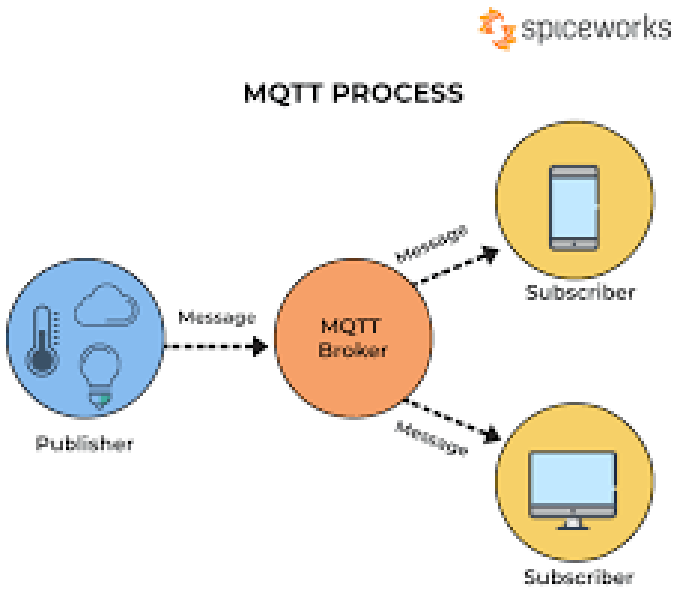
Mqtt_Laboratory

Today you will have 2 papers to work on in this laboratory, so let's get started



What Is MQTT?

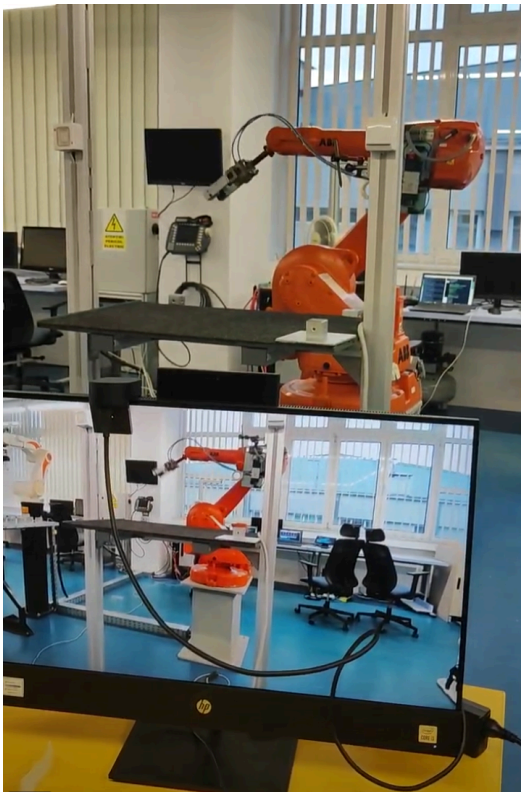
Message queuing telemetry support (MQTT) is defined as a low bandwidth consumption machine-to-machine protocol that helps IoT devices communicate with each other, with minimal code requirements and network footprint.

source:https://www.spiceworks.com/tech/iot/articles/what-is-mqtt/

TASK 1- Connection interface Python and Robot IRB1600

Video:https://youtu.be/E-Gvn6QCAYc

Code: Mqtt_Socket



```
import socket
import sys
import random
import paho.mqtt.client as mqtt

broker = 'broker.emqx.io'
port = 1883
topic = "button_topic/mqtt"
client_id = f'python-mqtt-{random.randint(0, 100)}'

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)

def on_message(client, userdata, msg):
    print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
    trimitere_primire_mesaje(msg.payload.decode())

def connect_mqtt() -> mqtt.Client:
    client = mqtt.Client(client_id)
    client.on_connect = on_connect
    client.on_message = on_message
```

```
    client.connect(broker, port)
    client.subscribe(topic)
    return client

def trimitere_primire_mesaje(topiccc):
    adresa_serverului = ("192.168.125.1", 1500)
    print(sys.stderr, "Conectare la adresa %s si portul %s" % adresa_serverului)
    priza_ethernet = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    priza_ethernet.connect(adresa_serverului)
    print("Ca sa actionezi robotul ai de ales intre doua aplicatii")
    print("Tasteaza aplicatia_1 sau aplicatia_2")
    print("Orice alt mesaj va duce robotul in pozitia de parcare")

    mesajul_introdus = topiccc.encode()
    if mesajul_introdus == b"1":
        print("Ai ales prima aplicatie1")
        a = b"aplicatia_1"
    elif mesajul_introdus == b"2":
        print("Ai ales a doua aplicatie2")
        a = b"aplicatia_2"
    else:
        print("Ai ales sa parchezi robotul")
        a = b"aplica"
    print(sys.stderr, "Se va trimite mesajul %s" % a)
    priza_ethernet.sendall(a)
    mesaj_primit = repr(priza_ethernet.recv(1025))
    print("Mesajul primit este: %s" % mesaj_primit)
    priza_ethernet.close()

if __name__ == "__main__":
    client = connect_mqtt()
    try:
        client.loop_forever()
    except KeyboardInterrupt:
        client.disconnect()
        client.loop_stop()
```

