



Future Institute of Technology

A Project Report on  
**Home Automation System using MQTT  
Protocol**

**Course:**

Project (PROJ-681)

**Semester:** 6th

**Submitted by:**

Soumyajit Ghosh (34230921015)

Oishee Deb (34230920010)

Sushil Kumar (34230920003)

Guddoo Kumar (34230920001)

**Mentored By:**

Sanjoy Banerjee

# **CERTIFICATE OF APPROVAL**

This is to certify that the project entitled “Home Automation System by using MQTT protocol” being submitted

by

Soumyajit Ghosh (University Roll No-34230921015)

Oishee Deb (University Roll No-34230920010)

Sushil Kumar (University Roll No-34230920003)

Guddoo Kumar (University Roll No-34230920001)

of the Maulana Abul Kalam Azad University of Technology (formerly West Bengal University of Technology), is hereby approved as a creditable work for the degree of Bachelor of Technology.

They have carried out the project work and presented it in a manner to warrant its acceptance as a prerequisite for the degree for which it has been submitted. It is understood that, by this approval, the undersigned does not endorse or approve any statements made, opinions expressed or conclusions drawn therein, but approved the project for which it has been submitted.

---

Pradipta Kumar Banerjee

Head of the Department

(CSE - AIML & IOT)

---

Sanjoy Banerjee

Project Mentor

(CSE - AIML & IOT)

# ACKNOWLEDGEMENT

We would like to express our profound gratitude and thanks to our project mentor **Mr. Sanjay Banerjee**, Assistant Professor of Computer Science (AIML & IOT) department for their constant support and inspiration and guidance throughout the process which has led to the successful completion of the project.

---

Soumyajit Ghosh  
(34230921015)

---

Oishee Deb  
(34230920010)

---

Sushil Kumar  
(34230920003)

---

Guddoo Kumar  
(34230920001)

# **ABSTRACT**

In today's fast-paced world, technology has become an integral part of our lives, transforming the way we interact with our surroundings.

But we often use the traditional system that requires manual operation, where users have to physically interact with switches, knobs, or buttons to control devices, adjust settings, or monitor their status. This limits the ability to control devices remotely or automate tasks based on specific conditions or schedules. Traditional systems often rely on wired connections, which can limit their reach and flexibility.

In contrast to IoT ecosystems, it leverages wireless technologies such as Wi-Fi, Bluetooth, and cellular networks, providing broader connectivity options and enabling devices to communicate over long distances.

IoT ecosystems provide remote accessibility and control. Users can monitor and control devices remotely from anywhere at any time through smartphones, tablets, or web interfaces, bringing convenience, flexibility, and efficiency.

The objective of our home automation project is to empower homeowners with the ability to effortlessly control and monitor various aspects of their living space. From those days when manually adjusting lights, fans, thermostats, and security systems or worrying about leaving appliances running when you're not home. With our system, you can achieve a new level of comfort, convenience, and peace of mind.

## Table of Contents

<b>1.</b>	<b>Introduction.....</b>	<b>6</b>
<b>1.1.</b>	<b>INTERNET OF THINGS (IOT) .....</b>	<b>6</b>
<b>1.2.</b>	<b>MQTT Protocol.....</b>	<b>8</b>
<b>2.</b>	<b>Project Description .....</b>	<b>12</b>
<b>2.1</b>	<b>Components Required .....</b>	<b>12</b>
<b>2.1.1.</b>	<b>Node MCU ESP8266 .....</b>	<b>12</b>
<b>2.1.2.</b>	<b>Motor .....</b>	<b>16</b>
<b>2.1.3.</b>	<b>Motor-Driver .....</b>	<b>18</b>
<b>2.1.4.</b>	<b>Breadboard.....</b>	<b>20</b>
<b>2.1.5.</b>	<b>External Power Supply.....</b>	<b>21</b>
<b>2.1.6.</b>	<b>Connecting Jumper wires.....</b>	<b>23</b>
<b>2.1.7.</b>	<b>Switch .....</b>	<b>23</b>
<b>2.1.8.</b>	<b>Potentiometer.....</b>	<b>25</b>
<b>2.2</b>	<b>Software Requirement.....</b>	<b>26</b>
<b>2.2.1.</b>	<b>Arduino IDE.....</b>	<b>26</b>
<b>2.2.2.</b>	<b>EMQX Cloud .....</b>	<b>29</b>
<b>3.</b>	<b>Block Diagram.....</b>	<b>30</b>
<b>4.</b>	<b>Circuit Diagram .....</b>	<b>30</b>
<b>5.</b>	<b>Flow Chart .....</b>	<b>31</b>
<b>6.</b>	<b>Working Flow of the Project .....</b>	<b>31</b>
<b>7.</b>	<b>Working Procedure of the Code.....</b>	<b>32</b>
<b>7.1.</b>	<b>Set up the NodeMCU ESP8266:.....</b>	<b>32</b>
<b>7.2.</b>	<b>Install MQTT Library: .....</b>	<b>32</b>
<b>7.3.</b>	<b>Connect to Wi-Fi:.....</b>	<b>33</b>
<b>7.4.</b>	<b>Establish an MQTT Connection: .....</b>	<b>33</b>
<b>7.5.</b>	<b>Publish Messages: .....</b>	<b>34</b>
<b>7.6.</b>	<b>Subscribe to Topics: .....</b>	<b>37</b>
<b>7.7.</b>	<b>Handle Incoming Messages:.....</b>	<b>37</b>
<b>7.8.</b>	<b>Disconnect from the MQTT Broker: .....</b>	<b>38</b>
<b>8.</b>	<b>MQTT Public/Subscribe Client Dashboard.....</b>	<b>38</b>
<b>9.</b>	<b>Project Code .....</b>	<b>39</b>
<b>10.</b>	<b>Advantage.....</b>	<b>44</b>
<b>11.</b>	<b>Limitations .....</b>	<b>45</b>
<b>12.</b>	<b>Future Scope.....</b>	<b>46</b>
<b>13.</b>	<b>Conclusion.....</b>	<b>47</b>
<b>14.</b>	<b>Bibliography .....</b>	<b>47</b>

# 1. Introduction

Welcome to the world of home automation, where the power to control and manage your living space is at your fingertips. Imagine a home where lights, appliances like fans, security systems, and even climate control are seamlessly integrated, allowing you to create a personalized, convenient, and energy-efficient environment. This introduction presents a captivating overview of a home automation project, highlighting its purpose, benefits, and potential impact on daily living.

Home automation takes this concept to the next level by harnessing the power of cutting-edge technologies such as Internet of Things (IoT), artificial intelligence, and smart devices to create intelligent homes that cater to our needs and preferences.

Imagine arriving home after a long day, and as you approach your front door, the lights automatically illuminate your path. You step inside to find the temperature set to your preferred level, creating a welcoming atmosphere. With a simple voice command or tap on your smartphone, you can adjust the lighting, lower the blinds, or even start your favorite playlist.

## 1.1. INTERNET OF THINGS (IOT)

The Internet of things (IoT) describes physical objects (or groups of such objects) with sensors, processing ability, software and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks. Internet of things has been considered a misnomer because devices do not need to be connected to the public internet, they only need to be connected to a network, and be individually addressable.

The field has evolved due to the convergence of multiple technologies, including ubiquitous computing, commodity sensors, increasingly powerful embedded systems, as well as machine learning. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), independently and collectively enable the Internet of things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", including devices and appliances (such as lighting fixtures, thermostats, home security systems, cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers. IoT is also used in healthcare systems.

There are a number of concerns about the risks in the growth of IoT technologies and products, especially in the areas of privacy and security, and consequently, industry and governmental moves to address these concerns have begun, including the development of international and local standards, guidelines, and regulatory frameworks.



### **1.1.1. Some Applications of IoT**

Let's look at some applications of IoT systems:

- **Connected Cars**

There are many ways vehicles, such as cars, can be connected to the internet. It can be through smart dashcams, infotainment systems, or even the vehicle's connected gateway. They collect data from the accelerator, brakes, speedometer, odometer, wheels, and fuel tanks to monitor both driver performance and vehicle health. Connected cars have a range of uses:

- Monitoring rental car fleets to increase fuel efficiency and reduce costs.
- Helping parents track the driving behavior of their children.
- Notifying friends and family automatically in case of a car crash.
- Predicting and preventing vehicle maintenance needs.

- **Home Automation**

Smart home devices are mainly focused on improving the efficiency and safety of the house, as well as improving home networking. Devices like smart outlets monitor electricity usage and smart thermostats provide better temperature control. Hydroponic systems can use IoT sensors to manage the garden while IoT smoke detectors can detect tobacco smoke. Home security systems like door locks, security cameras, and water leak detectors can detect and prevent threats, and send alerts to homeowners.

Connected devices for the home can be used for:

- Automatically turning off devices not being used.
- Rental property management and maintenance.
- Finding misplaced items like keys or wallets.
- Automating daily tasks like vacuuming, making coffee, etc.

