

Desarrollo de un control de motores de continua con un joystick mediante el protocolo mqtt



CIFO La Violeta
Centro de Innovación y Formación Ocupacional

Trabajo realizado por:
Yago Granados Gomes

Dirigido por:
Joan Masdemont Fontàs

Creación de prototipos de IoT con Raspberry

Barcelona, 02/11/2023

Contenido

1	Introducción	3
2	Creación de un Access Point con el ESP32	4
3.	Instalación del programa Mosquitto con identificación	10
4.	Optimización del Access Point.....	12
5.	Publicación y suscripción de datos.....	13
6.	Código e instalación para los motores en DC.....	17
7.	Código e instalación del joystick	20
8.	Códigos unificados.....	22
9.	Conclusiones.....	28
10.	Web gráfica.....	29

1 Introducción

El presente proyecto consiste en el diseño de un vehículo a radio control que pueda ser operado en situaciones con visibilidad reducida. Por lo tanto será necesario el uso de diferentes sensores y actuadores para que el operador pueda decidir y conocer la situación en cada momento del automóvil. En un principio, será necesario el uso de palancas y motores para el control y movilización del artefacto respectivamente.

Afortunadamente, el CIFO¹ La Violeta cuenta con todo lo necesario para poder hacer este proyecto realidad. Es decir, se eliminan los problemas económicos y temporales permitiendo centrarse únicamente en la realización y memoria del proyecto.

¹ Centros de Innovación y Formación Ocupacional

2 Creación de un Access Point con el ESP32

La mejor opción para establecer una conexión sin hilos entre un ESP32 y un Raspberry Pi 3 Model V1.2 ha sido crear un Access Point en el microcontrolador. Sin embargo, antes de poner en marcha cualquier programa es necesario instalar el ESP32 en el programa Arduino IDE. El primer paso es abrir la pestaña File → Preferences y copiar en el apartado Additional boards manager URLs el siguiente enlace:

- https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

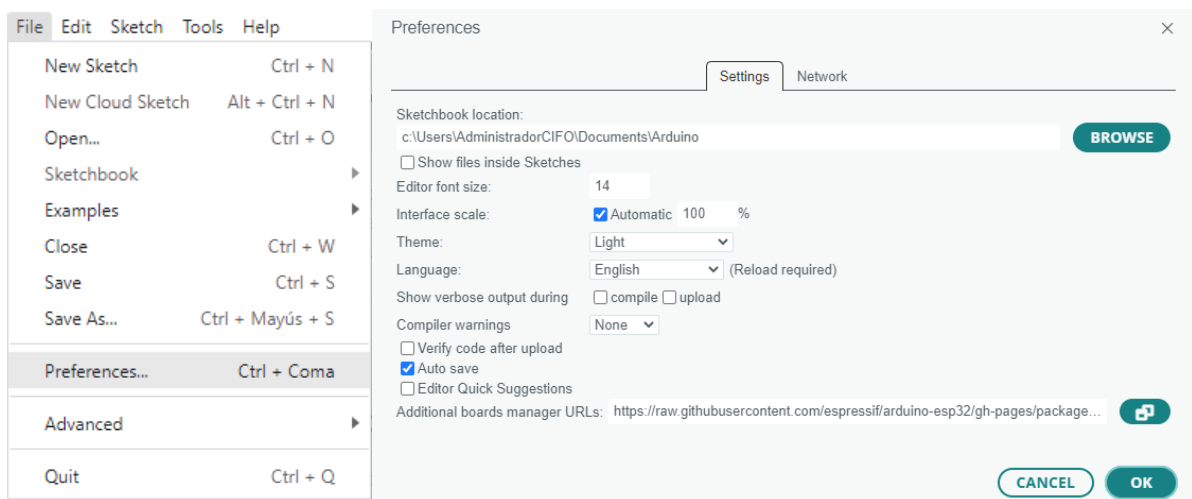


Ilustración 1: Instalación ESP32

El siguiente paso es abrir la pestaña Tools → Board: → Boards Manager... y escribir esp32 en el buscador y descargar la librería esp32 by Espressif Systems.

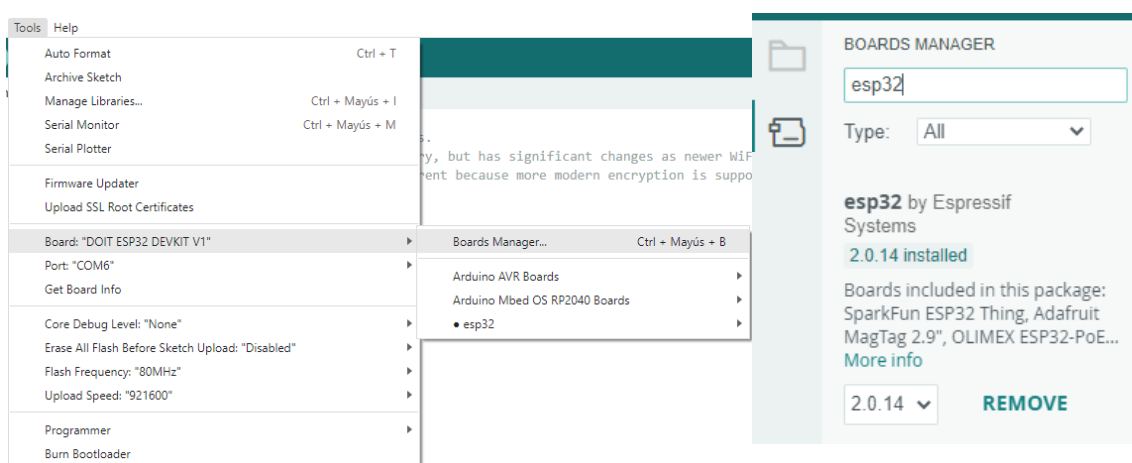


Ilustración 2: Descarga de librería esp32

Una vez instalado el programa hay que seleccionar el dispositivo en el programa, para hacer esto hay que seleccionar Tools → Board: → esp32 → DOIT ESP32 DEVKIT V1. No obstante, hay escoger un puerto por el cual se cargan los programas yendo a Tools → Port: → COMX.