Explained

This code establishes a connection to an MQTT broker and listens for messages on a specified topic, interacting with a robot through a TCP socket based on the received messages. Here's a breakdown of the code:

1. Imports

python

Copy code

```
import socket
import sys
import random
import paho.mqtt.client as mqtt
```

- **`socket`**: Used for networking, specifically to connect to a TCP server.
- **`sys`**: Provides access to some system-related functions like `stderr` for error output.
- **`random`**: Generates random numbers (used here for generating a unique MQTT client ID).
- **`paho.mqtt.client`**: A Python library for implementing MQTT clients for communication with MQTT brokers.

2. MQTT Broker and Topic Setup

python
Copy code

```
broker = 'broker.emqx.io'
port = 1883
topic = "button_topic/mqtt"
client_id = f'python-mqtt-{random.randint(0, 100)}'
```

- **`broker`**: Specifies the MQTT broker address (`broker.emqx.io` is a free public MQTT broker).
- **`port`**: MQTT typically uses port 1883.
- **`topic`**: The topic on which this client will subscribe to receive messages.
- **`client_id`**: A unique client ID is generated using `random.randint(0, 100)` to avoid conflicts.

3. Callback Functions

These functions are called when specific MQTT events happen.

python
Copy code

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)
```

- **`on_connect`**: Triggered when the client successfully connects to the broker. If the connection is successful (`rc == 0`), it prints a success message. Otherwise, it prints an error message.

python
Copy code
```
def on_message(client, userdata, msg):
    print(f"Received `{msg.payload.decode()}` from `{msg.topic}`
topic")
    trimitere_primire_mesaje(msg.payload.decode())
```

- **`on_message`**: Triggered when a message is received from the subscribed topic. It decodes the message and calls the function `trimitere_primire_mesaje` to handle it.

4. MQTT Connection and Subscription

python
Copy code
```
def connect_mqtt() -> mqtt.Client:
    client = mqtt.Client(client_id)
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(broker, port)
    client.subscribe(topic)
    return client
```

- **`connect_mqtt`**: Creates a new MQTT client, assigns the callback functions for connection and message events, connects to the broker, and subscribes to the specified topic.

5. Handling Messages with a TCP Socket

python
Copy code
```
def trimitere_primire_mesaje(topiccc):
    adresa_serverului = ("192.168.125.1", 1500)
    print(sys.stderr, "Conectare la adresa %s si portul %s" %
adresa_serverului)
    priza_ethernet = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    priza_ethernet.connect(adresa_serverului)
    print("Ca sa actionezi robotul ai de ales intre doua aplicatii")
    print("Tasteaza aplicatia_1 sau aplicatia_2")
```

