

Vulcano  
v.0.3 2024

PROBOT

3 de abril de 2024



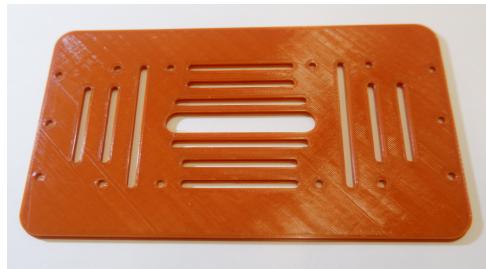
# Conteúdo

<b>1 Material</b>	<b>3</b>
1.1 Material para o chassis . . . . .	3
1.2 Electrónica . . . . .	5
1.3 Montagem Componentes eletrónicos . . . . .	7
1.4 Ligações electrónicas . . . . .	7
1.4.1 Ligações dos motores aos controladores. . . . .	7
1.4.2 Ligações dos sensores de chama . . . . .	7
1.4.3 Ligações dos sensores I2C - Temperatura e cor . . . . .	9
1.5 Programação . . . . .	10
1.5.1 Teste dos motores . . . . .	10
1.5.2 Sensores de chama . . . . .	11
1.5.3 Teste de sensores I2C (Inter-Integrated Circuit) . . . . .	11
1.5.4 MQTT . . . . .	13

# 1 Material

Para montar o nosso Vulcano vamos precisar do material que a seguir apresentamos.  
**Nota que neste desafio estás a receber o robô Vulcano parcialmente montado.**

## 1.1 Material para o chassis



Duas Bases



Dois suportes de bases



Quatro suportes de motor



Quatro whegs



Suporte sensores chama



Suporte controladores



Parafusos Motores 3x30 (8)



Parafuso suporte controladores 3x16 (6)

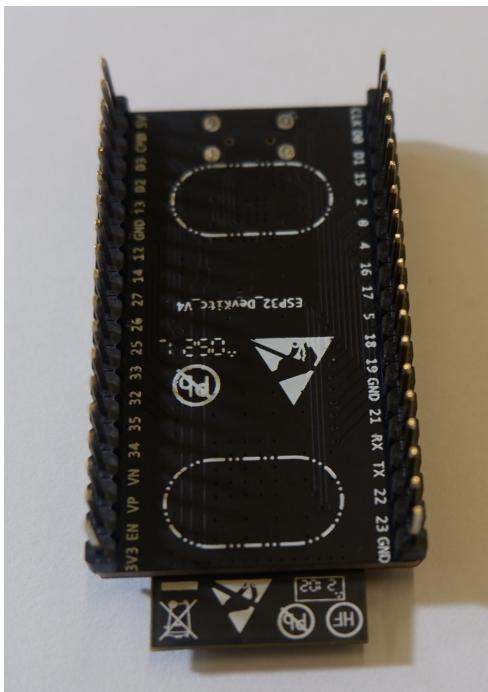


Parafusos chassis 3x10 (30)

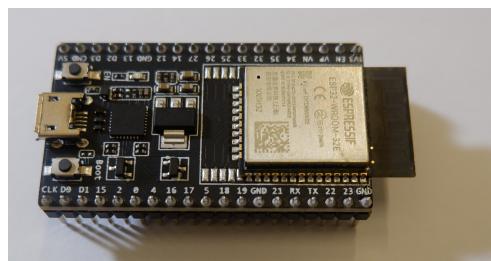


Porcas parafusos chassis (30)