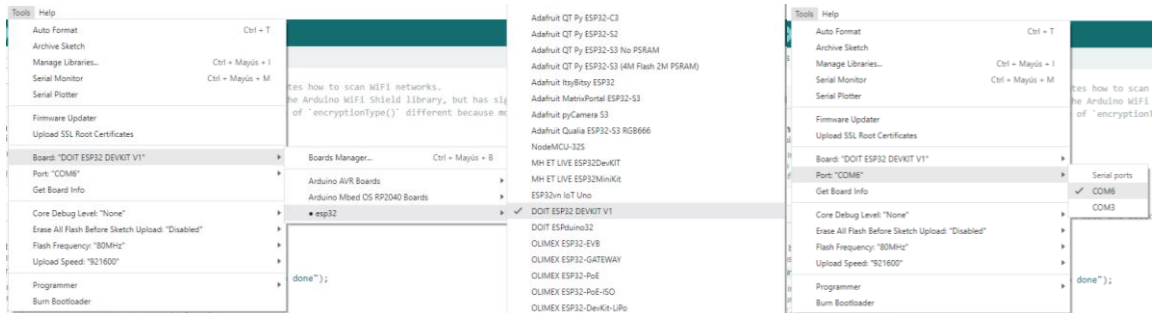


Ilustración 3: Asignación de ESP32 y puerto de comunicación

A continuación se copia el programa, modificamos nombre y contraseña y se pone en marcha. Entonces buscamos con el teléfono inteligente la red para comprobar que se ha creado correctamente.

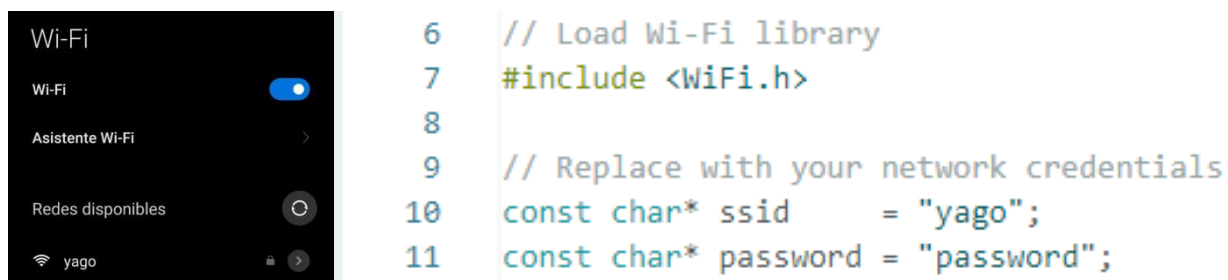


Ilustración 4: Creación de Access Point

El programa utilizado se muestra a continuación:

```

/*****
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
*****/

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "ESP32-Access-Point";
const char* password = "123456789";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request

```

```
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output26, OUTPUT);
  pinMode(output27, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output26, LOW);
  digitalWrite(output27, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Setting AP (Access Point)...");
  // Remove the password parameter, if you want the AP (Access Point) to be open
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.begin();
}

void loop(){
  WiFiClient client = server.available(); // Listen for incoming clients

  if (client) { // If a new client connects,
    Serial.println("New Client."); // print a message out in the serial port
    String currentLine = ""; // make a String to hold incoming data from the client
    while (client.connected()) { // loop while the client's connected
      if (client.available()) { // if there's bytes to read from the client,
        char c = client.read(); // read a byte, then
        Serial.write(c); // print it out the serial monitor
        header += c;
        if (c == '\n') { // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
```

```

client.println();
// turns the GPIOs on and off
if (header.indexOf("GET /26/on") >= 0) {
  Serial.println("GPIO 26 on");
  output26State = "on";
  digitalWrite(output26, HIGH);
} else if (header.indexOf("GET /26/off") >= 0) {
  Serial.println("GPIO 26 off");
  output26State = "off";
  digitalWrite(output26, LOW);
} else if (header.indexOf("GET /27/on") >= 0) {
  Serial.println("GPIO 27 on");
  output27State = "on";
  digitalWrite(output27, HIGH);
} else if (header.indexOf("GET /27/off") >= 0) {
  Serial.println("GPIO 27 off");
  output27State = "off";
  digitalWrite(output27, LOW);
}
// Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">>");
// CSS to style the on/off buttons
// Feel free to change the background-color and font-size attributes to fit
your preferences
client.println("<style>html { font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;});");
client.println(".button { background-color: #4CAF50; border: none; color:
white; padding: 16px 40px;});");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;});");
client.println(".button2 {background-color: #555555;}</style></head>");

// Web Page Heading
client.println("<body><h1>ESP32 Web Server</h1>");

// Display current state, and ON/OFF buttons for GPIO 26
client.println("<p>GPIO 26 - State " + output26State + "</p>");
// If the output26State is off, it displays the ON button
if (output26State=="off") {
  client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
}

// Display current state, and ON/OFF buttons for GPIO 27
client.println("<p>GPIO 27 - State " + output27State + "</p>");

```

```
// If the output27State is off, it displays the ON button
if (output27State=="off") {
    client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
}
client.println("</body></html>");

// The HTTP response ends with another blank line
client.println();
// Break out of the while loop
break;
} else { // if you got a newline, then clear currentLine
    currentLine = "";
}
} else if (c != '\r') { // if you got anything else but a carriage return
character,
    currentLine += c; // add it to the end of the currentLine
}
}
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}
```

Posteriormente se tiene que modificar un archivo de texto plano para la configuración llamado `dhcpcd.conf` dentro en el directorio `etc`. Para acceder a este archivo se realiza la siguiente ruta:

- 1 `pwd`
- 2 `cd etc`
- 3 `sudo su`
- 4 `nano dhcpcd.conf`

Se añaden las siguientes líneas en el archivo y se reinicia el Raspberry Pi 3 Model V1.2 con el comando `reboot`.

