Hybrid Smart Home Assistant with Offline Voice Control and Manual Operation via Node-RED Dashboard

In this project, we have built a **smart home automation system** that allows you to control electrical appliances using **both voice commands (offline) and a web-based dashboard (Node-RED).** The system uses:

VC02 Voice Recognition Module → To recognize voice commands offline
NodeMCU (ESP8266) → To process commands and control relays
MQTT Protocol → To communicate between NodeMCU and Raspberry Pi
Raspberry Pi 3 Model B+ → Acts as an MQTT broker and runs Node-RED
Four-Channel Relay Module (with Optocoupler) → Controls electrical appliances
Node-RED Dashboard → Provides a web interface for manual control

🔹 How It Works

1. **Voice Control (VC02 + NodeMCU)**

- The **VC02 voice module** listens for specific voice commands.
- If a command is recognized (e.g., "Turn on the light"), it is processed by **NodeMCU**.
- The NodeMCU activates the **corresponding relay**, switching the appliance ON/OFF.

2. **Manual Control via Node-RED (Raspberry Pi + MQTT + NodeMCU)**

- A **Node-RED dashboard** (hosted on the Raspberry Pi) provides ON/OFF buttons.
- When a button is pressed, an **MQTT message** is sent to the NodeMCU.
- The **NodeMCU reads the MQTT message** and controls the respective relay accordingly.

3. **MQTT Communication (Raspberry Pi as Broker, NodeMCU as Client)**

- The **Raspberry Pi runs an MQTT broker (Mosquitto)** to manage messages.
- The **NodeMCU subscribes to MQTT topics** (e.g., `relay/1`, `relay/2` etc.).
- When a message is published (ON/OFF), the NodeMCU updates the relay state.

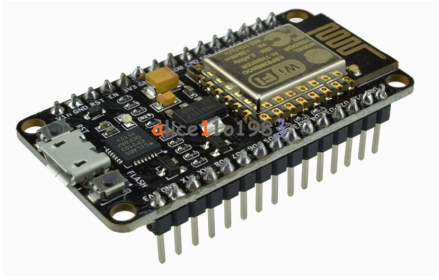
1. **Voice Control:** You can turn appliances ON/OFF using predefined voice commands.
2. **Manual Control:** You can control appliances from a **web-based dashboard** on any device.
3. **Real-Time Feedback:** The system instantly updates relay states based on voice or manual input.
4. **Offline Operation:** The **VC02 module works without the internet**, making voice control reliable.
5. **Smart IoT Integration:** Using **MQTT and Node-RED**, this system is scalable for future automation.

Components Used in the Project

VC 02 - Offline voice recognition module

The Seeed Studio Grove – Offline Voice Recognition Module is a compact and versatile device designed for integrating voice control into various electronic projects. This module utilizes advanced algorithms to recognize a wide range of spoken commands, enabling users to interact with their projects without requiring an internet connection. With its offline functionality, the Grove module ensures privacy and reliability, making it suitable for applications where data security is paramount. Its Grove interface allows for easy integration with a variety of microcontrollers and development boards, simplifying the prototyping process for hobbyists and professionals alike.

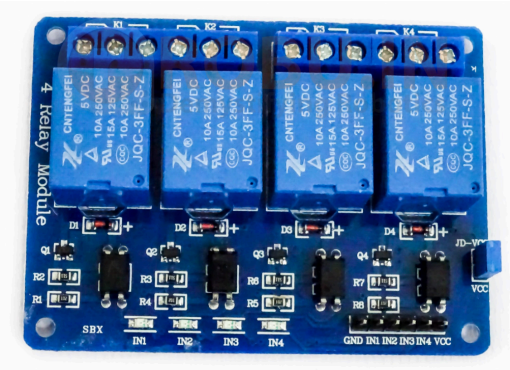
NodeMCU ESP8266



The ESP8266 NodeMCU CP2102 board has ESP8266 which is a highly integrated chip designed for the needs of a new connected world. It offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor.