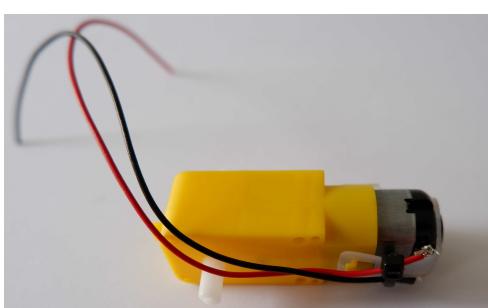
## 1.2 Electrónica



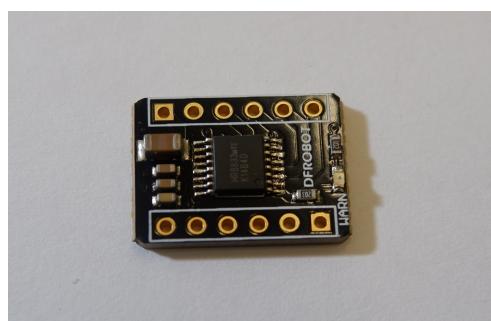
ESP32 - Vista frontal



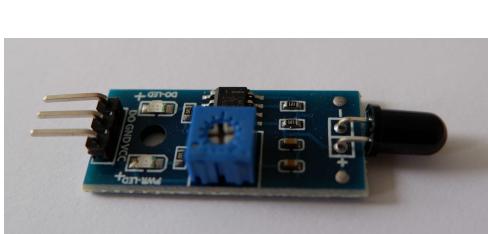
ESP32 - Vista lateral



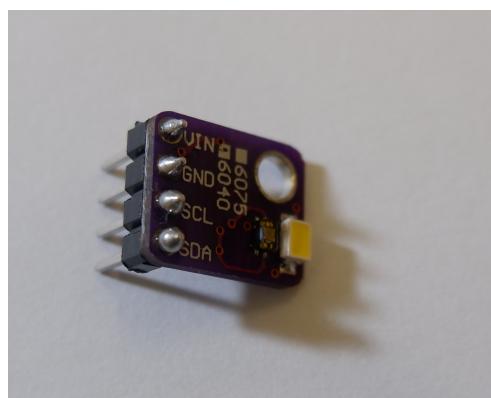
Motores (4)



Controlador de motor (1)



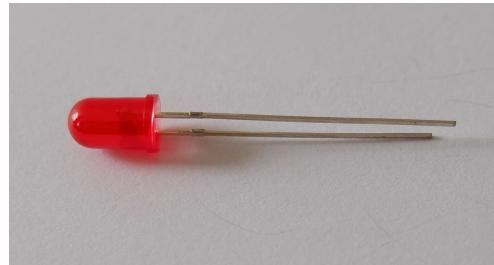
Sensor de chama (3)



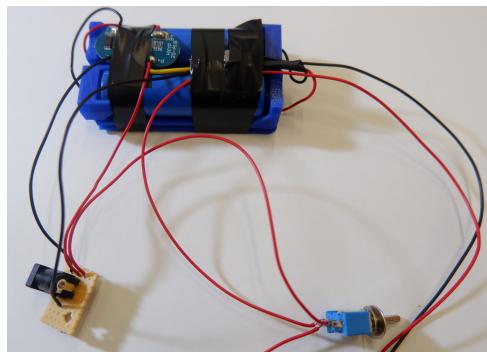
Sensor de cor (1)



Sensor de Temperatura (1)



Led (1)



Bateria (1)



Carregador (1)

## 1.3 Montagem Componentes eletrónicos

Esta é uma parte fundamental para o bom funcionamento do Vulcano. Uma ligação errada ou defeituosa pode pôr em causa o desempenho do robô ou levar mesmo à avaria de um ou mais componentes. Deves por isso ter atenção aos seguintes passos:

- Antes de ligar a bateria confirma todas as ligações efetuadas;
- Efetua as ligações passo a passo. Por exemplo:
  - liga em primeiro lugar os motores;
  - confirma as ligações;
  - liga a bateria;
  - confirma o funcionamento dos motores usando um dos exemplos de programação fornecidos.

Depois de tudo confirmado passa para a montagem dos primeiros sensores e assim sucessivamente até teres o teu Vulcano pronto para o desafio. Por uma questão de segurança desliga a bateria sempre que estiveres a proceder a novas ligações.

## 1.4 Ligações electrónicas

### 1.4.1 Ligações dos motores aos controladores.

Pinos do ESP32:

Motores lado esquerdo: Pinos 17 e 18.

Motores lado direito: Pinos 26 e 27.