- `interface wlan0`
- `static ip_address=192.168.4.2/24`


```
# Example static IP configuration:
interface wlan0
static ip_address=192.168.4.2/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1

interface eth0
static ip_address=10.199.160.162/24
static routers=10.199.160.254
static domain_name_servers=8.8.8.8 8.8.4.4
```

Ilustración 5: Modificación de características

Para finalizar se hace la función ping en el microordenador mientras el microcontrolador está conectado y con el programa en marcha para comprobar que hay una comunicación Wi-Fi entre los dos dispositivos.

```
pi@raspberrypi:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=15.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=11.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=20.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=12.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=114 time=20.8 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=114 time=12.2 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=114 time=11.2 ms
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 11.176/14.756/20.773/3.841 ms
pi@raspberrypi:~$ ping 10.199.160.162
PING 10.199.160.162 (10.199.160.162) 56(84) bytes of data.
64 bytes from 10.199.160.162: icmp_seq=1 ttl=64 time=0.137 ms
64 bytes from 10.199.160.162: icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from 10.199.160.162: icmp_seq=3 ttl=64 time=0.078 ms
pi@raspberrypi:~$ ping 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=255 time=199 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=255 time=9.95 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=255 time=11.6 ms
64 bytes from 192.168.4.1: icmp_seq=4 ttl=255 time=15.0 ms
64 bytes from 192.168.4.1: icmp_seq=5 ttl=255 time=16.2 ms
^C
--- 192.168.4.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 9.948/50.392/199.322/74.498 ms
pi@raspberrypi:~$ ping 192.168.4.2
PING 192.168.4.2 (192.168.4.2) 56(84) bytes of data.
64 bytes from 192.168.4.2: icmp_seq=1 ttl=64 time=0.128 ms
64 bytes from 192.168.4.2: icmp_seq=2 ttl=64 time=0.098 ms
64 bytes from 192.168.4.2: icmp_seq=3 ttl=64 time=0.097 ms
64 bytes from 192.168.4.2: icmp_seq=4 ttl=64 time=0.093 ms
64 bytes from 192.168.4.2: icmp_seq=5 ttl=64 time=0.093 ms
```

Ilustración 6: Ping a diferentes ip

3. Instalación del programa Mosquitto con identificación

En este apartado se muestra como instalar el programa Mosquitto en el Raspberry Pi 3 Model V1.2 y se hace con los siguientes cuatro comandos:

- `sudo apt update`
- `sudo apt install -y mosquitto mosquitto-clients`
- `sudo systemctl enable mosquitto.service`
- `mosquitto -v`

```
pi@raspberrypi:~$ sudo apt update
Get:1 http://rasbian.raspberrypi.org/raspbian bullseye InRelease [15.0 kB]
Get:2 https://deb.nodesource.com/node_18.x nodistro InRelease [12.1 kB]
Get:3 http://rasbian.raspberrypi.org/raspbian bullseye/main armhf Packages [13.2 MB]
Get:4 https://deb.nodesource.com/node_18.x nodistro/main armhf Packages [6,155 B]
Get:5 http://archive.raspberrypi.org/debian bullseye InRelease [23.6 kB]
Get:6 http://archive.raspberrypi.org/debian bullseye/main armhf Packages [313 kB]
Fetched 13.6 MB in 15s (829 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
70 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi@raspberrypi:~$ sudo apt install -y mosquitto mosquitto-clients
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mosquitto is already the newest version (2.0.11-1+deb11u1).
mosquitto-clients is already the newest version (2.0.11-1+deb11u1).
0 upgraded, 0 newly installed, 0 to remove and 70 not upgraded.
pi@raspberrypi:~$ sudo systemctl enable mosquitto.service
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
pi@raspberrypi:~$ mosquitto -v
1699271931: mosquitto version 2.0.11 starting
1699271931: Using default config.
1699271931: Starting in local only mode. Connections will only be possible from clients running on this machine.
1699271931: Create a configuration file which defines a listener to allow remote access.
1699271931: For more details see https://mosquitto.org/documentation/authentication-methods/
1699271931: Opening ipv4 listen socket on port 1883.
1699271931: Opening ipv6 listen socket on port 1883.
1699271931: Error: Address already in use
1699271931: mosquitto version 2.0.11 running
^C1699271949: mosquitto version 2.0.11 terminating
pi@raspberrypi:~$
```

Ilustración 7: Instalación mosquitto

En este momento se establece el usuario pi con credencial raspberry con los siguientes comandos.

- `sudo mosquitto_passwd -c /etc/mosquitto/passwd pi`
- `sudo nano /etc/mosquitto/mosquitto.conf`
- `sudo systemctl restart mosquitto`
- `sudo systemctl status mosquitto`