> **ESP8266 has powerful on-board processing and storage capabilities that allow it to be integrated with the sensors and other application-specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. Its high degree of on-chip integration allows for minimal external circuitry, and the entire solution, including the front-end module, is designed to occupy minimal PCB area.**

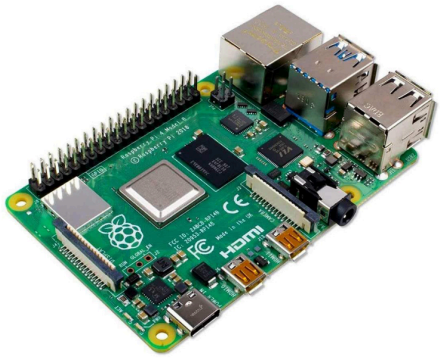
Four-Channel Relay Module (with Optocoupler)



It is a 4 Channel Isolated 5V 10A Relay Module Opto-coupler, A wide range of microcontrollers such as Arduino, AVR, PIC, ARM and so on can control it. It is also able to control various appliances and other types of equipment with a large current.

> **Relay output maximum contact is AC250V 10A and DC30V 10A. One can connect a microcontroller with standard interface directly to it. Red working status indicator lights are conducive to the safe use. It has a wide range of applications such as all MCU control, industrial sector, PLC control, smart home control.**
>
> ***Ready to get switching on your Raspberry Pi?! This neat relay module features 4 x 5V relays rated at 10A/250V each. It is designed to switch up to 4 high currents (10A) or high voltage (250V) loads with the help of microcontroller!***

Raspberry Pi 3 Model B+



This is Raspberry Pi 4 Model-B with 1 GB RAM. The Raspberry Pi 4 Model B is a single-board computer that was released by the Raspberry Pi Foundation in June 2019. It is an upgraded version of its predecessor, the Raspberry Pi 3 Model B+. The Raspberry Pi 4 Model B offers significant improvements in terms of performance, connectivity, and features.

> One of the key enhancements in the Raspberry Pi 4 Model B is its improved processing power. It is powered by a quad-core ARM Cortex-A72 CPU running at 1.5GHz, providing a substantial boost in performance compared to the previous models. This increased processing power enables the Raspberry Pi 4 to handle more demanding tasks and applications.
>
> In this project, we used a combination of **hardware components** (for physical control) and **software tools** (for communication and dashboard control). Below is a detailed list:

🔹 1. Hardware Components

Component	Purpose
VC02 Voice Recognition Module	Recognizes offline voice commands and sends signals to NodeMCU
NodeMCU ESP8266	Acts as the main microcontroller to process voice/manual commands and control relays
Four-Channel Relay Module (with Optocoupler)	Controls electrical appliances (ON/OFF switching)
Raspberry Pi 3 Model B+	Runs the MQTT broker (Mosquitto) and Node-RED for dashboard control
Wi-Fi Router	Provides network connectivity between Raspberry Pi and NodeMCU
Electrical Appliances (Fan, Light, etc.)	Devices that we control using relays
Jumper Wires	Used to connect components
Power Adapter for Raspberry Pi (5V, 2.5A)	Powers the Raspberry Pi
Micro USB Cable for NodeMCU	Used to power and program NodeMCU

🔹 2. Software & Protocols Used

Software/Protocol	Purpose
Arduino IDE	To write and upload code to NodeMCU
PubSubClient Library (MQTT)	Enables MQTT communication between NodeMCU and Raspberry Pi
Node-RED	Provides a graphical dashboard for manual relay control
Mosquitto MQTT Broker	Runs on Raspberry Pi to manage MQTT communication
ESP8266WiFi Library	Enables Wi-Fi communication for NodeMCU
GroveOfflineSensor Library	Used for processing voice commands from VC02 module

🔹 3. Communication and Control Mechanisms

Technology	Purpose
MQTT (Message Queuing Telemetry Transport)	Protocol for sending ON/OFF commands between Raspberry Pi and NodeMCU
Wi-Fi Communication	Connects Raspberry Pi, NodeMCU, and dashboard over a local network
GPIO (General Purpose Input/Output)	Used to control the relay module from NodeMCU
Optocoupler Circuit in Relay	Provides electrical isolation for safe switching of appliances

Pin Configuration

1 NodeMCU (ESP8266) Pin Connections

NodeMCU Pin	Connected To	Purpose
Vin (5V)	USB Power / 5V Adapter	Power Supply
GND	Common Ground	Ground Connection
D1 (GPIO5)	Relay Module IN1	Control Relay 1
D2 (GPIO4)	Relay Module IN2	Control Relay 2
D3 (GPIO0)	Relay Module IN3	Control Relay 3
D4 (GPIO2)	Relay Module IN4	Control Relay 4
D7 (GPIO13)	VC02 TX	Receive Data from VC02
D8 (GPIO15)	VC02 RX	Send Data to VC02
3.3V	VC02 VCC	Power Supply for VC02
GND	VC02 GND	Common Ground

Note: The relay module operates on active LOW logic, meaning:

- Sending LOW (0V) to relay turns it ON.
- Sending HIGH (5V) to relay turns it OFF.