- **Smart Cities**

IoT applications have made urban planning and infrastructure maintenance more efficient. Governments are using IoT applications to tackle problems in infrastructure, health, and the environment. IoT applications can be used for:

- Measuring air quality and radiation levels.
- Reducing energy bills with smart lighting systems.
- Detecting maintenance needs for critical infrastructures such as streets, bridges, and pipelines.
- Increasing profits through efficient parking management.

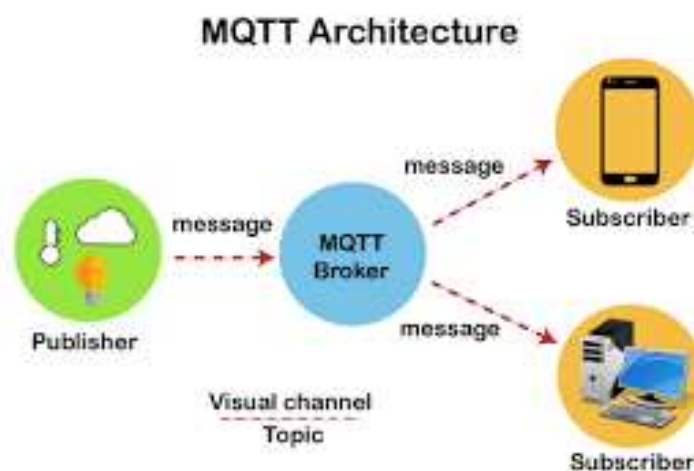
## **1.2. MQTT Protocol**

MQTT (Message Queuing Telemetry Transport) is a lightweight open messaging protocol that was developed for constrained environments such as M2M (Machine to Machine) and IoT (Internet of Things), where a small code footprint is required. MQTT is based on the Pub/Sub messaging principle of publishing messages and subscribing to topics. The protocol efficiently packs messages to minimize overhead. The MQTT specification recommends TLS as a transport option to secure the protocol using port 8883 (secure-mqtt), as the MQTT protocol



does not provide security on its own. Constrained devices can benefit from using TLS session resumption to reduce the reconnection cost.

The MQTT broker is the center of every Publish / Subscribe protocol. Depending on the implementation, a broker can manage up to thousands of simultaneously connected MQTT clients. The broker is responsible for receiving all messages, filtering the messages, determining who subscribed to each message and sending the message to those subscribed clients. The Broker also holds the sessions of all persistent clients, including subscriptions and missed messages. Another task of the Broker is the authentication and authorization of clients. Usually the broker is extensible, which facilitates custom authentication, authorization and integration with backend systems. Integration is especially important, because the Broker is often the component directly exposed on the Internet, serves many clients and has to forward messages to downstream analysis and processing systems. In short, the Broker is the central hub through which every message must be routed. It is therefore important that your broker is highly scalable, can be integrated into back-end systems, is easy to monitor and, of course, is fail-safe.



### **1.2.1. MQTT Components**

MQTT implements the publish/subscribe model by defining clients and brokers as below.

- **MQTT Client**

An MQTT client is any device from a server to a microcontroller that runs an MQTT library. If the client is sending messages, it acts as a publisher, and if it is receiving messages, it acts as a receiver. Basically, any device that

communicates using MQTT over a network can be called an MQTT client device.

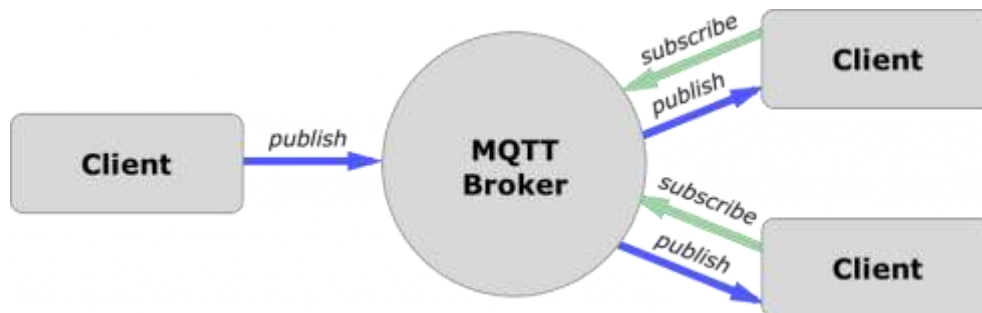
#### ▪ MQTT broker

The MQTT broker is the backend system which coordinates messages between the different clients. Responsibilities of the broker include receiving and filtering messages, identifying clients subscribed to each message, and sending them the messages. It is also responsible for other tasks such as:

- Authorizing and authenticating MQTT clients
- Passing messages to other systems for further analysis
- Handling missed messages and client sessions

#### ▪ MQTT connection

Clients and brokers begin communicating by using an MQTT connection. Clients initiate the connection by sending a CONNECT message to the MQTT broker. The broker confirms that a connection has been established by responding with a CONNACK message. Both the MQTT client and the broker require a TCP/IP stack to communicate. Clients never connect with each other, only with the broker.



### 1.2.2. The Principle behind MQTT

The MQTT protocol works on the principles of the publish/subscribe model. In traditional network communication, clients and servers communicate with each other directly. The clients request resources or data from the server, then the server processes and sends back a response. However, MQTT uses a publish/subscribe pattern to decouple the message sender (publisher) from the message receiver (subscriber). Instead, a third component, called a message broker, handles the communication between publishers and subscribers. The broker's job is to filter all incoming messages from publishers and distribute them correctly to subscribers.

The broker decouples the publishers and subscribers as below:

- **Space decoupling**

The publisher and subscriber are not aware of each other's network location and do not exchange information such as IP addresses or port numbers.

- **Time decoupling**

The publisher and subscriber don't run or have network connectivity at the same time.

- **Synchronization decoupling**

Both publishers and subscribers can send or receive messages without interrupting each other. For example, the subscriber does not have to wait for the publisher to send a message.

### **1.2.3. Importance MQTT protocol**

The MQTT protocol has become a standard for IoT data transmission because it delivers the following benefits:

- **Lightweight and efficient**

MQTT implementation on the IoT device requires minimal resources, so it can even be used on small microcontrollers. For example, a minimal MQTT control message can be as little as two data bytes. MQTT message headers are also small so that you can optimize network bandwidth.

- **Scalable**

MQTT implementation requires a minimal amount of code that consumes very little power in operations. The protocol also has built-in features to support communication with a large number of IoT devices. Hence, you can implement the MQTT protocol to connect with millions of these devices.

- **Reliable**

Many IoT devices connect over unreliable cellular networks with low bandwidth and high latency. MQTT has built-in features that reduce the time the IoT device takes to reconnect with the cloud. It also defines three different

quality-of-service levels to ensure reliability for IoT use cases— at most once (0), at least once (1), and exactly once (2).

- **Secure**

MQTT makes it easy for developers to encrypt messages and authenticate devices and users using modern authentication protocols, such as OAuth, TLS1.3, Customer Managed Certificates, and more.

- **Well-supported**

Several languages like Python have extensive support for MQTT protocol implementation. Hence, developers can quickly implement it with minimal coding in any type of application.

## 2. Project Description

To implement home automation using MQTT (Message Queuing Telemetry Transport), we will need the following hardware and software components:

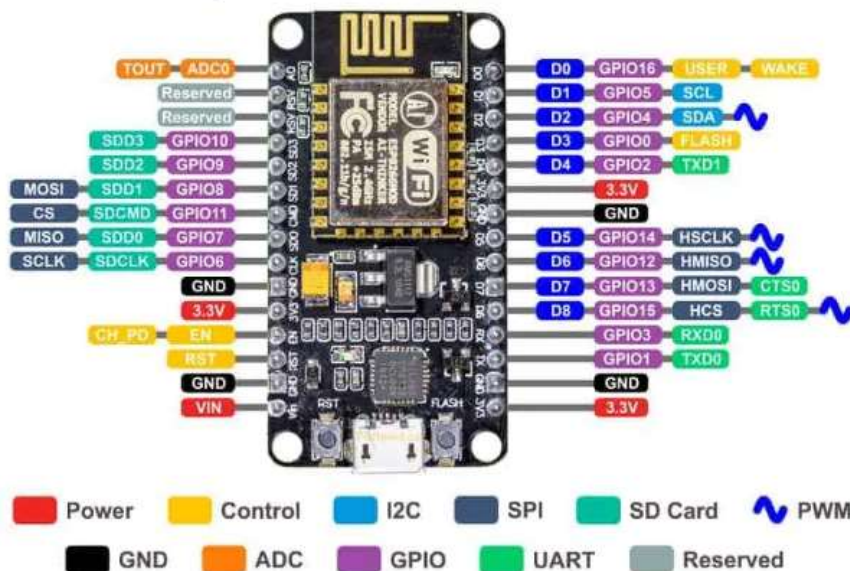
### 2.1 Components Required

#### 2.1.1. Node MCU ESP8266

Node MCU is an open source firmware for which open source prototyping board designs are available. The name "Node MCU" combines "node" and "MCU" (micro-controller unit). Strictly speaking, the term "Node MCU" refers to the firmware rather than the associated development kits. Both the firmware and prototyping board designs are open source.