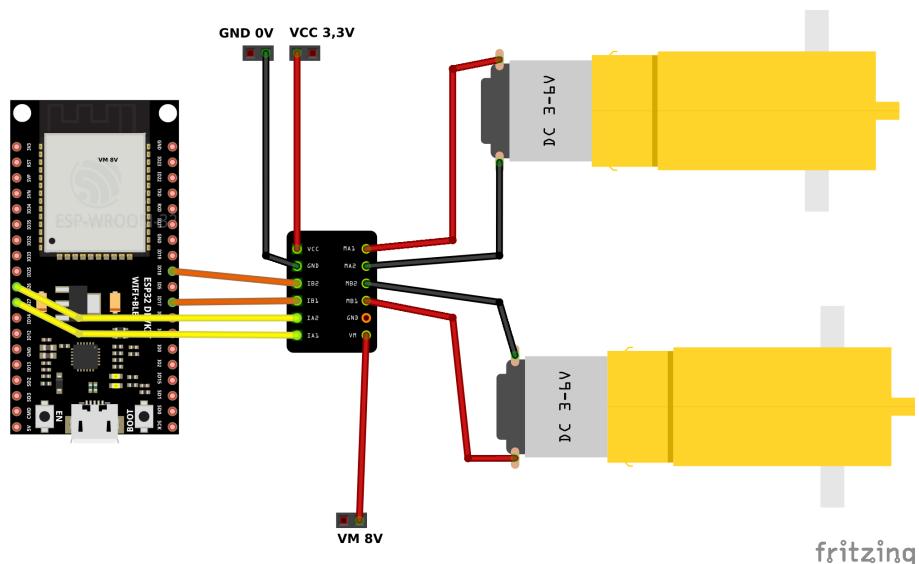


Figura 1: Ligação dos motores

É agora altura de testares o funcionamento dos motores usando a programação disponibilizada em 1.5.1.

### 1.4.2 Ligações dos sensores de chama

A figura 2 mostra um esquema das ligações dos sensores de chama ao controlador ESP32. Os sensores têm saída digital e é necessário declarar na função *setup* o modo dos mesmos – *por exemplo*:

```
void setup(){
    pinMode (34, INPUT);
}
```

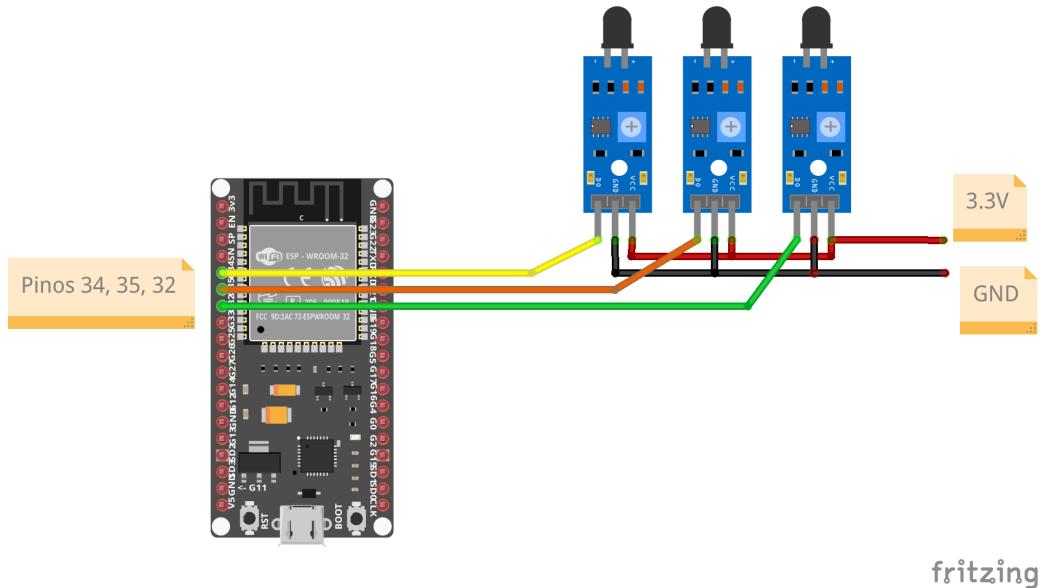


Figura 2: Ligações dos sensores

Podes testar os sensores de chama usando o código disponibilizado. Nota que é apenas um exemplo, terás de adaptar o código para mais de um sensor de acordo com as ligações que efetuares:

```
#include <Arduino.h>
#define Sensor_chama 32
void setup() {
    Serial.begin(115200);
    pinMode(Sensor_chama, INPUT);
}

void loop() {
    int val = digitalRead(Sensor_chama);
    Serial.println(val);
    delay(200);
}
```

#### 1.4.3 Ligações dos sensores I2C - Temperatura e cor

A figura 3 mostra o esquema das ligações dos sensores de temperatura e cor ao controlador ESP32. Os pinos do ESP32 devem ser sempre o pino 21, correspondente a SDA (Serial Data) e o pino 22 (Serial Clock).

