

SembradOS

Javier Fortuño y Christian Vinader

1. Idea del proyecto.....	2
2. Estructura del proyecto.....	2
3. Manual de Usuario.....	3
4. Programación en Arduino.....	6
4.1 Librerías.....	6
4.2 Variables globales.....	6
4.3 Setup.....	7
4.4 La clase App.....	8
4.5 Loop.....	13
4.5 Envío de datos a ThingSpeak.....	14
4.6 Recepción de datos MQTT.....	15
5. Plataforma ThingSpeak.....	18
5.1 Web de ThingSpeak.....	18
5.2 App móvil de ThingSpeak.....	21
5.3 Alertas en ThingSpeak.....	22
6. TinyML.....	24
6.1 Scrapping.....	24
6.2 Entrenamiento del modelo.....	27
6.3 Inferencia.....	29
6.4 Código Main de la Inferencia.....	30
7. Dockerización de la inferencia.....	33
8. Raspberry Pi.....	34
9. Resultados.....	38
9.1 Video de la Demo.....	38
9.1 Estructura de la entrega.....	38
10. Problemas encontrados.....	39
10.1 Código espagueti.....	39
10.2 Problemas con los Hilos.....	40
10.3 Problemas con el entrenamiento de la red neuronal.....	46
10.4 Problemas con la inferencia en la Wio Terminal.....	47
10.5 Imagen docker para la Raspberry pi.....	50
11. Bibliografía.....	53

1. Idea del proyecto

Nuestra idea con este proyecto es que mediante la pantalla del Wio Terminal y con unos sensores, poder registrar la temperatura, la humedad, la humedad del suelo y la lectura de luz para el correcto crecimiento y cuidado de una planta. Además de poder enviar estos datos a una plataforma IoT para su monitoreo, adicionalmente, se inferirá el tiempo meteorológico actual/en el corto plazo gracias a un modelo de red neuronal que hemos entrenado con los datos meteorológicos de la ciudad de Valencia.

Una forma novedosa y moderna para el cuidado de las plantas.

2. Estructura del proyecto

El proyecto se puede dividir en siete partes, siendo estas:

- Interfaz gráfica.
- Lectura de sensores.
- Comunicación con la plataforma de visualización de datos.
- Comunicación con el broker MQTT.
- Configuración de la plataforma de visualización de datos.
- Entrenamiento de la red neuronal.
- Inferencia de la red neuronal.

Para cumplir con la idea del proyecto, hemos terminado ideando el siguiente sistema (el porqué de esto se explica en el apartado de problemas “10.4 Problemas con la inferencia en la Wio Terminal”):



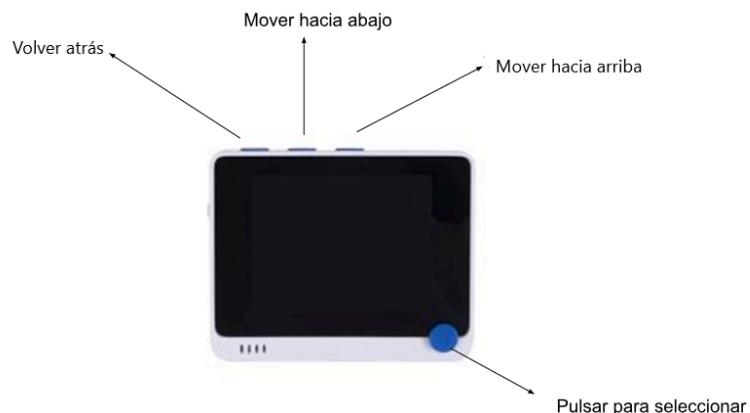
La explicación es la que sigue: La Wio Terminal recogerá datos de los sensores (temperatura, humedad, humedad de la tierra y nivel de luz), que el usuario podrá consultar

en tiempo real dentro del menú “Sensores” del propio dispositivo, además, la Wio Terminal enviará de forma periódica estos datos a la plataforma, ThingSpeak, donde el usuario podrá consultar la evolución de estos datos a intervalos configurables, adicionalmente, si se detecta algún dato anómalo, se enviará un correo de alerta.

Por otra parte, una Raspberry Pi leerá periódicamente las siete últimas muestras de temperatura, humedad, y mes en el que fueron tomadas las mediciones para de esta forma predecir el tiempo actual, una vez realice la inferencia, enviará el resultado de esta a el broker MQTT HiveMQ, finalmente la Wio Terminal recogerá el resultado de la predicción, haciéndola disponible al usuario en la ventana de “Predicción”.

3. Manual de Usuario.

El funcionamiento del wio terminal es muy simple, en este manual procederemos a explicarlo, la aplicación SembradOS hace uso de cuatro de los botones de la Wio Terminal:

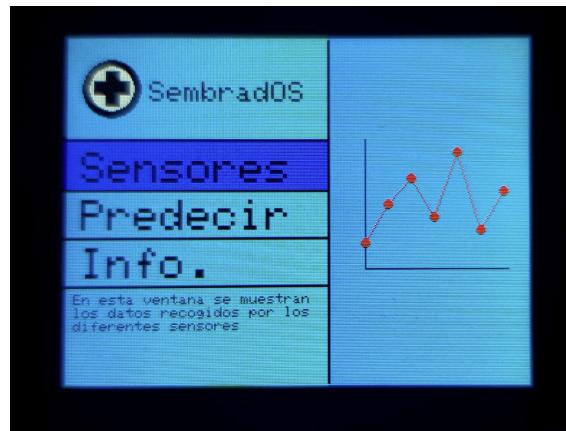


Empezamos encendiendo el dispositivo, bajando el botón lateral, una vez encendida, nos indicará que se está iniciando y mostrará el avance de la carga, además, nos recibirá con su logo y una breve melodía.

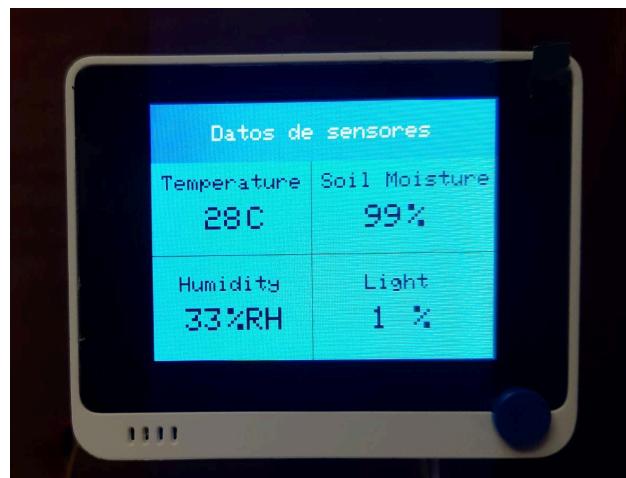


Una vez en el menú principal de la aplicación, podemos movernos entre sus opciones, tenemos disponibles tanto el menú para lectura de sensores, como el menú donde se muestra las predicciones, y por último el menú de información, que indica el estado de las conexiones.

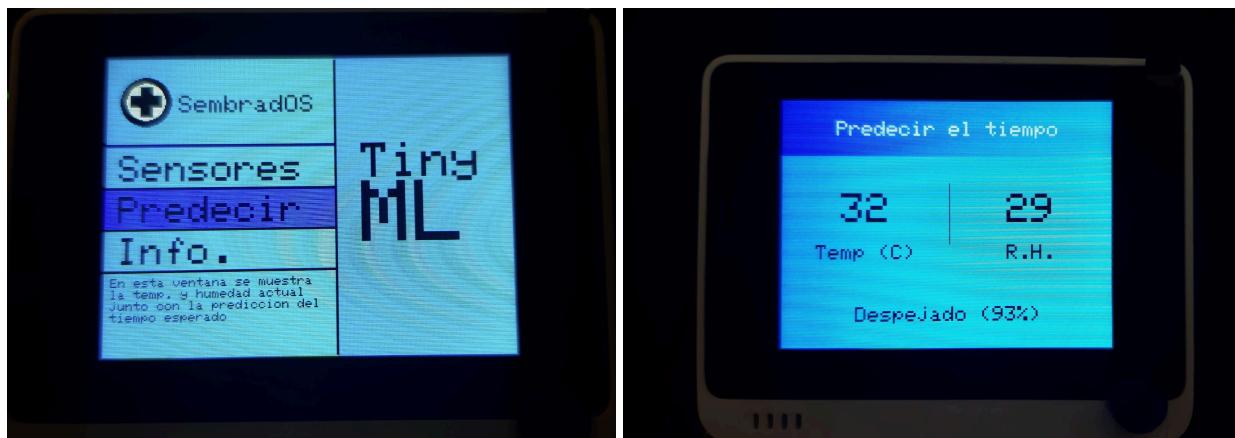
Además en la parte inferior se muestra una pequeña explicación de cada ventana, y a la derecha se muestra una imagen de ejemplo adecuada a la ventana que está seleccionando.



La primera ventana que podemos seleccionar es la ventana de *Sensores*, en ella podemos ver las lecturas de los sensores. Nos muestra las lecturas de temperatura, humedad, humedad del suelo y luz.

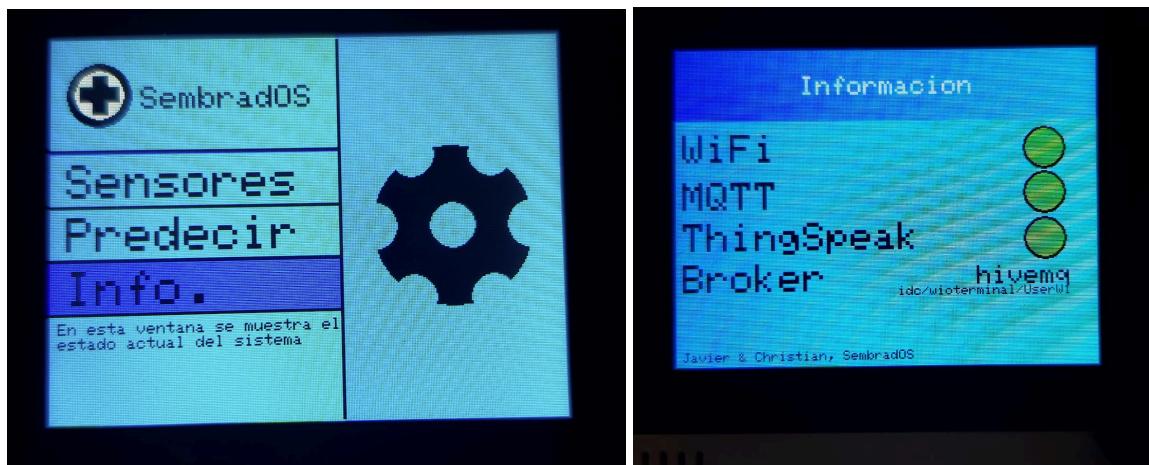


La segunda ventana es la de *Predicción*, en ella nos muestra la temperatura y humedad actuales, además de una predicción del tiempo actual/en el corto plazo.



La última ventana muestra la información de la aplicación:

- Estado conexión WiFi.
- Estado de la conexión con el broker MQTT.
- Estado de la conexión con la plataforma ThingSpeak.
- Broker MQTT al cual queremos conectarnos y el tópico al que estamos suscritos.



Por ejemplo, si la Wio Terminal no tuviese conexión con el broker MQTT y por lo tanto no pudiese recibir los datos de la predicción, ocurriría lo siguiente:



4. Programación en Arduino

Hemos decidido desarrollar el código de la Wio Terminal en lenguaje C++ usando el Arduino IDE y sus librerías, hemos escogido esta plataforma por diversas razones: Arduino proporciona una interfaz amigable que facilita la programación para los desarrolladores. Además, la extensa biblioteca de Arduino ofrece módulos preconstruidos y ejemplos que aceleran el desarrollo, reduciendo el tiempo necesario para implementar funcionalidades complejas. Así como la amplia existencia de librerías optimizadas para hardware específico, como el Wio Terminal.

Por último, la capacidad de integración con sensores, módulos y servicios web a través de bibliotecas bien mantenidas, permite desarrollar soluciones IoT robustas y versátiles.

Para el caso de la Wio Terminal, esta dispone de una extensa documentación proporcionada por la propia marca para Arduino:

<https://wiki.seeedstudio.com/Wio-Terminal-Getting-Started/>

A continuación se procederá a explicar de forma breve cada sección del código, se advierte de que el código mostrado podría variar del presente en la versión final del proyecto, pero el propósito de cada sección no habrá cambiado.

4.1 Librerías

Empecemos, por el principio, la Wio Terminal dispone de una pantalla con una resolución de 320x240, por lo que es necesaria una biblioteca para controlar esta pantalla, además necesitaremos librerías para las conexiones HTTP y MQTT, así como para leer los datos de los sensores, entre otras.

```
#include "TFT_eSPI.h" //TFT LCD library
#include "DHT.h" // DHT library
#include <rpcWiFi.h>
#include <SPI.h>
#include "RTC_SAMD51.h"
#include "DateTime.h"
#include <PubSubClient.h>
#include <ArduinoJson.h>
```

4.2 Variables globales

Para crear la aplicación hemos necesitado crear un conjunto de variables, primero unas definiciones para almacenar el nombre de la App, los pines de los sensores y inicializar el controlador de la pantalla y el sensor dht, así como el reloj RTC.

```
#define NAME_APP "SembradOS"
#define BUZZER_PIN WIO_BUZZER
#define DHTPIN 0 //Define signal pin of DHT sensor
// #define DHTPIN PIN_WIRE_SCL //Use I2C port as Digital Port */
#define DHTTYPE DHT11 //Define DHT sensor type
DHT dht(DHTPIN, DHTTYPE);
TFT_eSPI tft; // Inicializar TFT LCD
RTC_SAMD51 rtc;
DateTime now = DateTime(F(__DATE__), F(__TIME__));
```

A continuación, la configuración del WiFi, junto con los valores para conectarse a la plataforma de ThingSpeak.

```
//Config. Wifi
const char* ssid = "MIWIFI_5G_z5wx";
```

```

const char* password = "xxxxxxxxxxxxxx";
// Configuración de ThingSpeak
const char* api_key = "TZNMIJZKWEIX2BS6"; // Tu clave de API de ThingSpeak
const char* server = "api.thingspeak.com";
const int port = 80;
bool TSstate = false; //Variable para guardar el estado de la conexión

```

Luego se crean las variables de los sensores, se especifica la configuración del broker MQTT y se crea el WiFiClient que gestionará su conexión:

```

float temperatura; //Assign variable to store temperature
float humedad; //dht.readHumidity(); //Assign variable to store humidity
int light; //Assign variable to store light sensor values
int soil_moisture;
int sensorPin = A1;
int sensorValue = 0;
int gatherDataFromSensor1 = true;

//Broker MQTT
const char *ID = "Wio-Terminal-Client"; // Name of our device, must be unique
const char *subTopic = "idc/wioterminal/UserW1"; // Topic to subscribe to
const char *serverMQTT = "broker.hivemq.com"; // Server URL
String nombreMQTT = "hivemq";
bool MQTTstate = false; //Estado de la conexión con el broker
String err_msg = "No hay conexión"; //Mensaje predeterminado

WiFiClient wifiClient;
PubSubClient client(wifiClient);

```

Finalmente, se crean las variables que guardan la predicción del modelo de red neuronal, además de crear los enumerables para los valores de los menús, botones y Outputs de la predicción.

```

String prediction = "No llueve";
float probability;
// Enums para los menús y botones
enum MenuType {
    MAIN_MENU,
    SENSORS_MENU,
    PREDICT_MENU,
    SETTINGS_MENU
};
enum ButtonType {
    BUTTON_UP,
    BUTTON_DOWN,
    BUTTON_SELECT,
    BUTTON_BACK
};
const char* OUTPUTS[] = {
    "Despejado",
    "Podría llover",
    "Llueve"
};

