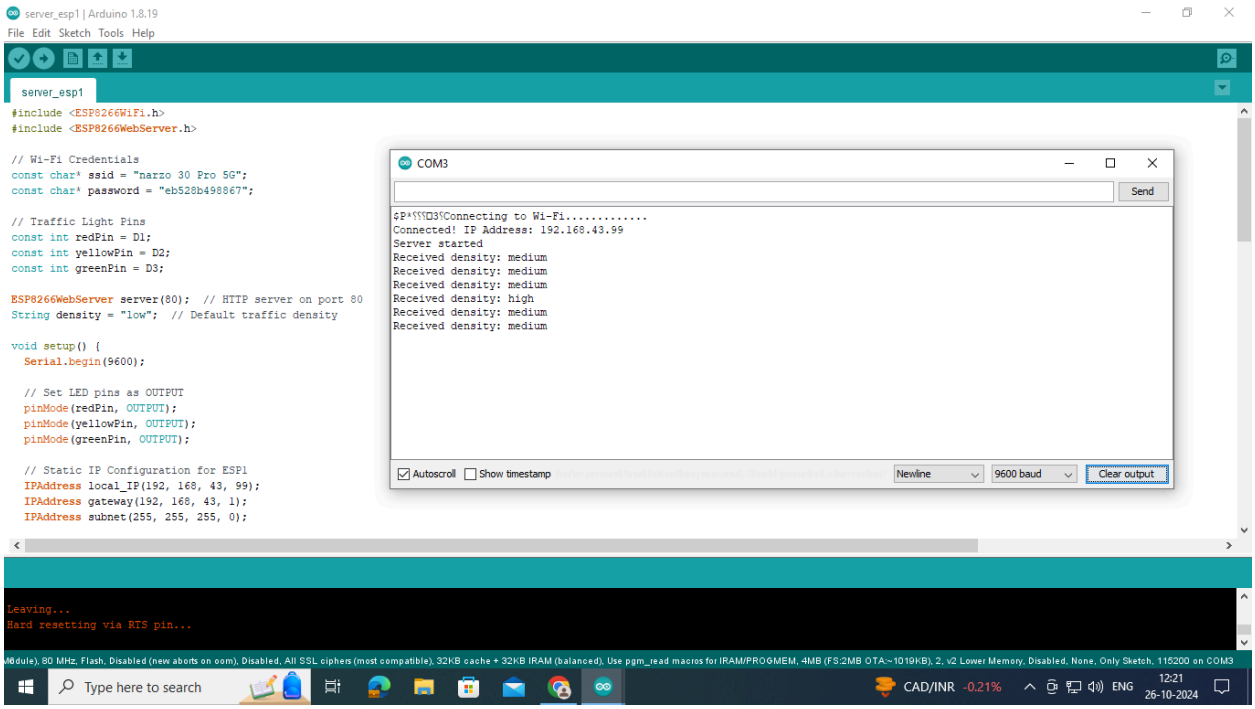
```
http.begin(client, url);
http.addHeader("Content-Type","application/x-www-form-urlencoded";
http.setTimeout(1000); // Set timeout to 10 seconds 6
String postData = "density=" + density;
int httpResponseCode = http.POST(postData);
Serial.println("Sending Density... ");
http.end(); // Close connection
}
delay(5000); // Wait 5 seconds before next request
}

// Function to get vehicle count from IR sensors
int getVehicleCount() {
int count = 0;
if (digitalRead(irSensor1) == LOW) { // Vehicle detected on lane 1
count++;
}
if (digitalRead(irSensor2) == LOW) { // Vehicle detected on lane 2
count++;
}
return count;
}

// Function to determine traffic density
String getDensity(int vehicleCount) {
if (vehicleCount >= 2) {
return "high";
} else if (vehicleCount == 1) {
return "medium";
} else {
return "low";
}
}
```