The firmware uses the Lua scripting language. The firmware is based on the Lua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented.

The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications.



### ○ NodeMCU ESP8266 Board Guide

The NodeMCU ESP8266 supports two types of I/O nodes with each GPIO Pin: Digital & Analog. The NodeMCU ESP8266 has 16 GPIO pins and one analog input pin shown in the image bellow. However only 10 of these GPIO pins can be used for digital input and output operations.

### ○ Advantages over other microcontrollers

The NodeMCU ESP8266, an open-source IoT development board, offers several advantages over other microcontrollers. Here are some of the key advantages of the NodeMCU ESP8266:

- **Integrated Wi-Fi Connectivity:**

One of the significant advantages of the NodeMCU ESP8266 is its built-in Wi-Fi module. This allows the board to connect to Wi-Fi networks

and communicate with other devices and services over the internet. The integrated Wi-Fi capability makes it an ideal choice for IoT applications that require wireless connectivity.

- **Low Cost:**

The NodeMCU ESP8266 is known for its affordability. Compared to other microcontrollers with similar capabilities, the NodeMCU ESP8266 is relatively inexpensive. This makes it accessible to a wide range of developers, hobbyists, and students who are looking to prototype and develop IoT projects without breaking the bank.

- **High Computing Power:**

Despite its low cost, the NodeMCU ESP8266 offers impressive computing power. It is equipped with a powerful 32-bit Tensilica processor, providing sufficient processing capabilities for most IoT applications. The board also has ample memory, including both RAM and Flash memory, enabling the execution of complex tasks and storage of program code and data.

- **Large Community and Support:**

The NodeMCU ESP8266 has gained significant popularity among the maker and IoT communities. As a result, it has a large and active community of developers and enthusiasts who share their knowledge, projects, and troubleshooting tips. The availability of extensive online resources, tutorials, and forums makes it easier for users to get started and find assistance when working with the NodeMCU ESP8266.

- **Arduino-IDE-Compatible:**

The NodeMCU ESP8266 can be programmed using the Arduino IDE, which is a widely-used development environment with a rich ecosystem of libraries and examples. This compatibility allows developers already familiar with Arduino programming to leverage their existing knowledge and codebase when working with the NodeMCU ESP8266, making the transition to this platform relatively seamless.

## ○ **Limitations of NodeMCU ESP8266**

While the NodeMCU ESP8266 offers numerous advantages, it also has a few disadvantages to consider:

- **Limited GPIO Pins:**

The NodeMCU ESP8266 has a limited number of GPIO pins available for interfacing with other devices. Typically, it offers around 10 to 11 GPIO pins, which may be insufficient for complex projects that require numerous connections to sensors, actuators, or other peripherals.

- **Lack of Analog Inputs:**

The NodeMCU ESP8266 has only one analog input pin (A0), limiting the number of analog sensors it can directly interface with. If your project requires multiple analog inputs, you may need additional analog-to-digital converters (ADCs) or alternative hardware solutions.

- **Power Consumption:**

The NodeMCU ESP8266 can consume relatively high power, especially during Wi-Fi transmissions. This can be a concern for battery-powered or low-power applications, as it may reduce the overall battery life or require additional power management circuitry.

- **Wi-Fi Interference:**

Since the NodeMCU ESP8266 operates on the 2.4GHz frequency, it can be susceptible to Wi-Fi interference from nearby devices, such as routers and other wireless networks. This interference may affect the stability and reliability of Wi-Fi connections and communication.

### 2.1.2. Motor

A motor is an electro-mechanical device that converts electrical energy into mechanical motion. It consists of various components, including a rotor, a stator, and a power supply, which work together to generate rotational or linear movement.



#### ▪ **Working Principle:**

Motors operate based on the principles of electromagnetism. When an electric current is passed through the motor's windings, a magnetic field is created, which interacts with the magnetic field of the permanent magnets or other windings. This interaction generates a force that causes the rotor to rotate or move.

#### ▪ **Types of Motors:**

The types of Electric motors are available in three main segments like AC motor, DC motor, & special purpose motors

- **DC Motors:**

DC motors, or direct current motors, are powered by a direct current source. They feature a split-ring commutator or electronic commutation system to change the direction of the current flow in the motor's windings, producing continuous rotational motion.

- **AC Motors:**

AC motors, or alternating current motors, are powered by an alternating current source. They include induction motors, synchronous motors, and various



specialized types. AC motors are commonly used in household appliances, industrial machinery, and power systems.

- **Brushed Motors:**

Brushed motors use brushes and a commutator to create the magnetic field that drives the rotor's motion. These motors are simpler and less expensive but tend to have lower efficiency and require periodic maintenance.

- **Brushless Motors:**

Brushless motors use electronic commutation instead of brushes and a commutator. They offer higher efficiency, longer lifespan, and smoother operation compared to brushed motors. They are widely used in applications such as drones, electric vehicles, and computer cooling systems.

- **Stepper Motors:**

Stepper motors are specialized motors that move in discrete steps rather than continuous rotation. They are widely used in robotics, 3D printers, CNC machines, and other applications that require precise positioning and control.

- **Servo Motors:**

Servo motors are compact motors equipped with built-in feedback mechanisms. They are widely used in control systems that require accurate position control, such as robotic arms, RC models, and industrial automation.



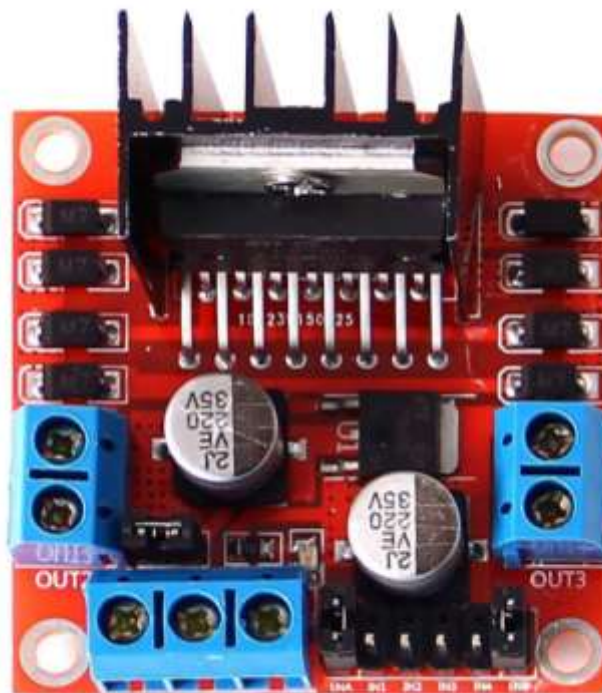
We are using this brushless cooling fan.

### 2.1.3. Motor-Driver

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. Let's take a closer look at the pin out of L298N module and explain how it works

#### Features & Specifications

- **Driver Model:** L298N 2A
- **Driver Chip:** Double H Bridge L298N
- **Motor Supply Voltage (Maximum):** 46V
- **Motor Supply Current (Maximum):** 2A
- **Logic Voltage:** 5V
- **Driver Voltage:** 5-35V
- **Driver Current:** 2A
- **Logical Current:** 0-36mA
- **Maximum Power (W):** 25W
- **Heatsink** for better performance
- **Power-On LED** indicator



- **Advantages of L298N motor driver over other motor drivers**

The L298N motor driver module offers several advantages over other motor drivers. Here are some of the key advantages of the L298N motor driver:

- **Dual H-Bridge Configuration:**

The L298N motor driver is designed with a dual H-bridge configuration, which allows it to control the speed and direction of two DC motors independently. This feature makes it suitable for applications that require precise control and coordination of multiple motors, such as robotic platforms, CNC machines, and motorized vehicles.

- **High Current Handling Capability:**

The L298N motor driver can handle relatively high current levels, making it suitable for driving motors with higher power requirements. It can handle a continuous current of up to 2A per channel and can handle peak currents of up to 3A per channel. This capability enables the driver to power a wide range of motors, including medium-sized and high-torque motors.

- **Voltage Compatibility:**

The L298N motor driver can operate with a wide range of input voltages, typically between 7V and 35V. This flexibility allows it to be used with various power sources, such as batteries, power supplies, or other voltage-regulated sources. The wide voltage compatibility makes the L298N motor driver versatile and adaptable to different project requirements.

- **Built-in Protection Mechanisms:**

The L298N motor driver incorporates built-in protection mechanisms to safeguard the driver and connected components from potential damage. It includes built-in diodes that protect against reverse voltage and voltage spikes generated during motor operation. These protection features enhance the reliability and durability of the motor driver and help prevent damage due to electrical transients.

- **Easy to Use and Interface:**

The L298N motor driver module is relatively straightforward to use and interface with a microcontroller or other control systems. It has clearly labelled input and output pins, making it easy to connect and control the driver. Additionally, it offers flexible control options, including both direct control and pulse-width modulation (PWM) control for precise speed regulation.

- **Widely Available and Affordable:**

The L298N motor driver module is widely available from various suppliers and is often offered at an affordable price point. Its popularity and availability make it easily accessible for hobbyists, makers, and developers, who can integrate it into their projects without significant cost constraints.