2 Raspberry Pi 3 Model B+ Pin Connections

Raspberry Pi Pin	Connected To	Purpose
5V (Pin 2 or 4)	Relay Module VCC	Power Supply for Relays
GND (Pin 6, 9, or 14)	Relay Module GND	Common Ground
GPIO14 (TX, Pin 8)	MQTT to NodeMCU	Data Communication
GPIO15 (RX, Pin 10)	MQTT from NodeMCU	Data Communication

Important Notes:

- The 5V from Raspberry Pi is used to power the Relay Module.
- The NodeMCU communicates with Raspberry Pi via MQTT over Wi-Fi, so there are no direct GPIO connections between Pi and NodeMCU.
- Ensure the Pi and NodeMCU are on the same Wi-Fi network for MQTT communication.

3 Four-Channel Relay Module Pin Connections

Relay Module Pin	Connected To	Purpose
VCC	5V from Raspberry Pi	Power for Relay Module
GND	GND (NodeMCU & Raspberry Pi)	Common Ground
IN1	D1 (GPIO5) on NodeMCU	Control Relay 1
IN2	D2 (GPIO4) on NodeMCU	Control Relay 2
IN3	D3 (GPIO0) on NodeMCU	Control Relay 3
IN4	D4 (GPIO2) on NodeMCU	Control Relay 4

Important Notes:

- If the relay does not switch, check if it needs an external 5V power supply instead of the Raspberry Pi's 5V.
- Optocoupler isolation is built-in, so no extra circuit is needed.

4 VC02 Voice Recognition Module Pin Connections

VC02 Pin	Connected To	Purpose
VCC	3.3V from NodeMCU	Power for VC02
GND	GND	Common Ground
TX	D7 (GPIO13) on NodeMCU	Sends Voice Command Data
RX	D8 (GPIO15) on NodeMCU	Receives Response

How it works:

- The VC02 **recognizes voice commands offline**.
- It sends the recognized command to **NodeMCU via serial (D7, D8)**.
- The **NodeMCU processes the command** and turns the relay ON/OFF accordingly.

VC 02 TEST CODE

```
#include "GroveOfflineSensor.h"

#include <SoftwareSerial.h>

#define RX_VC02 D7

#define TX_VC02 D6

SoftwareSerial groveSerial(RX_VC02, TX_VC02); // RX, TX

void setup() {

  Serial.begin(115200);

  while (!Serial); // wait for serial port to connect. Needed for native USB port only , This port is for displaying data Grove Sensor sends

  groveSerial.begin(115200); // Make sure to set the baud rate to match your communication

}

void loop() {

   uint8_t *voiceData = detectVoiceFromGroveSensor(&groveSerial);

  if(voiceData != NULL){

    String response = getCommandInString(voiceData);

    Serial.println(response);

  }

  delay(1000);

}
```

What Does This Code Do?

This Arduino (NodeMCU) code listens for voice commands from the VC02 offline voice recognition module, processes them, and prints the recognized command to the serial monitor.

Code Breakdown

1. Include Required Libraries

cpp

```cpp
#include "GroveOfflineSensor.h"

#include <SoftwareSerial.h>
```

- `#include "GroveOfflineSensor.h"` → This library provides functions to communicate with the VC02 voice recognition module.
- `#include <SoftwareSerial.h>` → Allows the creation of a software-based serial port on any digital pins of an Arduino-compatible board (like NodeMCU).

2. Define VC02 Communication Pins

cpp

```cpp
#define RX_VC02 D7

#define TX_VC02 D6
```

- `D7` → RX (Receive) pin of NodeMCU, which will receive data from the TX pin of the VC02 module.

- `D6` → TX (Transmit) pin of NodeMCU, which will send data to the RX pin of the VC02 module.

Why do we need these?

The NodeMCU (ESP8266) has only one hardware UART (Serial) port (`TX0`, `RX0`). Since this is used for debugging via USB, we create a software serial port to communicate with VC02.