```

4.3 Setup

En el Setup del código se usan diversas de las variables anteriores para arrancar la aplicación, a grandes rasgos, se encarga de iniciar las conexiones necesarias, inicializar los

pines con sus respectivos modos, los sensores, la pantalla tft, los valores iniciales necesarios y de arrancar la aplicación así como llamar al menu principal.

```
App app = App(tft);
void setup() {
    Serial.begin(9600);
    rtc.begin();
    if (!Serial){delay(2000);}

    dht.begin(); //Start DHT sensor
    tft.begin();
    tft.setRotation(3); //set TFT LCD rotation
    tft.fillScreen(TFT_BLACK);
    tft.setTextPadding(320);
    tft.setTextColor(TFT_WHITE, TFT_BLACK);
    tft.setTextSize(2);
    tft.drawString("Iniciando.", 5, 220);

    startWiFi();
    tft.drawString("Iniciando...", 5, 220);
    rtc.adjust(now);
    now = rtc.now();

    pinMode(WIO_5S_UP, INPUT);
    pinMode(WIO_5S_DOWN, INPUT);
    pinMode(WIO_KEY_A, INPUT_PULLUP);
    pinMode(WIO_KEY_C, INPUT);
    pinMode(WIO_KEY_B, INPUT);
    pinMode(WIO_5S_PRESS, INPUT_PULLUP);
    pinMode(WIO_BUZZER, OUTPUT);
    pinMode(WIO_LIGHT, INPUT); //Set light sensor pin as INPUT

    //iniciar aplicación
    app.start();
    tft.drawString("Iniciando.....", 5, 220);
    //Primera lectura de sensores
    temperatura = dht.readTemperature();
    humedad = dht.readHumidity();
    sensorValue = analogRead(sensorPin); //Store sensor values
    soil_moisture = map(sensorValue,1023,400,0,100); //Map sensor values
    light = analogRead(WIO_LIGHT);
    light = map(light,0,1023,0,100);

    //Configuramos la conexión con el broker MQTT
    client.setServer(serverMQTT, 1883);
    client.setCallback(callback);
    reconnect();
    app.setMenu(MAIN_MENU);
}
```

4.4 La clase App

A medida que añadíamos características a la aplicación, se volvía imposible navegar por el código y añadir nuevas funcionalidades sin depender de comprobar estados con numerosos if's y funciones sobredimensionadas, se profundiza más en esto en el apartado de problemas encontrados, pero resumidamente decidimos intentar darle un lavado de cara a

la aplicación, creando una nueva clase App donde manejar toda la interfaz gráfica de nuestra aplicación, de forma que fuese más sencillo agregar métodos, menús y funcionalidades.

```
3 class App {
4     public:
5         //App() {}
6         App(TFT_eSPI &tft) //Constructor...
7     }
8     //Iniciar la aplicación
9     void start(){...}
10    void setMenu(MenuType menu) {...}
11    //Manejo de inputs de botones a nivel interno de la app
12    void handleButtonPress(ButtonType button) {...}
13    void update() { ...}
14    //Tonos (Sonido)
15    void playTone(int tone, int duration) {...}
16    //Notas (Sonido)
17    void playNote(char note, int duration) {...}
18    //Función pública para actualizar los valores de los sensores
19    void updateSensorsData(){...}
20
21 private:
22     //Printeo de cada menú de la aplicación
23     void printMainMenu() {...}
24     void printSensorsMenu() {...}
25     void printPredictMenu() {...}
26     void printSettingsMenu() {...}
27     //Actualiza el menú sensores con los nuevos datos de los sensores
28     void updateSensorsMenu() {...}
29     void updatePredictMenu(){...}
30     void updateSettingMenu(){...}
31     //Printeo de logos en el menú principal
32     void drawGraphLogo() {...}
33     void drawIALogo() {...}
34     void drawOptionsLogo() {...}
35     //Manejo de input de los botones a nivel visual de los menús
36     void buttonPressedMainMenu() {...}
37     void buttonPressedSensorsMenu() {//vacio...}
38     void buttonPressedPredictMenu() {//vacio...}
39     void buttonPressedSettingsMenu() {//vacio...}
40
41     //Función NOP
42     void nop(){}
43 }
```

Vamos a explicar a grandes rasgos cómo funciona esta clase, el código interno de algunas funciones se obviará por ser repetitivo, pero si se explicará su objetivo.

La clase App dispone de un constructor al que se le pasa como argumento el controlador/objeto de la pantalla tft, que es sobre el que trabajará la aplicación, este constructor.

Antes de ver el código, debemos entender que cada ventana de la aplicación realiza tres funciones principales: dibujarse, manejar los inputs del usuario y actualizarse.

Por eso, la clase App dispone de tres variables que almacenan las direcciones a las funciones que realizan estas tres acciones, que serán diferentes dependiendo del menú en el que se encuentre el usuario, estos tres punteros se actualizarán cada vez que nos

movamos entre los menús para apuntar a las nuevas funciones encargadas de manejar estas tres acciones de los menús.

```
class App {  
    public:  
        App(TFT_eSPI &tft) : tft(tft), spr(&tft), spr_sensores(&tft), currentMenu(MAIN_MENU),  
menuSelectedOption(0) { //asignaciones del tft, spr, currentMenu y MenuSelectedOption  
            //setMenu(MAIN_MENU);  
            printMenuFunction = &App::nop; //Función nop del objeto App que es basicamente  
una función vacía  
            buttonPressedFunction = &App::nop;  
            updateMenuFunction = &App::nop;  
        }  
}
```

Las variables privadas de la clase son las siguientes, entre ellas destacan los punteros a funciones anteriormente comentadas y dos sprites que ayudarán a dibujar información en pantalla de forma dinámica y rápida.

```
//Variables de la APP  
TFT_eSPI &tft;  
TFT_eSprite spr;  
TFT_eSprite spr_sensores;  
MenuType currentMenu;  
int menuSelectedOption;  
uint16_t colorFondo1 = 0x04FA;  
  
//Punteros a funciones DE la propia App (no sirven externas)  
void (App::*printMenuFunction)();  
void (App::*buttonPressedFunction)();  
void (App::*updateMenuFunction)();
```

Una vez se instancia App, se llama a App.start() (en el Setup), esta función se encarga de arrancar la aplicación, nos mostrará su logo mediante el uso de la librería tft y además tocará una melodía dándonos la bienvenida haciendo uso de otras dos funciones `void playTone(int tone, int duration)` y `void playNote(char note, int duration)` las cuales se encargan de los tonos y las notas de la melodía.

```
void start(){  
    tft.fillScreen(TFT_BLACK);  
    tft.fillCircle(160, 120, 50, 0x4380); // Círculo verde  
    tft.fillCircle(160, 120, 40, TFT_WHITE); // Círculo blanco  
    tft.fillRect(140, 110, 40, 20, 0x4380); // Raya horizontal  
    tft.fillRect(150, 100, 20, 40, 0x4380); // Raya vertical  
    tft.drawString("Iniciando...", 5, 220);  
  
    tft.setTextColor(TFT_WHITE);  
    tft.setTextSize(2);  
    tft.setTextDatum(TC_DATUM); //Centrado  
    tft.drawString(NAME_APP, 160, 184);  
    tft.setTextDatum(TL_DATUM);  
  
    char notes[] = "aabbc";  
    int beats[] = { 1, 1, 1, 1, 2 };  
    int tempo = 200;
```

```

        for(int i = 0; i < sizeof(notes)/sizeof(notes[0]); i++) {
            this->playNote(notes[i], beats[i] * tempo);
            delay(tempo / 5); /* delay between notes */
        }
        tft.drawString("Iniciando.....", 5, 220);
    }
}

```

A continuación disponemos de la función que cambia entre menús, la cual recibe como entrada el menu al que se quiere mover el usuario y se encarga de actualizar los punteros printMenuFunction, printMenuFunction y updateMenuFunction además de ejecutar las funciones a las que apuntan estos punteros para dibujar y actualizar el nuevo menú, se muestra una parte del código, con solo el caso para el menú de muestra de datos de sensores:

```

void setMenu(MenuType menu) {
    //if(xSemaphoreTake(changingMenuSemaphore, portMAX_DELAY) == pdTRUE) {
    spr.deleteSprite();
    spr_sensores.deleteSprite();
    currentMenu = menu;
    Serial.print(menu);
    switch (menu) {
        //Asignamos las direcciones de las funciones de gestión del menú
        case MAIN_MENU:
            printMenuFunction = &App::printMainMenu; //Le decimos que está dentro de App
o no lo aceptará
            buttonPressedFunction = &App::buttonPressedMainMenu;
            updateMenuFunction = &App::nop;
            menuSelectedOption = 0;
            break;
        case SENSORS_MENU:
            printMenuFunction = &App::printSensorsMenu;
            buttonPressedFunction = &App::buttonPressedSensorsMenu;
            updateMenuFunction = &App::updateSensorsMenu;
            break;
        case PREDICT_MENU:
            printMenuFunction = &App::printPredictMenu;
            buttonPressedFunction = &App::buttonPressedPredictMenu;
            updateMenuFunction = &App::updatePredictMenu;
            break;
        case SETTINGS_MENU:
            printMenuFunction = &App::printSettingsMenu;
            buttonPressedFunction = &App::buttonPressedSettingsMenu;
            updateMenuFunction = &App::updateSettingMenu;
            break;
    }
    Serial.println(menu);
    //printMenuFunction es un puntero a función, así que hay que desreferenciarlo y
entonces ejecutarlo
    (this->*printMenuFunction)();
    (this->*updateMenuFunction)();
    //xSemaphoreGive(changingMenuSemaphore);
}

```

Luego tenemos una función que gestiona las acciones de los botones por los menús, a nivel de aplicación, esta función recibe el botón que se ha pulsado y registra los cambios en el estado de las variables de la aplicación, para que esta luego se actualice acorde a la acción

realizada mediante la llamada a la función buttonPressedFunction, se muestra el caso para la ventana principal:

```
void handleButtonPress(ButtonType button) {
    //Manejo acciones botones en menu main
    if(currentMenu == MAIN_MENU) {
        switch (button) {
            case BUTTON_UP:
                menuSelectedOption = (menuSelectedOption - 1 + 3) % 3;
                playNote('a', 30);
                break;
            case BUTTON_DOWN:
                menuSelectedOption = (menuSelectedOption + 1) % 3;
                playNote('a', 30);
                break;
            case BUTTON_SELECT:
                setMenu(static_cast<MenuType>(menuSelectedOption + 1)); //casteo a MenuType para
pasar como argumento
                playNote('f', 30);
                return;
            break;
        }
        ...
    }
    (this->*buttonPressedFunction)(); //actualizamos la pantalla
}
```

Y finalmente, dentro de esta sección de métodos públicos disponemos de la función.

```
void update() {
    (this->*updateMenuFunction)();
}
```

Que simplemente hace una llamada a la función que actualiza el menú actual para que los cambios que se hayan hecho (por ejemplo, nuevas lecturas de un sensor) se vean reflejados.

A continuación, pasamos a la sección de las funciones privadas de la clase App, aquí se encuentran todas las funciones encargadas de dibujar cada menú, actualizarlos periódicamente, dibujar diversos logos y gestionar el impacto visual de pulsar los botones, estas son:

```
private:
    //Printeo de cada menú de la aplicación
    void printMainMenu() {...}
    void printSensorsMenu() {...}
    void printPredictMenu() {...}
    void printSettingsMenu() {...}
    //Actualiza el menú sensores con los nuevos datos de los sensores
    void updateSensorsMenu() {...}
    void updatePredictMenu() {...}
    void updateSettingMenu() {...}
    //Printeo de logos en el menú principal
    void drawGraphLogo() {...}
    void drawIALogo() {...}
    void drawOptionsLogo() {...}
    //Manejo de input de los botones a nivel visual de los menús
    void buttonPressedMainMenu()
```

```

void buttonPressedSensorsMenu() {...}
void buttonPressedPredictMenu() {...}
void buttonPressedSettingsMenu() {...}
//Función NOP
void nop() {}

```

4.5 Loop

El loop se encarga de lanzar las funciones necesarias para que la aplicación funcione correctamente, así como hacer las comprobaciones del estado de las conexiones y ejecutar acciones de forma periódica, como comunicarse con el servidor ThingSpeak, el broker MQTT o reiniciar la conexión WiFi cada cierto tiempo para evitar problemas, para ello dispone de diversos contadores y variables.

```

unsigned long lastTimeSensorRead = millis();
unsigned long lastTimeSensorSend = millis();
unsigned long lastWifiRefresh = millis();
unsigned long reconnectDelay = millis();
void loop() {
    //client es la conexión con el broker MQTT
    //Para que no se quede en bucle en reconnect si no hay conexión
    if (!client.connected() && WiFi.status() == WL_CONNECTED && (labs((long)(millis() - reconnectDelay)) > 20000)) {
        reconnect();
        delay(100);
        reconnectDelay = millis();
        //Si está conectado, revisa el MQTT por nuevos mensajes
    } else if (client.connected() && WiFi.status() == WL_CONNECTED) {
        client.loop();
    }

    //Detecta si se están pulsando botones
    while(TaskButtonHandler()){} //Puede que el usuario quiera seguir interactuando por el menú

    //Lee datos de sensores cada 5 segundos
    if(labs((long)(millis() - lastTimeSensorRead)) > 5000){//cada 5s
        TaskReadSensors();
        lastTimeSensorRead = millis();
    }

    //Envía datos de sensores a ThingSpeak cada x minutos
    if(labs((long)(millis() - lastTimeSensorSend)) > 900000){//cada 15 minutos = 900000ms
        //Si no está conectado al Wifi, que se conecte.
        if(WiFi.status() != WL_CONNECTED){
            startWiFi();
        }
        //Si está conectado al Wifi, que envie datos a TS
        if (WiFi.status() == WL_CONNECTED){
            TaskSendDataToThingSpeak();
            int maxcount = 0;
            //Si no es capaz de establecer la conexión con TS
            while(!TSstate && maxcount<2 && WiFi.status() == WL_CONNECTED){ //Si no ha podido enviar datos, que lo vuelva a intentar 2 veces
                //startWiFi();
                TaskSendDataToThingSpeak();
            }
        }
    }
}

```

```

        delay(500);
        maxcount = maxcount+1;
    }
}
lastTimeSensorSend = millis();
}

//Refrescar conexión WiFi cada x tiempo para evitar problemas
if(labs((long)(millis() - lastWifiRefresh)) > 1700000){//cada ~30min
    startWiFi();
    reconnect();
    lastWifiRefresh = millis();
}

//actualizamos la aplicación
app.update();
delay(50);
}

```

4.5 Envío de datos a ThingSpeak

La plataforma de ThingSpeak recibe los datos mediante una consulta HTTP POST a una ruta definida por nuestra API Writing Key, por lo que se crea un cliente WiFi llamado clientTS para manejar la conexión, se comprueba si la conexión con el servidor TS es posible, si no lo es, intentará reconectarse dos veces más y marcará que la conexión no ha sido exitosa con la variable TSstate, si sí lo es, hacemos una petición POST mediante ClientTS.print(), pasándole en el cuerpo del HTTP los campos que queremos actualizar (que corresponden con el campo de la temperatura, humedad, humedad de la tierra y nivel de luz), luego espera a la respuesta del servidor, si esta llega, la imprime por Serial y indica con la variable TSstate que la conexión ha sido exitosa.

```

//Tarea que envía los datos a la plataforma
void TaskSendDataToThingSpeak(){
    WiFiClient clientTS;

    //String solo para printear por Serial
    String url = "GET /update?api_key=" + String(api_key);
    url += "&field1=" + String(temperatura);
    url += "&field2=" + String(humedad);
    url += "&field3=" + String(soil_moisture);
    url += "&field4=" + String(light) + " HTTP/1.1\n\n";

    Serial.print("Solicitando URL: ");
    Serial.println(url);

    //Nos conectamos a TS, si no podemos, detenemos el envío y reintentamos
    if (!clientTS.connect(server, port)) {
        Serial.println("Connection failed.");
        Serial.println("Waiting 5 seconds before retrying...");
        delay(4000);
        TSstate = false; //No hemos podido conectarnos
        clientTS.stop();
        return;
    }
}

```

```

//Subida de datos
Serial.println("Soil_moisture: "+String(round(soil_moisture))+" ,light:
"+String(round(light)));
String postStr = String(api_key);
postStr += "&field1=" + String(temperatura);
postStr += "&field2=" + String(humedad);
postStr += "&field3=" + String(round(soil_moisture));
postStr += "&field4=" + String(round(light));
postStr += "\r\n\r\n";

//make an HTTP POST request
clientTS.print("POST /update HTTP/1.1\n");
clientTS.print("Host: api.thingspeak.com\n");
clientTS.print("Connection: close\n");
clientTS.print("X-THINGSPEAKAPIKEY: "+String(api_key)+"\n");
clientTS.print("Content-Type: application/x-www-form-urlencoded\n");
clientTS.print("Content-Length: ");
clientTS.print(postStr.length());
clientTS.print("\n\n");
clientTS.print(postStr); //aquí los datos anteriores

int maxloops = 0;

//wait for the server's reply to become available
while (!clientTS.available() && maxloops < 1500) {
    maxloops++;
    delay(1); //delay 1 msec
}
if (clientTS.available() > 0) {
    //read back one line from the server
    String line = clientTS.readString(); // Read from the server response
    // Proceed various line-endings
    line.replace("\r\n", "\n");
    line.replace('\r', '\n');
    line.replace("\n", "\r\n");
    Serial.println(line);
    TSstate = true;
} else {
    Serial.println("client.available() timed out ");
    TSstate = false;
}

Serial.println("Closing connection.");
clientTS.stop();
}