```
print("Orice alt mesaj va duce robotul in pozitia de parcare")
```

- **`trimitere_primire_mesaje`**: Handles the message received via MQTT. It sets up a TCP connection to a server with the IP `192.168.125.1` on port `1500`.
    - The message is interpreted as a command to control a robot (based on whether it's `1`, `2`, or something else).
    - Depending on the message, it sends commands like `aplicatia_1` or `aplicatia_2` to the server to control the robot.
    - If the message is not `1` or `2`, it sends a message to park the robot.

The code then sends the selected command through the socket using:

python
Copy code
```
priza_ethernet.sendall(a)
```

It waits for a response from the server and prints the received message:

python
Copy code
```
mesaj_primit = repr(priza_ethernet.recv(1025))
print("Mesajul primit este: %s" % mesaj_primit)
```

Finally, the connection to the server is closed:

python
Copy code
```
priza_ethernet.close()
```

6. Main Loop

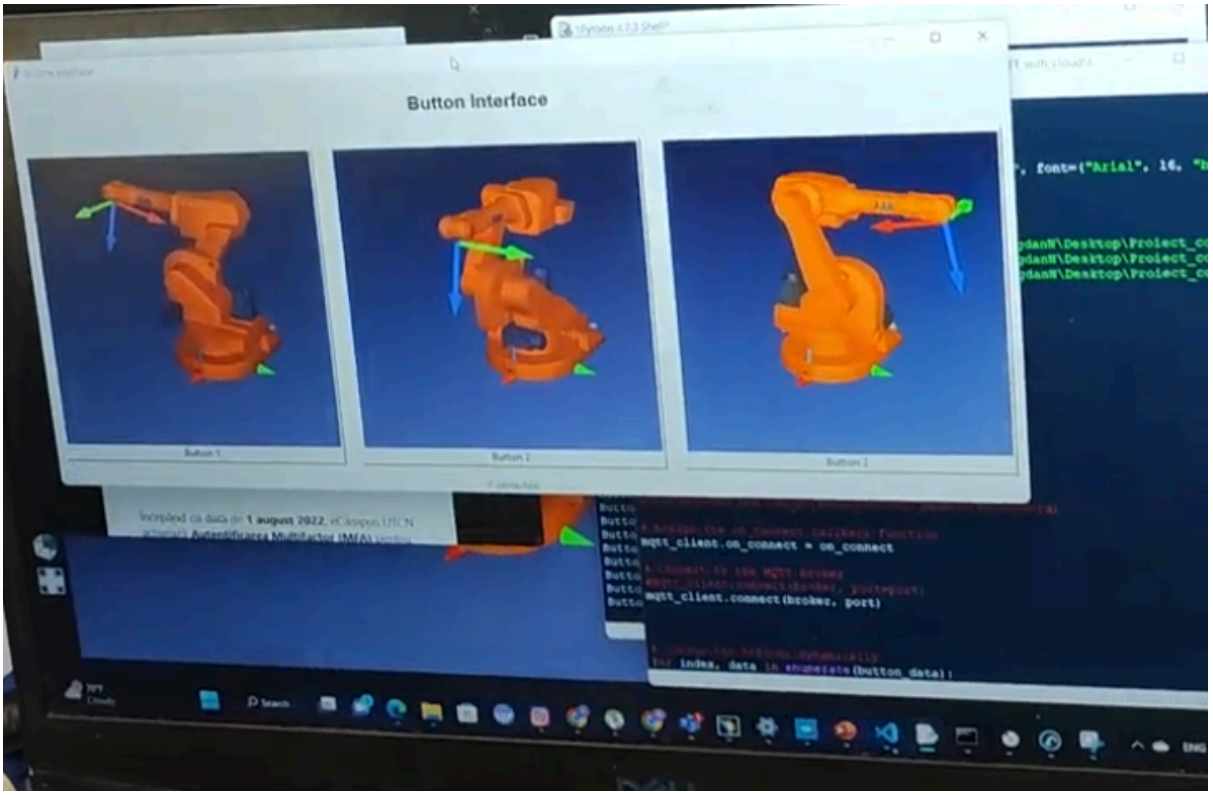
python
Copy code
```
if __name__ == "__main__":
    client = connect_mqtt()
    try:
        client.loop_forever()
    except KeyboardInterrupt:
        client.disconnect()
        client.loop_stop()
```

- **Main loop**: The MQTT client connects to the broker and enters an infinite loop to listen for incoming messages (`client.loop_forever()`).
- When you stop the program (e.g., via a keyboard interrupt), it disconnects from the broker gracefully.

Summary

- The script connects to an MQTT broker and listens to messages on the topic `button_topic/mqtt`.
- When a message is received, it's processed and sent over a TCP connection to a robot control server.
- Based on the message received, the robot is instructed to perform one of two tasks or park itself.

Publisher Mqtt_Interface




```
import tkinter as tk
from PIL import Image, ImageTk
import paho.mqtt.client as mqtt
import random