3. Create a Software Serial Port for VC02

cpp

```
SoftwareSerial groveSerial(RX_VC02, TX_VC02); // RX, TX
```

- Creates a software serial port named `groveSerial`.
- The RX and TX pins are assigned as per our earlier definitions (`D7` for RX, `D6` for TX).
- This will allow NodeMCU to communicate with the VC02 module without interfering with the USB serial port.

4. Initialize Serial Communication

cpp

```
void setup() {

  Serial.begin(115200);

  while (!Serial); // wait for serial port to connect. Needed for native USB port only
```

- `Serial.begin(115200);` → Initializes the hardware serial port for debugging at 115200 baud rate (speed of data transfer).
- `while (!Serial);` → Waits for the USB serial connection to be established (useful when connected to a PC for debugging).

5. Initialize Software Serial for VC02

cpp

```
 groveSerial.begin(115200); // Make sure to set the baud rate to match your communication

}
```

- `groveSerial.begin(115200);` → Initializes the software serial port for communicating with VC02 at 115200 baud rate.
- The baud rate must match the default baud rate of the VC02 module for proper communication.

At this stage:

- The NodeMCU's hardware serial port is ready for debugging.

- The software serial port is ready to communicate with the VC02 module.

6. Start the Main Loop

cpp

```cpp
void loop() {

  // Your main code here
```

- The `loop()` function runs continuously after `setup()`.
- Here, we will receive voice commands from the VC02 module and process them.

7. Detect Voice Commands

cpp

```cpp
uint8_t *voiceData = detectVoiceFromGroveSensor(&groveSerial);
```

- Calls the `detectVoiceFromGroveSensor()` function from the `GroveOfflineSensor` library.
- Passes the software serial object (`groveSerial`) to read incoming voice commands.
- Returns a pointer (`uint8_t *voiceData`) to the detected voice command data.

At this point:

- If the user speaks a command, `voiceData` will store it.
- If no command is detected, `voiceData` will be `NULL`.

8. Process the Received Voice Command

cpp

```cpp
 if (voiceData != NULL) {

    String response = getCommandInString(voiceData);

    Serial.println(response);

  }
```

- Checks if a voice command is detected (`voiceData != NULL`).

Converts the received byte array (`voiceData`) into a readable string using:

cpp

```
String response = getCommandInString(voiceData);
```

- Prints the voice command to the serial monitor (`Serial.println(response);`).

At this point:

- If the user says "Turn on the fan", `response` might store "Turn on the fan".
- If the user says "Turn off the light", `response` might store "Turn off the light".

9. Add a Small Delay

cpp

```
 delay(1000);

}
```

- Waits for 1 second (`1000ms`) before checking for the next voice command.
- Prevents CPU overuse and ensures smooth voice detection.

Example Serial Monitor Output

Scenario 1: User Says "Turn on the fan"

```
Turn on the fan
```

Scenario 2: User Says "Turn off the light"

```
Turn off the light
```

Relay control Code

```
#include "GroveOfflineSensor.h"

#include <SoftwareSerial.h>

#define RX_VC02 D7

#define TX_VC02 D8

#define RELAY1 D1  // GPIO5

#define RELAY2 D2  // GPIO4

#define RELAY3 D3  // GPIO0

#define RELAY4 D4  // GPIO2

SoftwareSerial groveSerial(RX_VC02, TX_VC02); // RX, TX

void setup() {

  Serial.begin(115200);

  pinMode(D1, OUTPUT);

  pinMode(D2, OUTPUT);

  pinMode(D3, OUTPUT);

  pinMode(D4, OUTPUT);

  digitalWrite(D1, HIGH);

  digitalWrite(D2, HIGH);

  digitalWrite(D3, HIGH);

  digitalWrite(D4, HIGH);
```

```
  while (!Serial); // wait for serial port to connect. Needed for native USB port only , This port is for displaying data Grove Sensor sends

  groveSerial.begin(115200); // Make sure to set the baud rate to match your communication

}



void loop() {

   uint8_t *voiceData = detectVoiceFromGroveSensor(&groveSerial);

  if(voiceData != NULL){

    String response = getCommandInString(voiceData);

    Serial.println(response);

    if(response == "ok, turn on the light")

    {

      digitalWrite(D1, LOW);

    }

    if(response == "ok, turn off the light")

    {

      digitalWrite(D1, HIGH);

    }



    if(response == "ok, warm light turn on")

    {
```

```
    digitalWrite(D2, LOW);

  }

  if(response == "ok, warm light turn off")

  {

    digitalWrite(D2, HIGH);

  }

  if(response == "ok, low fan")

  {

    digitalWrite(D3, LOW);

  }

  if(response == "ok, high fan")

  {

    digitalWrite(D3, HIGH);

  }

      if(response == "ok, start to fan")

  {

    digitalWrite(D4, LOW);

  }

  if(response == "ok, stop to fan")

  {

    digitalWrite(D4, HIGH);
```

```
    }

  }

  delay(1000);

}
```

