**DISTANCE MEASURE**

**ABSTRACT:**

This code integrates an ESP8266 or ESP32 microcontroller with an ultrasonic sensor and MQTT communication to create a versatile distance monitoring system. Here's a more detailed abstract:

The project combines hardware components and software libraries to enable remote distance sensing and monitoring. The ESP8266 or ESP32 microcontroller serves as the central processing unit, responsible for reading data from the ultrasonic sensor and transmitting it over a Wi-Fi network using MQTT protocol.

Upon startup, the microcontroller initializes the Wi-Fi connection by connecting to a specified SSID and password. Once connected, it establishes communication with an MQTT broker running on a local or remote server.

The ultrasonic sensor, connected to the microcontroller via digital pins, emits ultrasonic pulses and measures the time taken for the pulses to bounce back after hitting an object. This information is converted into distance measurements in centimeters using the NewPing library, which provides accurate and reliable distance readings.

The main loop of the program continuously performs the following steps:

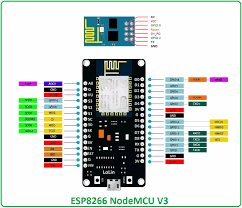
1. Checks if the microcontroller is connected to the MQTT broker. If not, attempts to reconnect.
2. Reads the distance from the ultrasonic sensor by triggering a pulse and calculating the time it takes for the echo to return.
3. Checks the validity of the distance reading. If the reading is zero, it indicates a failed measurement or that the target is out of range.
4. Prints the distance reading to the serial monitor for local debugging and monitoring.
5. Publishes the distance reading to a predefined MQTT topic, allowing remote devices or applications to subscribe and receive real-time updates on the sensed distance.

Through MQTT communication, the distance measurements are made available to other devices or systems connected to the same MQTT broker, enabling various applications such as home automation, industrial monitoring, or environmental sensing.

Overall, this project demonstrates the integration of IoT (Internet of Things) principles to create a scalable and efficient distance monitoring system that can be deployed in diverse scenarios with minimal hardware and software resources

**COMPONENTS:**

**ESP8266**: The ESP8266 is a low-cost Wi-Fi microcontroller chip with full TCP/IP stack and microcontroller capability. It is widely used in IoT projects due to its ability to connect to a Wi-Fi network and process data. In this project, the ESP8266 handles Wi-Fi connectivity, receives control commands, and interfaces with both the servo motor and the LCD display.



**ULTRASONIC SENSOR:** The ultrasonic sensor emits sound waves and measures the time it takes for them to bounce back, enabling precise distance calculation. Integrated with the ESP8266, it provides real-time distance data wirelessly to the Blynk app, offering remote distance monitoring on mobile devices.



**Jumper Wires**: Jumper wires are insulated wires with connectors (typically male or female pins) at each end, used to create temporary or semi-permanent connections between different components on a breadboard or between a breadboard and other devices.



**CODE:**

#ifdef ESP8266

#include <ESP8266WiFi.h> /\* WiFi library for ESP8266 \*/

#else

#include <WiFi.h> /\* WiFi library for ESP32 \*/

#endif

#include <Wire.h>

#include <PubSubClient.h>

#include <NewPing.h> /\* Ultrasonic sensor library \*/

// Define pins for ultrasonic sensor

#define TRIGGER\_PIN 5

#define ECHO\_PIN 4

#define MAX\_DISTANCE 200 // Maximum distance we want to ping for (in centimeters)

// Define Wi-Fi and MQTT settings

#define wifi\_ssid "moto g54 5G\_3480"

#define wifi\_password "9500697232"

#define mqtt\_server "192.168.253.87"

// Define MQTT topics

#define distance\_topic "sensor/ultrasonic/distance\_cm"

// Create objects for Wi-Fi, MQTT, and Ultrasonic sensor

WiFiClient espClient;

PubSubClient client(espClient);

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE);

void setup() {

Serial.begin(9600);

setup\_wifi();

client.setServer(mqtt\_server, 1883);

}

void setup\_wifi() {

delay(10);

// Connecting to a WiFi network

Serial.println();

Serial.print("Connecting to ");

Serial.println(wifi\_ssid);

WiFi.begin(wifi\_ssid, wifi\_password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

void reconnect() {

// Loop until we're reconnected

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

if (client.connect("ESP8266Client")) {

Serial.println("connected");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

delay(5000);

}

}

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

// Wait a few seconds between measurements

delay(2000);

// Read distance from ultrasonic sensor

unsigned int distance = sonar.ping\_cm();

// Check if the reading is valid

if (distance == 0) {

Serial.println("Failed to read from ultrasonic sensor or out of range!");

return;

}

// Print and publish distance

Serial.print("Distance: ");

Serial.print(distance);

Serial.println(" cm");

client.publish(distance\_topic, String(distance).c\_str(), true);

}



**CODE EXPALNATION**

#### Libraries and Credentials

* **Includes Necessary Libraries for WiFi and MQTT:** The project begins by including the necessary libraries for WiFi and MQTT communication. These libraries enable the ESP32 to connect to a WiFi network and communicate with the MQTT broker.
* **Stores WiFi and MQTT Credentials:** The WiFi SSID and password, along with MQTT broker username and password, are stored in variables. This allows the ESP32 to connect to the specified WiFi network and authenticate with the MQTT broker.