```

4.6 Recepción de datos MQTT

La aplicación también debe recibir las predicciones del broker MQTT, para esto, como se puede observar en el setup, se inicia una conexión con dicho broker, que se va actualizando periodicamente en el loop, para esto, disponemos de una función *reconnect*, encargada de reconectar nuestra aplicación al broker MQTT, esto se intenta tres veces

antes de desistir, la conexión consiste en suscribirnos al tópico especificado en la variable global *subTopic*, además de marcar mediante la variable *MQTTstate* que la conexión ha sido exitosa.

```
//Reconectarse al broker MQTT
void reconnect() {
    // Loop until we're reconnected
    int maxloops = 0;
    //en un principio se intentaba reconectar 3 veces, pero por ahora que lo intente solo 1 vez
    while (!client.connected() && maxloops<1)
    {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect(ID)) {
            //client.setKeepAlive(3600);
            Serial.println("connected");
            client.subscribe(subTopic);
            Serial.print("Subscribed to: ");
            Serial.println(subTopic);
            MQTTstate = true;
        }
        else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            //Serial.println(" try again in 5 seconds");
            MQTTstate = false;
            // Wait 5 seconds before retrying
            delay(5000);
        }
        maxloops = maxloops+1;
    }
}
```

Para la recepción de los mensajes MQTT, disponemos de la función *callback*, la cual recibe el payload del mensaje MQTT, lo lee y guarda en una variable que luego deserializa en un documento Json mediante la librería Json de Arduino, se comprueba si en este JSON existe un campo “message”, su existencia indicaría que ha ocurrido un error, y si no se encuentra, almacena los valores de los campos “pred” y “prediction” para luego ser mostrados en el menú de Predicción, finalmente limpia el documento JSON. Adicionalmente, la variable *err_msg* mantiene registro del error, si ocurriese.

```
//Documento JSON de como máximo 2 campos
const size_t bufferSize = JSON_OBJECT_SIZE(2);
DynamicJsonDocument jsonDoc(bufferSize);
//Callback de la conexión al broker MQTT
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    String payl = "";
    //guardamos el payload
    for (int i=0;i<length;i++) {
        payl += (char)payload[i];
```

```

    Serial.print((char)payload[i]);
}
//Deserializamos el Json
DeserializationError error = deserializeJson(jsonDoc, payl);
if (error) {
    Serial.print("Error al parsear JSON: ");
    Serial.println(error.c_str());
    err_msg = "Algo ha salido mal";
    jsonDoc.clear();
    return;
}
//Si hay un campo "message" es porque ha ocurrido algo
if(!jsonDoc["message"].isNull()){
    err_msg = jsonDoc["message"].as<String>();
    jsonDoc.clear();
    return;
}

int pred = jsonDoc["pred"].as<int>();
//pred es un número, por lo que hay que sacar la predicción que indica
prediction = OUTPUTS[pred];
probability = jsonDoc["prec"].as<float>();
err_msg = "";
jsonDoc.clear();
Serial.println();
}

```

La estructura de los mensajes MQTT es la siguiente:

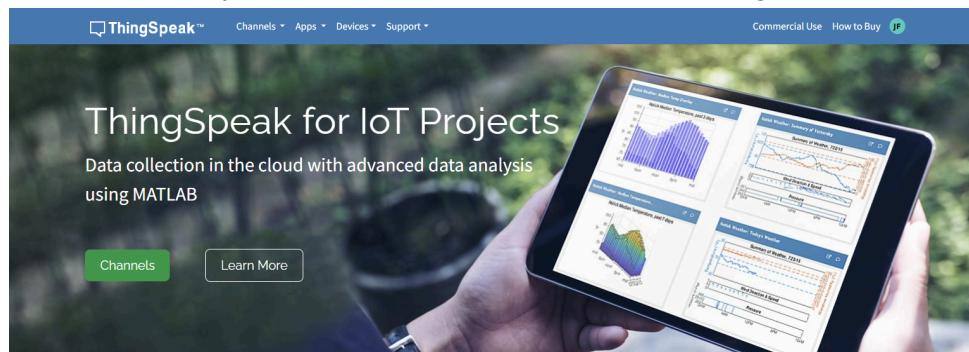
UserW1 = {"pred": "0", "prec": "0.93887115"}

El tópico, como se observa en la sección de variables globales es:

idc/wioterminal/UserW1

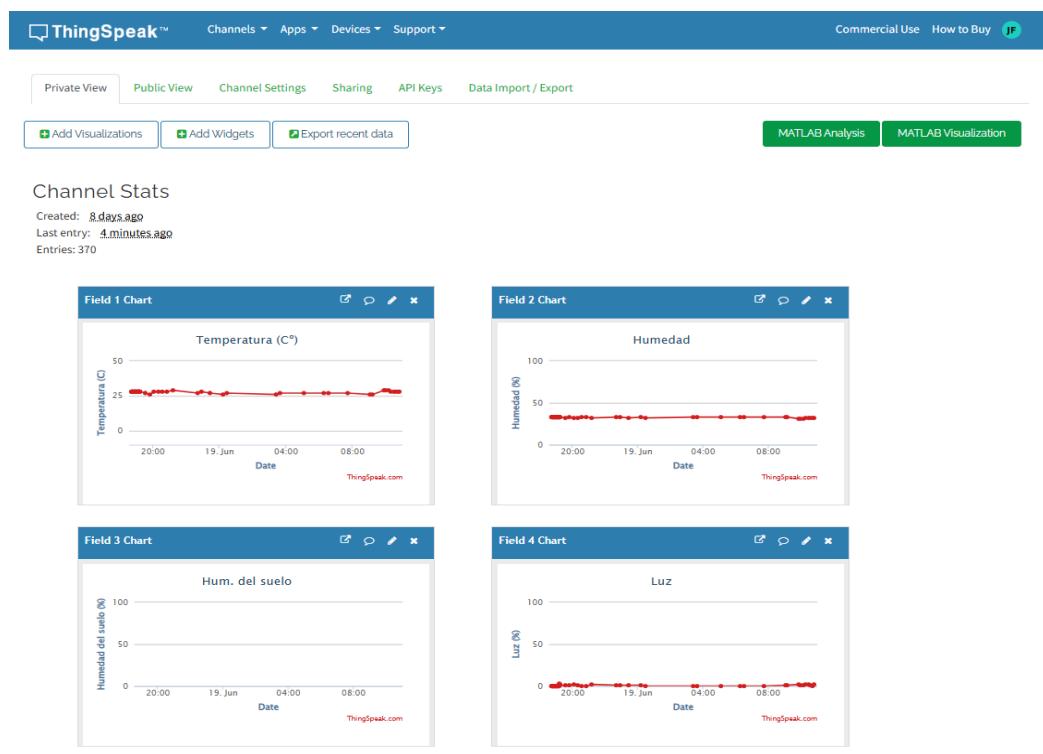
5. Plataforma ThingSpeak

ThingSpeak es una plataforma de Internet de las Cosas (IoT) que permite la recopilación, visualización y análisis de datos en tiempo real provenientes de dispositivos y sensores. Facilita el almacenamiento de datos en la nube y su posterior procesamiento mediante gráficos, widgets y la integración con MATLAB para análisis avanzados. Con ThingSpeak, podemos monitorizar los datos de manera continua, configurar alertas basadas en umbrales específicos, y acceder a sus datos desde cualquier lugar.

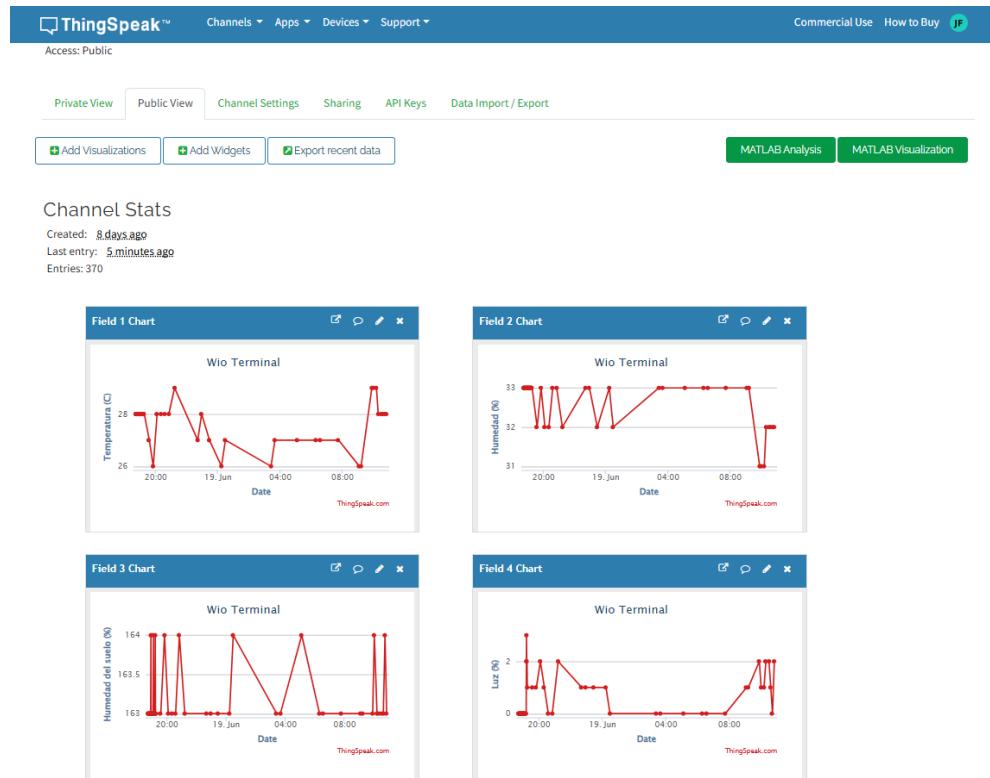


5.1 Web de ThingSpeak

Esta es la principal forma de acceder a ThingSpeak, se trata de una plataforma gratuita que permite enviar datos desde dispositivos conectados a internet y visualizar esos datos a través de gráficos y otro tipo de visualizaciones (en nuestro caso será mediante gráficos). Nosotros hemos creado y configurado un Canal nuevo, llamado Wio Terminal, en él contamos con 4 campos donde se muestran los datos de temperatura, humedad, humedad del suelo y luz. Cada gráfica está creada de forma personalizada para cada dato que almacena.



También disponemos de una ventana con la vista pública, la cual puede configurarse de forma diferente a la anterior, que se trataba de la vista privada.



En la pestaña de “*Channel Settings*” podemos ver y personalizar las opciones del canal, en nuestro caso queda de la siguiente forma:

Field	Mapping
Field 1	Temperatura (C)
Field 2	Humedad (%)
Field 3	Humedad del suelo (%)
Field 4	Luz (lx)
Field 5	(disabled)

Además nos da la ID del canal necesaria para poder enviar los datos. Como se puede observar, podemos configurar hasta 8 campos, pero para este proyecto será suficiente con 4.

La siguiente ventana “API Keys” muestra las llaves necesarias para poder escribir y leer datos de los campos. Write API Key nos permite enviar datos a los campos, y Read API Key es la que nos permite leer los datos de los campos.

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Write API Key

Key TZNMIJZKWEIX2BS6

Generate New Write API Key

Read API Keys

Key SDFC4QG0TT44L3C3

Note

Save Note Delete API Key

Add New Read API Key

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click [Generate New Write API Key](#).
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click [Generate New Read API Key](#) to generate an additional read key for the channel.
- **Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

API Requests

Write a Channel Feed
GET https://api.thingspeak.com/update?api_key=TZNMIJZKWEIX2BS6&field1=0

Read a Channel Feed
GET <https://api.thingspeak.com/channels/2573962/feeds.json?results=2>

Read a Channel Field
GET <https://api.thingspeak.com/channels/2573962/fields/1.json?results=2>

Read Channel Status Updates
GET <https://api.thingspeak.com/channels/2573962/status.json>

Learn More

Alerts API Key TAK8d0Y8G509d8/regM

Además, en la información de nuestra cuenta disponemos de una Alert API Key para configurar las alertas por correo.

En la última pestaña disponible “Data Import/ Export” simplemente hemos configurado la zona horaria a la de España.

Wio Terminal

Channel ID: **2573962**
Author: [mwa000034218789](#)
Access: Public

Datos de los sensores del Wio Terminal

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Import

Upload a CSV file to import data into this channel.

File Elegir archivo No se ha sele...ningún archivo

Time Zone (GMT+01:00) Madrid

Upload

Help

The correct format for data import is provided in this [CSV Import Template File](#). Use the field names *field1*, *field2*, and so on, instead of custom field names.

CSV Import Format

```
created_at,field1,field3,field4,field8,elevation  
2019-01-01T10:11:12-05:00,11,33,44,88,10
```

Other Import and Export Options

You can also use MATLAB, the REST API, or the MQTT API to import and export channel data.

[Read Data](#)

[Write Data](#)

Export

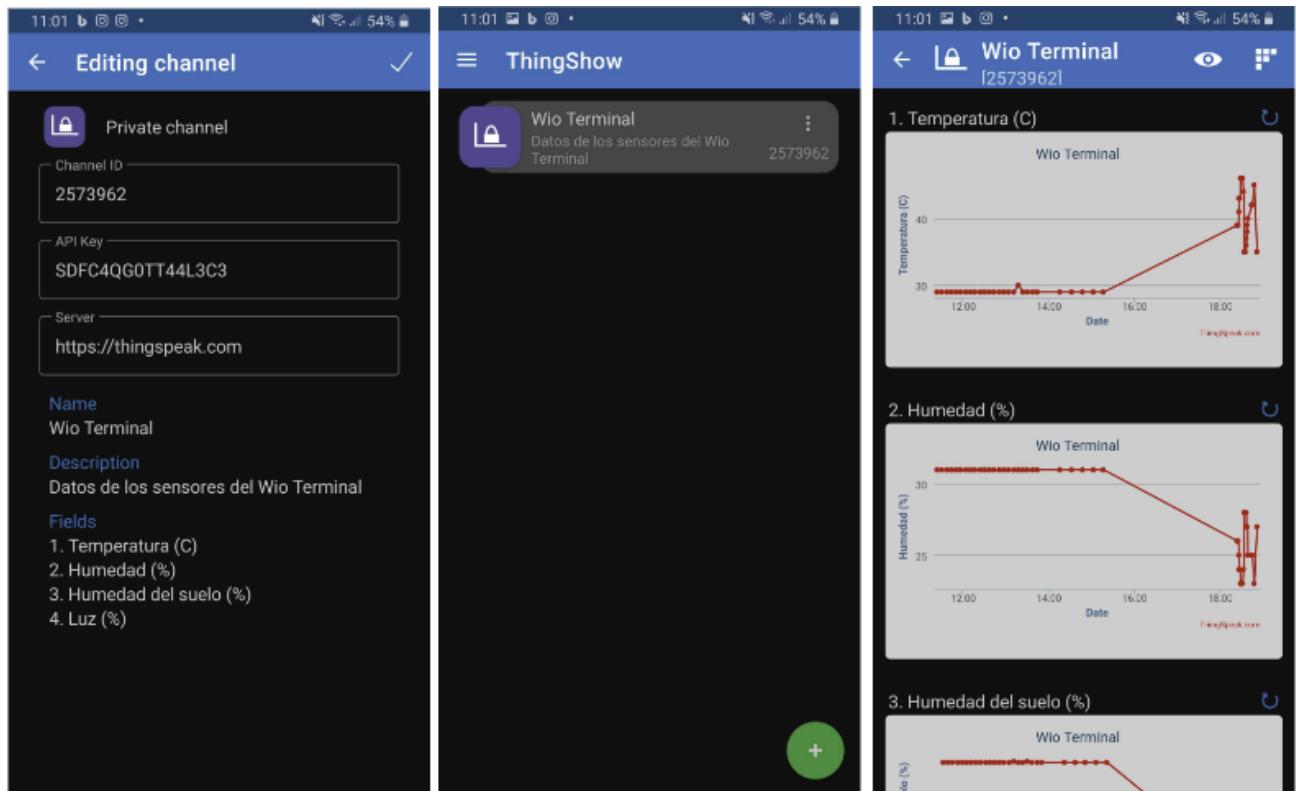
Download all of this Channel's feeds in CSV format.

Time Zone (GMT+01:00) Madrid

Download

5.2 App móvil de ThingSpeak

Una de las razones para escoger la plataforma de ThingSpeak era su flexibilidad a la hora de acceder a los datos, un ejemplo de esto es las numerosas aplicaciones para móvil que existen y que nos permiten añadir nuestro canal mediante su ID y la clave de lectura para así consultar nuestros datos desde cualquier lugar, en esta ocasión hemos decidido usar una aplicación llamada "[ThingShow](#)".