# MQTT broker settings
broker = 'broker.emqx.io'
port = 1883
```

```
topic = "button_topic/mqtt"
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 1000)}'
# username = 'emqx'
# password = 'public'

def button_pressed(number):
    mqtt_client.publish(topic, number)
    print(f"Button {number} pressed and sent to MQTT broker")

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        connection_label.config(text="Connected", fg="green")
    else:
        connection_label.config(text="Not Connected", fg="red")

# Create the main window
window = tk.Tk()
window.title("Button Interface")

# Create a title label
title_label = tk.Label(window, text="Button Interface", font=("Arial", 16, "bold"))
title_label.pack(pady=20)

# List of button labels and image file paths
button_data = [
    {"label": "Button 1", "image_path":
r"C:/Users/PCRoboti/Downloads/Proiect_conectare_MQTT_with_cloud/Proiect_conectare_
MQTT_with_cloud/Picture/Figure1.jpg"},
    {"label": "Button 2", "image_path":
r"C:/Users/PCRoboti/Downloads/Proiect_conectare_MQTT_with_cloud/Proiect_conectare_
MQTT_with_cloud/Picture/Figure2.jpg"},
    {"label": "Button 3", "image_path":
r"C:/Users/PCRoboti/Downloads/Proiect_conectare_MQTT_with_cloud/Proiect_conectare_
MQTT_with_cloud/Picture/Figure3.jpg"}
]

# Create a frame to hold the buttons
button_frame = tk.Frame(window)
button_frame.pack()

# Configure grid weight for resizing
button_frame.grid_columnconfigure(0, weight=1)
button_frame.grid_columnconfigure(1, weight=1)
button_frame.grid_columnconfigure(2, weight=1)

# Create the MQTT client
mqtt_client = mqtt.Client()
```

```
# Set username and password for MQTT broker
#mqtt_client.username_pw_set(username, password=password)

# Assign the on_connect callback function
mqtt_client.on_connect = on_connect

# Connect to the MQTT broker
#mqtt_client.connect(broker, port=port)
mqtt_client.connect(broker, port)



# Create the buttons dynamically
for index, data in enumerate(button_data):
    label = data["label"]
    image_path = data["image_path"]

    # Load the image and resize it
    image = Image.open(image_path)
    image = image.resize((350, 350))  # Adjust the dimensions as needed

    # Create the photo image object
    photo = ImageTk.PhotoImage(image)

    # Create the button with image and label
    button = tk.Button(button_frame, text=label, image=photo, compound=tk.TOP,
                       command=lambda num=index+1: button_pressed(num))
    button.image = photo  # Store a reference to the image to prevent garbage collection

    # Place the button in the frame
    button.grid(row=0, column=index, padx=10, pady=10, sticky="nsew")

# Create a label to indicate the connection status
connection_label = tk.Label(window, text="Not Connected", fg="red")
connection_label.pack()

# Start the MQTT loop
mqtt_client.loop_start()

# Start the GUI event loop
window.mainloop()

# Disconnect from the MQTT broker
mqtt_client.disconnect()
```

Explained

This code creates a graphical user interface (GUI) using `Tkinter` that displays a set of buttons with images. When a button is clicked, a message is sent to an MQTT broker using the `paho-mqtt` library. The code includes the following components:

1. Imports

python
Copy code

```
import tkinter as tk
from PIL import Image, ImageTk
import paho.mqtt.client as mqtt
import random
```

- **`tkinter`**: A built-in Python module for creating GUIs.
- **`PIL.Image` and `PIL.ImageTk`**: Part of the Python Imaging Library (Pillow), used for handling image files and displaying them in the GUI.
- **`paho.mqtt.client`**: A library for interacting with an MQTT broker.
- **`random`**: Used to generate a unique MQTT client ID.

2. MQTT Broker Settings

python
Copy code

```
broker = 'broker.emqx.io'
port = 1883
topic = "button_topic/mqtt"
client_id = f'python-mqtt-{random.randint(0, 1000)}'
```

- **`broker`**: The MQTT broker address (`broker.emqx.io` is a free broker).
- **`port`**: Standard MQTT port (`1883`).
- **`topic`**: The topic to which button-press messages will be sent.
- **`client_id`**: A unique ID for the MQTT client to avoid conflicts.

3. Button Press Handler

python
Copy code

```
def button_pressed(number):
    mqtt_client.publish(topic, number)
    print(f"Button {number} pressed and sent to MQTT broker")
```

- **`button_pressed(number)`**: Publishes a message to the MQTT broker whenever a button is pressed. The message contains the button number.

4. MQTT Connection Callback

python
Copy code

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        connection_label.config(text="Connected", fg="green")
    else:
        connection_label.config(text="Not Connected", fg="red")
```

- **`on_connect(client, userdata, flags, rc)`**: A callback that runs when the MQTT client connects to the broker. If the connection is successful (`rc == 0`), it updates the label in the GUI to show "Connected." Otherwise, it shows "Not Connected."