#### WiFi and MQTT Setup

* **Connecting to WiFi:** A function is defined to handle the connection to the WiFi network. It continuously attempts to connect until a connection is established, providing feedback via the serial monitor.
* **Connecting to MQTT Broker:** Another function handles the connection to the MQTT broker. It also continuously attempts to connect until successful and subscribes to a specific MQTT topic to control the buzzer.

#### MQTT Callback

* **Handling Received MQTT Messages:** A callback function is defined to handle messages received on the subscribed MQTT topic. When a message is received, the function checks the topic and payload. If the message is for the buzzer control topic and contains "1", the buzzer is turned on. If the message contains "0", the buzzer is turned off. The buzzer's status is then published to a separate MQTT topic.

#### Ultrasonic Distance Measurement

* **Measuring Distance Using the Ultrasonic Sensor:** A function is defined to measure the distance using the ultrasonic sensor. It triggers the sensor to send out a pulse and measures the time it takes for the echo to return. This time is used to calculate the distance to an object in centimeters.

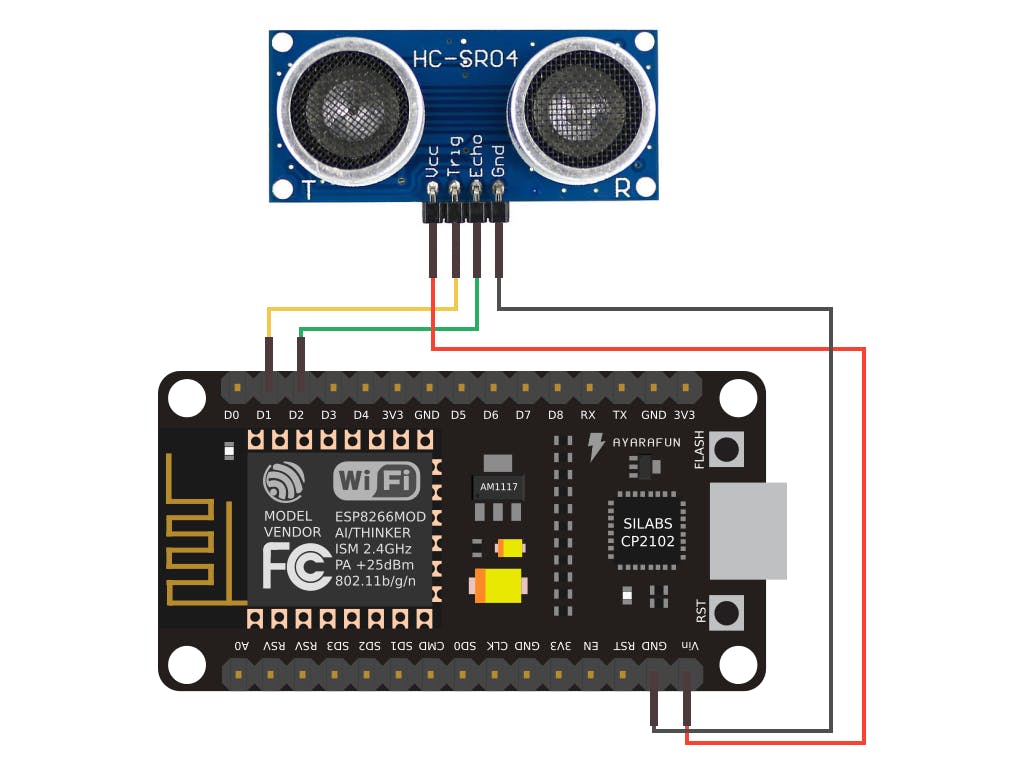
#### Setup Function

* **Initialization of Serial Communication, WiFi, MQTT Connections, and Pin Modes:** In the setup function, the serial communication is initialized for debugging purposes. The ESP32 attempts to connect to the specified WiFi network and MQTT broker. The pin modes for the buzzer and ultrasonic sensor are set up to ensure proper operation.