5.3 Alertas en ThingSpeak

Adicionalmente a lo anterior, ThingSpeak también permite configurar alertas, aunque este proceso es algo más complejo que en otras plataformas debido a que hay que hacer uso de Matlab.

Empezamos añadiendo un script en MATLAB Analysis.

The screenshot shows the ThingSpeak MATLAB Analysis interface. At the top, there's a navigation bar with 'ThingSpeak™', 'Channels', 'Apps', 'Devices', and 'Support'. Below the navigation bar, a breadcrumb trail shows 'Apps / MATLAB Analysis / TriggerPrueba1 /'. A dropdown menu for 'Apps' is open, showing 'All Apps' and 'MATLAB Analysis'. The main area displays a table with two columns: 'Name' and 'Created'. There is one row in the table, which contains the name 'TriggerTemperaturaHumedad' and the creation date '2024-06-10'.

Name	Created
TriggerTemperaturaHumedad	2024-06-10

Dentro de este script, usaremos nuestra Clave de Alertas para obtener los permisos de enviar un correo de alerta, así como la ID de nuestro canal, para recoger los datos que nos interesen. Luego configuraremos lo necesario para generar un aviso, obtenemos el último dato de temperatura y humedad, y si estos están por encima o por debajo de un límite, se generará una alerta por correo.

```
% Store the channel ID for the moisture sensor channel.
channelID = 2573962;

% Provide the ThingSpeak alerts API key. All alerts API keys start with TAK.
alertApiKey = 'TAK8d0Y8G509d8/regM';

% Set the address for the HTTP call
alertUrl="https://api.thingspeak.com/alerts/send";

% webwrite uses weboptions to add required headers. Alerts needs a
% ThingSpeak-Alerts-API-Key header.
options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key", alertApiKey ]);

% Set the email subject.
alertSubject = sprintf("Alerta del estado de la planta");
alertBody = "";
% Read the recent data.
tempData = thingSpeakRead(channelID, 'NumDays', 30, 'Fields', 1);
moistureData = thingSpeakRead(channelID, 'NumDays', 30, 'Fields', 3);
% Check to make sure the data was read correctly from the channel.
if isempty(moistureData)
    alertBody = ' No data read from plant. ';
elseif isempty(tempData)
    alertBody = ' No data read from plant. ';
else
    % Calculate a 10% threshold value based on recent data.
    tspan = max(tempData) - min(tempData);
    tValue = 0.1 * tspan + min(tempData);

    smspan = max(moistureData) - min(moistureData);
    smValue = 0.1 * smspan + min(moistureData);

    % Get the most recent point in the array of moisture data.
    tlastValue = tempData(end);
```

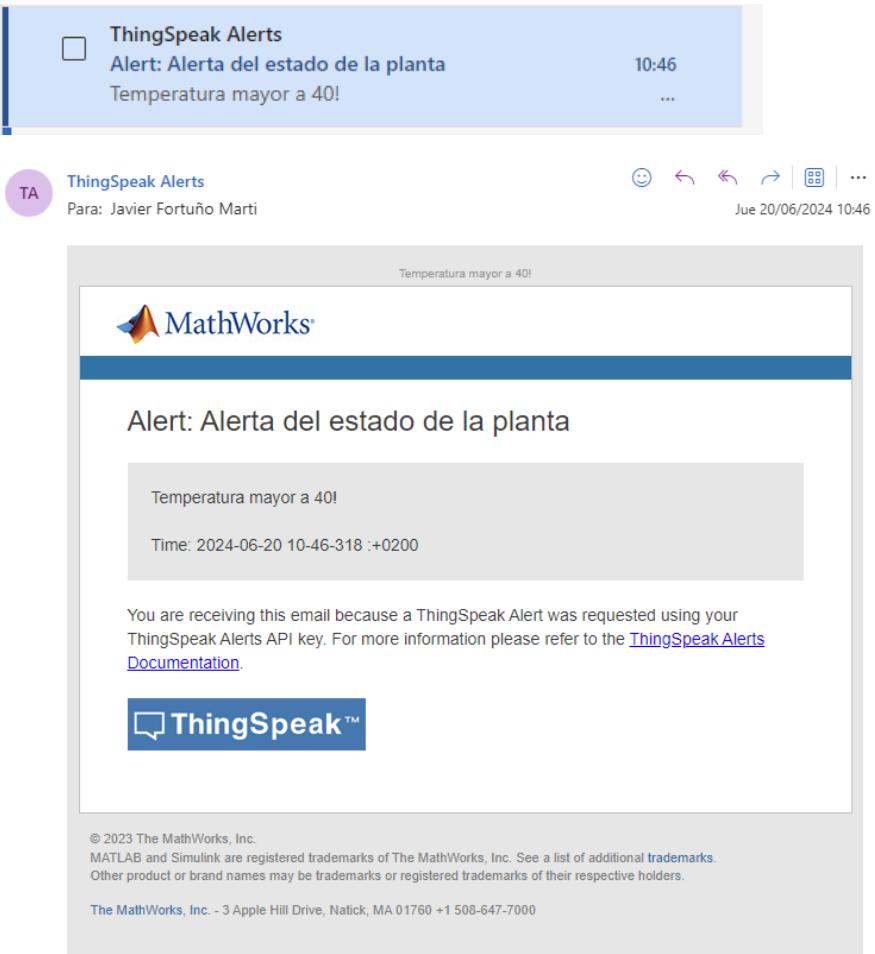
```

smlastValue = moistureData(end);

% Set the outgoing message
%fprintf("ejecutando");
if (tlastValue>40 || smlastValue<20)
    if(tlastValue>40)
        alertBody = 'Temperatura mayor a 40!\n';
        fprintf("TEMPERATURA > 50");
    end
    if(smlastValue<20)
        alertBody += 'Nivel de humedad muy bajo, riegame!\\n';
        fprintf("Soil Moisture < 20");
    end
try
    webwrite(alertUrl , "body", alertBody, "subject", alertSubject, options);
catch someException
    %fprintf("Failed to send alert: %s\\n", someException.message);
end
end
end

```

Para probar si funciona, hemos escrito un corto script en Python que envía una temperatura de 50 al primer campo, el resultado es un correo avisándonos de esto.



Ahora solo falta configurar la plataforma para que ejecute el script Matlab, por ejemplo, cada 15 minutos, esto se hace mediante la herramienta “*Time Control*” de ThingSpeak, en ella creamos un nuevo TimeControl llamado TempCheck que lanzará el script especificado (TriggerTemperaturaHumedad) cada x periodo de tiempo.

The screenshot shows the ThingSpeak web interface. At the top, there is a navigation bar with links for 'Channels', 'Apps', 'Devices', and 'Support'. The 'Apps' link is currently selected, and a dropdown menu is open, listing 'All Apps', 'MATLAB Analysis', 'MATLAB Visualizations', 'Plugins', 'ThingTweet', and 'TimeControl'. The 'TimeControl' option is highlighted.

In the main content area, there is a sub-navigation bar with 'Apps / TimeControl' and a 'New TimeControl' button. Below this, the title 'Recurring TimeControls' is displayed. A table lists a single recurring time control:

Name	Recurrence	Last Ran	Run At
<input checked="" type="checkbox"/> TempCheck	Every 15 minutes	2024-06-19 10:51 pm	2024-06-19 11:05 pm

Below the table, there is another sub-navigation bar with 'Apps / TimeControl / TempCheck' and an 'Edit TimeControl' button. The edit form contains the following fields:

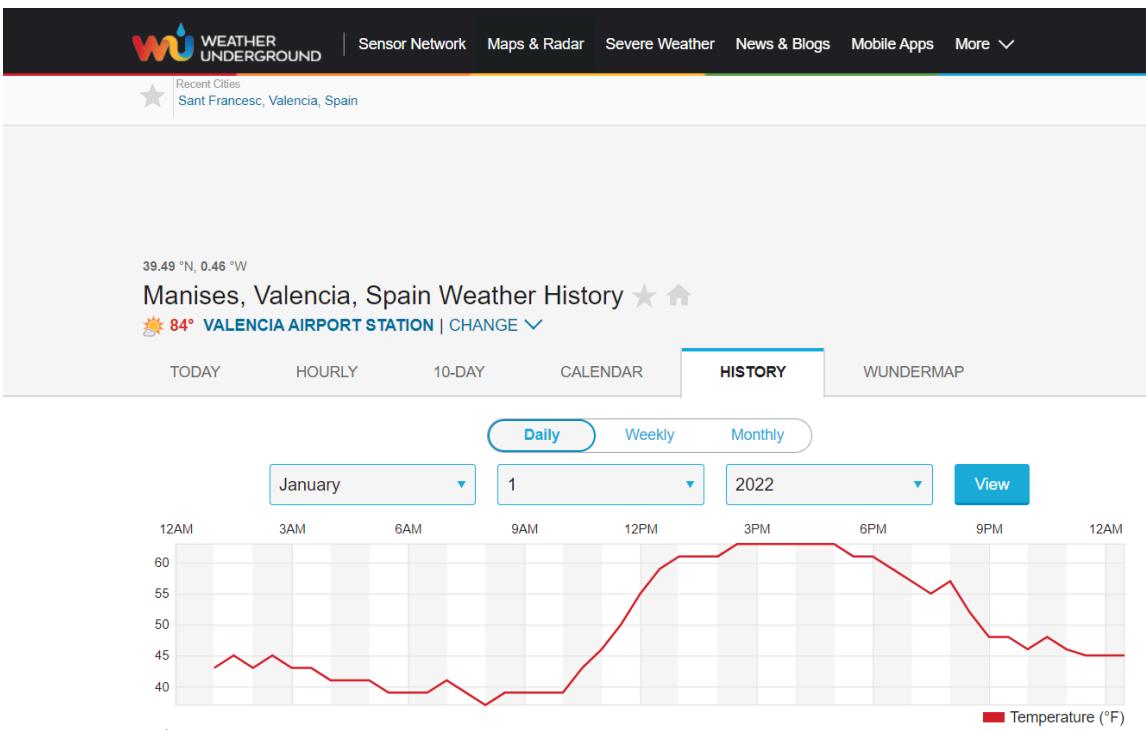
Name:	TempCheck
Frequency:	Every 15 minutes
Time Zone:	Madrid (edit)
Last Ran:	2024-06-19 10:51 pm
Run At:	2024-06-19 11:05 pm
Fuzzy Time:	± 0 minutes
MATLAB Analysis:	TriggerTemperaturaHumedad
Created:	2024-06-10 1:29 pm

6. TinyML

Nuestra aplicación debe ser capaz de inferir el tiempo actual/en el corto plazo, por lo que es necesario entrenar un modelo de red neuronal que se encargue de esto, este proceso empieza con el *scrapping* del dataset a utilizar, luego habrá que filtrar limpiar estos datos, prepararlos para el entrenamiento, entrenar la red neuronal, optimizarla, guardarla y usarla.

6.1 Scrapping

Hemos utilizado los datos de los últimos 2 años de temperatura y humedad de la ciudad de Valencia a intervalos de media hora, para ello hemos usado la página Web de [Wunderground](#), la cual almacena de forma accesible los datos meteorológicos de un gran número de ciudades.



Daily Observations

Time	Temperature	Dew Point	Humidity	Wind	Wind Speed	Wind Gust	Pressure	Precip.	Condition
1:00 AM	43 °F	41 °F	93 %	W	5 mph	0 mph	30.09 in	0.0 in	Fair
1:30 AM	45 °F	41 °F	87 %	NW	3 mph	0 mph	30.09 in	0.0 in	Fair
2:00 AM	43 °F	41 °F	93 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
2:30 AM	45 °F	41 °F	87 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
3:00 AM	43 °F	41 °F	93 %	WNW	6 mph	0 mph	30.09 in	0.0 in	Fair
3:30 AM	43 °F	39 °F	87 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
4:00 AM	41 °F	37 °F	87 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
4:30 AM	41 °F	37 °F	87 %	W	2 mph	0 mph	30.09 in	0.0 in	Fair
5:00 AM	41 °F	37 °F	87 %	W	2 mph	0 mph	30.09 in	0.0 in	Fair
5:30 AM	39 °F	37 °F	93 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
6:00 AM	39 °F	37 °F	93 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
6:30 AM	39 °F	37 °F	93 %	NW	5 mph	0 mph	30.09 in	0.0 in	Fair
7:00 AM	41 °F	37 °F	87 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
7:30 AM	39 °F	37 °F	93 %	W	5 mph	0 mph	30.09 in	0.0 in	Fair
8:00 AM	37 °F	36 °F	93 %	WNW	3 mph	0 mph	30.09 in	0.0 in	Fair
8:30 AM	39 °F	36 °F	87 %	NW	5 mph	0 mph	30.09 in	0.0 in	Fair
9:00 AM	39 °F	36 °F	87 %	NW	5 mph	0 mph	30.09 in	0.0 in	Fair
9:30 AM	39 °F	36 °F	87 %	W	5 mph	0 mph	30.09 in	0.0 in	Fair

Hemos utilizado la técnica de “Web Scraping” para poder obtener los datos de la página web. Para ello hemos usado un corto script de python que obtiene los datos que necesitamos mediante la URL de la página, el rango de fechas para obtener los datos y la ruta al ejecutable de ChromeDriver y su configuración, este es un ejecutable que permite a los navegadores Chrome interactuar con scripts de automatización. Facilita la navegación, pruebas y automatización de tareas en Chrome a través de comandos enviados desde estos scripts.

El código busca la tabla que devuelve la consulta de la url anterior, y dentro de esta guarda los encabezados de la tabla (th), recorre cada fila de ésta (tr) y almacena los valores de sus celdas (td), que va guardando en la lista *data*.

Cuando ya tenemos todos los datos, los almacenamos en un archivo .csv para usarlos en el entrenamiento.

```
# Bucle para recorrer las fechas y obtener los datos
while start_date <= end_date:
    print('Gathering data from:', start_date)
    formatted_lookup_URL = lookup_URL.format(start_date.year, start_date.month,
start_date.day)

    driver.get(formatted_lookup_URL)
    print("Opened URL")

    # Espera hasta que las tablas estén presentes en la página
    tables = WebDriverWait(driver,
20).until(EC.presence_of_all_elements_located((By.CSS_SELECTOR, "table")))

    # Parsear el contenido de la página con BeautifulSoup
    soup = BeautifulSoup(driver.page_source, 'lxml')
    tables = soup.find_all('table')
    if len(tables) > 1: # Asegúrate de que la tabla existe
        table = tables[1]

        # Obtener los encabezados de la tabla
        table_head = table.findAll('th')
        output_head = []
        for head in table_head:
            output_head.append(head.text.strip())

        # Escribir las filas de la tabla
        output_rows = []
        rows = table.findAll('tr')
        for j in range(1, len(rows)):
            table_row = rows[j]
            columns = table_row.findAll('td')
            output_row = []
            for column in columns:
                value = column.text.strip()
                output_row.append(value)

            if len(output_row) > 0:
                output_row = [start_date.date()] + output_row
                data.append(output_row)

    # Incrementar la fecha en un día
    start_date += timedelta(days=1)

# Escribir los datos en un archivo CSV
print("Now, Write the Data.")
output_head = ["Date"] + output_head
df = pd.DataFrame(data, columns=output_head)
df.to_csv("./WeatherData.csv", index=False)
print("Raw data has been output as CSV for future use.")

# Cerrar el driver
driver.quit()
print("Program Complete!")
```

6.2 Entrenamiento del modelo.

El siguiente paso es entrenar nuestro modelo mediante TensorFlow. Para ello hemos utilizado el entorno de Python Notebooks (Jupyter Notebooks).