5. Tkinter GUI Setup

Main Window:
python
Copy code

```
window = tk.Tk()
window.title("Button Interface")
```

- Creates the main window for the GUI.

Title Label:
python
Copy code

```
title_label = tk.Label(window, text="Button Interface",
font=("Arial", 16, "bold"))
title_label.pack(pady=20)
```

- Adds a title at the top of the window.

Button Data:
python
Copy code

```
button_data = [
    {"label": "Button 1", "image_path": r"path_to_image1"},
    {"label": "Button 2", "image_path": r"path_to_image2"},
    {"label": "Button 3", "image_path": r"path_to_image3"}
]
```

- This is a list of dictionaries where each entry contains:
    - `label`: The button's label.
    - `image_path`: The path to the image associated with each button.

Button Frame:
python
Copy code
```
button_frame = tk.Frame(window)
button_frame.pack()
```

- Creates a frame to hold the buttons.

Grid Configuration:
python
Copy code
```
button_frame.grid_columnconfigure(0, weight=1)
button_frame.grid_columnconfigure(1, weight=1)
button_frame.grid_columnconfigure(2, weight=1)
```

- Sets up a grid for the buttons, allowing the frame to resize the buttons equally.

6. MQTT Client Setup

python
Copy code
```
mqtt_client = mqtt.Client()
mqtt_client.on_connect = on_connect
mqtt_client.connect(broker, port)
```

- `mqtt_client = mqtt.Client()`: Initializes the MQTT client.
- `mqtt_client.on_connect = on_connect`: Assigns the `on_connect` callback to handle MQTT connection events.
- `mqtt_client.connect(broker, port)`: Connects to the MQTT broker.

7. Creating Buttons with Images

python
Copy code
```
for index, data in enumerate(button_data):
    label = data["label"]
    image_path = data["image_path"]
    image = Image.open(image_path)
    image = image.resize((350, 350))  # Resize image
    photo = ImageTk.PhotoImage(image)
    button = tk.Button(button_frame, text=label, image=photo,
compound=tk.TOP,
```

```
                    command=lambda num=index+1:
button_pressed(num))
    button.image = photo  # Prevents the image from being garbage
collected
    button.grid(row=0, column=index, padx=10, pady=10,
sticky="nsew")
```

- `enumerate(button_data)`: Loops through each button's data (label and image path).
- **Image Handling**: Loads the image from the path and resizes it to fit within the GUI.
- **Button Creation**: A button is created for each entry in `button_data`, displaying the image and text. The `command` argument binds the button press to the `button_pressed` function, passing the button's index as the argument.

8. Connection Status Label

python
Copy code

```
connection_label = tk.Label(window, text="Not Connected", fg="red")
connection_label.pack()
```

A label is added to the window to display the connection status of the MQTT client.

9. Starting MQTT and GUI Loops

python
Copy code

```
mqtt_client.loop_start()
window.mainloop()
```

- `mqtt_client.loop_start()`: Starts the MQTT client loop in a background thread. This is necessary for handling incoming MQTT messages and maintaining the connection.
- `window.mainloop()`: Starts the Tkinter event loop, keeping the GUI open.

10. Disconnecting from the Broker

python
Copy code

```
mqtt_client.disconnect()
```

This disconnects the MQTT client when the GUI is closed.