Visual Documentation

1. **ESP1 Server**

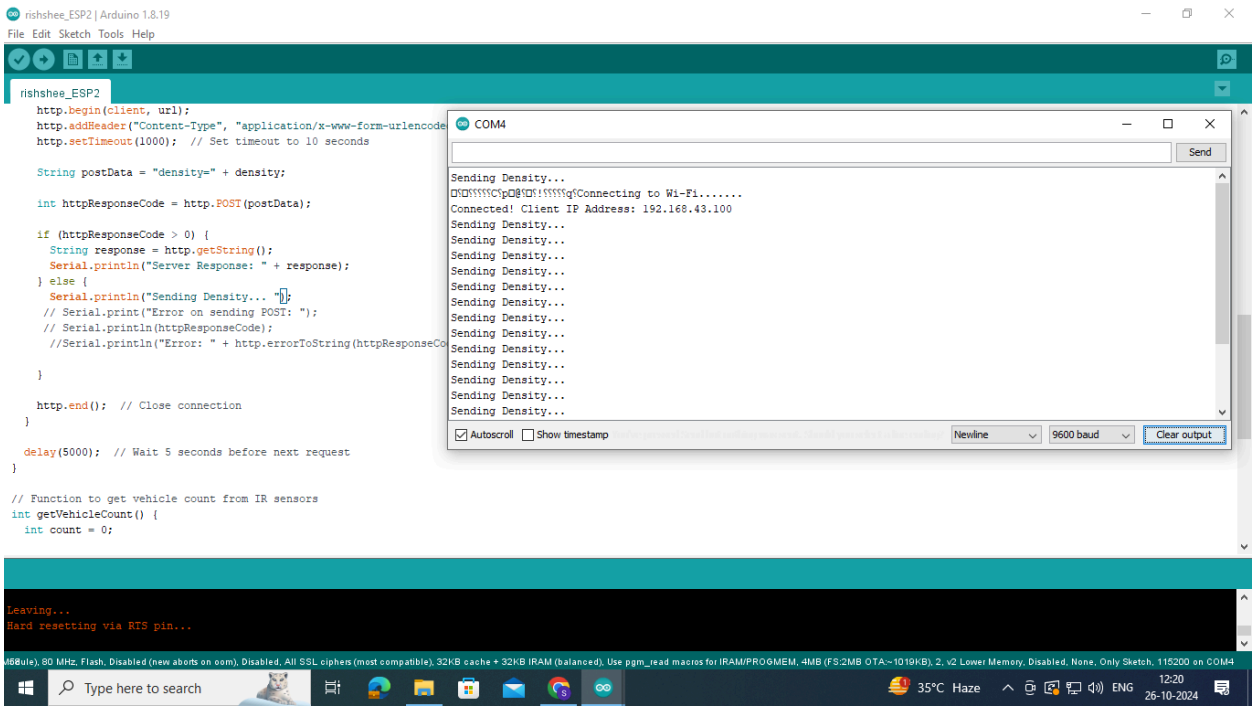


The image shows the serial monitor output of the ESP1 module configured as the server. Once the ESP1 successfully connects to the Wi-Fi network, the following details are displayed on the console:

1. **Wi-Fi Connection Status**: A confirmation message indicating that the ESP1 has connected to the specified Wi-Fi network.
2. **IP Address**: The static IP address assigned to ESP1, which is essential for communication with the client (ESP2).
3. **Server Initialization**: A message stating "Server started," indicating that the ESP1 is ready to receive data from the client.

This output verifies that the ESP1 module is functioning as expected and is ready to process traffic density data to control the traffic light sequence.

2. ESP2 Client



The image shows the serial monitor output of the ESP2 module configured as the client. This output verifies that ESP2 is functioning correctly and successfully communicating with ESP1. The key details displayed on the console are:

1. **Wi-Fi Connection Status**: A confirmation message indicating that the ESP2 has successfully connected to the specified Wi-Fi network. This ensures the client module is online and ready to send data.
2. **Client IP Address**: The static IP address assigned to ESP2, which confirms its unique identification on the network and enables seamless communication with ESP1.
3. **Sending Density Data**: A message confirming that ESP2 is actively sending vehicle density data ("low," "medium," or "high") to ESP1 based on inputs from the IR sensors. This step is crucial for dynamically controlling the traffic lights.

This output validates that the ESP2 client is gathering data and attempting to transmit it, ensuring the project's functionality.

3. Video Link of Project

[Click to watch video]

Problems Faced and Solutions

- **Connection Issues**: Faced WiFi disconnection issues; resolved by setting static IPs for both ESP modules. Also make sure you have a strong WiFi connection.
- **Timeout Errors**: Initial configuration issues with HTTPClient; replaced random number density generation with actual IR sensor values.
- **Getting this error: A fatal esptool.py error occurred: Failed to connect to ESP8266: Timed out waiting for packet header_**
    1. Firstly check all your hardware connections properly.
    2. Make sure all the required libraries are properly installed.
    3. You are using the latest version.
    4. Make sure all the required drivers are installed.
    5. Make sure USB cables are properly connected.

FAQs

1. **Q: What if ESP8266 fails to connect to Wi-Fi?**
    - **A:** Verify the SSID and password, and ensure the router is within range.
2. **Q: Why do LEDs not respond to density changes?**
    - **A:** Check connections and ensure IR sensors are correctly detecting density.
    - **A:** Also properly tune the IR sensors.