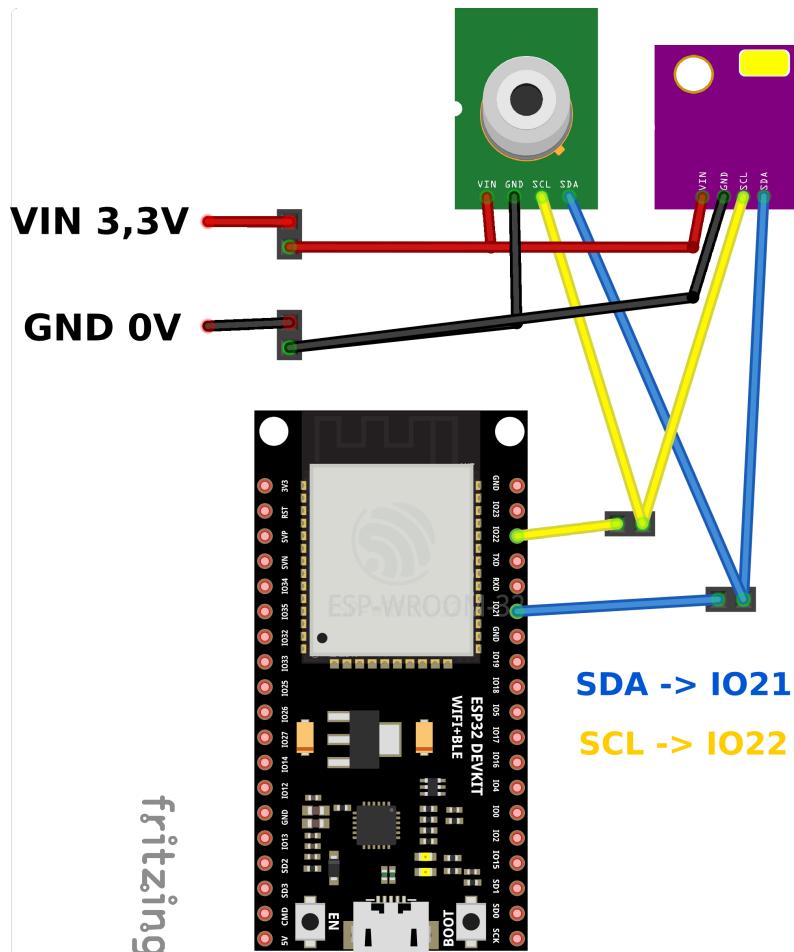


Figura 3: Ligações dos sensores de temperatura e cor

Para saberes um pouco mais sobre o protocolo I2C deves passar para secção 1.5.3.

## 1.5 Programação

Para programares o Vulcano podes usar o Visual Studio Code adicionando a extensão PlatformIO, ou em opção usar o Arduino IDE. Em qualquer dos casos deves começar por criar um novo projeto (VScode+Platformio) ou um novo Sketch (Arduino IDE). As bibliotecas deverão ficar na mesma pasta do ficheiro principal. A placa a escolher será ESP32 DEV MODULE. No caso de usares o Arduino IDE não esqueças de adicionar as placas ESP32 ao IDE. Em Arquivo/Preferências (File/Preferences) adicionar o seguinte link [https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json) na opção URL do gerenciador de placas adicionais (Additional boards manager URL's)

Na página do Vulcano podes descargar toda a documentação do desafio Vulcano onde encontrares pequenos exemplos de código para testares todos os componentes electrónicos do robô. Estes estão incluídos nas seguintes pastas:

- i2c \_scan
- motores
- sensor\_chama
- sensor\_rgb
- sensor\_temp
- temperatura\_led
- motores\_temperatura
- mqtt\_temperatura

Em quase todas as pastas existem diversos ficheiros. O ficheiro *main.cpp* é aquele onde desenvolvemos o nosso código. Os outros correspondem a bibliotecas necessárias à programação de sensores e motores e deverão constar dos ficheiros *main.cpp* através da diretiva *include* - no exemplo *motores* necessitamos da biblioteca ESP32MotorControl:

```
#include "ESP32MotorControl.h"
```

Se usares o Arduino IDE a extensão do ficheiro principal será *.ino* em vez de *.cpp*.