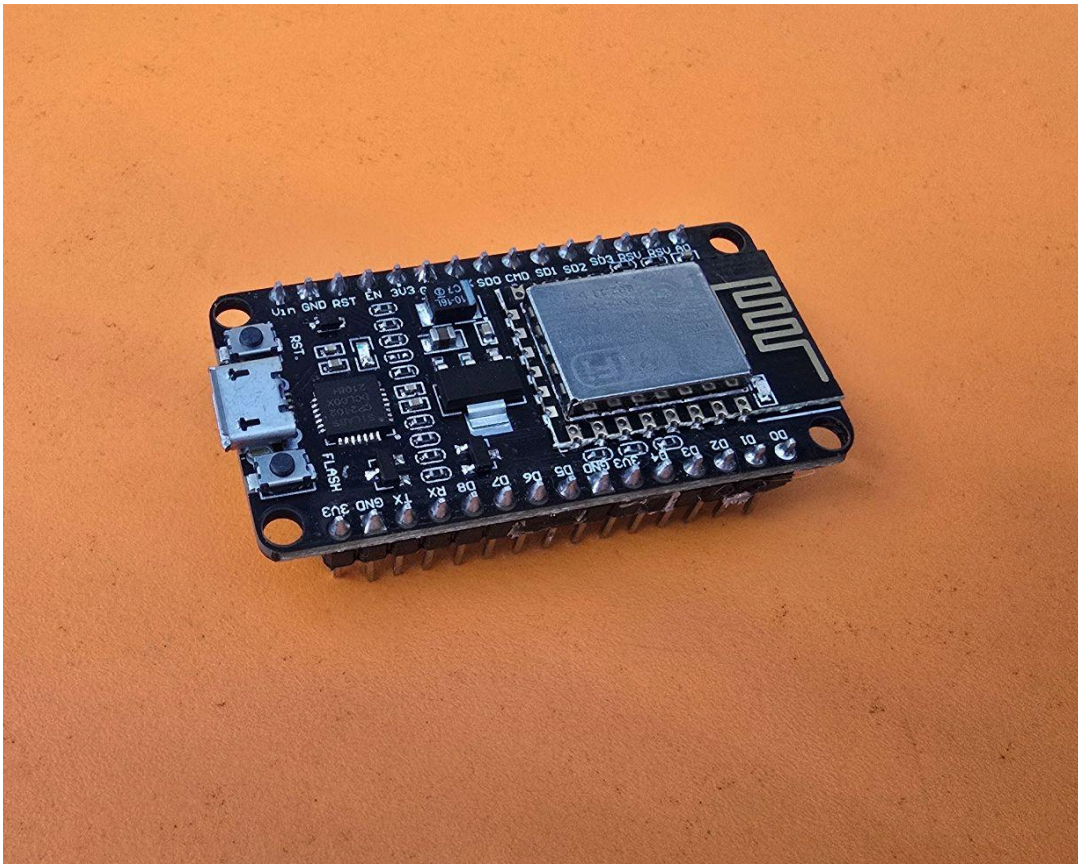
Summary

- The script creates a GUI with buttons that each send a message to an MQTT broker when pressed.
- Images are loaded and displayed on each button.
- The MQTT connection status is displayed in the GUI, and the MQTT client runs in the background while the GUI is active.