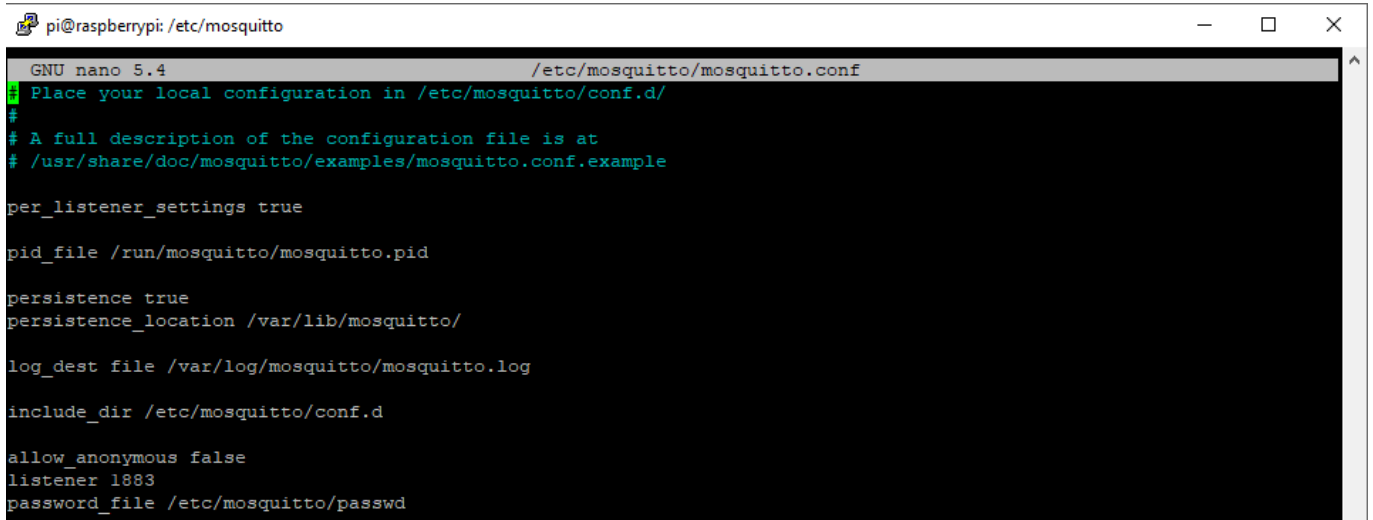
Cabe resaltar que el segundo comando modifica un archivo plano de texto para la configuración en que hay escribir tres líneas en la posición que se muestra en la ilustración 9.

- `per_listener_settings true`
- `allow_anonymous false`
- `listener 1883`
- `passwd_file /etc/mosquitto/passwd`

```
pi@raspberrypi:/etc/mosquitto $ sudo nano passwd
pi@raspberrypi:/etc/mosquitto $ sudo mosquitto_passwd -c /etc/mosquitto/passwd pi
Password:
Reenter password:
pi@raspberrypi:/etc/mosquitto $ sudo nano /etc/mosquitto/mosquitto.conf
pi@raspberrypi:/etc/mosquitto $ sudo systemctl restart mosquitto
pi@raspberrypi:/etc/mosquitto $ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-11-06 13:12:35 CET; 6s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 1098 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 1100 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 1101 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 1102 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
 Main PID: 1103 (mosquitto)
    Tasks: 1 (limit: 1595)
       CPU: 55ms
    CGroup: /system.slice/mosquitto.service
            └─1103 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Nov 06 13:12:35 raspberrypi systemd[1]: Starting Mosquitto MQTT Broker...
Nov 06 13:12:35 raspberrypi systemd[1]: Started Mosquitto MQTT Broker.
```

Ilustración 8: Status de mosquitto



The screenshot shows a terminal window with the nano text editor open to the file /etc/mosquitto/mosquitto.conf. The editor's title bar shows 'pi@raspberrypi: /etc/mosquitto'. The file content includes comments about configuration and a list of settings. The settings shown are: per_listener_settings true, pid_file /run/mosquitto/mosquitto.pid, persistence true, persistence_location /var/lib/mosquitto/, log_dest file /var/log/mosquitto/mosquitto.log, include_dir /etc/mosquitto/conf.d, allow_anonymous false, listener 1883, and password_file /etc/mosquitto/passwd.

```
GNU nano 5.4 /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

per_listener_settings true

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

Ilustración 9: Archivo mosquitto.conf

4. Optimización del Access Point

Para poder seguir trabajando en este código es necesario reducir el número de líneas porque no se utilizan. El nuevo código para crear el Access Point queda tal que así:

```
/******  
Rui Santos  
Complete project details at https://randomnerdtutorials.com  
*****/  
  
// Load Wi-Fi library  
#include <WiFi.h>  
  
// Replace with your network credentials  
const char* ssid = "yago";  
const char* password = "password";  
  
void setup() {  
    Serial.begin(115200);  
  
    // Connect to Wi-Fi network with SSID and password  
    Serial.print("Setting AP (Access Point)...");  
    // Remove the password parameter, if you want the AP (Access Point) to be open  
    WiFi.softAP(ssid, password);  
  
    IPAddress IP = WiFi.softAPIP();  
    Serial.print("AP IP address: ");  
    Serial.println(IP);  
  
}  
  
void loop(){  
}
```

5. Publicación y suscripción de datos

Para poder conectar el Raspberry Pi 3 Model V1.2 al ESP32 primero hay que instalar paho-mqtt con el comando `pip3 install paho-mqtt`. Después hay que escribir el código debajo de la ilustración en el microcontrolador.

```
pi@raspberrypi:~ $ pip3 install paho-mqtt
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: paho-mqtt in ./local/lib/python3.9/site-packages (1.6.1)
pi@raspberrypi:~ $
```

Ilustración 10: Instalación de paho-mqtt

```
import random
import time

from paho.mqtt import client as mqtt_client

broker = '192.168.4.2'
port = 1883
topic = "prueba"
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 1000)}'
username = 'pi'
password = 'raspberrypi'

def connect_mqtt():
    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)

    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client

def publish(client):
    msg_count = 0
    while True:
        time.sleep(1)
        msg = f"messages: {msg_count}"
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
```

```
    print(f"Send `{msg}` to topic `{topic}`")
else:
    print(f"Failed to send message to topic {topic}")
msg_count += 1

def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        while True:
            print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
            time.sleep(1)
    client.subscribe(topic)
    client.on_message = on_message

def run():
    client = connect_mqtt()
    publish(client)
    client.loop_forever()

if __name__ == '__main__':
    run()
```

También hay que escribir el siguiente código en el programa en Arduino IDE para introducirlo en el ESP32.