Lo primero que vamos hacer es cargar los datos del csv y eliminar las columnas que no vamos a necesitar, así como formatear con los valores adecuados las columnas que sí nos interesan, siendo estas Temperatura (en Kelvins), Humedad, Mes y Condición (tiempo meteorológico).

```
import pandas as pd
import numpy as np

data = pd.read_csv('WeatherData.csv')
data.iloc[:7]
data = data.drop(columns = ["Time", "Dew Point", "Wind", "Wind Speed", "Wind Gust",
"Pressure", "Precip."])
data["Temperature"] = data["Temperature"].apply( lambda x: round((int(x[:2])-32) * 5/9 + 273, 1) )
data["Humidity"] = data["Humidity"].apply(lambda x: int(x[:2]))
data['Date'] = pd.to_datetime(data['Date']) #Esto convierte la columna Date a un formato
datetime, lo que permite utilizar métodos como .dt.month
data['Month'] = data['Date'].dt.month #El método .dt.month extrae el número del mes de la
fecha.
data = data.drop(columns = ["Date"])
labels = data.Condition.unique()
labels
```

A continuación agrupamos todos los posibles estados del tiempo (Condition) en tres grandes grupos de estados, No llueve, Puede llover y Llueve y prepara el output de los datos, añadiendo estos estados/columnas a los datos originales y asignando a cada muestra su estado correspondiente de entre los tres. Finalmente, eliminamos la columna condición.

```
processed_data = data.copy()
norain = ['Mostly Cloudy', 'Fair', 'Partly Cloudy', 'Haze', 'Fog', 'Mostly Cloudy / Windy',
'Cloudy', 'Partly Cloudy / Windy']
mightrain = ['Showers in the Vicinity', 'Thunder in the Vicinity', 'Thunder']
rain = ['Rain Shower', 'Light Rain Shower', 'Light Rain', 'Rain', 'Heavy Rain', 'Light Rain
with Thunder', 'Heavy Rain Shower', 'Heavy T-Storm', 'T-Storm', 'Light Rain / Windy']

processed_data["NoRain"] = 0
processed_data["MightRain"] = 0
processed_data["Rain"] = 0

processed_data["NoRain"].loc[data.Condition.isin(norain)] = 1
processed_data["MightRain"].loc[data.Condition.isin(mightrain)] = 1
processed_data["Rain"].loc[data.Condition.isin(rain)] = 1
processed_data = processed_data.drop(columns="Condition")
processed_data
```

Seguidamente creamos un conjunto de datos de entrada 'x' y un conjunto de etiquetas 'y' a partir del DataFrame 'processed_data' anterior. Las etiquetas 'y' corresponden a las categorías climáticas ("NoRain", "MightRain", "Rain") excluyendo las primeras 7 filas, mientras que los datos de entrada 'x' se generan a partir de ventanas de 7 medidas de las demás columnas, aplanadas y acumuladas en un array. Finalmente, muestra las

dimensiones de 'x' y 'y', siendo el input 'x' un array de 18 elementos, 6 muestras (cada una con Temperatura, Humedad y Mes, 18 valores) por lo el input sería: [*temperatura muestra 1, humedad muestra 1, mes de la subida de la muestra 1, ..., temperatura muestra 6, humedad muestra 6, mes de la subida de la muestra 6*]. Mientras que el output se trata de un array de tres elementos donde cada posición corresponde con un estado meteorológico [*No llueve, Puede Llover, Llueve*], por ejemplo, si la predicción fuese que no llueve, la salida sería [1,0,0].

```
y = processed_data[["NoRain", "MightRain", "Rain"]].to_numpy()[7:]
rawx = processed_data.drop(columns = ["NoRain", "MightRain", "Rain"]).to_numpy()
x = []
temp = np.array(0)
for i in range(len(rawx)-7):
    temp = rawx[i:i+6].flatten()
    #print(temp)
    x.append(temp)
x = np.array(x)
x.shape, y.shape
((33237, 18), (33237, 3))
```

A continuación importamos las bibliotecas necesarias para empezar el entrenamiento: TensorFlow, Keras y funciones scikit-learn, luego dividimos los datos en conjuntos de entrenamiento y prueba.

```
import tensorflow as tf
import keras
from keras import layers
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, random_state=33,
stratify=y)
```

Ahora vamos a definir, compilar y entrenar un modelo de red neuronal utilizando TensorFlow y Keras. El modelo se entrena durante 40 épocas con un 10% de los datos de entrenamiento reservado para validación y un tamaño de lote de 128. Finalmente, el rendimiento del modelo se evalúa en el conjunto de prueba y el modelo entrenado se guarda en un archivo llamado 'model' para su posterior uso en la inferencia mediante python.

```
model = tf.keras.Sequential()
model.add(layers.Dense(14, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))
optimizer = tf.optimizers.Adam(learning_rate=0.0001)
model.compile(loss = 'categorical_crossentropy',
              optimizer = optimizer,
              metrics=['accuracy'])

model.fit(xtrain, ytrain, epochs=40, validation_split=0.1, batch_size=128)

test_results = model.evaluate(xtest, ytest, verbose=1)
```

```
model.save('model.h5')
```

6.3 Inferencia

Una vez obtenido el modelo, ya podemos empezar a trabajar en la inferencia, para la cual, aunque en un principio la habíamos realizado en Arduino C++, hemos terminado usando Python.

Para esto, empezamos importando la librería tensorflow y numpy, cruciales para realizar las inferencias. Luego de esto definiremos una función predict_with_model, la cual se llamará al momento de realizar la inferencia, esta tiene como input el array de datos sobre el cual se realizará esta (16 datos, 6 muestras), cargará un modelo pre entrenado de Keras desde el archivo "model.h5" y utiliza ese modelo para hacer las predicciones. La función convierte los elementos de entrada a flotantes, asegura que la entrada tenga la forma correcta (añadiendo una dimensión si es necesario), y luego realiza la predicción. La salida del modelo, que se asume como probabilidades de diferentes clases, se procesa para determinar la clase predicha y la confianza correspondiente. Si ocurre un error durante el proceso, la función captura la excepción, imprime un mensaje de error y devuelve False para que el script que la ha llamado pueda detectar el error.

```
import tensorflow as tf
import numpy as np

# Ruta al modelo preentrenado
MODEL_PATH = "model.h5"

def predict_with_model(input_array):
    try:
        # Cargar el modelo
        model = tf.keras.models.load_model(MODEL_PATH)
        print("Modelo cargado exitosamente.")

        # Asegurarse de que input_array tenga el formato adecuado
        input_array = np.array([float(x) for x in input_array])
        #input_array = np.asarray(input_array)

        if len(input_array.shape) == 1:
            # Añadir una dimensión extra si es un array de una sola dimensión
            input_array = np.expand_dims(input_array, axis=0)

        # Realizar predicción
        predictions = model.predict(input_array)

        # Asumiendo que la salida del modelo es de la forma [probabilidades]
        predicted_class = np.argmax(predictions, axis=-1)
        confidence = np.max(predictions, axis=-1)

        # Devuelve la clase predicha y la confianza
        return predicted_class[0], confidence[0]
    except Exception as e:
        print(f"Error al realizar la predicción: {e}")
        return False
```

6.4 Código Main de la Inferencia

La función de inferencia anterior será llamada por otro script, el Main, encargado de recibir los datos de la plataforma ThingSpeak y enviar el resultado de la predicción al broker MQTT (broker.hivemq.com), este es el script principal del proceso de inferencia.

En esta primera parte del código hemos importado las librerías necesarias además de la función de inferencia anterior.

Hacemos la configuración MQTT en la que definimos la dirección del broker y el topic (“idc/witerminal/UserW1”).

Luego definimos los callbacks que utiliza el cliente de MQTT.

1. “on_connect”: Se llamará cuando el cliente se conecte al broker MQTT. Imprime un mensaje de confirmación con el host y puerto del broker, así como las flags y el código de retorno de la conexión.
2. “on_publish”: Se llamará cuando el cliente publique un mensaje. Imprime el mensaje indicando que el mensaje se ha publicado, incluyendo el ID del mensaje (mid).

Los datos se van a obtener desde la plataforma ThingSpeak a la que previamente el Wio Terminal le habrá enviado los datos de lectura de los sensores, para esto, se realiza dos consultas HTTP GET a una url que contiene la ID de nuestro canal, que previamente hemos hecho público, así como el campo de datos que se solicita y el número de estos (empezando por el último) así como el mes en el que fueron subidas las muestras.

De todo esto último se encarga la función get_field_data que tiene como input la url del campo de datos, que se define en las variables url_field, ésta realiza una solicitud GET a una URL, procesa la respuesta JSON para extraer valores específicos de campos (temperatura y humedad) y los meses de subida de estos datos, también maneja posibles errores durante estos procesos. Al final devuelve dos listas: una con los valores de los campos y otra con los meses de carga.

```
import requests
from Inferencia import predict_with_model
import numpy as np
from datetime import datetime
import paho.mqtt.client as mqtt
import json
import time

#--CONFIGURACIÓN MQTT--
THE_BROKER = "broker.hivemq.com"
THE_TOPIC = "idc/witerminal/UserW1"

# Callback function used when the client receives a CONNACK response from the broker.
def on_connect(client, userdata, flags, rc):
    print("connected to ", client._host, "port: ", client._port)
    print("flags: ", flags, "returned code: ", rc)

# Callback function used when a message is published.
def on_publish(client, userdata, mid):
    print("msg published (mid={})".format(mid))

#--CONFIGURACIÓN ThingSpeak--
read_api_key = "SDFC4QG0TT44L3C3"
```

```

channel_id = "2573962"

num_results = 6 # Número de resultados a obtener

# URLs para los datos
url_field1 =
f"https://api.thingspeak.com/channels/{channel_id}/fields/1.json?results={num_results}"
url_field2 =
f"https://api.thingspeak.com/channels/{channel_id}/fields/2.json?results={num_results}"

def get_field_data(url):
    try:
        response = requests.get(url)
        response.raise_for_status() # Levanta un error si la solicitud falló
        data = response.json()
        feeds = data['feeds']

        data_values = [feed['field1'] if 'field1' in feed else feed['field2'] for feed in feeds]
        upload_months = [datetime.strptime(feed['created_at'], "%Y-%m-%dT%H:%M:%S") .month
for feed in feeds]

        return data_values, upload_months
    except requests.RequestException as e:
        print(f"Error en la solicitud: {e}")
        return [], []
    except ValueError as e:
        print(f"Error al procesar la respuesta: {e}")
        return [], []

```

Por último solo queda realizar la predicción y enviarla al broker MQTT. Esto se implementa un bucle principal *main* que realiza varias tareas:

- Obtiene los datos de los campos temperatura y humedad a través de la función “get_field_data” y las dos urls anteriormente definidas, verificando el formato de los datos.
- Hace predicciones llamando a la función explicada anteriormente que realiza la inferencia, esta función se llama mediante *predict_with_model(input array)*.
- Genera los mensajes de respuesta dependiendo del resultado de la inferencia.
- Realizar la conexión con el broker y publicar los resultados..

Explicado a detalle, el bucle principal se ejecuta continuamente “while True” para mantener la aplicación en funcionamiento de manera constante, este bucle se ejecutará cada 30 minutos (1800 segundos) gracias al *time.sleep* presente al final del bucle.

Se obtienen los datos de temperatura y humedad y los meses de las medidas de la gráficas de Thingspeak.

Se verifica que se han obtenido las muestras suficientes para hacer la predicción. Si falta alguna imprime un mensaje indicando que se necesitan más muestras y se prepara un mensaje MQTT de advertencia.

Si se tienen suficiente muestras, se crea un array (“predecir_arr”) que contiene datos de temperatura, humedad y meses, de las seis muestras.

Se llama a la función “predict_with_model” con el array de predicciones. Si la predicción es exitosa, se obtiene el resultado y se prepara un mensaje MQTT con la predicción y su confianza. Si la predicción falla, se prepara un mensaje MQTT indicando que se necesitan más muestras.

A continuación se configura un cliente MQTT(“mqtt.Client”) con conexiones y configuraciones necesarias, se genera un mensaje JSON a partir del mensaje con el resultado de la inferencia, se inicia el bucle del cliente MQTT(“client.loop_start()”), se publica el mensaje en el topic THE_TOPIC (siendo este `idc/wioterminal/UserW1`) y se detiene el bucle después de la publicación, por último, se espera 30 minutos antes de comenzar el próximo ciclo del bucle principal.

```

if(__name__=="__main__"):
    while True:
        # Obtener datos de ambos campos
        temperatura_ar, meses_subida_temp = get_field_data(url_field1)
        humedad_ar, _ = get_field_data(url_field2)
        message_to_mqtt = {}
        print(temperatura_ar)
        print(meses_subida_temp)
        print(humedad_ar)
        if (len(temperatura_ar) != num_results or len(humedad_ar) != num_results):
            print("Se necesitan más muestras")
            message_to_mqtt = {"message": "Se necesitan más muestras"}
        else:
            print(f"Últimos {num_results} datos del campo 1: {temperatura_ar}")
            print(f"Últimos {num_results} datos del campo 2: {humedad_ar}")
            print(f"Últimos {num_results} datos del campo 3: {meses_subida_temp}")
            #Ahora hay que crear el array de predicciones
            predecir_arr = []
            for i in range(num_results):
                predecir_arr.append(temperatura_ar[i])
                predecir_arr.append(humedad_ar[i])
                predecir_arr.append(meses_subida_temp[i])

            print(len(predecir_arr))
            print(predecir_arr)
            #message_to_mqtt = {"pred":"Llueve", "prec":"0.45"}
            result = predict_with_model(predecir_arr)
            if result is not False:
                prediction, confidence = result
                print(f"Predicción: {prediction}, Confianza: {confidence}")
                message_to_mqtt = {
                    "pred":str(prediction),
                    "prec": str(confidence)
                }
            else:
                print("Falló la predicción.")
                message_to_mqtt = {"message": "Se necesitan más muestras"}

    client = mqtt.Client(client_id="",
                        clean_session=True,
                        userdata=None,
                        protocol=mqtt.MQTTv311,
                        transport="tcp")

```

```

client.on_connect = on_connect
client.on_publish = on_publish

client.username_pw_set(username=None, password=None)
client.connect(THE_BROKER, port=1883, keepalive=60)

json_message = json.dumps(message_to_mqtt)

client.loop_start()

client.publish(THE_TOPIC,
              payload=json_message,
              qos=0,
              retain=True)
client.loop_stop()
time.sleep(1800)

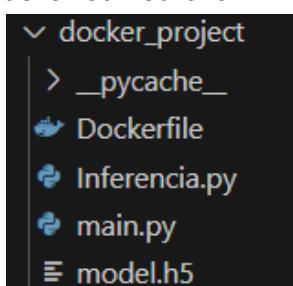
```

7. Dockerización de la inferencia

Después de los pasos anteriores, disponemos de un código que obtiene los datos, realiza una inferencia y publica su resultado.

Pero, ¿quién se encarga de ejecutarlo? La respuesta es: cualquiera, en nuestro caso hemos pensado en usar un dispositivo discreto y capaz de mantenerse encendido durante muchas horas con un bajo consumo, pero esto será explicado en el próximo punto, para llegar a ejecutarlo sobre cualquier máquina, es necesario encapsular el código, para ello hemos creado una imagen Docker que se encargará de solucionar todas las dependencias y ejecutar nuestro código gracias a los contenedores Docker.

El contenido de este contenedor constará de nuestro código main que realiza las conexiones y llama a la inferencia, el código que realiza la propia inferencia y el modelo .h5 de la red neuronal.