Detailed Line-by-Line Explanation of Your Code

This Arduino (NodeMCU ESP8266) program controls four relays using the VC02 Offline Voice Recognition module. When the user speaks predefined commands, the corresponding relay turns ON or OFF.

Code Breakdown (Line by Line)

1. Header File

cpp

```
#include "GroveOfflineSensor.h"

#include <SoftwareSerial.h>
```

- `#include "GroveOfflineSensor.h"`
  - Includes the library for the Grove VC02 Offline Voice Recognition Sensor.
  - This library provides functions to detect voice commands and convert them into text.
- `#include <SoftwareSerial.h>`
  - Enables software-based serial communication.
  - The ESP8266 has only one hardware UART (`Serial`), so `SoftwareSerial` is used to communicate with the VC02 module.

2. Defining Pins for Voice Module (VC02)

cpp

```
#define RX_VC02 D7

#define TX_VC02 D8
```

- `D7` (GPIO13) is used as the Receive (RX) pin.
- `D8` (GPIO15) is used as the Transmit (TX) pin.
- These are the software serial pins for communicating with the VC02 module.

3. Defining Relay Pins

cpp

```cpp
#define RELAY1 D1  // GPIO5

#define RELAY2 D2  // GPIO4

#define RELAY3 D3  // GPIO0

#define RELAY4 D4  // GPIO2
```

- Each relay module is connected to a different GPIO pin on the NodeMCU.
- The NodeMCU sends HIGH (OFF) or LOW (ON) signals to these pins to control the relays.

4. Initializing Software Serial

cpp

```cpp
SoftwareSerial groveSerial(RX_VC02, TX_VC02);
```

- Creates a software serial object named `groveSerial`.
- This allows communication with the VC02 voice module using the specified TX/RX pins.

5. Setup Function (Runs Once at Startup)

cpp

```cpp
void setup() {

  Serial.begin(115200);
```

- Starts the hardware serial at `115200` baud rate for debugging and printing messages.

cpp

```cpp
  pinMode(D1, OUTPUT);
```

```
  pinMode(D2, OUTPUT);

  pinMode(D3, OUTPUT);

  pinMode(D4, OUTPUT);
```

- Sets all relay pins as OUTPUT, so they can control the relay module.

cpp

```
 digitalWrite(D1, HIGH);

  digitalWrite(D2, HIGH);

  digitalWrite(D3, HIGH);

  digitalWrite(D4, HIGH);
```

- Turns OFF all relays initially by setting them to HIGH.
- Most relay modules are active LOW, meaning:
  - `HIGH` (1) → Relay OFF
  - `LOW` (0) → Relay ON

cpp

```
 while (!Serial);
```

- Waits until the serial monitor is ready (used for debugging).
- This is not necessary for ESP8266, but it is helpful in boards like Arduino Due with a native USB port.

cpp

```
 groveSerial.begin(115200);
```

- Starts the software serial (`groveSerial`) at `115200` baud rate to communicate with the VC02 module.

6. Loop Function (Runs Continuously)

cpp

```
void loop() {
```

- This function runs in an infinite loop to continuously listen for voice commands and control the relays.

7. Detecting Voice Commands

cpp

```
  uint8_t *voiceData = detectVoiceFromGroveSensor(&groveSerial);
```

- Calls the function `detectVoiceFromGroveSensor()`, which listens to the VC02 module for a voice command.
- If a voice command is detected, it returns the command as raw data (`uint8_t*`). If no command is detected, it returns `NULL`.

8. Processing the Voice Command

cpp

```
 if(voiceData != NULL){

    String response = getCommandInString(voiceData);

    Serial.println(response);
```

- If a voice command is detected, it is converted into a human-readable string using `getCommandInString()`.
- The recognized command is printed on the serial monitor for debugging.

9. Controlling Relays Based on Commands

Each voice command is checked, and the corresponding relay is turned ON/OFF.