```
/*
*****
Rui Santos
Complete project details at https://randomnerdtutorials.com
*****/

// Load Wi-Fi library
#include <WiFi.h>
#include <PubSubClient.h>

// Replace with your network credentials
const char* ssid = "yago";
const char* password = "password";

// MQTT Broker
const char *mqtt_broker = "192.168.4.2";
const char *topic = "prueba";
const char *mqtt_username = "pi";
const char *mqtt_password = "raspberry";
const int mqtt_port = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
```

```

Serial.begin(115200);

// Connect to Wi-Fi network with SSID and password
Serial.print("Setting AP (Access Point)...");
// Remove the password parameter, if you want the AP (Access Point) to be open
WiFi.softAP(ssid, password);

IPAddress IP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(IP);

client.setServer(mqtt_broker, mqtt_port);
client.setCallback(callback);
while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public MQTT broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
        Serial.println("Public EMQX MQTT broker connected");
    } else {
        Serial.print("failed with state ");
        Serial.print(client.state());
        delay(2000);
    }
}

// Publish and subscribe
client.publish(topic, "Hi, I'm ESP32 ^^");
client.subscribe(topic);
}

void callback(char *topic, byte *payload, unsigned int length) {
    Serial.print("Message arrived in topic: ");
    Serial.println(topic);
    Serial.print("Message:");
    for (int i = 0; i < length; i++) {
        Serial.print((char) payload[i]);
    }
    Serial.println();
    Serial.println("-----");
}

void loop(){
    client.subscribe(topic);
    delay(50);
    //Serial.println(topic);
    /***
    client.publish(topic, "Hi, I'm ESP32 !!");
    delay(1000);
    client.publish(topic, "Say, It again!!!");

```

```
    delay(1000);****/  
    client.loop();  
}
```

Una vez copiado y con los dos programas en marcha se imprime en el ESP32 un mensaje estableciendo una conexión sin hilos entre los dos dispositivos.

6. Código e instalación para los motores en DC

A continuación se muestra en código utilizado para mover los motores hacia derecha, izquierda, adelante y atrás aislado del código para crear el Access Point. Hay que destacar que se instala en con el ESP32 y es necesario un chip L293D para el control de los motores en diferentes direcciones.

```
#include <Arduino.h> // Incluye la biblioteca de Arduino

// Definición de los pines conectados al L293
const int IN11 = 27; // Pin de control 1 (Input 1)
const int IN12 = 12;
const int EN1 = 14; // Pin de control 2 (Input 2)
const int IN21 = 25; // Pin de control 1 (Input 1)
const int IN22 = 33;
const int EN2 = 26;

void setup() {
  // Configura los pines como salidas
  Serial.begin(9600);
  pinMode(IN11, OUTPUT);
  pinMode(IN12, OUTPUT);
  pinMode(EN1, OUTPUT);
  pinMode(IN21, OUTPUT);
  pinMode(IN22, OUTPUT);
  pinMode(EN2, OUTPUT);
}

void loop() {
  // Gira el motor en una dirección
  Serial.println("arranca");
  digitalWrite(IN11, HIGH);
  digitalWrite(IN12, LOW);
  digitalWrite(EN1, HIGH);
  digitalWrite(IN21, HIGH);
  digitalWrite(IN22, LOW);
  digitalWrite(EN2, HIGH);

  // Puedes agregar un retraso para que el motor se mueva durante un tiempo determinado
  delay(2000); // Motor gira durante 2 segundos

  // Detiene el motor
  digitalWrite(EN1, LOW);
  digitalWrite(EN2, LOW);

  // Espera antes de cambiar de dirección
  delay(1000); // Espera 1 segundo

  // Gira el motor derecha
```

```
Serial.println("derecha");
digitalWrite(IN11, HIGH);
digitalWrite(IN12, LOW);
digitalWrite(EN1, HIGH);
digitalWrite(IN21, LOW);
digitalWrite(IN22, HIGH);
digitalWrite(EN2, HIGH);

// Puedes agregar un retraso para que el motor se mueva durante un tiempo determinado
delay(2000); // Motor gira durante 2 segundos

// Detiene el motor
digitalWrite(EN1, LOW);
digitalWrite(EN2, LOW);

// Espera antes de cambiar de dirección
delay(1000); // Espera 1 segundo

// Gira el motor izquierda
Serial.println("arranca");
digitalWrite(IN11, LOW);
digitalWrite(IN12, HIGH);
digitalWrite(EN1, HIGH);
digitalWrite(IN21, HIGH);
digitalWrite(IN22, LOW);
digitalWrite(EN2, HIGH);

// Puedes agregar un retraso para que el motor se mueva durante un tiempo determinado
delay(2000); // Motor gira durante 2 segundos

// Detiene el motor
digitalWrite(EN1, LOW);
digitalWrite(EN2, LOW);

// Espera antes de cambiar de dirección
delay(1000); // Espera 1 segundo

// Gira el motor en la dirección opuesta
digitalWrite(IN11, LOW);
digitalWrite(IN12, HIGH);
digitalWrite(EN1, HIGH);
digitalWrite(IN21, LOW);
digitalWrite(IN22, HIGH);
digitalWrite(EN2, HIGH);

// Puedes agregar un retraso nuevamente
delay(2000); // Motor gira durante 2 segundos en la dirección opuesta