Seguidamente se escribe el archivo Dockerfile que creará nuestra imagen, instalando las dependencias necesarias, configurando el entorno de ejecución y lanzando el script *main*:

```

# Usar una imagen base de Python
FROM python:3.9-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copiar los archivos necesarios al directorio de trabajo del contenedor
COPY main.py Inferencia.py model.h5 /app/

# Instalar las dependencias necesarias
RUN pip install requests paho-mqtt tensorflow numpy

# Comando por defecto para ejecutar el contenedor
CMD ["python", "main.py"]

```

Y finalmente se ejecuta la siguiente secuencia de comandos dentro del directorio donde se encuentra el Dockerfile con la finalidad de crear la imagen y subirla al repositorio de Docker, haciéndola pública a cualquiera que quiera descargarla.

```
> docker build -t idcwioterminal
> docker login
> docker tag idcwioterminal javifortu/idcwioterminal:latest
> docker push javifortu/idcwioterminal:latest
```

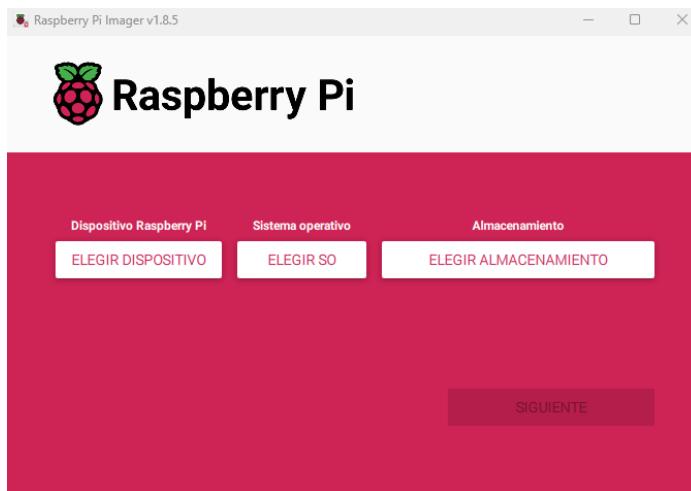
```
PS D:\Documentos\Arduino\ProyectoV2\WioTerminal_TinyMLWeatherStation-main\docker_project> docker push javifortu/idcwioterminal:latest
The push refers to repository [docker.io/javifortu/idcwioterminal]
79b3e1d9d058: Pushed
2b93544db42d: Pushed
c07a9cc290a8: Layer already exists
543693446bda: Layer already exists
50df9224ea2a: Layer already exists
c0a8bf9d6dab: Layer already exists
2d8c3949de61: Layer already exists
1387079e86ad: Layer already exists
latest: digest: sha256:80ff5549d7ffb72056077ee9586aad5d0cbe6bf10696e3419033010c4c118e45 size: 1998
```

Aquel que quiera ejecutar el contenedor para realizar la inferencia en su máquina tan solo debe instalar docker y ejecutar **docker run javifortu/idcwioterminal:latest**

```
C:\Users\javic>docker run javifortu/idcwioterminal:latest
2024-06-19 20:27:49.577370: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-06-19 20:27:49.579366: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-06-19 20:27:49.669392: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-06-19 20:27:50.018549: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-06-19 20:27:51.249901: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
['39.00', '40.00', '42.00', '42.00', '45.00', '35.00']
[6, 6, 6, 6, 6, 6]
['27.00', '25.00', '25.00', '25.00', '23.00', '27.00']
Últimos 6 datos del campo 1: ['39.00', '40.00', '42.00', '42.00', '45.00', '35.00']
Últimos 6 datos del campo 2: ['27.00', '25.00', '25.00', '25.00', '23.00', '27.00']
Últimos 6 datos del campo 3: [6, 6, 6, 6, 6, 6]
18
['39.00', '27.00', 6, '40.00', '25.00', 6, '42.00', '25.00', 6, '42.00', '25.00', 6, '45.00', '23.00', 6, '35.00', '27.00', 6]
Modelo cargado exitosamente.
1/1 0s 87ms/step
/app/main.py:88: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
    client = mqtt.Client(client_id=""),
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
Predicción: 0, Confianza: 0.8066191077232361
msg published (mid=1)
```

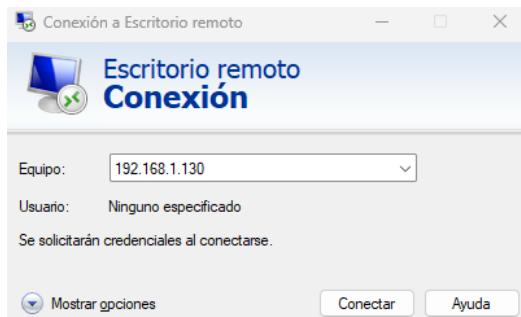
8. Raspberry Pi

Para este proyecto hemos decidido hacer uso de una Raspberry Pi para realizar la inferencia, para ello, primero debemos instalar su sistema operativo, Raspbian, en la tarjeta microSD mediante la herramienta oficial Raspberry Pi Imager.

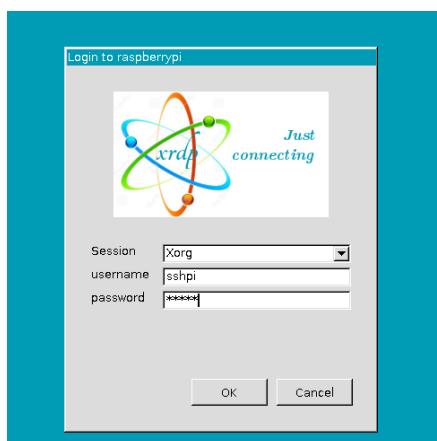


Una vez hecho esto, introducimos la tarjeta en la Raspberry, la encendemos y la conectamos a un monitor mediante un cable HDMI-microHDMI, una vez tengamos imagen, podemos configurar el ssh para conectarnos a ella remotamente, aunque esto también puede hacerse en el momento de la instalación del SO.

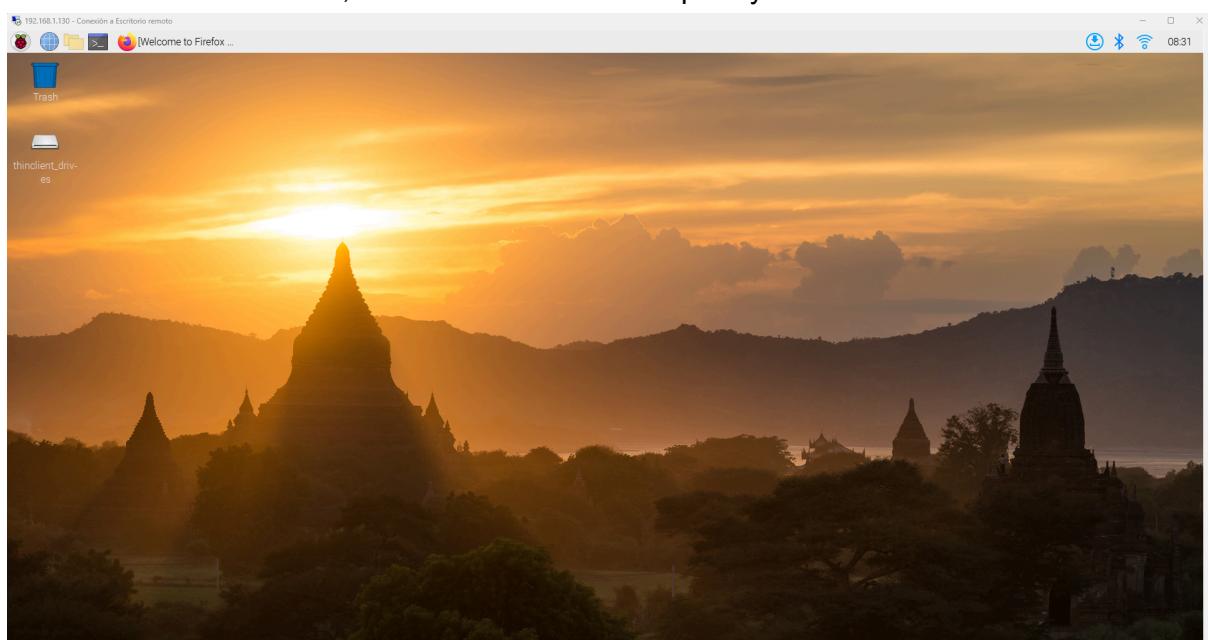
Sea como sea, una vez configurado el ssh, podemos usar el comando ssh en PowerShell de Windows para conectarnos a la raspberry, o si queremos ver el escritorio, podemos realizar una conexión remota:



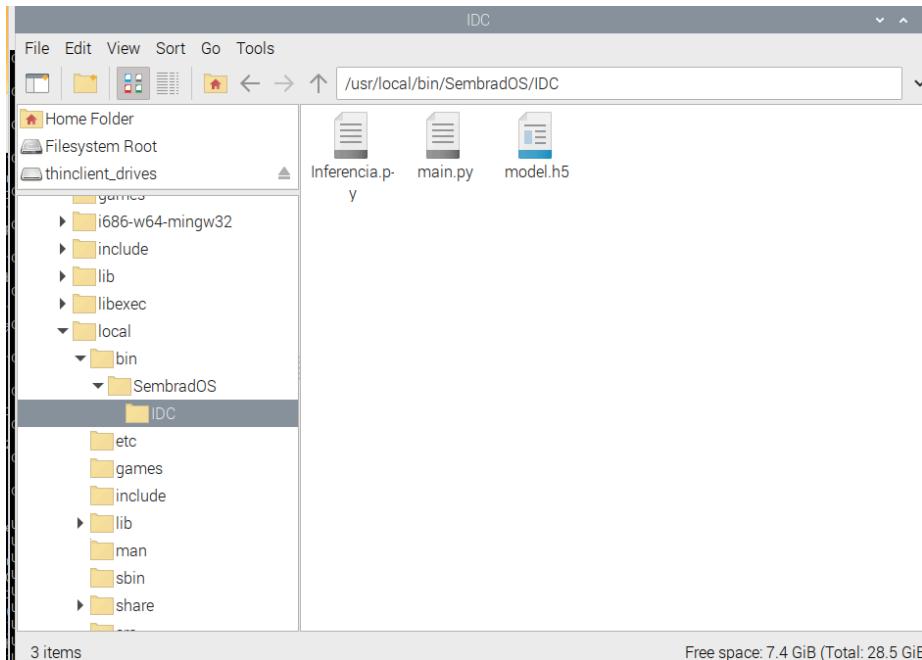
Si todo está correctamente configurado, podremos iniciar sesión en la Raspberry pi, en nuestro caso hemos creado un usuario específico para conexiones ssh, llamado *sshpi*.



Una vez iniciada la sesión, estamos dentro de la raspberry.



Ahora lo lógico sería instalar Docker y descargar nuestra imagen, pero por el problema que se relata en el punto **10.5 Imagen docker para la Raspberry pi**, no ha sido posible crear una imagen compatible con la raspberry pi, por lo que hemos optado por pasar manualmente los archivos, para ello hemos creado una carpeta con estos en la dirección /usr/local/bin:



A continuación se ha instalado python mediante la consola, además, necesitaremos crear un entorno de ejecución de python para aislar la ejecución del script, configurarlo con sus permisos adecuados (en este caso, que sus permisos los adquiera el superusuario javifortu), e instalar las dependencias necesarias, como la biblioteca libhdf5-dev mediante `sudo apt install libhdf5-dev` para que pueda ser capaz de leer el modelo, dentro del entorno deberemos instalar las librerías de tensorflow, requests, numpy y paho-mqtt.

```
javifortu@raspberrypi:~$ ls -ld /ent_vir_python/
drwxr-xr-x 5 root root 4096 Jun 20 13:42 /ent_vir_python/
javifortu@raspberrypi:~$ sudo chown -R javifortu:javifortu /ent_vir_python/
javifortu@raspberrypi:~$ ls -ld /ent_vir_python/
drwxr-xr-x 5 javifortu javifortu 4096 Jun 20 13:42 /ent_vir_python/
javifortu@raspberrypi:~$ source /ent_vir_python/bin/activate
(ent_vir_python) javifortu@raspberrypi:~$ pip install --upgrade pip
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: pip in /ent_vir_python/lib/python3.11/site-packages (23.0.1)
Collecting pip
  Using cached https://www.piwheels.org/simple/pip/pip-24.0-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.0.1
    Uninstalling pip-23.0.1:
      Successfully uninstalled pip-23.0.1
Successfully installed pip-24.0
(ent_vir_python) javifortu@raspberrypi:~$ pip install tensorflow requests numpy paho-mqtt
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting tensorflow
  Downloading tensorflow-2.16.1-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl.metadata (4.1 kB)
Collecting requests
  Downloading https://www.piwheels.org/simple/requests/requests-2.32.3-py3-none-any.whl (64 kB)
                                             64.9/64.9 kB 972.6 kB/s eta 0:00:00
Collecting numpy
  Downloading numpy-1.25.3-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (14.5 MB)
Collecting paho-mqtt
  Downloading paho_mqtt-3.6.0-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (1.1 MB)
```

Ahora, si queremos que la Raspberry ejecute el script automáticamente cada vez que se encienda, podemos crear un nuevo servicio dentro de /etc/systemd/system/ llamado SembradOS.service y habilitar este servicio mediante `sudo systemctl enable`

`SembradOS.service`, como se ha comentado anteriormente, habrá que configurarlo con los permisos y direcciones adecuadas.

```
javifortu@raspberrypi: /home/sshpi
File Edit Tabs Help
GNU nano 7.2
/etc/systemd/system/SembradOS.service
[Unit]
Description=SembradOS - Script de inicialización
After=network.target

[Service]
User=javifortu
Group=javifortu
WorkingDirectory=/usr/local/bin/SembradOS/IDC
ExecStart=/ent_vir_python/bin/python /usr/local/bin/SembradOS/IDC/main.py
Environment="PATH=/ent_vir_python/bin:/usr/local/bin:/usr/bin:/bin"
Restart=on-failure
StandardOutput=journal+console
StandardError=journal+console

[Install]
WantedBy=multi-user.target
```

Ahora habilitamos el servicio, lo arrancamos, recargamos systemctl y comprobamos el estado de nuestro servicio.

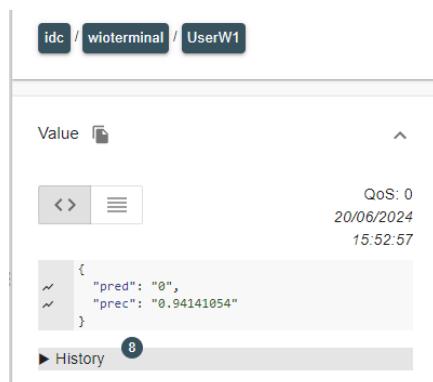
```
javifortu@raspberrypi:~/home/sshp1 $ sudo systemctl daemon-reload
javifortu@raspberrypi:~/home/sshp1 $ sudo systemctl restart SembradOS.service
javifortu@raspberrypi:~/home/sshp1 $ sudo systemctl status SembradOS.service
● SembradOS.service - SembradOS - Script de inicialización
    Loaded: loaded (/etc/systemd/system/SembradOS.service; enabled; preset: enabled)
    Active: active (running) since Thu 2024-06-20 15:49:39 CEST; 3s ago
      Main PID: 16447 (python)
        Tasks: 4 (limit: 3699)
       CPU: 3.593s
      CGroup: /system.slice/SembradOS.service
              └─16447 /ent_vir_python/bin/python /usr/local/bin/SembradOS/IDC/main.py

Jun 20 15:49:39 raspberrypi systemd[1]: Started SembradOS.service - SembradOS - Script de inicializaci
```

A medida que va pasando el tiempo podemos ir viendo que el servicio se está ejecutando y realizando la inferencia.

```
javifortu@raspberrypi:~/home/sshpi $ sudo systemctl status SembradOS.service
● SembradOS.service - SembradOS - Script de inicialización
  Loaded: loaded (/etc/systemd/system/SembradOS.service; enabled; preset: enabled)
  Active: active (running) since Thu 2024-06-20 15:49:39 CEST; 3min 31s ago
    Main PID: 16447 (python)
      Tasks: 14 (limit: 3699)
        CPU: 10.845s
       CGroub: /system.slice/SembradOS.service
               └─16447 /ent_vir_python/bin/python /usr/local/bin/SembradOS/IDC/main.py

Jun 20 15:52:57 raspberrypi python[16447]: ['28.00', '28.00', '28.00', '28.00', '28.00', '28.00']
Jun 20 15:52:57 raspberrypi python[16447]: [6, 6, 6, 6, 6, 6]
Jun 20 15:52:57 raspberrypi python[16447]: ['32.00', '32.00', '32.00', '32.00', '32.00', '32.00']
Jun 20 15:52:57 raspberrypi python[16447]: Últimos 6 datos del campo 1: ['28.00', '28.00', '28.00', '28.00', '28.00', '28.00']
Jun 20 15:52:57 raspberrypi python[16447]: Últimos 6 datos del campo 2: ['32.00', '32.00', '32.00', '32.00', '32.00', '32.00']
Jun 20 15:52:57 raspberrypi python[16447]: Últimos 6 datos del campo 3: [6, 6, 6, 6, 6, 6]
Jun 20 15:52:57 raspberrypi python[16447]: 18
Jun 20 15:52:57 raspberrypi python[16447]: ['28.00', '32.00', 6, '28.00', '32.00', 6, '28.00', '32.00', 6, '28.00', '32.00', 6, '28.00',
Jun 20 15:52:57 raspberrypi python[16447]: Modelo cargado exitosamente.
Jun 20 15:52:57 raspberrypi python[16447]: [1968 blob data]
```



En esta última imagen podemos comprobar que se están subiendo las predicciones al tópico del broker MQTT

9. Resultados

9.1 Video de la Demo

Los resultados pueden verse en el [video de la demo](#)

9.1 Estructura de la entrega

Para la entrega hemos enviado un archivo .zip, este contiene los archivos usados para la creación de la red neuronal y el código Arduino:

📁 Red Neuronal	20/06/2024 20:45	Carpeta de archivos
📁 SembradOS	20/06/2024 20:47	Carpeta de archivos
📄 link_video_demo.txt	20/06/2024 21:01	Documento de te... 1 KB

El código de la red neuronal consta del archivo de Scrappeo de los datos meteorológicos, el archivo Juniper Notebooks usado para el entrenamiento de la RN, el csv con los datos meteorológicos.

📁 Docker_Inferencia	17/06/2024 19:22	Carpeta de archivos
📄 BuildTFLiteModel.ipynb	17/06/2024 11:06	Archivo de origen ... 35 KB
📄 Scrapper_GetWeatherData.py	16/06/2024 20:14	Archivo PY 3 KB
📄 WeatherData.csv	16/06/2024 20:14	Archivo de origen ... 3.074 KB

La carpeta Docker_Inferencia contiene el Dockerfile, el modelo y los archivos que realizan la inferencia, recepción de ThingSpeak y envío de datos al broker MQTT.