TASK 2 - Connection between phone and ESP8266

Phone configuration

Subscriber Mqtt




```
#include <ESP8266WiFi.h>  // For ESP8266 boards
#include <PubSubClient.h>
#include <ArduinoJson.h>  // For parsing JSON messages

const char* ssid = "BogdanS24";  // Your Wi-Fi network
const char* password = "12345678";  // Wi-Fi password
const char* mqtt_server = "broker.emqx.io";
const int mqtt_port = 1883;  // Use 1883 for non-secure connections
const char* topic = "xarm/variables";  // The topic to subscribe to

WiFiClient net;
PubSubClient client(net);

unsigned long previousMillis = 0; // To track time for blinking
unsigned long blinkInterval = 0;   // Interval for blinking the LED
int blinkCount = 0;  // To track how many times to blink
bool ledState = LOW;
bool isBlinking = false;  // Flag to track if blinking is happening

// Message buffer (simple queue)
```

```
#define MAX_QUEUE_SIZE 5
struct Message {
  String variable1;
  int variable2;
};
Message messageQueue[MAX_QUEUE_SIZE];  // Buffer to store incoming
messages
int queueStart = 0, queueEnd = 0;      // Pointers to manage the queue

void messageReceived(char* topic, byte* payload, unsigned int length) {
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, payload, length);

  if (error) {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
    return;
  }

  String variable1 = doc["variable1"];  // Get 'a' or 'r'
  int variable2 = doc["variable2"];  // Get number of blinks

  Serial.println("Message received:");
  Serial.print("variable1: ");
  Serial.println(variable1);
  Serial.print("variable2: ");
  Serial.println(variable2);

  // Add the message to the queue if space is available
  if ((queueEnd + 1) % MAX_QUEUE_SIZE != queueStart) {
    messageQueue[queueEnd].variable1 = variable1;
    messageQueue[queueEnd].variable2 = variable2;
    queueEnd = (queueEnd + 1) % MAX_QUEUE_SIZE;
  } else {
    Serial.println("Message queue is full! Ignoring new message.");
  }
}

void processMessageQueue() {
  // If not currently blinking and the queue has messages, process the
next one
  if (!isBlinking && queueStart != queueEnd) {
    Message msg = messageQueue[queueStart];
```

```
    queueStart = (queueStart + 1) % MAX_QUEUE_SIZE;

    // Set the blink interval based on variable1
    if (msg.variable1 == "a") {
      blinkInterval = 1000;  // 1 second for 'a'
    } else if (msg.variable1 == "r") {
      blinkInterval = 2000;  // 2 seconds for 'r'
    }

    blinkCount = msg.variable2;  // Set the number of blinks
    isBlinking = true;  // Set the flag to indicate that blinking is in
progress
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);  // Onboard LED as output

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Configure MQTT client
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(messageReceived);

  // Connect to the MQTT broker
  reconnectMQTT();
  client.subscribe(topic);  // Subscribe after connecting
}

void loop() {
  if (!client.connected()) {
    reconnectMQTT();
  }
  client.loop();
```

```
  // Process the next message from the queue if blinking is not in
progress
  processMessageQueue();

  // Handle the blinking process
  if (isBlinking) {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= blinkInterval) {
      previousMillis = currentMillis;

      // Toggle LED state
      ledState = !ledState;
      digitalWrite(LED_BUILTIN, ledState);

      // If LED is off, reduce the blink count (we count one complete
cycle)
      if (ledState == LOW) {
        blinkCount--;
      }

      // If the blink count reaches 0, stop blinking
      if (blinkCount <= 0) {
        isBlinking = false;
        Serial.println("Blinking finished.");
      }
    }
  }
}

void reconnectMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("arduinoClient")) {
      Serial.println("Connected to MQTT");
      client.subscribe(topic);  // Re-subscribe after connecting
    } else {
      Serial.print("Failed to connect, rc=");
      Serial.print(client.state());
      Serial.println(" Trying again in 5 seconds...");
      delay(5000);
    }
  }
}
```

Explained

This code is designed to run on an ESP8266 microcontroller, which connects to an MQTT broker and handles JSON messages received over the MQTT protocol. The messages instruct the device to blink an onboard LED with specific intervals and durations. Here's a breakdown of the code:

1. Includes and Constants

cpp
Copy code

```
#include <ESP8266WiFi.h>  // For ESP8266 boards
#include <PubSubClient.h>  // For MQTT functionality
#include <ArduinoJson.h>   // For parsing JSON messages
```

- **`ESP8266WiFi.h`**: Allows the ESP8266 to connect to Wi-Fi.
- **`PubSubClient.h`**: A library for MQTT client functionality.
- **`ArduinoJson.h`**: Used to deserialize JSON-formatted payloads.

cpp
Copy code

```
const char* ssid = "BogdanS24";  // Your Wi-Fi network
const char* password = "12345678";  // Wi-Fi password
const char* mqtt_server = "broker.emqx.io";
const int mqtt_port = 1883;
const char* topic = "xarm/variables";  // The topic to subscribe to
```

- **Wi-Fi credentials**: `ssid` and `password` represent the Wi-Fi network details.
- **`mqtt_server`**, **`mqtt_port`**: The MQTT broker address and port for non-secure connections (port 1883).
- **`topic`**: The MQTT topic on which messages are received.

2. Wi-Fi and MQTT Setup

cpp
Copy code

```
WiFiClient net;
PubSubClient client(net);
```

- **`WiFiClient net`**: Used to manage the Wi-Fi connection.
- **`PubSubClient client`**: Handles the MQTT communication.