Relay 1 "Light"

cpp

```
if(response == "ok, turn on the light")
 {
   digitalWrite(D1, LOW);
 }
 if(response == "ok, turn off the light")
 {
   digitalWrite(D1, HIGH);
 }
```

- If the user says "ok, turn on the light", the relay turns ON (`LOW`).
- If the user says "ok, turn off the light", the relay turns OFF (`HIGH`).

Relay 2 - "Warm Light"

cpp

```
if(response == "ok, warm light turn on")
 {
   digitalWrite(D2, LOW);
 }


 if(response == "ok, warm light turn off")
 {
   digitalWrite(D2, HIGH);
```

```
}
```

- **Similar to Relay 1, but for controlling a warm light.**

Relay 3 - "Fan Speed Control"

cpp

```
if(response == "ok, low fan")
 {
   digitalWrite(D3, LOW);
 }
 if(response == "ok, high fan")
 {
   digitalWrite(D3, HIGH);
 }
```

- The third relay controls a fan.
- `"ok, low fan"` turns it ON (`LOW`).
- `"ok, high fan"` turns it OFF (`HIGH`).

Relay 4 - "Fan Start/Stop"

cpp

```
if(response == "ok, start to fan")
 {
   digitalWrite(D4, LOW);
 }
```

```
    if(response == "ok, stop to fan")

    {

      digitalWrite(D4, HIGH);

    }
```

- **The fourth relay starts/stops the fan.**

Loop Delay

cpp

```
 delay(1000);

}
```

- **Waits for 1 second before listening for the next command.**
- **This prevents rapid false triggers.**

Summary: What Does This Code Do?

Listens for voice commands from the VC02 module.
Identifies the spoken command as a string.
Controls relays based on voice commands:

- Relay 1 → Light ON/OFF
- Relay 2 → Warm Light ON/OFF
- Relay 3 → Fan LOW/HIGH
- Relay 4 → Fan START/STOP
  ✅ Prints the recognized command on the serial monitor.

Example: Expected Serial Monitor Output

Scenario 1: User Says "ok, turn on the light"

```
ok, turn on the light
```

➡ Relay 1 turns ON (Light turns ON)

Scenario 2: User Says "ok, stop to fan"

```
ok, stop to fan
```

➡ Relay 4 turns OFF (Fan stops)

Next Steps

Would you like me to modify this code to also send MQTT messages to a Raspberry Pi-based MQTT broker? That way, you can control the relays both via voice and a Node-RED dashboard!

Step-by-Step Procedure for Our Smart Home Automation Project

We will go through every step in detail, from setting up the hardware to writing the code and testing the project.

Set Up Raspberry Pi as an MQTT Broker

Install Mosquitto MQTT Broker

1. Connect Raspberry Pi to the internet.
2. Open the terminal and run:

bash

```
sudo apt update
```

```
sudo apt install mosquitto mosquitto-clients -y
```

3. Start the Mosquitto service:

bash

```
sudo systemctl enable mosquitto
```

```
sudo systemctl start mosquitto
```

Check if MQTT Broker is Running

Run this command to check the status:

bash

```
sudo systemctl status mosquitto
```

If running correctly, you will see `active (running)`.

Set Up Node-RED on Raspberry Pi

Install Node-RED

Run the following commands in the Raspberry Pi terminal:

bash

```
sudo apt update
```

```
sudo apt install -y node-red
```

Start Node-RED

Start Node-RED using:

bash

```
node-red
```

Now, open a browser and go to `http://<your-raspberry-ip>:1880` to access Node-RED.

Build Node-RED Dashboard for Manual Control

Add UI Nodes

1. Install the Node-RED dashboard:

bash

```
npm install node-red-dashboard
```

2. Restart Node-RED:

bash

```
node-red-stop
```

```
node-red-start
```

Create UI Buttons to Control Relays

1. Open Node-RED (`http://<your-raspberry-ip>:1880`).
2. Drag and drop:

    - Dashboard Button Nodes (for ON/OFF control).
    - MQTT Out Nodes (to publish relay commands).

3. Configure topics:

    - relay/1 → Relay 1
    - relay/2 → Relay 2
    - relay/3 → Relay 3
    - relay/4 → Relay 4

Now, when we press a button, Node-RED will publish an MQTT message (`ON` or `OFF`) to the topic.