### 1.5.1 Teste dos motores

Na pasta *motores* tens um exemplo que permite ver o funcionamento dos motores.

```
#include "ESP32MotorControl.h"
#include <Arduino.h>

// Inicialização dos motores
ESP32MotorControl MotorControl = ESP32MotorControl();

/* Nas seguintes funções assumimos que:
** Motor esquerdo - 0
** Motor direito - 1
*/
void robot_forward(int speedLeft, int speedRight) {
    MotorControl.motorForward(0, speedLeft);
    MotorControl.motorForward(1, speedRight);
}
void robot_reverse(int speedLeft, int speedRight) {
    MotorControl.motorReverse(0, speedLeft);
    MotorControl.motorReverse(1, speedRight);
}
void robot_turn_left(int speedLeft, int speedRight) {
    MotorControl.motorForward(1, speedRight);
    MotorControl.motorReverse(0, speedLeft);
```

```

}

void robot_turn_right(int speedLeft, int speedRight) {
    MotorControl.motorForward(0, speedLeft);
    MotorControl.motorReverse(1, speedRight);
}

void robot_stop() {
    MotorControl.motorStop(0);
    MotorControl.motorStop(1);
}

void setup() {
    // 17,18,26,27 pinos dos motores no ESP32
    // correspondentes aos motores esquerdos e direitos
    MotorControl.attachMotors(17, 18, 26, 27);
}

void loop() {
    robot_forward(50,50);
    delay(5000);
    robot_turn_left(50,50);
    delay(5000);
    robot_turn_right(50,50);
    delay(5000);
    robot_stop();
    delay(2000);
}

```

Se tudo estiver corretamente ligado estes funcionarão da seguinte forma:

- Em frente durante 5 segundos (velocidade 50);
- Vira à esquerda durante 5 segundos;
- Vira à direita durante 5 segundos;
- Para durante 2 segundos.

Se na fase inicial em que os motores funcionam **em frente**, os motores de um lado rodam para trás, deves trocar os fios que ligam o ESP32 ao controlador dos motores – se for do lado esquerdo, trocar os pinos 17 e 18, se for do lado direito, trocar os pinos 26 e 27.

### 1.5.2 Sensores de chama

Os sensores de chama usados no Vulcano têm uma saída digital que envia um sinal 0 quando em presença de radiação infravermelha e 1 quando não deteta essa radiação. Poderás usá-los para orientar o teu Vulcano na subida.

### 1.5.3 Teste de sensores I2C (Inter-Integrated Circuit)

I2C (Inter-Integrated Circuit) é um bus de comunicação em série que utiliza uma arquitectura multi-master-slave.

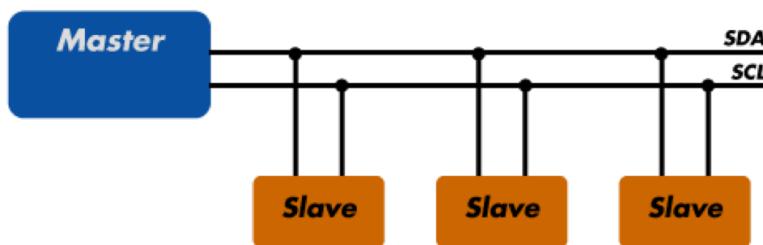


Figura 4: I2C

Devido à sua simplicidade, o I2C é amplamente utilizado na comunicação entre microcontroladores e conjuntos de sensores, ecrãs, dispositivos IoT, EEPROMs, etc.