// Detiene el motor
digitalWrite(EN1, LOW);
digitalWrite(EN2, LOW);
```

```
// Espera antes de volver a iniciar el ciclo  
delay(1000); // Espera 1 segundo antes de volver a girar  
}
```

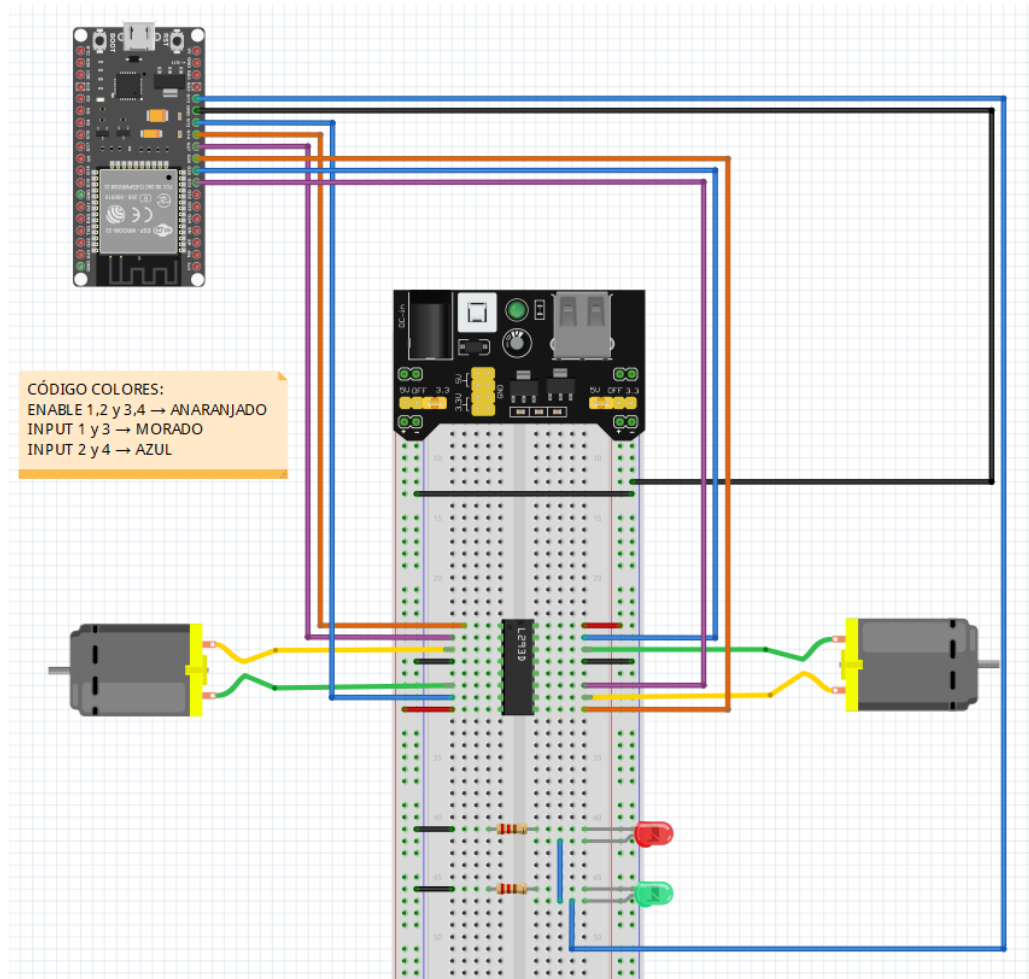


Ilustración 11: Esquema eléctrico ESP32

7. Código e instalación del joystick

En este punto se muestra el código utilizado para la lectura del eje de coordenadas del joystick que se simplificará en los siguientes cuatro movimientos.

- Arriba → Adelante
- Abajo → Atrás
- Derecha → Derecha
- Izquierda → Izquierda

En este caso, el Raspberry Pi 3 Model V1.2 solo cuenta con pines de lectura digital y el joystick emite salidas analógicas por lo que hará falta un chip ADC², más concretamente, un mcp3008.

```
import RPi.GPIO as GPIO
import time
#import threading

pin_eje_x = 21
pin_eje_y = 20
pin_sw_j = 16

def setup():

    GPIO.setmode(GPIO.BCM)
    GPIO.setup([pin_eje_x, pin_eje_y, pin_sw_j],GPIO.IN)

def lectura():
    #print(f"A continuación se muestra la lectura del joystick: \n")
    #print(f"El eje x tiene un valor de: {pin.read(pin_eje_x)}\n")
    #print(f"El eje y tiene un valor de: {pin.read(pin_eje_y)}\n")
    valor_x = GPIO.input(pin_eje_x)
    valor_y = GPIO.input(pin_eje_y)

    print(valor_x)
    print(valor_y)

def main():
    setup()
    while True:
        lectura()
        time.sleep(1)