Connect & Configure NodeMCU

Flash Firmware to NodeMCU

1. Install Arduino IDE on your PC.
2. Add ESP8266 Board Manager (`Tools > Board > ESP8266`).
3. Install PubSubClient Library for MQTT communication.

Write Code for NodeMCU

This code does the following:
✔ Connects to Wi-Fi
✔ Connects to MQTT Broker
✔ Subscribes to relay topics (`relay/1`, `relay/2`, etc.)
✔ Controls relays based on messages received

Complete NodeMCU Code

cpp

```cpp
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

const char* ssid = "Rajaseelan";

const char* password = "charanguru";

const char* mqtt_server = "192.168.0.50";  // Raspberry Pi's IP

WiFiClient espClient;
```

```
PubSubClient client(espClient);

// Relay pins

#define RELAY1 D1

#define RELAY2 D2

#define RELAY3 D3

#define RELAY4 D4

void setup() {

  pinMode(RELAY1, OUTPUT);

  pinMode(RELAY2, OUTPUT);

  pinMode(RELAY3, OUTPUT);

  pinMode(RELAY4, OUTPUT);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) delay(500);

  client.setServer(mqtt_server, 1883);

  client.setCallback(callback);

}



void loop() {

  if (!client.connected()) reconnect();

  client.loop();

}
```

```
void callback(char* topic, byte* payload, unsigned int length) {

  String message = "";

  for (int i = 0; i < length; i++) {

    message += (char)payload[i];

  }

  if (String(topic) == "relay/1") digitalWrite(RELAY1, message == "ON" ? LOW :
HIGH);

  if (String(topic) == "relay/2") digitalWrite(RELAY2, message == "ON" ? LOW :
HIGH);

  if (String(topic) == "relay/3") digitalWrite(RELAY3, message == "ON" ? LOW :
HIGH);

  if (String(topic) == "relay/4") digitalWrite(RELAY4, message == "ON" ? LOW :
HIGH);

}
```

Connect VC02 Voice Recognition Module

Configure VC02 to Recognize Commands

1. Train the VC02 module using predefined commands:
   - "Turn on the fan" → Sends `relay/1 ON`
   - "Turn off the fan" → Sends `relay/1 OFF`
   - "Turn on the light" → Sends `relay/2 ON`
2. The NodeMCU processes the voice command and publishes an MQTT message.

Final Testing

Test Manual Control via Node-RED

Open Node-RED Dashboard (`http://<your-raspberry-ip>:1880/ui`).
Press Relay 1 ON → Check if the relay switches ON.
Press Relay 1 OFF → Check if the relay switches OFF.

Test Voice Control

Say "Turn on the fan" → VC02 should send a message to NodeMCU, which will turn ON the relay.
Say "Turn off the fan" → The relay should switch OFF.

Automate MQTT Broker Startup

To make sure the broker starts automatically, enable Mosquitto service:

bash

```
sudo systemctl enable mosquitto
```

To make Node-RED start on boot, run:

bash

```
sudo systemctl enable nodered.service
```

How MQTT Worked in Our Project

Overview of MQTT in Our Project

MQTT (Message Queuing Telemetry Transport) is the core communication protocol in our Smart Home Automation project. It enables the Raspberry Pi (as the MQTT broker), NodeMCU (as the MQTT client), and Node-RED (as the dashboard control system) to communicate efficiently over Wi-Fi.

1. Raspberry Pi acts as the MQTT Broker (Mosquitto).
2. NodeMCU (ESP8266) acts as an MQTT Client (Subscriber & Publisher).
3. Node-RED (Running on Raspberry Pi) acts as another MQTT Client (Publisher & Subscriber).
4. The Four-Channel Relay is connected to NodeMCU to switch ON/OFF electrical appliances.

How MQTT Works in Our Project – Step-by-Step

1. Setting Up the MQTT Broker (Raspberry Pi)

I. What is the Broker?
The broker is a central server that manages all MQTT messages. It ensures reliable communication between NodeMCU and Node-RED.

We installed Mosquitto MQTT Broker on the Raspberry Pi to act as the central hub.

II. Why Raspberry Pi?

- Acts as a local MQTT server.
- Can run Node-RED to provide a dashboard interface.
- Handles communication between different devices efficiently.

2. NodeMCU Connects to Wi-Fi and MQTT Broker

How does NodeMCU connect to MQTT?

- First, NodeMCU connects to Wi-Fi (`Rajaseelan`).
- Then, it connects to the MQTT broker running on Raspberry Pi (`192.168.0.50`).
- It subscribes to specific relay control topics (`relay/1`, `relay/2`, etc.).