#### **2.1.4. Breadboard**

A breadboard, also known as a prototyping board or solderless breadboard, is a versatile tool used by electronics enthusiasts, hobbyists, and engineers for building and testing electronic circuits. It provides a convenient platform for quickly and temporarily connecting electronic components without the need for soldering.

**Here are some key features and benefits of using a breadboard:**

- **Contact Points:**

A breadboard consists of a plastic board with a grid of interconnected metal clips or sockets. These metal clips or sockets provide contact points for inserting and connecting electronic components such as resistors, capacitors, integrated circuits (ICs), and wires. The grid is usually organized into rows and columns, allowing for easy circuit layouts and connections.

- **Solderless Connections:**

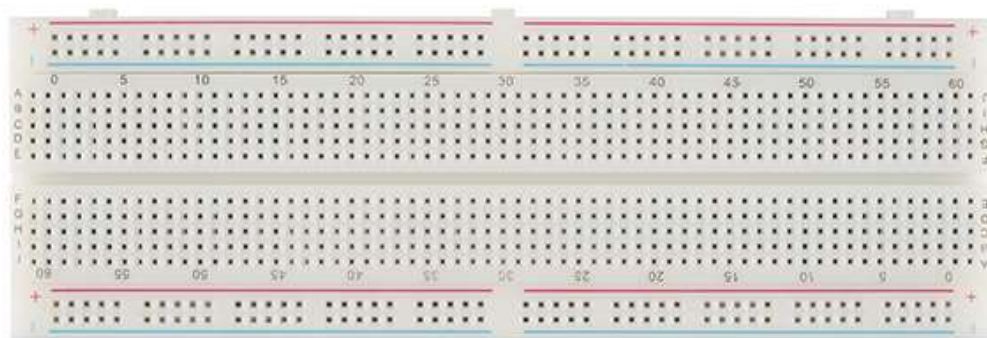
The main advantage of a breadboard is that it allows for solderless connections. Components can be inserted into the contact points and held securely by metal clips or sockets. This enables rapid prototyping, experimentation, and modifications without the need for soldering or permanent connections, making it ideal for iterative circuit design and testing.

- **Reusability:**

Since no soldering is involved, components can be easily removed and reused on a breadboard. This makes breadboards a flexible and cost-effective solution for testing and reconfiguring circuits, as components can be quickly swapped or replaced as needed.

- **Easy Circuit Design:**

Breadboards provide a straightforward platform for circuit design. The interconnected rows and columns allow components to be easily arranged and connected in a logical manner. The breadboard's layout often follows a standard pattern, with power supply rails on the sides and a central channel for connecting ICs. This organized layout simplifies the process of creating and troubleshooting circuits.



### **2.1.5. External Power Supply**

An external power supply, also known as an AC adapter, is a device used to provide electrical power to electronic devices or systems. It is typically connected to the device through a cable and plugs into a power outlet or another power source. The external power supply converts the input voltage from the power source to the specific voltage and current required by the device it is powering.

**Here are some key features and benefits of external power supplies:**

- **Voltage Conversion:**

External power supplies are designed to convert the input voltage from the power source to the appropriate voltage required by the device being powered. This allows devices with different voltage requirements to be powered from a standard power outlet or other power sources.

- **Current Regulation:**

External power supplies also regulate the current output to ensure that the device receives a stable and consistent power supply. This helps protect the device from damage due to overvoltage or fluctuations in the power source.

- **Portable and Compact:**

External power supplies are often compact and lightweight, making them portable and easy to carry. This is especially beneficial for powering portable electronic devices such as laptops, mobile phones, and tablets, allowing users to use their devices while on the go.

- **Versatility:**

External power supplies are available in a variety of output voltages and current ratings, allowing them to be used with a wide range of devices. They can power everything from small electronic devices like routers and modems to larger appliances like televisions and gaming consoles.

- **Safety Features:**

External power supplies are built with safety features to protect the device and the user. These may include overload protection, short circuit protection, and overvoltage protection, which help prevent damage to the device and reduce the risk of electrical hazards.



### **2.1.6. Connecting Jumper wires**

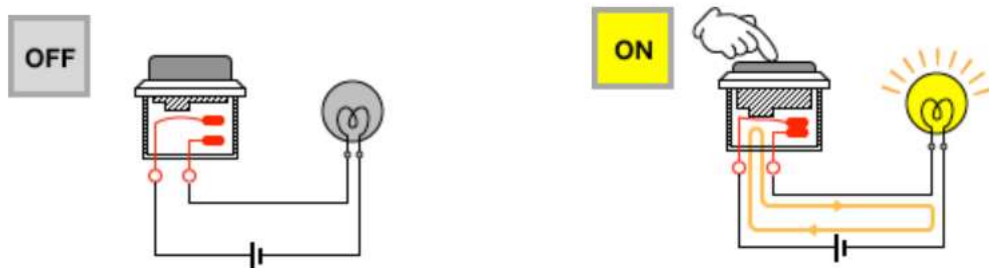
A jump wire (also known as jumper, jumper wire, DuPont wire) is an electrical wire, or group of them in a cable, with a connector or pin at each end (or sometimes without them – simply "tinned"), which is normally used to interconnect the components of a breadboard or other prototype or test circuit, internally or with other equipment or components, without soldering.

Individual jump wires are fitted by inserting their "end connectors" into the slots provided in a breadboard, the header connector of a circuit board, or a piece of test equipment.



### **2.1.7. Switch**

A switch is a simple yet essential electrical component used to control the flow of electricity in a circuit. It is designed to open or close a pathway within the circuit, allowing or interrupting the current flow. Switches are commonly used to turn on or off lights, appliances, electronic devices, and various other electrical systems.



## **Here are some key features of switches:**

- **On/Off Functionality:**

The primary purpose of a switch is to toggle the circuit's state between two positions: on and off. When the switch is in the "on" position, it allows the flow of current through the circuit, activating the connected device or system. When switched to the "off" position, it breaks the circuit, cutting off the current flow and turning off the device.

- **Manual Operation:**

Switches are typically manually operated, meaning they require physical interaction by pressing, toggling, or rotating the switch mechanism. However, some switches can also be remotely operated or activated through electronic control systems.

## **Here are some types of switches:**

- **Single Pole, Single Throw (SPST) Switch:**

An SPST switch is the simplest and most common type of switch. It has two terminals and provides a single pathway for the current when switched on. It is commonly used in basic on/off applications.

- **Single Pole, Double Throw (SPDT) Switch:**

An SPDT switch has three terminals and can connect one common terminal to either of the other two terminals. This switch type is commonly used for selecting between two different circuit paths or devices.

- **Push Button Switch:**

A push button switch is a momentary switch that requires continuous pressure to maintain the circuit connection. It is often used for applications where temporary activation or signalling is needed, such as doorbells or push-to-start buttons.



- **Toggle Switch:**

A toggle switch is a lever-operated switch that maintains its position until manually switched again. It is commonly used in applications that require a persistent on/off state, such as lighting controls or power switches.

### **2.1.8. Potentiometer**

A potentiometer, often referred to as a pot, is a three-terminal variable resistor that allows for precise control of electrical resistance. It consists of a resistive element and a movable contact, which can be adjusted to change the resistance value.



**Here are some key features and uses of potentiometers:**

- **Resistance Control:**

The primary function of a potentiometer is to control resistance. By rotating the shaft or slider of the potentiometer, the position of the movable contact along the resistive element changes. This movement alters the effective length of the resistive path, resulting in a variable resistance value between the fixed terminal and the movable contact.

- **Voltage Divider:**

Potentiometers can be used as voltage dividers. By connecting the outer terminals of the potentiometer across a voltage source and the wiper terminal to a load or circuit, the output voltage across the load can be varied by adjusting the position of the wiper. This feature is commonly used for volume control,

brightness adjustment, and other applications where variable voltage control is needed.

- **Analog Input:**

Potentiometers can be used as analog input devices in electronic systems. By connecting the wiper terminal to an analog-to-digital converter (ADC) input of a microcontroller or other digital device, the position of the potentiometer can be read as a voltage value. This allows for user-controlled input, such as in audio mixing consoles, joysticks, or other devices where continuous adjustment is required.