#### Main Loop

* **Maintaining WiFi and MQTT Connections and Publishing Distance Measurements:** The main loop continuously checks and maintains the WiFi and MQTT connections. If either connection is lost, it attempts to reconnect. The loop also measures the distance using the ultrasonic sensor at regular intervals and publishes the measured distance to a specific MQTT topic. This ensures that the distance data is continuously sent to the MQTT broker.

**CIRCUIT DIAGRAM:**



**STEPS TO CREATE A PROJECT IN ARDUINO IOT CLOUD**

**Set Up the Hardware:**

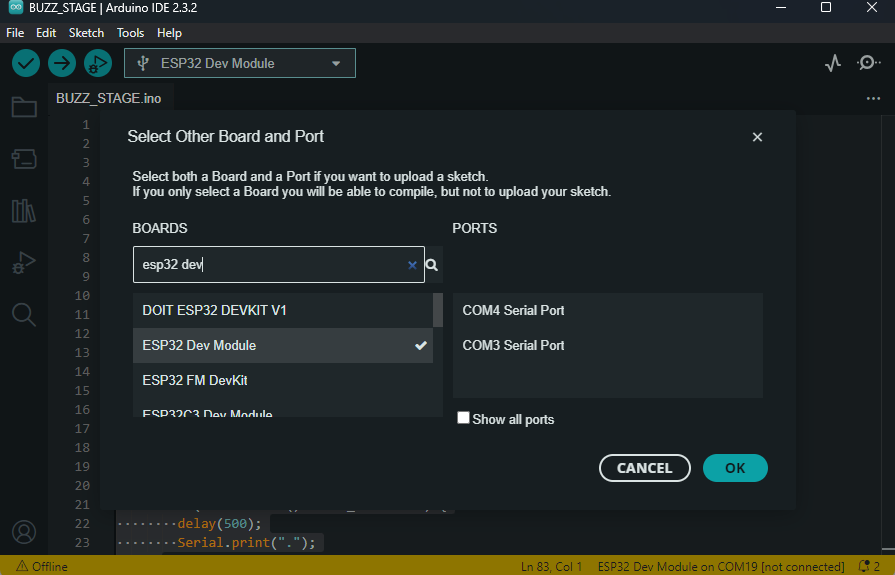
* Connect the ultrasonic sensor to GPIO pins defined for trigger and echo signals on the ESP32.
* Connect the trigger pin of the ultrasonic sensor to a GPIO pin on the ESP32.
* Connect the echo pin of the ultrasonic sensor to another GPIO pin on the ESP32.
* Ensure the sensor's VCC pin is connected to a 5V pin on the ESP32, and the GND pin to a GND pin on the ESP32.

**Install Arduino IDE:**

* Download and install the Arduino IDE from the official website (https://www.arduino.cc/en/software).

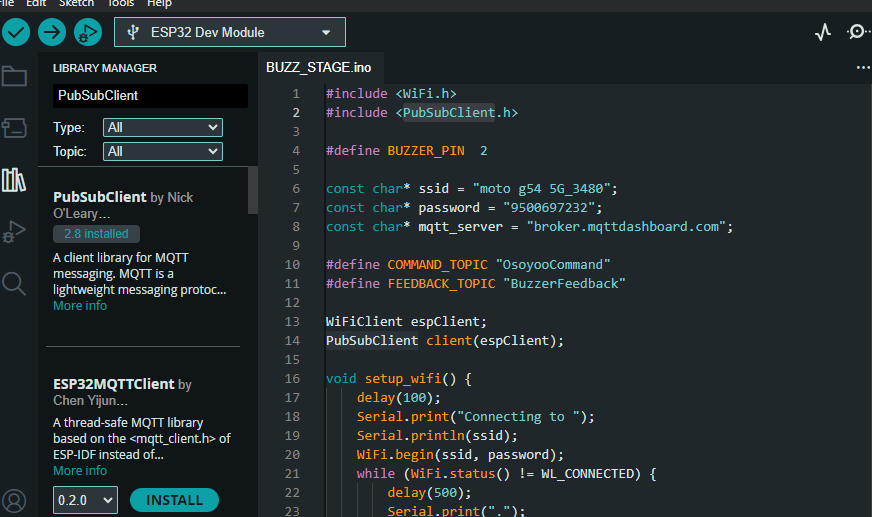
**Install ESP32 Board in Arduino IDE:**

* Open the Arduino IDE.
* Go to Tools > Board > Boards Manager.
* Search for ESP32 and install the esp32 by Espressif Systems.



**Install PubSubClient Library:**

* In the Arduino IDE, go to Sketch > Include Library > Manage Libraries.
* Search for PubSubClient and install it.