Code snippet to connect to MQTT broker:

cpp

```cpp
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
const char* ssid = "Rajaseelan";
const char* password = "charanguru";
const char* mqtt_server = "192.168.0.50";  // Raspberry Pi IP

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void setup() {
  setup_wifi();  // Connect to Wi-Fi
  client.setServer(mqtt_server, 1883);  // Connect to MQTT Broker
  client.setCallback(callback); // Function to process incoming MQTT messages
```

```
}
```

3. NodeMCU Subscribes to MQTT Topics

i. What are Topics?
Topics are message channels used in MQTT communication.

ii. Why Subscribe?
Since NodeMCU controls the relays, it must listen for messages sent from Node-RED.

Topics used in our project:

- `"relay/1"` → Controls Relay 1
- `"relay/2"` → Controls Relay 2
- `"relay/3"` → Controls Relay 3
- `"relay/4"` → Controls Relay 4

Code snippet for subscribing to topics:

cpp

```cpp
void reconnect() {
  while (!client.connected()) {
    if (client.connect("ESP8266Client")) {
      client.subscribe("relay/#");  // Subscribe to all relay topics
    } else {
      delay(5000);
    }
  }
}
```

4. Node-RED Sends MQTT Messages to NodeMCU

What is Node-RED?

- A graphical interface running on Raspberry Pi that allows manual control of relays.
- We created buttons in Node-RED Dashboard to turn relays ON/OFF.

How does Node-RED control the relays?

- When you press a button in Node-RED, it publishes an MQTT message.
- The message (`ON` or `OFF`) is sent to a specific relay topic (`relay/1`, `relay/2`, etc.).

Example Message from Node-RED:

```
Topic: relay/1

Payload: ON
```

Example Node-RED Flow (for Relay 1):

- Inject Node (to trigger messages).
- Switch Node (ON/OFF logic).
- MQTT Out Node (publishes the ON/OFF message to `relay/1`).

5. NodeMCU Receives the Message & Switches Relays

How does NodeMCU process MQTT messages?

- When it receives a message (`ON` or `OFF`) on a relay topic (`relay/1`), it toggles the relay accordingly.

Code snippet to process MQTT messages:

cpp

```cpp
void callback(char* topic, byte* payload, unsigned int length) {
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];  // Convert payload to string
  }

  if (String(topic) == "relay/1") controlRelay(RELAY1, message);
  if (String(topic) == "relay/2") controlRelay(RELAY2, message);
  if (String(topic) == "relay/3") controlRelay(RELAY3, message);
  if (String(topic) == "relay/4") controlRelay(RELAY4, message);
}
```

Function to Control Relay:

cpp

```cpp
void controlRelay(int relayPin, String command) {
```

```
  if (command == "ON") {
    digitalWrite(relayPin, LOW); // Relay ON
  } else if (command == "OFF") {
    digitalWrite(relayPin, HIGH); // Relay OFF
  }
}
```

6. Voice Commands from VC02 are Converted to MQTT Messages

What is the VC02 Module?
The VC02 is an offline voice recognition module that recognizes pre-defined voice commands.

How does VC02 control relays?

- When you say a command (`"ok, turn on the light"`), the VC02 module recognizes it.
- The command is processed in NodeMCU and converted into an MQTT message.
- NodeMCU publishes the message to the MQTT broker (Raspberry Pi).

Voice command processing code:

cpp

```
uint8_t *voiceData = detectVoiceFromGroveSensor(&groveSerial);
if (voiceData != NULL) {
    String response = getCommandInString(voiceData);
    if (response == "ok, turn on the light") controlRelay(RELAY1, "ON");
}
```

7. Relay Module Controls Electrical Appliances

What is a Four-Channel Relay Module?

- It acts as a switch to control high-voltage appliances.
- The optocoupler inside the relay isolates the low-voltage NodeMCU from high-voltage AC appliances.

How Relays are Switched?

- When the relay receives a LOW signal, it turns ON the connected device.
- When the relay receives a HIGH signal, it turns OFF the connected device.

✅ Relay Connections with NodeMCU:

Relay	NodeMCU Pin
Relay 1	D1 (GPIO5)
Relay 2	D2 (GPIO4)
Relay 3	D3 (GPIO0)
Relay 4	D4 (GPIO2)

Images