## 2.2 Software Requirement

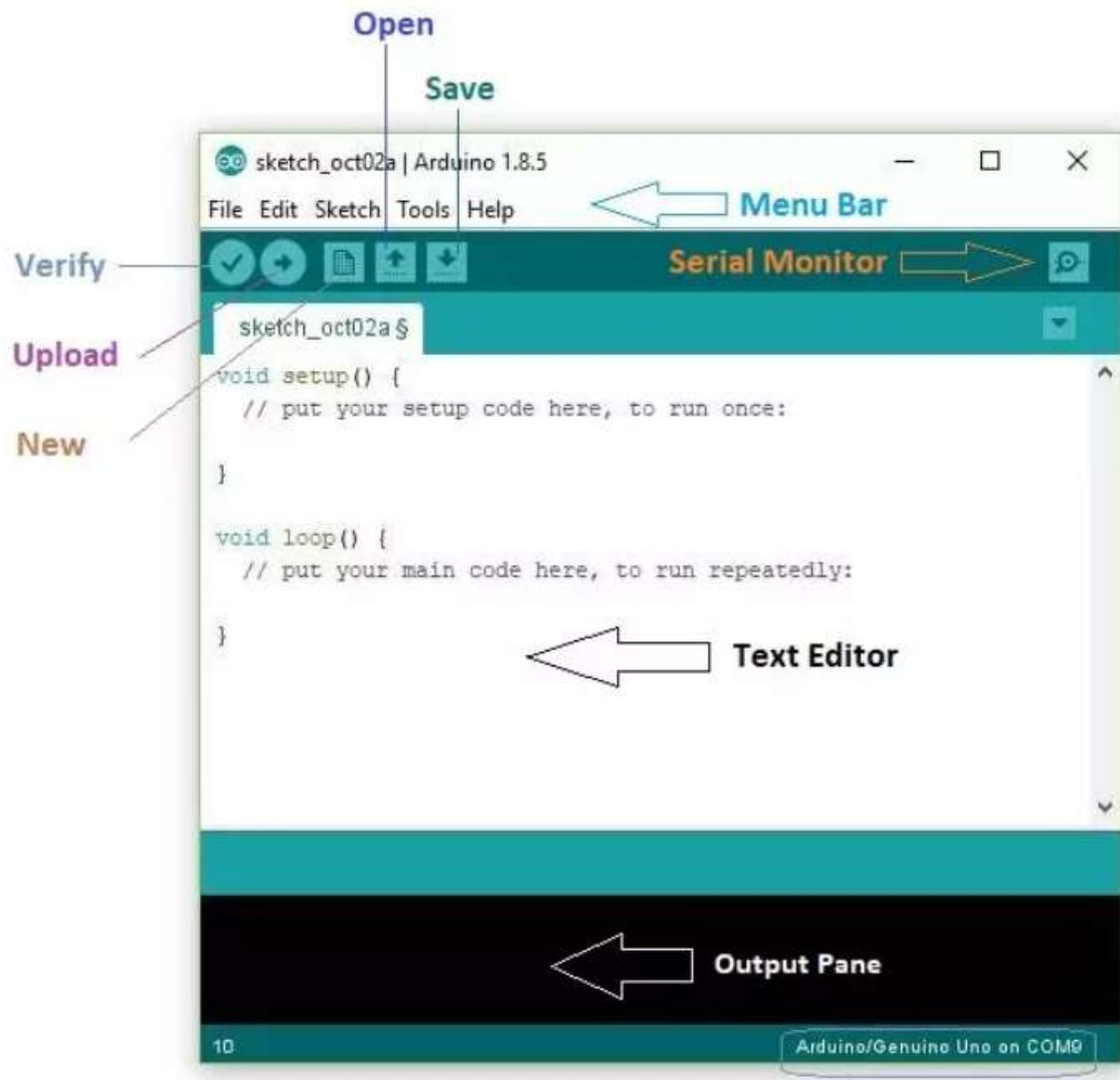
In order to build the project we need some softwares, that will help to complete the project. Those softwares are given bellow:

### 2.2.1. Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

- **Writing Sketches**

Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.



The Arduino Integrated Development Environment (IDE) has many advantages for programming and developing projects using Arduino microcontrollers. Here are some key advantages of using the Arduino IDE:

- **Simplicity and Ease of Use:**

The Arduino IDE is designed to be user-friendly, making it accessible for beginners and experienced developers alike. It provides a simple and intuitive interface that simplifies the process of writing, compiling, and uploading code to Arduino boards. The IDE's straightforward structure and extensive documentation make it easy to get started with Arduino programming.

- **Cross-Platform Compatibility:**

The Arduino IDE is available for multiple operating systems, including Windows, macOS, and Linux, ensuring broad compatibility across different platforms. This allows users to develop Arduino projects on their preferred operating system without any limitations.

- **Library Support:**

The Arduino IDE comes with a vast collection of pre-written libraries that provide ready-to-use functions and code snippets for various hardware modules and sensors. These libraries simplify the development process by offering high-level functions for common tasks, such as controlling servos, reading sensor data, and communicating with external devices. The extensive library support saves time and effort by providing reusable code solutions.

- **Integrated Tools:**

The Arduino IDE integrates various tools to streamline the development process. It includes a code editor with syntax highlighting, auto-completion, and error-checking features that help identify and rectify coding mistakes. The IDE also includes a serial monitor for debugging and communication with Arduino boards, making it easy to send and receive data for testing and troubleshooting purposes.

- **Large Community and Support:**

Arduino has a thriving community of developers, enthusiasts, and experts who actively contribute to forums, online resources, and projects. This vibrant community provides ample support and resources for users of the Arduino IDE. Whether you're a beginner seeking guidance or an experienced developer looking for advanced techniques, the Arduino community offers a wealth of knowledge, tutorials, and example projects.

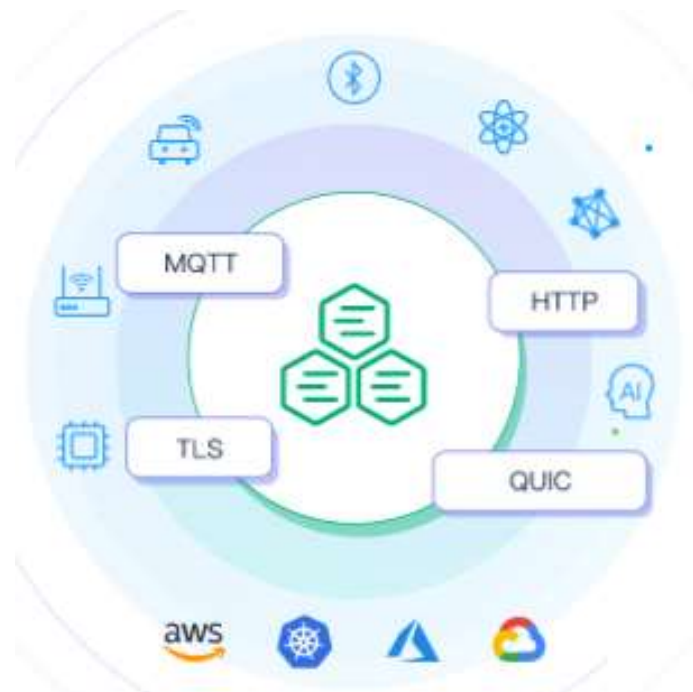
- **Open-Source Platform:**

Arduino IDE is an open-source platform, which means that the source code is freely available for modification and customization. This allows developers to extend the functionality of the IDE, create custom libraries, and contribute to

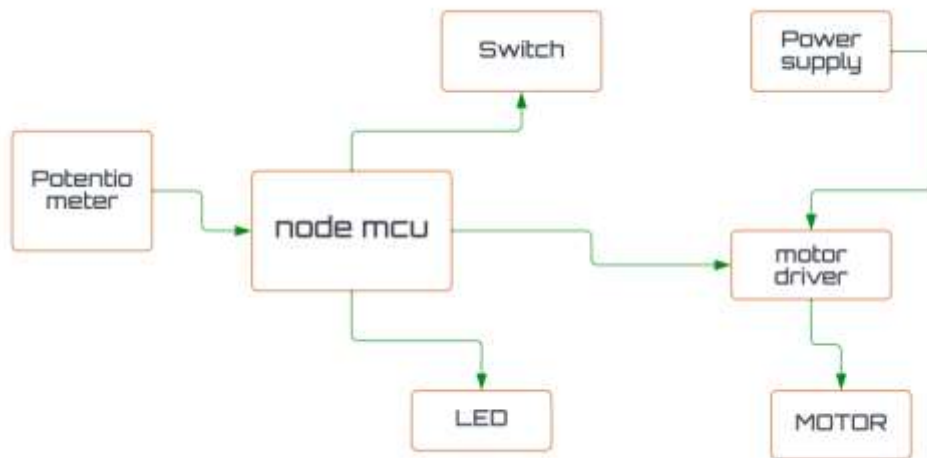
the ongoing development of the Arduino ecosystem. The open-source nature of Arduino promotes collaboration and innovation within the community.