main()
```

² Convertidor Analógico Digital

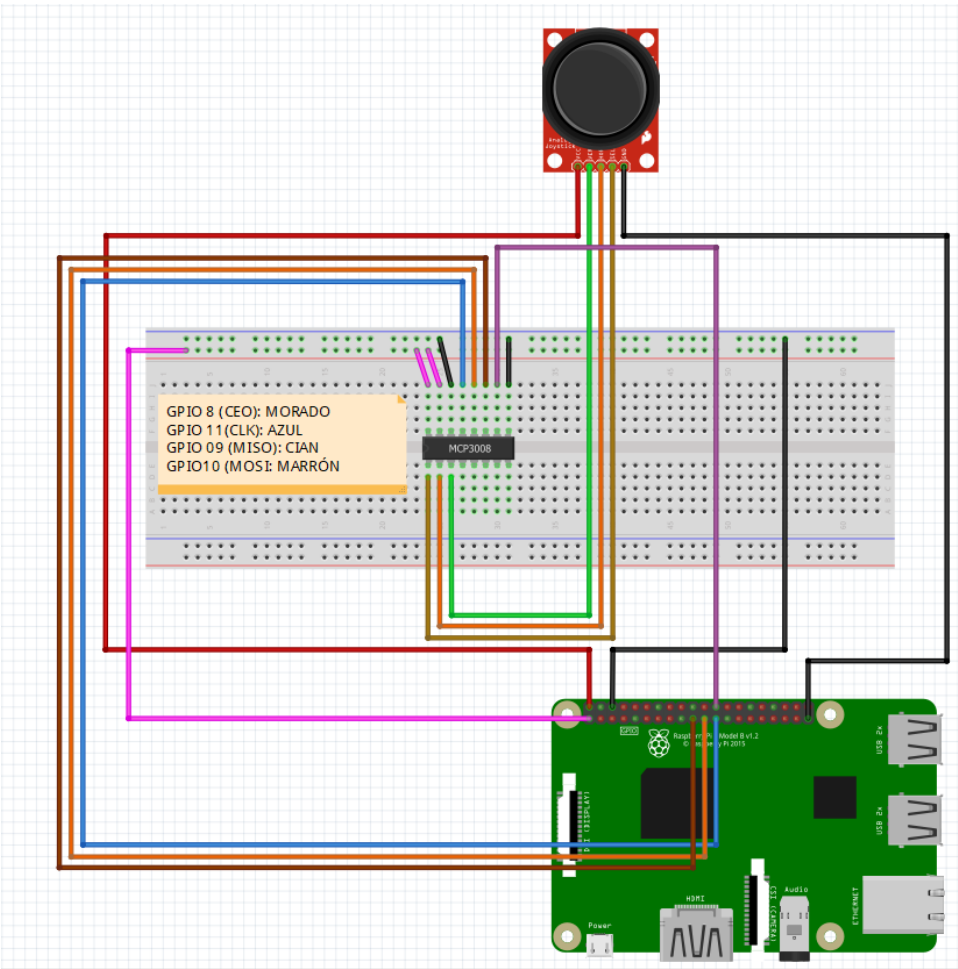


Ilustración 12: Esquema eléctrico Raspberry

8. Códigos unificados

En primer lugar se muestra el código final del ESP32 unificando todas las funcionalidades desarrolladas en este proyecto.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <string.h>
#include <Arduino.h>
#include <TaskScheduler.h>

Scheduler runner;

// Replace with your network credentials
const char* ssid      = "yago";
const char* password = "password";

String myString;

// MQTT Broker
const char *mqtt_broker = "192.168.4.2";
const char *topic = "prueba";
const char *mqtt_username = "pi";
const char *mqtt_password = "raspberry";
const int mqtt_port = 1883;

// Motores definición de los pines conectados al L293
const int IN11 = 27;
const int IN12 = 12;
const int EN1 = 14;
const int IN21 = 25;
const int IN22 = 33;
const int EN2 = 26;

const int LED = 13;

int x;
int y;

WiFiClient espClient;
PubSubClient client(espClient);

void callback(char *topic, byte *payload, unsigned int length) {
    myString = "";
    for (int i = 0; i < length; i++) {
        myString += (char) payload[i];
    }
}
```

```

    }

}

Task receptor_mqtt(0, TASK_FOREVER, []() {
client.subscribe(topic);

    char *token;
    int values[3];
    int index = 0;

    char lista[50];
    myString.toCharArray(lista, 50);

    token = strtok(lista, ",");
    while (token != NULL && index < 3) {
values[index] = atoi(token); // Convert the token to an integer
token = strtok(NULL, ","); // Get the next token
        index++;
    }
    x = values[0];
    y = values[1];
    client.loop();

});

```

```

Task direccion_motor(0, TASK_FOREVER, []() {
    if (x >= 600 && x <= 900){
        if ( y >= 600 && y <= 900 ){
            Serial.println("para");
            digitalWrite(EN1, LOW);
            digitalWrite(EN2, LOW);
        }
        else if ( y >= 0 && y <= 599 ){
            Serial.println("izquierda");
            digitalWrite(IN11, LOW);
            digitalWrite(IN12, HIGH);
            digitalWrite(EN1, HIGH);
            digitalWrite(IN21, HIGH);
            digitalWrite(IN22, LOW);
            digitalWrite(EN2, HIGH);
        }
        else if ( y >= 901 && y <= 1025 ){
            Serial.println("derecha");
            digitalWrite(IN11, HIGH);
            digitalWrite(IN12, LOW);
            digitalWrite(EN1, HIGH);
            digitalWrite(IN21, LOW);
            digitalWrite(IN22, HIGH);
        }
    }
}

```

```
        digitalWrite(EN2, HIGH);
    }
}

else {
    if ( x >= 901 && x <= 1025 ){
        Serial.println("arranca");
        digitalWrite(IN11, HIGH);
        digitalWrite(IN12, LOW);
        digitalWrite(EN1, HIGH);
        digitalWrite(IN21, HIGH);
        digitalWrite(IN22, LOW);
        digitalWrite(EN2, HIGH);
    }
    else if( x >= 0 && x <= 599){
        Serial.println("atras");
        digitalWrite(IN11, LOW);
        digitalWrite(IN12, HIGH);
        digitalWrite(EN1, HIGH);
        digitalWrite(IN21, LOW);
        digitalWrite(IN22, HIGH);
        digitalWrite(EN2, HIGH);
    }
}
});

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

    // Pines motores como salidas
    pinMode(IN11, OUTPUT);
    pinMode(IN12, OUTPUT);
    pinMode(EN1, OUTPUT);
    pinMode(IN21, OUTPUT);
    pinMode(IN22, OUTPUT);
    pinMode(EN2, OUTPUT);

    pinMode(LED, OUTPUT);

    digitalWrite(LED, HIGH);

    // Connect to Wi-Fi network with SSID and password
    Serial.print("Setting AP (Access Point)...");
    // Remove the password parameter, if you want the AP (Access Point) to be open
    WiFi.softAP(ssid, password);

    IPAddress IP = WiFi.softAPIP();

    client.setServer(mqtt_broker, mqtt_port);
    client.setCallback(callback);
    while (!client.connected()) {
```



```

String client_id = "esp32-client-";
client_id += String(WiFi.macAddress());

Serial.printf("The client %s connects to the public MQTT broker\n",
client_id.c_str());
if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
    Serial.println("Public EMQX MQTT broker connected");
} else {
    Serial.print("failed with state ");
    Serial.print(client.state());
    delay(1000);
}
}

// Publish and subscribe
client.publish(topic, "Hi, I'm ESP32 ^^");
client.subscribe(topic);
runner.init();
runner.addTask(receptor_mqtt);
runner.addTask(direccion_motor);
receptor_mqtt.enable();
direccion_motor.enable();
}

void loop(){
    runner.execute();
}