O I2C é constituído por duas linhas, SCL (Serial Clock) e SDA (Serial Data). O I2C suporta múltiplos slaves, ou seja, podem ser ligados múltiplos dispositivos I2C slave a um controlador I2C. Estes diferentes dispositivos I2C slave têm diferentes endereços de dispositivo, pelo que o controlador principal I2C pode aceder a um determinado dispositivo especificado através do seu endereço. No nosso caso o ESP32 será o controlador e os sensores de temperatura e cor os slaves.

O sensor de temperatura MLX90614 tem o endereço 0x5A (hexadecimal) e o sensor de cor tem o endereço 0x10 (hexadecimal). Para saberes se os sensores do teu robô estão ligados corretamente podes usar o ficheiro da pasta **i2c\_scan** e utilizando o monitor serial terás a informação dos sensores ligados e o respectivo endereço:

```
#include <Arduino.h>
#include <Wire.h>

void setup()
{
    Serial.begin (115200);
    pinMode(22, OUTPUT);           // Clock goes from master to peripherical
    pinMode(21, INPUT_PULLUP);     // Data should be internally pulled up
    Wire.begin(21,22);
}

void Scanner ()
{
    Serial.println ();
    Serial.println ("I2C scanner. Scanning ...");
    byte count = 0;

    Wire.begin();
    for (byte i = 8; i < 120; i++)
    {
        Wire.beginTransmission (i);           // Begin I2C transmission Address (i)
        if (Wire.endTransmission () == 0)    // Receive 0 = success (ACK response)
        {
            Serial.print ("Found address: ");
            Serial.print (i, DEC);
            Serial.print (" (0x");
            Serial.print (i, HEX);           // PCF8574 7 bit address
            Serial.println ());
            count++;
        }
    }
    Serial.print ("Found ");
    Serial.print (count, DEC);           // numbers of devices
    Serial.println (" device(s).");
}

void loop()
{
    Scanner ();
    delay (500);
}
```

Nota que este programa apenas te permite saber se os sensores estão corretamente ligados e a comunicar com o microcontrolador. Para obteres dados destes sensores terá de usar os outros exemplos disponibilizados para cada um dos sensores. De seguida usando os exemplos fornecidos podes confirmar os seu funcionamento através das leituras efetuadas pelos mesmos, por exemplo para o sensor de cor (RGB):

```
#include <Arduino.h>
```

```

#include "Wire.h"
#include "veml6040.h"

VEML6040 RGBWSensor;

void setup() {
    Serial.begin(115200);
    Wire.begin();
    if(!RGBWSensor.begin()) {
        Serial.println("ERROR: couldn't detect the sensor");
        while(1){}
    }

    /*
     * init RGBW sensor with:
     * - 320ms integration time
     * - auto mode
     * - color sensor enable
     */
}

RGBWSensor.setConfiguration(VEML6040_IT_320MS+VEML6040_AF_AUTO+VEML6040_SD_ENABLE);

delay(1500);
}

void loop() {
    Serial.print("RED: ");
    Serial.print(RGBWSensor.getRed());
    Serial.print(" GREEN: ");
    Serial.print(RGBWSensor.getGreen());
    Serial.print(" BLUE: ");
    Serial.print(RGBWSensor.getBlue());
    Serial.print(" WHITE: ");
    Serial.print(RGBWSensor.getWhite());
    Serial.print(" CCT: ");
    Serial.print(RGBWSensor.getCTT());
    Serial.print(" AL: ");
    Serial.println(RGBWSensor.getAmbientLight());
    delay(400);
}

```

#### 1.5.4 MQTT

MQTT, sigla de Message Queuing Telemetry Transport, é um protocolo de mensagens leve para sensores e pequenos dispositivos móveis otimizado para redes TCP/IP.

Neste protocolo um conjunto de equipamentos - sensores, microcontroladores, computadores, ... - comunicam enviando mensagens (*publisher*) e recebendo mensagens (*subscribers*). Qualquer um dos equipamentos pode ser ao mesmo tempo *publisher* e *subscriber*. As mensagens são separadas por tópicos. Qualquer *publisher* pode criar qualquer tipo de tópico. Os *subscribers* terão de conhecer os tópicos publicados para poderem aceder às respectivas mensagens.