### **2.2.2. EMQX Cloud**

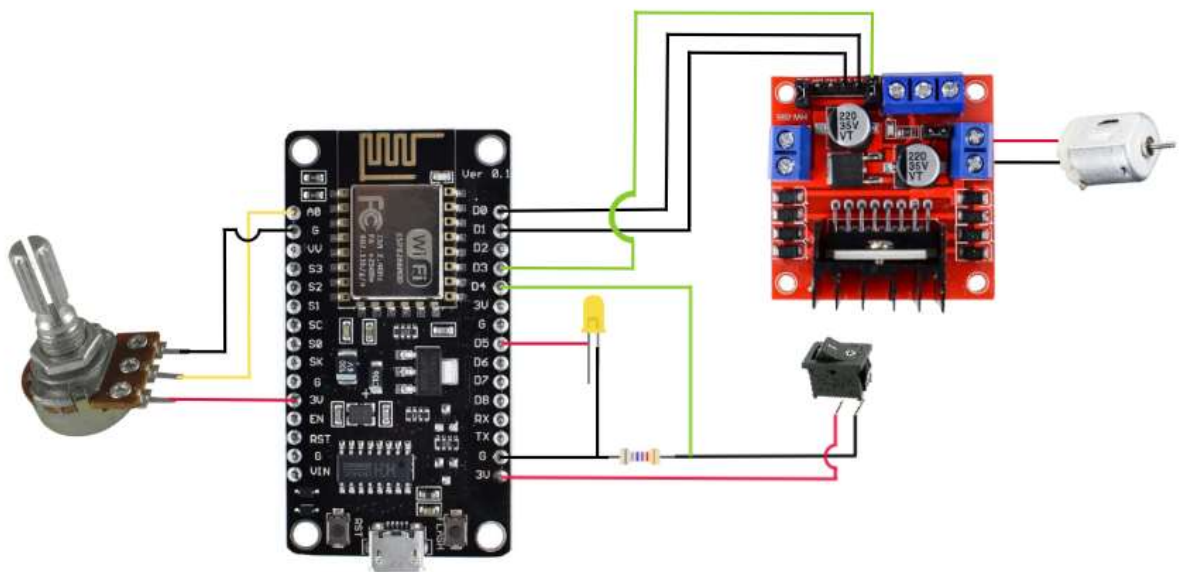
EMQX is an open-source, distributed MQTT (opens new window)messaging broker that can support millions of concurrent MQTT connections. EMQX can be used for connecting to devices that support MQTT protocol, and EMQX can also be used for delivering messages from a server to a client (e.g., a device)



### 3. Block Diagram



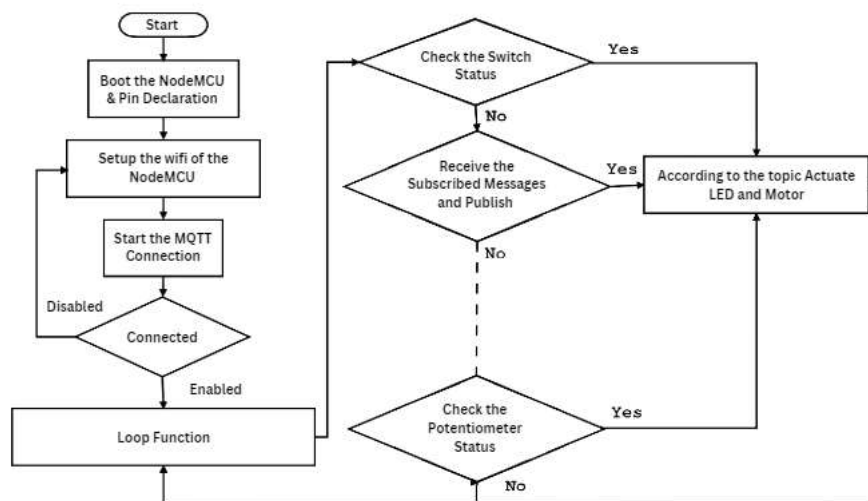
### 4. Circuit Diagram



#### Pin Declaration-

- I. Motor INPUT-1 pin as D0 (As an OUTPUT pin)
- II. Motor INPUT-2 pin as D1 (As an OUTPUT pin)
- III. Motor ENABLE pin as D3 (As an OUTPUT pin)
- IV. Led pin as D5 (As an OUTPUT pin)
- V. Switch pin as D4 (As an INPUT\_PULLUP pin)
- VI. Potentiometer pin as A0 (As an INPUT pin)

## 5. Flow Chart



## 6. Working Flow of the Project

At first, we have to make the circuit where the motor's input pins are connected to D0 and D1 and the enable pin is with D3. Then the led pin is connected with D5, the switch pin is with D4 and the potentiometer pin is with A0.

Then we will boot the NodeMCU ESP8266 and declare all the pins for Switch, Potentiometer, Led and Motor after that we will connect the node MCU with WIFI, and while successfully connected with the WIFI then the MQTT will be connected, if the WIFI fails to connect the process will repeat until the WIFI is connected with the system.

After completion of the setup now comes the actual part of controlling the speed of the motor and light, it first checks the status of the switch if it's a yes then accordingly it actuates led but if the system does not find any position of the switch it then check if there is any status in the Subscribe message if there is any information found it then actuate the speed of the motor accordingly but if the system still does not get any information then it checks the potentiometer status and acts accordingly if it also fails then the system repeats all the above process still it does not get any information from the client.

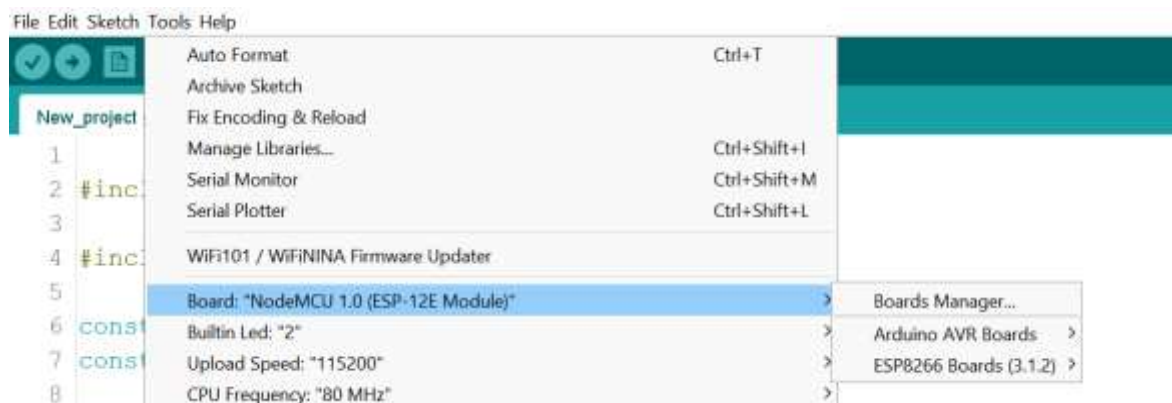
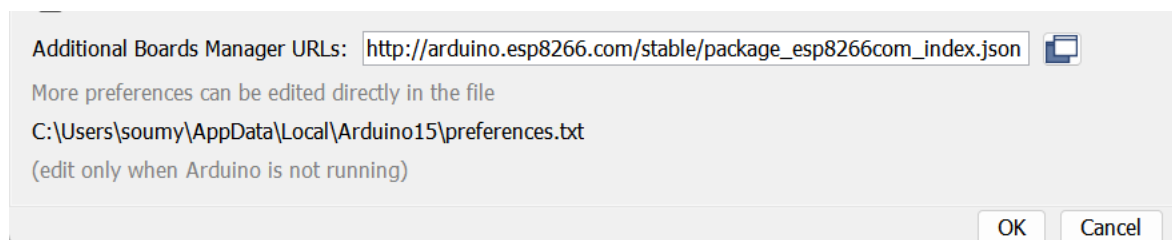
It is an IOT protocol using this we can change the speed and control the lights fan of our house without any human interference. As our technology grows, we should also go on par with the technology and reduce human interference.

## 7. Working Procedure of the Code

To use the MQTT protocol with NodeMCU ESP8266, you'll need to follow a series of steps to establish a connection, publish and subscribe to topics, and handle messages. Here's a general overview of the working procedure:

### 7.1. Set up the NodeMCU ESP8266:

Ensure that you have the necessary development environment set up for programming the NodeMCU ESP8266. This includes installing the Arduino IDE, adding the ESP8266 board support, and configuring the required libraries.



### 7.2. Install MQTT Library:

Install the MQTT library for the Arduino IDE, such as the PubSubClient library, which provides MQTT functionality. You can install the library by navigating to "Sketch" -> "Include Library" -> "Manage Libraries" in the Arduino IDE and searching for the MQTT library.

```
#include <PubSubClient.h>
```



### 7.3. Connect to Wi-Fi:

Configure the NodeMCU ESP8266 to connect to the desired Wi-Fi network. Set the Wi-Fi credentials (SSID and password) in your code and establish a connection to the network using the appropriate functions from the ESP8266WiFi library.

```
1 #include <ESP8266WiFi.h>
2
3 const char *ssid = "G-Family";
4 const char *password = "9903694766";
5
6 WiFiClient espClient;
7 PubSubClient client(espClient);
8
27 void setup_wifi() {
28
29     delay(10);
30
31     Serial.println();
32     Serial.print("Connecting to ");
33     Serial.println(ssid);
34
35     WiFi.begin(ssid, password);
36
37     while (WiFi.status() != WL_CONNECTED) {
38         delay(500);
39         Serial.print("Connecting to WiFi..");
40     }
41
42     randomSeed(micros());
43
44     Serial.println("");
45     Serial.println("WiFi connected");
46     Serial.println("IP address: ");
47     Serial.println(WiFi.localIP());
48 }
```