Nombre	Fecha de modificación	Tipo	Tamaño
📁 __pycache__	17/06/2024 19:22	Carpeta de archivos	
📄 Dockerfile	20/06/2024 20:40	Archivo	1 KB
📄 Inferencia.py	17/06/2024 11:32	Archivo PY	2 KB
📄 main.py	19/06/2024 23:14	Archivo PY	5 KB
📄 model.h5	17/06/2024 11:05	Archivo H5	34 KB

Mientras que la carpeta SembradOS contiene el código Arduino

arduino SembradOS.ino	20/06/2024 20:39	Archivo INO	33 KB
-----------------------	------------------	-------------	-------

10. Problemas encontrados

10.1 Código espagueti

El primero de los problemas con el que nos topamos se trataba de nuestro propio código, cuando la aplicación empezó a adquirir cierta complejidad, entender y moverse por el código resultaba muy complicado ya que se basaba en ciertas funciones sobredimensionadas que realizaban múltiples funciones y dependían de comprobar muchos estados.

Para solucionar esto ideamos la clase App, la cual se encargaría de ordenar todas las funcionalidades gráficas de la aplicación, además, cada funcionalidad nueva se implementaría en su propia función, el código de la clase App al final, sin expandir, es el siguiente:

```
3 class App {
4     public:
5         //App() {}
6     >     App(TFT_eSPI &tft) //Constructor...
7     }
8     //Iniciar la aplicación
9     >     void start(){...}
10    }
11    void setMenu(MenuType menu) { ...}
12    }
13    //Manejo de inputs de botones a nivel interno de la app
14    >     void handleButtonPress(ButtonType button) {...}
15    }
16    >     void update() { ...}
17    }
18    //Tonos (Sonido)
19    >     void playTone(int tone, int duration) { ...}
20    }
21    //Notas (Sonido)
22    >     void playNote(char note, int duration) { ...}
23    }
24    //Función pública para actualizar los valores de los sensores
25    >     void updateSensorsData(){...}
26    }
27
28 private:
29     //Printeo de cada menú de la aplicación
30     >     void printMainMenu() {...}
31     }
32     void printSensorsMenu() { ...}
33     >     void printPredictMenu() { ...}
34     }
35     >     void printSettingsMenu() { ...}
36     }
37     //Actualiza el menú sensores con los nuevos datos de los sensores
38     >     void updateSensorsMenu() {...}
39     }
40     >     void updatePredictMenu(){ ...}
41     }
42     >     void updateSettingMenu(){ ...}
43     }
44     //Printeo de logos en el menú principal
45     >     void drawGraphLogo() {...}
46     }
47     >     void drawIALogo() { ...}
48     }
49     >     void drawOptionsLogo() { ...}
50     }
51     //Manejo de input de los botones a nivel visual de los menús
52     >     void buttonPressedMainMenu() { ...}
53     }
54     >     void buttonPressedSensorsMenu() { //vacio...
55     }
56     >     void buttonPressedPredictMenu() { //vacio...
57     }
58     >     void buttonPressedSettingsMenu() { //vacio...
59     }
60     }
61     //Función NOP
62     >     void nop(){}
```

```

//Función NOP
void nop(){}

//Variables de la APP
TFT_eSPI &tft;
TFT_eSprite spr;
TFT_eSprite spr_sensores;
MenuType currentMenu;
int menuSelectedOption;
uint16_t colorFondo1 = 0x04FA;

//Punteros a funciones DE la propia App (no sirven externas)
void (*App::*printMenuFunction)();
void (*App::*buttonPressedFunction)();
void (*App::*updateMenuFunction)();

};


```

Y fuera de App, se tiene esto:

```

644 //Documento JSON de como máximo 2 campos
645 const size_t bufferSize = JSON_OBJECT_SIZE(2);
646 DynamicJsonDocument jsonDoc(bufferSize);
647
648 > void callback(char* topic, byte* payload, unsigned int length) { ...
649 }
650
651 > void reconnect() { ...
652 }
653
654 > void startWiFi(){ ...
655 }
656
657 App app = App(tft);
658 > void setup() { ...
659 }
660
661
662 unsigned long lastTimeSensorRead = millis();
663 unsigned long lastTimeSensorSend = millis();
664 unsigned long lastWifiRefresh = millis();
665 unsigned long reconnectDelay = millis();
666
667 > void loop() { ...
668 }
669
670
671 //Tarea que gestiona los sensores
672 > void TaskReadSensors() { ...
673 }
674
675 //Tarea que gestiona los inputs de los botones a nivel de inputs
676 > bool TaskButtonHandler() { ...
677 }
678
679 > void TaskSendDataToThingSpeak(){ ...
680 }


```

Se observa que la aplicación final en Arduino consta de unas ~900 líneas de código, por lo que hubiese sido muy difícil seguir con el proyecto sin una estructura *medio clara*.

10.2 Problemas con los Hilos

Nuestra aplicación SembradOS requiere de escuchar los inputs del usuario, enviar datos a la plataforma y recibir datos del broker MQTT, es por esto que decidimos hacer uso de concurrencia, mediante FreeRTOS, con su librería compatible explicada en [Wio Terminal FreeRTOS](#) después modificamos el código para hacerlo concurrente y creamos las tareas y semáforos necesarios:

```

#include <Seeed_Arduino_FreeRTOS.h>
...
SemaphoreHandle_t changingMenuSemaphore;
SemaphoreHandle_t usingSensorDataSemaphore;
SemaphoreHandle_t firstSensorDataReadSemaphore;
...
void setup() {
    ...
    changingMenuSemaphore = xSemaphoreCreateBinary();
    xSemaphoreGive(changingMenuSemaphore);
    usingSensorDataSemaphore = xSemaphoreCreateBinary();
}


```

```

    xSemaphoreGive(usingSensorDataSemaphore);
    firstSensorDataReadSemaphore = xSemaphoreCreateBinary(); //lectura inicial de los
sensores
    xSemaphoreGive(firstSensorDataReadSemaphore);
    ...
xTaskCreate(
    TaskReadSensors,      // Función de tarea
    "ReadSensors",        // Nombre de la tarea
    1000,                 // Tamaño de la pila
    NULL,                 // Parámetros de la tarea
    2,                    // Prioridad de la tarea
    NULL                 // Handle de la tarea
);

xTaskCreate(
    TaskButtonHandler,   // Función de tarea
    "ButtonHandler",     // Nombre de la tarea
    6000,                // Tamaño de la pila
    NULL,                // Parámetros de la tarea
    1,                   // Prioridad de la tarea
    NULL                 // Handle de la tarea
);
vTaskStartScheduler();
}

```

Nos dimos cuenta de que al momento de integrar este código junto con las funcionalidades WiFi, mediante la librería rpcWifi, creando su respectiva tarea, la aplicación dejaba de funcionar y la Wio Terminal se bloqueaba hasta el punto de que había que encenderla en modo Bootloader para recuperar la conexión con el ordenador, esto pasaba incluso solo por importar la librería rpcWifi, sin nisiquiera hacer uso de alguna de sus funciones.

Revisando el foro oficial de Wio Terminal, descubrimos que esto podía deberse a un bug, pues dicha librería usa FreeRTOS para realizar las conexiones WiFi, por lo que al usar FreeRTOS encima de otra librería que ya lo usa, la WioTerminal colapsa:

Possible bug in FreeRTOS and AtWiFi Arduino library

■ Products & Technology ■ Wio Terminal



ONLYA

jul. '20

I am trying to implement FreeRTOS with WiFi feature. However, I find that the task cannot be created even if I only include the AtWiFi.h. Here is the code:



rmrf

oct. '23

I second that. rpcWiFi is doing some RTOS manipulation under the hood making use of tasks impossible.

I managed to run tasks for some time by removing vTaskStartScheduler(); and locating xTaskCreate before any WiFi statements, but after connection to WiFi tasks stop.

It is really sad that this is not resolved for such a long time for such a great devices lineup. 😞

De forma resumida, probamos una gran número de alternativas, entre ellas probar a usar ZephyrOS, para el cual había que instalar diversos programas como CMake, wget y 7zip.

Zephyr SDK 0.16.8		
Downloads		
SDK Bundle		
OS	Minimal ^[1]	Full
Linux	[AArch64][zephyr-sdk-0.16.8_linux-aarch64_minimal.tar.xz] / [x86-64][zephyr-sdk-0.16.8_linux-x86_64_minimal.tar.xz]	[AArch64][zephyr-sdk-0.16.8_linux-aarch64.tar.xz] / [x86-64][zephyr-sdk-0.16.8_linux-x86_64.tar.xz]
macOS	[AArch64][zephyr-sdk-0.16.8_macos-aarch64_minimal.tar.xz] / [x86-64][zephyr-sdk-0.16.8_macos-x86_64_minimal.tar.xz]	[AArch64][zephyr-sdk-0.16.8_macos-aarch64.tar.xz] / [x86-64][zephyr-sdk-0.16.8_macos-x86_64.tar.xz]
Windows	[x86-64][zephyr-sdk-0.16.8_windows-x86_64_minimal.7z]	[x86-64][zephyr-sdk-0.16.8_windows-x86_64.7z]

Además, llegamos a configurar el proyecto de Zephyr.

```
PS C:\Users\javic> $env:ZEPHYR_BASE = "D:\Documentos\zephyrproject\zephyr"
PS C:\Users\javic> $env:PATH += ";$env:ZEPHYR_BASE\scripts\windows"
PS C:\Users\javic>
```

```
PS D:\Documentos\que> python3 -m west init zephyrproject
--- Initializing in D:\Documentos\que\zephyrproject
--- Cloning manifest repository from https://github.com/zephyrproject-rtos/zephyr
Cloning into 'D:\Documentos\que\zephyrproject\.west\manifest-tmp'...
remote: Enumerating objects: 1063503, done.
remote: Counting objects: 100% (303/303), done.
remote: Compressing objects: 100% (180/180), done.
Receiving objects: 22% (244082/1063503), 179.21 MiB | 5.50 MiB/s
```

```
zephyrproject/
├── src/
│   └── my_wio_project/
│       ├── CMakeLists.txt
│       ├── prj.conf
│       └── src/
│           └── main.c
```

Una vez configurado el proyecto nos dimos cuenta de que si seguíamos por este camino, debíamos reescribir todo el código con librerías compatibles con Zephyr, lo cual era una tarea demasiado grande para nosotros.

Es por esto probamos a hacer uso de Azure RTOS ThreadX, otra librería de creación de hilos compatible con Arduino y Wio Terminal:

[README](#) [License](#) [Security](#)

Azure RTOS ThreadX For Arduino

[Arduino CI](#) no status

This is a port of Azure RTOS ThreadX to Arduino as a Library. For more information about Azure RTOS, please visit Microsoft Doc and source code on Github.

 A new [Azure RTOS ThreadX for Arduino 101: Threads](#) is available on hackster.io.

Hardware support

The port and provided demo is verified on following board and Arduino Core.

Board	Chip	Architecture	Verified Arduino Core
Seeeduino XIAO	ATSAMD21	Cortex-M0+	Seeed Studio/ArduinoCore-samd 1.8.3
Seeeduino Wio Terminal	ATSAMD51	Cortex-M4	Seeed Studio/ArduinoCore-samd 1.8.3
B-L4S5I-IOT01A	STM32L4SS	Cortex-M4	stm32duino/Arduino_Core_STM32 2.3.0
32F746GDISCOVERY	STM32F746	Cortex-M7	stm32duino/Arduino_Core_STM32 2.3.0

Esta librería proporcionaba ejemplos de su implementación, como <https://github.com/xiongyu0523/AzureRTOS-ThreadX-For-Arduino/blob/main/examples/dem>

[o_blink_serialread/demo_blink_serialread.ino](#), en los cuales nos basamos para adaptar nuestro código. Después de volver a modificar adecuadamente los hilos y semáforos de la aplicación, adaptándolos a Azure RTOS, resulta que como era evidente, la librería FreeRTOS que usa rpcWiFi entra en conflicto con AzureRTOS, por lo que no hemos solucionado nada, hay que buscar otra alternativa.

Usar otras librerías WiFi?

WiFiNINA

Communication

Enables network connection (local and Internet) with the Arduino MKR WiFi 1010, Arduino MKR VIDOR 4000, Arduino Uno WiFi Rev.2 and Nano 33 IoT.

With this library you can instantiate Servers, Clients and send/receive UDP packets through WiFi. The board can connect either to open or encrypted networks (WEP, WPA). The IP address can be assigned statically or through a DHCP. The library can also manage DNS.

[Go to repository](#)

Pero no compatible directamente con Wio Terminal, por suerte hay una librería genérica que a lo mejor sí es compatible.

WiFiNINA_Generic (Library for WiFiNINA modules/shields to support many more boards)

[Library Manager](#) [WiFiNINA_Generic 1.8.15-1](#) release [v1.8.15-1](#) license [MIT](#) contributions [welcome](#) issues [0 open](#)

Currently Supported Boards

This [WiFiNINA_Generic](#) library currently supports these following boards:

1. nRF52 boards, such as AdaFruit Feather nRF52832, nRF52840 Express, BlueFruit Sense, Itsy-Bitsy nRF52840 Express, Metro nRF52840 Express, NINA_B302_ublox, NINA_B112_ublox, etc.
2. SAM DUE
3. SAMD21
 - Arduino SAMD21: ZERO, MKRs, NANO_33_IOT, etc.
 - Adafruit SAMD21 (M0): ItsyBitsy M0, Feather M0, Feather M0 Express, Metro M0 Express, Circuit Playground Express, Trinket M0, PIRkey, Hallowing M0, Crickit M0, etc.
 - Seeeduino: LoRaWAN, Zero, Femto M0, XIAO M0, Wio GPS Board, etc.

También viene con ejemplos para aprender a usarla. El problema es que al final resulta no ser compatible con Wio Terminal:

```
Start WiFiPing on SAMD SEEED_WIO_TERMINAL
WiFiNINA_Generic v1.8.15-1
Communication with WiFi module failed!
255
```

khoih-prog commented on Sep 20, 2021

Owner

WIO Terminal can't use WiFiNINA or [WiFiNINA_Generic](#).
You have to use [Seeed_Arduino_rpcWiFi Library](#).
These libraries of mine are supporting WIO_Terminal

1. [WIOTerminal_WiFiManager](#)
2. [WebSockets_Generic](#)
3. [SinricPro_Generic](#)

Go to [Seeedstudio Forum](#) for more support.
Good Luck,

El propio creador de la librería ofrece tres alternativas, pero todas usan rpcWiFi, que es precisamente la biblioteca que está causando el problema. Parece ser que la única biblioteca compatible con Wio Terminal es rpcWiFi.

A continuación preguntamos en el foro por ayuda, aunque esta tardó demasiado en llegar, el proyecto ya estaba suficientemente avanzado como para volver atrás e intentar volver a implementar los hilos.

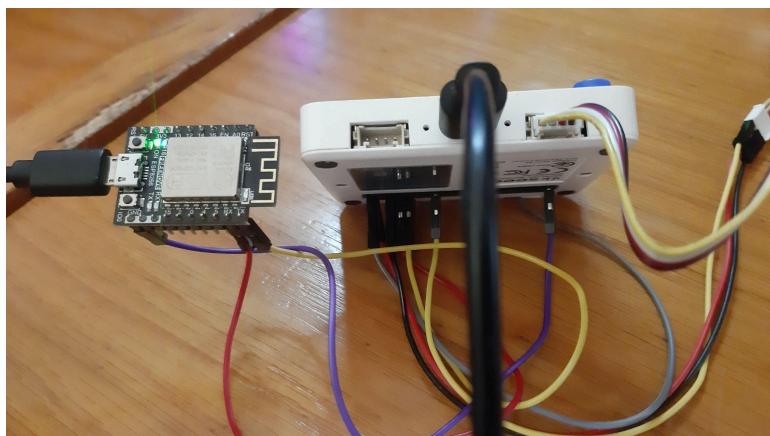
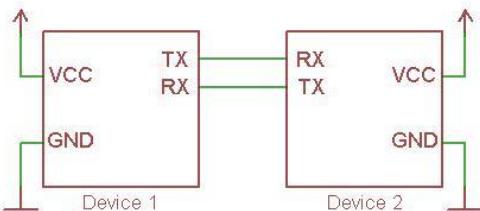
M MaFi 1 min

Any way to use FreeRTOS and rpcWiFi at the same time, I need to create FreeRTOS tasks for my project and do web requests at the same time.
Wio Terminal stops working when the FreeRTOS tasks are created and the rpcWiFi library is being used even though it's not even initialized with Wifi.begin(). 😞

Responder

Así que buscamos soluciones más variadas.