```

En segundo lugar se muestra el código final en el microordenador unificando todas las funcionalidades desarrolladas en este proyecto.

```

from random import randint          #conexión
from paho.mqtt import client as mqtt_client #conexión
import spidev #Joystick
#import os    #Joystick

from time import sleep,time #Mutuo

#t_anterior_publish = time.time()

#interval_publish = 1

broker = '192.168.4.2'
port = 1883
topic = "prueba"
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{randint(0, 1000)}'

```

```
username = 'pi'
password = 'raspberry'

# Define Axis Channels (channel 3 to 7 can be assigned for more buttons / joysticks)
swt_channel = 0
vrx_channel = 1
vry_channel = 2
#Time delay, which tells how many seconds the value is read out
delay = 1

# Spi oeffnen
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz=1000000
# Function for reading the MCP3008 channel between 0 and 7

def readChannel(channel):
    val = spi.xfer2([1,(8+channel)<<4,0])
    data = ((val[1]&3) << 8) + val[2]
    return data

# endless loop

'''
def lectura():

    # Determine position
    vrx_pos = readChannel(vrx_channel)
    vry_pos = readChannel(vry_channel)
    # SW determine
    swt_val = readChannel(swt_channel)
    # output
    #print("VRx : {} VRY : {} SW : {}".format(vrx_pos,vry_pos,swt_val))
    # wait
    sleep(delay)
'''

def connect_mqtt():
    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)

    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
```

```
    return client

def publish(client):
    msg_count = 0

    # Determine position
    vrx_pos = readChannel(vrx_channel)
    vry_pos = readChannel(vry_channel)
    # SW determine
    swt_val = readChannel(swt_channel)
    # output
    #print("VRx : {} VRy : {} SW : {}".format(vrx_pos,vry_pos,swt_val))
    # wait
    #sleep(delay)

    lista = f"{vrx_pos}" + "," + f"{vry_pos}" + "," + f"{swt_val}"

    #msg = f" {msg_count}"
    result = client.publish(topic, lista)
    # result: [0, 1]
    status = result[0]
    if status == 0:
        pass#print(f"Send `{lista}` to topic `{topic}`")
    else:
        pass
        #print(f"Failed to send message to topic {topic}")
    #msg_count += 1

def subscribe(client: mqtt_client):
    def on_message(client, userdata, lista):
        print(f"Received `{lista.payload.decode()}` from `{lista.topic}` topic")
        sleep(1)
    client.subscribe(topic)
    client.on_message = on_message

def run():
    client = connect_mqtt()
    #print("hola")
    #sleep(1)
    while True:
        #subscribe(client)
        publish(client)

        #client.loop_forever()

if __name__ == '__main__':
    run()
```

9. Conclusiones

Con este proyecto he desarrollado y he adquirido los conocimientos obtenidos en el curso Creación de prototipos de IoT con Raspberry impartido por el profesor Joan Masdemont Fontàs. Un ejemplo de ello es la utilización de motores de corriente pero utilizando el lenguaje de programación C++.

Por un lado, el microordenador no cuenta con pines de entrada analógicos por lo que es necesario el uso del chip mcp3008 para convertir cualquier entrada analógica a digital. Esto es una gran desventaja contra microcontroladores como el ESP32 o el Arduino UNO.

Por otro lado, programar en el lenguaje de programación en C++ es mucho más complicado que programar en Python. En el primero hay que tener más conocimientos de cómo se hacen los procesos mientras que en Python hay funciones diseñadas específicamente para muchos procesos.

Finalmente, el mayor punto débil de este proyecto es la transmisión de datos entre el ESP32 y el Raspberry Pi 3 Model V1.2. En un principio eran de tres segundos de espera pero si el programa estaba en marcha durante unos minutos podía ser superior a ser a los diez segundos. Este hecho era muy incómodo y desesperante porque era un tiempo de intriga en que no se sabía si funciona correctamente o no. Para intentar solucionar este problema se ha mejorado el código eliminado el máximo de prints y time.sleep o delays posibles. También se ha utilizado la librería TaskScheduler.h en el ESP32 para que el microcontrolador sea capaz de hacer más de una acción a la vez. El resultado de esto no ha sido el esperado porque solo ha sido posible tener un retraso de tres segundos estables y por lo tanto el responsable de esto ha sido el protocolo mqtt.

10. Web grafía

- [raspbian stretch - Setting up a Raspberry Pi as an access point - the easy way - Raspberry Pi Stack Exchange](#)
- [Installing ESP32 in Arduino IDE \(Windows, Mac OS X, Linux\) | Random Nerd Tutorials](#)
- [https://randomnerdtutorials.com/esp32-access-point-ap-web-server/](#)
- [How to Connect an ESP32 WiFi Microcontroller to a Raspberry Pi Using IoT MQTT \(predictabledesigns.com\)](#)
- [Installing ESP32 in Arduino IDE \(Windows, Mac OS X, Linux\) | Random Nerd Tutorials](#)
- [https://www.emqx.com/en/blog/esp32-connects-to-the-free-public-mqtt-broker](#)
- [https://dev.to/emqx/how-to-use-mqtt-in-python-paho-3b8](#)
- [ESP32 MQTT Publish Subscribe with Arduino IDE | Random Nerd Tutorials](#)
- [Installing ESP32 in Arduino IDE \(Windows, Mac OS X, Linux\) | Random Nerd Tutorials](#)
- [Getting Started with Node-RED on Raspberry Pi | Random Nerd Tutorials](#)
- [Getting Started with Node-RED Dashboard | Random Nerd Tutorials](#)
- [Install Mosquitto Broker Raspberry Pi | Random Nerd Tutorials](#)
- [Running a MQTT Broker on Raspberry Pi - Hackster.io](#)
- [Use a Joystick on the Raspberry Pi \(with MCP3008\) \(tutorials-raspberrypi.com\)](#)
- [ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials](#)
- [Split String in Arduino | Delft Stack](#)
- [Arduino convierte cadena a char | Delft Stack](#)
- [https://community.home-assistant.io/t/after-mosquitto-update-increased-connection-time/113735](#)