### 7.4. Establish an MQTT Connection:

Create an instance of the MQTT client and configure it with the necessary settings. This includes specifying the MQTT broker's IP address, port number, and any authentication credentials if required. Connect the MQTT client to the broker using the client's `connect()` function.

```

122 | client.setServer(mqtt_server, mqtt_port);
123 | client.setCallback(callback);

129 | if (!client.connected()) {
130 |     reconnect();
131 | }

84 | void reconnect() {
85 |     // Loop until we're reconnected
86 |     while (!client.connected()) {
87 |         Serial.print("Attempting MQTT connection...");
88 |
89 |         // Create a random client ID
90 |         String client_id = "esp8266-client-";
91 |         client_id += String(WiFi.macAddress());
92 |
93 |         // Attempt to connect
94 |         if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
95 |             Serial.println("Public emqx mqtt broker connected");
96 |             // Once connected, publish an announcement...
97 |             client.subscribe("device/motor");
98 |             // ... and resubscribe
99 |             client.subscribe("device/led");
100 |         } else {
101 |             Serial.print("failed with state ");
102 |             Serial.print(client.state());
103 |             Serial.println(" try again in 3 seconds");
104 |             // Wait 3 seconds before retrying
105 |             delay(3000);
106 |         }
107 |     }
108 | }

```

## 7.5. Publish Messages:

Use the MQTT client's `publish()` function to send messages to specific topics. Specify the topic and message payload to be published. You can publish sensor data, device status updates, or any other information you want to share with other devices or subscribers.

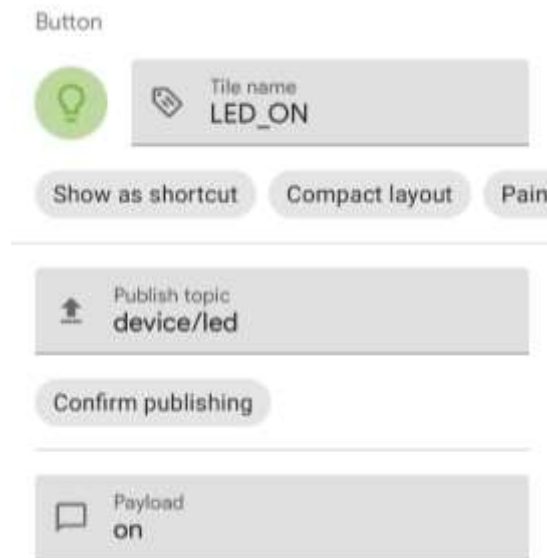
```

24 | #define sub1 "device/led"
25 | #define sub2 "device/motor"

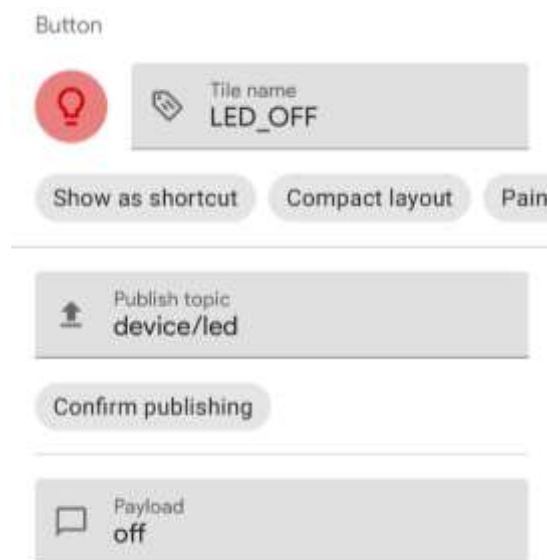
109 | client.subscribe(sub2);
110 | client.subscribe(sub1);

```

- In our project if we press the **LED\_ON Button** the it will send “on” messages in the topic “device/led” to broker and it will forward it to the subscribed clients.




- And if we press the **LED\_OFF Button** the it will send “off” messages in the topic “device/led” to broker and it will forward it to the subscribed clients.




- In our project if we toggle the **Motor Bar** the it will send the speed “0-100” as a messages in the topic “device/motor” to broker and it will forward it to the subscribed clients.

Progress




Tile name  
**MOTOR\_SPEED**


---



Subscribe topic  
**device/motorstatus**



Publish topic  
**device/motor**



☒ Enable publishing

Warning. The tile status gets updated only on incoming messages on the subscribe topic; if you plan to use a different publish topic the tile probably won't be updated.

123
 

Min. value  
 0


123
 

Max. value  
 100

☒ Display current value


- And if we toggle **Motor Bar** to “0” then it will send the speed as “0” as a message in the topic “device/motor” to broker and it will forward it to the subscribed clients.
- And there is also a **Motor\_OFF Button** which will send the value as “0” which will turn off the motor.

Button




Tile name  
**MOTOR\_OFF**

---



Publish topic  
**device/motor**

---



Payload  
 0

## 7.6. Subscribe to Topics:

Use the MQTT client's `subscribe()` function to subscribe to specific topics and receive messages published to those topics. Specify the topic(s) you want to subscribe to, and provide a callback function that will be triggered whenever a message is received on the subscribed topic(s).

```
26 | #define pub1 "device/ledstatus"
27 | #define pub2 "device/motorstatus"

111 |         client.subscribe(pub1);
112 |         client.subscribe(pub2);
```

## 7.7. Handle Incoming Messages:

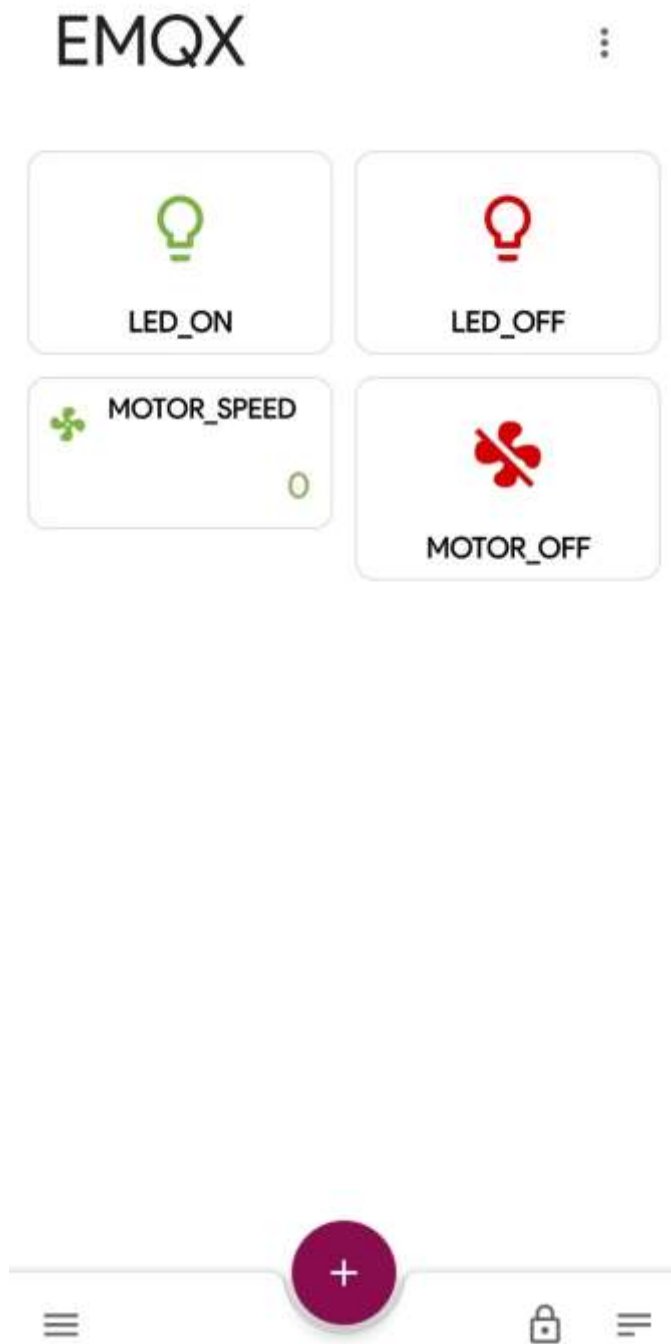
Implement the callback function that you specified in the subscription. This function will be executed whenever a message is received on the subscribed topic(s). Inside the callback, you can extract and process the message payload as needed. You can also perform actions based on the received messages, such as updating device states, triggering events, or sending commands to other components.

```
61 | void callback(char *topic, byte *payload, unsigned int length) {
62 |     Serial.print("Message arrived [");
63 |     Serial.print(topic);
64 |     Serial.print("] ");
65 |
66 |
67 |     if(strstr(topic, sub1)) {
68 |         if (!strcmp((char *)payload, "on", length)) {
69 |             digitalWrite(ledpin, HIGH);
70 |             state = 0;
71 |         }
72 |         else if (!strcmp((char *)payload, "off", length)) {
73 |             digitalWrite(ledpin, LOW);
74 |             state = 1;
75 |         }
76 |     }
77 |     if (strcmp((char*)topic, "device/motor")==0){
78 |         payload[length]='\0';
79 |
80 |         char* charPointer = (char *)payload;
81 |         double f = 0;
82 |         int i =0;
83 |         if(isDigit(charPointer[0])){
84 |             f=atof(charPointer);
85 |             i=atoi(charPointer);
86 |         }
87 |         if ( 100 >= f >= 0) {
88 |             i = map(f, 0, 100, 0, 255);
89 |             analogWrite(en1,i);
90 |             digitalWrite(in1, HIGH);
91 |             digitalWrite(in2, LOW);
92 |         }
```

### 7.8. Disconnect from the MQTT Broker:

When you're finished with the MQTT communication, call the MQTT client's `disconnect()` function to gracefully disconnect from the broker. This frees up system resources and terminates the MQTT connection.