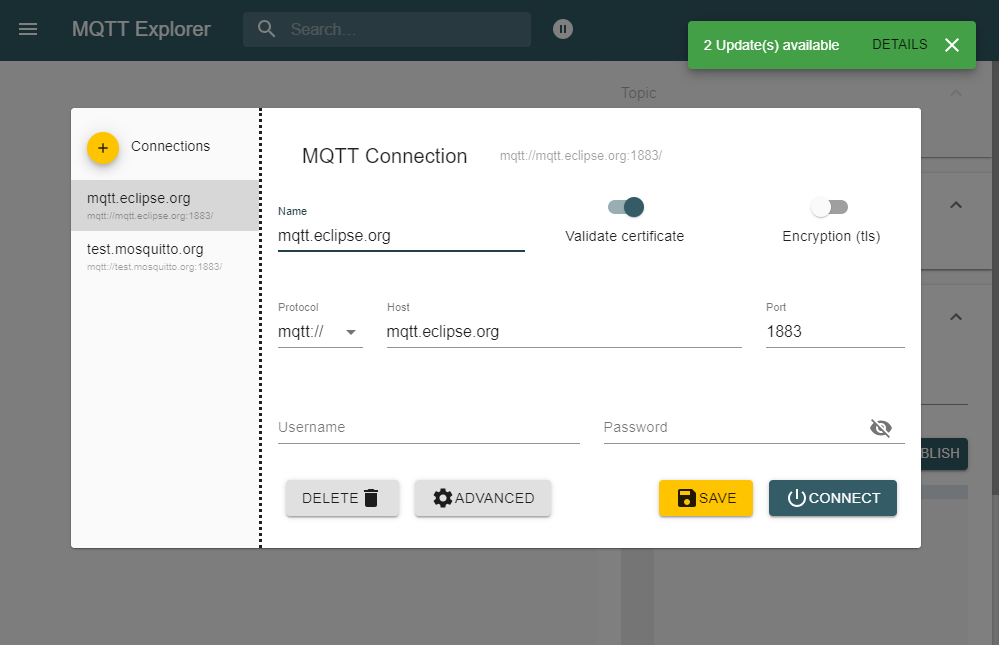


**Write the Code:**

* Open a new sketch in Arduino IDE and copy the following code:
* Connect your ESP32 to your computer using a USB cable.
* Select the correct board and port in the Arduino IDE:
* Go to Tools > Board > ESP32 Dev Module.
* Go to Tools > Port and select the COM port to which the ESP32 is connected.
* Click on the upload button to compile and upload the code to the ESP32.

**Set Up MQTT Explorer:**

* Download and install MQTT Explorer from its official website.
* Open MQTT Explorer and create a new connection.
* Configure the connection with the following settings:
  + Connection Name: Any name you prefer.
  + Broker Address: broker.mqttdashboard.com (or your MQTT broker's address).
  + Port: 1883 (default MQTT port).
  + Client ID: Any unique identifier.
  + Username and Password: If your MQTT broker requires authentication, provide the credentials.
  + Leave other settings as default unless your setup requires specific configurations.



**Interact with MQTT Topics:**

* Once connected, MQTT Explorer will display a list of available topics on the broker.
* Subscribe to the feedback topic (BuzzerFeedback) to monitor sensor status.
* Publish messages to the command topic (OsoyooCommand) to trigger the ultrasonic sensor.
* Use MQTT Explorer's interface to send and receive messages, visualize topics, and monitor sensor feedback.

**CONCLUSION:**

In conclusion, the provided code presents a robust solution for real-time distance measurement using an ultrasonic sensor integrated with a NodeMCU microcontroller and hosted on a web server. By leveraging the capabilities of the NodeMCU and the versatility of web technologies, the system enables users to remotely monitor distances from any web-enabled device with ease.

Throughout the implementation, the ultrasonic sensor serves as a reliable tool for accurately measuring distances, while the NodeMCU acts as the central processing unit, facilitating sensor interfacing and web server hosting functionalities. The integration of an asynchronous web server on the NodeMCU platform enhances accessibility, allowing users to access distance measurement functionality seamlessly via a web interface.

The dynamic nature of the web interface, coupled with real-time data updates, provides users with immediate access to distance measurements, empowering them with timely insights for informed decision-making and operational control. Additionally, the simplicity and user-friendly design of the web interface contribute to an intuitive user experience, making distance monitoring accessible to a wide range of users.

Overall, this project demonstrates the potential of IoT technologies in facilitating remote monitoring and control applications. With its practical utility and versatility, the system finds applications across various domains, including home automation, industrial monitoring, robotics, and beyond. Through continuous innovation and refinement, similar systems can further enhance efficiency, productivity, and safety in diverse operational environments.