3. Message Queue and Blinking Variables

cpp
Copy code

```
unsigned long previousMillis = 0;  // Tracks time for blinking
unsigned long blinkInterval = 0;   // Interval for blinking the LED
int blinkCount = 0;                // Number of blinks
bool ledState = LOW;               // LED state (on or off)
bool isBlinking = false;           // Flag to indicate if blinking
is happening
```

- These variables track the current state of the LED (on or off), manage blinking intervals, and control how many times the LED should blink.

cpp
Copy code

```
#define MAX_QUEUE_SIZE 5
struct Message {
  String variable1;
  int variable2;
};
Message messageQueue[MAX_QUEUE_SIZE];
int queueStart = 0, queueEnd = 0;
```

- **Message Queue**: A simple queue to store up to 5 messages received from the MQTT broker. Each message contains two variables: `variable1` (string) and `variable2` (integer).

4. Handling Incoming MQTT Messages

cpp
Copy code

```
void messageReceived(char* topic, byte* payload, unsigned int
length) {
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, payload,
length);

  if (error) {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
    return;
  }
```

```
  String variable1 = doc["variable1"];
  int variable2 = doc["variable2"];
```

- **`messageReceived`**: Callback function that is invoked whenever a new MQTT message is received. It processes the JSON payload, extracting two values:
  - **`variable1`**: Controls the blinking interval (`"a"` or `"r"`).
  - **`variable2`**: The number of times the LED should blink.

cpp
Copy code

```
 if ((queueEnd + 1) % MAX_QUEUE_SIZE != queueStart) {
    messageQueue[queueEnd].variable1 = variable1;
    messageQueue[queueEnd].variable2 = variable2;
    queueEnd = (queueEnd + 1) % MAX_QUEUE_SIZE;
  } else {
    Serial.println("Message queue is full! Ignoring new message.");
  }
```

- Adds the message to the queue if there's space. The queue works in a circular buffer style.

5. Processing the Message Queue

cpp
Copy code

```
void processMessageQueue() {
  if (!isBlinking && queueStart != queueEnd) {
    Message msg = messageQueue[queueStart];
    queueStart = (queueStart + 1) % MAX_QUEUE_SIZE;

    if (msg.variable1 == "a") {
      blinkInterval = 1000;
    } else if (msg.variable1 == "r") {
      blinkInterval = 2000;
    }

    blinkCount = msg.variable2;
    isBlinking = true;
  }
}
```

- **`processMessageQueue()`**: Dequeues messages from the queue and configures the blink interval and the number of blinks. It checks if blinking is not already in progress.

6. Setup Function

cpp
Copy code

```
void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(messageReceived);
  reconnectMQTT();
  client.subscribe(topic);
}
```

- **`setup()`**: Initializes the ESP8266:
  - Connects to Wi-Fi.
  - Sets up the MQTT client and its callback function (`messageReceived`).
  - Subscribes to the MQTT topic after connecting to the broker.

7. Main Loop

cpp
Copy code

```
void loop() {
  if (!client.connected()) {
    reconnectMQTT();
  }
  client.loop();
  processMessageQueue();

  if (isBlinking) {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= blinkInterval) {
```

```
      previousMillis = currentMillis;
      ledState = !ledState;
      digitalWrite(LED_BUILTIN, ledState);

      if (ledState == LOW) {
        blinkCount--;
      }

      if (blinkCount <= 0) {
        isBlinking = false;
        Serial.println("Blinking finished.");
      }
    }
  }
}
```

- `loop()`:
  - Ensures the MQTT client stays connected and processes incoming messages.
  - If there's a message in the queue and blinking is not in progress, it starts the blinking sequence.
  - Controls the LED blinking based on the current time (`millis()`), toggling the LED state and reducing the blink count with each cycle.

8. Reconnect MQTT Function

cpp
Copy code

```
void reconnectMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("arduinoClient")) {
      Serial.println("Connected to MQTT");
      client.subscribe(topic);
    } else {
      Serial.print("Failed to connect, rc=");
      Serial.print(client.state());
      Serial.println(" Trying again in 5 seconds...");
      delay(5000);
    }
  }
}
```

- `reconnectMQTT()`: Ensures the ESP8266 is connected to the MQTT broker. If the connection is lost, it attempts to reconnect every 5 seconds.

Summary

- The ESP8266 connects to an MQTT broker and subscribes to a topic.
- JSON-formatted MQTT messages control the blinking behavior of the onboard LED.
- The message specifies:
    - `variable1`: Determines the blink interval ("a" for 1 second, "r" for 2 seconds).
    - `variable2`: The number of times the LED should blink.
- Messages are processed in a queue to handle multiple commands sequentially.