## 8. MQTT Public/Subscribe Client Dashboard



## 9. Project Code

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

// Update these with values suitable for your network.
const char *ssid = "G-Family";
const char *password = "9903694766";

String device_name = "EMQX";
const char *mqtt_server = "broker.emqx.io";
const char *mqtt_username = "emqx";
const char *mqtt_password = "emqx";
const int mqtt_port = 1883;

//Declare the led pin and motor pins
#define ledpin D5
#define in1 D0
#define in2 D1
#define en1 D3
#define switch D4
#define pot A0
int x,i,state= 1,pot_value=0;

#define sub1 "device/led"
#define sub2 "device/motor"
#define pub1 "device/ledstatus"
#define pub2 "device/motorstatus"

WiFiClient espClient;
PubSubClient client(espClient);
```

```
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE      (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("Connecting to WiFi..");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char *topic, byte *payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
```



```

if(strstr(topic, sub1)) {
    if (!strcmp((char *)payload, "on", length)) {
        digitalWrite(ledpin, HIGH);
        state = 0;
    }
    else if (!strcmp((char *)payload, "off", length)) {
        digitalWrite(ledpin, LOW);
        state = 1;
    }
}

if (strcmp((char*)topic, "device/motor")==0){
    payload [length]='\0';

    char* charPointer = (char *)payload;
    double f = 0;
    int i =0;
    if(isDigit(charPointer[0])){
        f=atof(charPointer);
        i=atoi(charPointer);
    }
    if ( 100 >= f >= 0) {
        i = map(f, 0, 100, 0, 255);
        analogWrite(en1,i);
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
    }
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");

```

```

// Create a random client ID
String client_id = "esp8266-client-";
client_id += String(WiFi.macAddress());

// Attempt to connect
if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
    Serial.println("Public emqx mqtt broker connected");
    // Once connected, publish an announcement...
    client.subscribe(sub2);
    client.subscribe(sub1);
    client.subscribe(pub1);
    client.subscribe(pub2);
} else {
    Serial.print("failed with state ");
    Serial.print(client.state());
    Serial.println(" try again in 3 seconds");
    // Wait 3 seconds before retrying
    delay(3000);
}
}
}

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

    pinMode(ledpin, OUTPUT);
    pinMode(switch, INPUT_PULLUP);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(en1, OUTPUT);
    // pinMode(pot, INPUT);

```

```
setup_wifi();
client.setServer(mqtt_server, mqtt_port);
client.setCallback(callback);
digitalWrite(ledpin,HIGH);
}

void loop() {

    if (!client.connected()) {
        reconnect();
    }
    if(digitalRead(switch)== LOW){
        delay(200);
        if(state == 1){
            digitalWrite(ledpin, LOW);
            state = 0;
            client.publish(pub1, "Led ON");
        }
    }
    if(digitalRead(switch)== HIGH){
        if (state == 0){
            digitalWrite(ledpin, HIGH);
            state = 1;
            client.publish(pub1, "Led OFF");
        }
    }
    client.loop();

    Serial.print("Publish message: ");
    // client.publish(pub2, );
}
```

## 10. Advantage

Home automation using NodeMCU ESP8266 offers several advantages for homeowners. Here are some key benefits:

- **Cost-Effective Solution:**

NodeMCU ESP8266 is an affordable and readily available microcontroller board, making it a cost-effective choice for home automation projects. It provides a cost-efficient way to upgrade your home with smart features without significant investment.

- **Lightweight and Efficient:**

MQTT is a lightweight messaging protocol designed for constrained devices and low-bandwidth networks. NodeMCU ESP8266, being a low-cost and power-efficient microcontroller, is well-suited for MQTT-based home automation. This combination allows for efficient and reliable communication between devices while minimizing resource usage.

- **Ease of Use and Programming:**

NodeMCU ESP8266 is based on the popular Arduino programming framework, which offers a user-friendly development environment. It simplifies the process of programming and uploading code to the board, even for beginners or individuals with limited programming experience.

- **Wi-Fi Connectivity:**

NodeMCU ESP8266 comes with built-in Wi-Fi capabilities, allowing seamless integration with your home network. This enables the device to connect to the internet, access cloud services, and communicate with other smart devices, making it an ideal choice for building an interconnected smart home ecosystem.

- **Real-time Communication:**

MQTT protocol follows a publish-subscribe model, enabling real-time communication between devices. This means that devices can instantly send and receive messages, allowing for prompt updates and actions in a home automation system. Whether it's controlling lights, adjusting temperature, or receiving security alerts, MQTT facilitates near-instantaneous communication.

- **Reliability and Robustness:**

MQTT protocol provides reliable message delivery by supporting different levels of Quality of Service (QoS). This ensures that messages are delivered with the desired level of reliability, even in scenarios with intermittent network connectivity or unstable connections. By leveraging MQTT's QoS levels, home automation systems using NodeMCU ESP8266 can maintain data integrity and ensure critical messages are received.

- **Remote Access and Control:**

With NodeMCU ESP8266, you can remotely access and control your home automation system using smartphones, tablets, or computers. This enables you to monitor and manage your home devices, adjust settings, and receive real-time notifications from anywhere, enhancing convenience and providing peace of mind.

- **Energy Efficiency:**

NodeMCU ESP8266 facilitates energy monitoring and management within your home automation system. By integrating energy monitoring sensors or utilizing power-saving techniques, you can track and optimize energy usage, leading to potential energy savings and increased sustainability.

## 11. Limitations

There are few limitation of this project those are given bellow

- **Limited GPIO Pins:**

The Node MCU ESP8266 has a limited number of GPIO pins available (usually around 10). This can be a disadvantage if you need to control multiple motors or other peripherals simultaneously, as it may not have enough pins for your specific project requirements.

- **Limited Processing Power:**

The ESP8266 microcontroller on the NodeMCU board is not as powerful as some other microcontrollers or development boards. It has a relatively low clock speed and limited RAM, which can be a disadvantage when performing complex motor control algorithms or when handling other tasks concurrently.

- **Limited Power Output:**

The NodeMCU ESP8266 may not provide sufficient current or voltage to drive high-power motors directly. If you're working with motors that require higher power, you may need additional motor driver circuits or external power supplies to interface between the NodeMCU and the motors.

- **Noise and Interference:**

The NodeMCU ESP8266 operates on the 2.4GHz frequency, which is also used by other wireless devices such as Wi-Fi routers and Bluetooth devices. This can lead to interference and noise that may affect the stability and reliability of motor control signals.

## 12. Future Scope

- **Home Automation:**

Node MCU-based motor control can be integrated into home automation systems. By connecting motors to Node MCU devices, homeowners can remotely control motorized appliances such as fans, blinds, curtains, and gates. This allows for convenient and personalized control of various aspects of the home environment.

- **Smart Agriculture:**

Node MCU can find applications in smart agriculture systems. It can control motors in irrigation systems, greenhouse vents, and agricultural machinery. By integrating sensors and Node MCU, motor control can be optimized based on environmental conditions and data analytics, leading to improved efficiency and crop yield.

- **Energy Efficiency and Conservation:**

Node MCU-based motor control can contribute to energy management and conservation efforts. By monitoring motor usage, implementing control algorithms, and integrating with energy management systems, Node MCU can optimize energy consumption and reduce wastage in various settings, including industrial and residential environments.

## 13. Conclusion

In conclusion, controlling a motor using Node MCU, an IoT-enabled microcontroller board, offers several advantages and opportunities. Node MCU provides Wi-Fi connectivity, affordability, ease of use, and a large community of developers, making it a popular choice for motor control projects. By leveraging Node MCU, you can enable remote control, automation, and integration with other IoT devices, leading to enhanced functionality and convenience.

This project demonstrates how to control the speed of a motor using the Node MCU ESP8266 microcontroller board and the MQTT protocol. By leveraging MQTT, the motor can be controlled remotely and integrated into larger IoT systems for automation and monitoring purposes. This project provides a foundation for further exploration and integration of motor control into various IoT applications.

## 14. Bibliography

- ZDNET software ltd.

Available: <https://www.zdnet.com/home-and-office/smart-home/best-home-automation-system/>

- Smart Home Automation in India | Home Control & Smart from Schneider Electric

Available: <https://www.se.com/in/en/work/products/product-launch/smart-home-automation-wiser/>

- WiFi Based DC Motor Speed and Direction Control using NodeMCU from svsembedded

Available: <https://youtu.be/tEHrzWihEC4>