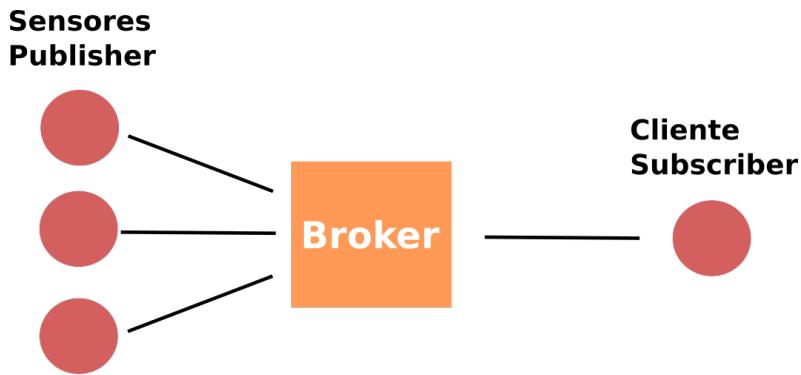


Figura 5: MQTT

Durante os dias do festival terás um *broker* MQTT a funcionar - o IP do computador em que o *broker* funciona será disponibilizado no primeiro dia. Para testares o envio de mensagens para o broker podes usar o seguinte software - <http://mqtt-explorer.com/> e publicar para os tópicos que entenderes. No desafio propriamente dito terás de publicar para os seguintes tópicos:

- **temperatureOBJ** -> o valor da temperatura do solo
- **temperatureAMB** -> o valor da temperatura ambiente
- **hot** -> uma string *HOT* indicando um ponto quente (opcional);
- **top** -> uma string *TOP* indicando a zona vermelha da rampa (opcional);

No ficheiros tens um exemplo da publicação de mensagens para os tópicos **temperatureOBJ** e **temperatureAMB**:

```

#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_MLX90614.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "wifi/WiFiMulti.h"

// SSID e password da rede wifi
#define WLAN_SSID "robotica"
#define WLAN_PASS "robotica2023"
// IP e porta do servidor MQTT
#define AIO_SERVER "10.129.35.194"
#define AIO_SERVERPORT 1883

//Variáveis da temperatura ambiente e objecto
float ambientCelsius = 0.0;
float objectCelsius = 0.0;

//Instância do sensor de temperatura
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
//Instância do cliente de wifi
WiFiClient client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT);
// temperatureOBJ e temperatureAMB são os nomes dos tópicos
// para onde vamos publicar
// Durante os testes podemos mudar para algo diferente
Adafruit_MQTT_Publish temperatureObj = Adafruit_MQTT_Publish(&mqtt, "temperatureOBJ");
Adafruit_MQTT_Publish temperatureAmb = Adafruit_MQTT_Publish(&mqtt, "temperatureAMB");

```

```

//Função para realizar a ligação MQTT
void MQTT_connect() {
    int8_t ret;

    // Stop if already connected.
    if (mqtt.connected()) {
        return;
    }

    Serial.print("Connecting to MQTT... ");

    uint8_t retries = 3;
    while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
        Serial.println(mqtt.connectErrorString(ret));
        Serial.println("Retrying MQTT connection in 5 seconds... ");
        mqtt.disconnect();
        delay(5000); // wait 5 seconds
        retries--;
        if (retries == 0) {
            // basically die and wait for WDT to reset me
            while (1);
        }
    }

    Serial.println("MQTT Connected!");
}

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

    // Connect to WiFi access point.
    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(WLAN_SSID);

    WiFi.begin(WLAN_SSID, WLAN_PASS);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println();
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    //Inicializamos o sensor de temperatura
    mlx.begin();
}

void loop() {
    MQTT_connect();

    ambientCelsius = mlx.readAmbientTempC();
    objectCelsius = mlx.readObjectTempC();
    Serial.print(F("\nSending temperature val "));

    if (!temperatureObj.publish(objectCelsius)) {
}

```

```
    Serial.println(F("Failed"));
}
else {
    Serial.println(F("OK!"));
}
if (!temperatureAmb.publish(ambientCelsius)) {
    Serial.println(F("Failed"));
}
else {
    Serial.println(F("OK!"));
}
delay(2000);
}
```

Esta é a fase mais complexa do desafio mas não te preocipes - durante o workshop daremos exemplos e lá estaremos para te apoiar!

Bom trabalho e acima de tudo diverte-te!