Usar un ESP8266 propio como puente para transmitir datos a ThingSpeak, el ESP puede alimentarse a través del pin de 3.3V de la Wio Terminal cuando no está conectado mediante USB, estos dos dispositivos se unen mediante los pins TX-RX y viceversa de la siguiente forma para que puedan comunicarse:



Problema: El ESP del que disponemos, envía una serie de caracteres al iniciarse, por lo que hace muy difícil la tarea de gestionar los mensajes ya que hay que tener en cuenta que cada vez que se encienda, va a enviar una serie de cadenas con información no relevante para nuestro caso.

Lo cual ya se advierte en la propia documentación de dicho ESP:

Después de varias horas de intentos fallidos, modificando baudios y demás, no hemos podido encontrar una forma fiable de transmitir los datos por dichos pines. Además, el ESP

trata tanto su Serial por consola como su Serial TX-RX como el mismo, complicando aún más el proceso ya que a veces de alguna forma, recogía lo que él mismo enviaba.

Captura de parte del código de prueba y su salida:

A la derecha se encuentra el código del Wio Terminal, éste iniciaba 2 serials, Serial y Serial1, de forma que el Serial1 no era más que la conexión serie con el ESP a través de los pines TX-RX, al inicio del loop se encargaba de limpiar los caracteres “basura” que envía el ESP al iniciarse, y a partir de ahí, debía enviarle una cabecera de alerta al ESP “b” y luego enviarle el mensaje solicitado al ESP, el ESP debía leer la cabecera, comprobar cuál era su valor y ejecutar la acción que correspondiere con esa cabecera, que en este caso era enviar “bbx” al Wio Terminal, donde x es el número de mensajes enviados por el ESP al Wio Terminal.

Pero como se ve en las capturas, no lo hemos logrado, la recepción de los mensajes no es fiable y no hemos conseguido solucionarlo, es algo posible de hacer, pero ahora mismo, no merece la pena seguir intentándolo cuando siempre existe la posibilidad de prescindir de los hilos, aunque implique reestructurar el código.

Último intento

Buscando por internet damos con un post en japonés de alguien que dice haber arreglado este problema con una API que ejecuta FreeRTOS en TOPPERS/ASP manteniendo la sintaxis de FreeRTOS.

Arduino IDEでTOPPERS/ASP、そしてWio TerminalのWi-Fi/BLE使用

このQiitaの記事で紹介されているように、Arduino IDEでTOPPERSのRTOSの一つ

TOPPERS/ASPが使えるようになっています。

ターゲットデバイスはWio Terminalです。

しかし、Wio TerminalにはWi-Fi/BLEが搭載されているのですが、[FreeRTOS](#)ライブラリに依存していて、TOPPERS/ASPとは共存できません。

そこで、FreeRTOSのAPIをTOPPERS/ASPで実装して、Wi-Fi/BLEを使えるようにするArduinoライ

Después de traducir la página para ver qué dice, instalamos las librerías necesarias según el post y lanzamos el programa ejemplo que trae, pero no compila.

```
Output
Using precompiled core: C:\Users\javi\AppData\Local\Temp\arduinocores\1ce882z\44c90d>C:\Users\javi\arduino\core.a
Linking everything together...
"C:\Users\javi\AppData\Local\Arduino15\packages\Seeduino\tools\arm-none-eabi-gcc\7-2017q4/bin/arm-none-eabi-g++" -L"C:\Users\javi\AppData\Local\Temp\arduino\sketches\B3005EB8FD7F967064C5EECA3C3921" -Os -Wl,--gc-sections
C:\Users\javi\AppData\Local\Temp\arduino\sketches\B3005EB8FD7F967064C5EECA3C3921\libraries\Seed_Arduino\rpcUnified\src\rpc_threading_freertos.cpp.o: In function `erpc::Semaphore::putFromISR()':
d:\documents\arduino\libraries\Seed_Arduino\rpcUnified\src\rpc_threading_freertos.cpp:209: undefined reference to `xSemaphoreGiveFromISR'
collect2.exe: error: ld returned 1 exit status
```

Probamos a usar las librerías rpcUnified y rpcWifi de 2021, que es cuando se creó el post. El programa ahora se lanza, pero al probar al probar a lanzar hilos, el programa se vuelve a colgar, no sirve, puede que por incompatibilidades entre otras librerías o quien sabe.

La única solución posible es deshacernos de los hilos, el problema de hacer esto es que mientras el Wio Terminal esté recogiendo los datos de los sensores, o enviando los datos a ThingSpeak, éste no reacciona a los inputs del usuario, pero poco más se puede hacer con el tiempo limitado que tenemos y después de haber perdido 3 días enteros intentando arreglar este problema.

10.3 Problemas con el entrenamiento de la red neuronal

Unos de los problemas que encontramos mientras realizamos el proyecto es uno relacionado con TinyML y el código python usado para entrenar el modelo.

En el código que utilizamos existía un problema y es que al crear el modelo de entrenamiento, este importaba librerías que no funcionaban.

```
import tensorflow as tf
from tensorflow.keras import layers
from keras.layers import Dense
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.20, random_state = 33)
```

Esta parte nos daba error ya que está importando “Dense” tanto desde keras como de “tensorflow.keras”, lo que hace hay conflicto.

La solución a la que hemos llegado es importar Keras de manera coherente y usando la función “train test split” de “sklearn.model_selection” para dividir los datos.

```
random_train_test_split de sklearn.model_selection para dividir los datos.
```

```
import tensorflow as tf
import keras
from keras import layers
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, random_state=33,
stratify=y)
```

Además, en el entrenamiento hemos añadido capas de Dropout, una técnica de regularización utilizada para prevenir el sobreajuste durante el entrenamiento.

```
model = tf.keras.Sequential()
model.add(layers.Dense(14, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))
optimizer = tf.optimizers.Adam(learning_rate=0.0001)
```

```

model.compile(loss = 'categorical_crossentropy',
              optimizer = optimizer,
              metrics=['accuracy'])

model.fit(xtrain, ytrain, epochs=40, validation_split=0.1, batch_size=128)

test_results = model.evaluate(xtest, ytest, verbose=1)

model.export('model')

```

El último error que hemos encontrado es en el método `convert`. En el que la documentación nos pone esta línea de código que convierte un modelo de keras a un formato compatible con TensorFlow Lite, el cual nos daba error al ejecutarlo.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
```

Decidimos cambiarlo al siguiente método que se utiliza para convertir un modelo que ya ha sido guardado previamente (el que hemos guardado con el código anterior) en el formato `SaveModel` de TensorFlow.

```
converter = tf.lite.TFLiteConverter.from_saved_model("model/")
```

10.4 Problemas con la inferencia en la Wio Terminal

En un principio, queríamos realizar la inferencia de nuestro modelo en la propia Wio Terminal basándonos en un código de ejemplo que modificamos para adaptar a nuestro caso, siendo el siguiente:

```

// Include TensorFlow Libraries
#include "TensorFlowLite.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

// Include Remaining Libraries
#include "DHT.h"
#include "TFT_eSPI.h"

TFT_eSPI tft;

// Our model
#include "model.h"

// DHT Sensor Variables
#define DHTTYPE DHT11
#define DHTPIN 0
#define DEBUG Serial
DHT dht(DHTPIN, DHTTYPE);

// TF Model Outputs
const char* OUTPUTS[] = {
    "No Rain",

```

```

    "Might Rain",
    "Rain"
};

int NUM_OUTPUTS = 3;
int array_count = 0;
float probability, reading[3];
float temp_hum_val[21];
long displayMarker, dataMarker;
String prediction;

// Tensor Flow Variables

tfLite::MicroErrorReporter tflErrorReporter;
tfLite::AllOpsResolver tflOpsResolver;
const tfLite::Model* tflModel = nullptr;
tfLite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;

// Create an area of memory to use for input, output, and other TensorFlow
// arrays. You'll need to adjust this by combining, running, and looking for errors.

constexpr int tensorArenaSize = 8 * 1024;
byte tensorArena[tensorArenaSize];

void millisDelay(int duration) {
    long mark = millis();
    while (millis() - mark < duration) {}
}

void setup() {
    // Wait for Serial to connect
    Serial.begin(115200);

    // Initialise LCD
    tft.begin();
    tft.setRotation(3);
    tft.fillScreen(TFT_BLACK);

    /// Get the TFL representation of the model byte array
    tflModel = tfLite::GetModel(model);
    if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
        Serial.println(F("Model schema mismatch!"));
        while (1);
    }

    // Create an interpreter to run the model
    tflInterpreter = new tfLite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena,
tensorArenaSize, &tflErrorReporter);

    // Allocate memory for the model's input and output tensors
    tflInterpreter->AllocateTensors();

    // Get pointers for the model's input and output tensors
    tflInputTensor = tflInterpreter->input(0);
    tflOutputTensor = tflInterpreter->output(0);
}

```

```

dht.begin();
dataMarker = millis();
displayMarker = millis();
tft.setTextDatum(TC_DATUM);
tft.setTextSize(2);
tft.drawString(F("Smart Weather Station"), 160, 20);
tft.drawFastVLine(160,80,60, TFT_WHITE);
}

void loop() {

    if (array_count == 0 || millis() - displayMarker > 2000) {
        displayMarker = millis();
        //float reading[2];
        if (!dht.readTempAndHumidity(reading)) {
            tft.setTextPadding(0);
            tft.setTextSize(2);
            tft.drawString(F("Temp (C)"), 80, 140);
            tft.drawString(F("R.H."), 240, 140);
            tft.setTextSize(4);
            tft.drawString(String(round(reading[1]*10)/10), 80, 90);
            tft.drawString(String(round(reading[0])), 240, 90);
        } else {
            //tft.drawString(F("Failed to get temprature and humidity value."),0,0);
        }
    }

    if (array_count == 0 || millis() - dataMarker > 4000) {
        dataMarker = millis();
        tft.setTextSize(2);
        tft.setTextPadding(320);

        for (int i=0; i<21; i++) {
            temp_hum_val[i] = temp_hum_val[i+3];
        }

        temp_hum_val[18] = reading[1] + 273.15;
        temp_hum_val[19] = reading[0];
        temp_hum_val[20] = 6;
        array_count++;
        if (array_count > 7) array_count = 7;

        if (array_count == 7) {

            // Copy array into tensor inputs
            for (int i=0; i<21; i++) {
                tfLiteInputTensor->data.f[i] = temp_hum_val[i];
                Serial.print(String(temp_hum_val[i]) + " ");
            }

            // Run inference on data
            TfLiteStatus invoke_status = tfLiteInterpreter->Invoke();
            if (invoke_status != kTfLiteOk) {
                Serial.println(F("Invoke Failed!"));
                while(1);
                return;
            }
        }
    }
}

```

```
// Read predicted y value from output, get max probability

probability = 0;
for (int i = 0; i < NUM_OUTPUTS; i++) {
    if (tflOutputTensor->data.f[i] > probability) {
        prediction = OUTPUTS[i];
        probability = tflOutputTensor->data.f[i];
    }
}

tft.drawString(prediction + " (" + String(probability) + ")", 160 ,200);
} else {
    tft.drawString("Need " + String(7-array_count) + " more readings.", 160, 200);
}
}
```

Para ello adaptamos el modelo de red neuronal al formato .h, probamos el código anterior y funcionaba a la perfección, por lo que decidimos implementarlo en nuestra aplicación, pero nos topamos con un problema: La memoria parecía desbordarse al tener que almacenar todas las librería que usabamos anteriormente, más las librerías de tensorflow lite específicamente compatibles con Wio Terminal (que se encuentran descontinuadas pero pudimos descargarlas mediante una url específica), más el modelo de 20KB, por lo que la Wio Terminal se colgaba al igual que con el problema de los hilos.

Detectado este problema, probamos a disminuir la pila de memoria usada para el modelo, a cuantizarlo y reducirlo, llegando a un peso de 13KB, pero no era suficiente, no parecía posible inferir en la propia Wio Terminal, con el modelo y las librerías, la Wio Terminal necesitaba más de 470 páginas para almacenarlo todo, lo que parece ser demasiado, mientras que sin la inferencia apenas llegaba a las 240, creemos que este era el problema.

Es por ello que terminamos decidiendo idear el sistema final haciendo uso de una Raspberry Pi y un broker MQTT.

10.5 Imagen docker para la Raspberry pi

En un principio habíamos pensado en usar docker y nuestra imagen `javifortu/idcwioterminal:latest` para lanzar el servicio de inferencia en la raspberry pi siguiendo los siguientes pasos:

Abrimos una terminal e instalamos docker siguiendo los pasos descritos en [Install Docker Engine on Debian](#), una vez hecho esto podemos comprobar si Docker ha sido correctamente instalado lanzando el contenedor *hello-world*.

```
sshpi@raspberrypi: ~
File Edit Tabs Help
sshpi@raspberrypi:~ $ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:d1b0b5888fbb59111dbf2b3ed698489c41046cb9d6d61743e37ef8d9f3dda06f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

Ahora tan solo tenemos que descargar nuestra imagen y ejecutarla en un contenedor, el problema es que nuestra imagen original no está hecha para sistemas Linux/arm como el de la raspberry pi 4.

```
sshpi@raspberrypi:~$ sudo docker run javifortu/idcwioterminal:latest
Unable to find image 'javifortu/idcwioterminal:latest' locally
latest: Pulling from javifortu/idcwioterminal
2cc3ae149d28: Pull complete
87c0ed5d65e2: Pull complete
9b72c177d608: Pull complete
74338068cff9: Pull complete
fa9e209a71e9: Pull complete
7977de41ac71: Pull complete
fe9788b11476: Pull complete
bae3bb1836c0: Pull complete
Digest: sha256:80ff5549d7ff72056077ee9586aad5d0cbe6bf10696e3419033010c4c118e45
Status: Downloaded newer image for javifortu/idcwioterminal:latest
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
exec /usr/local/bin/python: exec format error
sshpi@raspberrypi:~$
```

Por lo que primero hay que volver a construir la imagen, esta vez indicandole la plataforma arm64 e instalando las dependencias compatibles y necesarias para su correcto funcionamiento, el nuevo Dockerfile queda de la siguiente forma:

```
# Usa una base compatible con ARM y luego instala TensorFlow
FROM arm64v8/python:3.9
WORKDIR /app
# Instala las dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    pkg-config \
    libhdf5-dev \
    && rm -rf /var/lib/apt/lists/*
# Actualiza pip
RUN pip install --no-cache-dir --upgrade pip
# Copia los archivos de la aplicación al contenedor
COPY . .
# Instala las dependencias de Python
RUN pip install --no-cache-dir requests paho-mqtt tensorflow==2.10.0 numpy h5py==3.8.0
# Comando por defecto para ejecutar el contenedor
CMD ["python", "main.py"]
```

Ahora debemos habilitar y configurar 'buildx' de Docker para poder construir la imagen compatible con arm y finalmente volvemos a construir la imagen:

```
> docker run --privileged --rm tonistiigi/binfmt --install all
> docker buildx create --name mybuilder --use
> docker buildx build --platform linux/arm64 -t javifortu/idcwioterminal:arm --push .
```

```
PS D:\Documentos\Arduino\ProyectoV2\WioTerminal_TinyMLWeatherStation-main\docker_project> docker buildx build --platform linux/arm64 -t javifortu/idcwioterminal:arm --push .
[+] Building 342.7s (13/13) FINISHED
  => [internal] load build definition from Dockerfile
  => transferring dockerfile: 651B
  => [internal] load metadata for docker.io/arm64v8/python:3.9
  => [auth] arm64v8/python:pull token for registry-1.docker.io
  => [internal] load .dockercfg
  => transferring context: 2B
  => [1/6] FROM docker.io/arm64v8/python:3.9@sha256:584771f912983a61a768a0028a5e0bbefdee975327547496f0c1c68de967170d
  => => resolve docker.io/arm64v8/python:3.9@sha256:584771f912983a61a768a0028a5e0bbefdee975327547496f0c1c68de967170d
  => [internal] load build context
  => => transferring context: 8298
  => CACHED [2/6] WORKDIR /app
  => CACHED [3/6] RUN apt-get update && apt-get install -y     pkg-config     libhdf5-dev     && rm -rf /var/lib/apt/lists/*
  => [4/6] RUN pip install --no-cache-dir --upgrade pip
  => [5/6] COPY . .
  => [6/6] RUN pip install --no-cache-dir requests paho-mqtt tensorflow==2.10.0 numpy h5py==3.8.0
  => exporting to image
  => => exporting layers
  => => exporting manifest sha256:b2a360cce372a038bd7aabdee6bf1d8290182ca006a63240427987162b70d482
  => => exporting config sha256:8f9f0e90302acd8eada7dd5e01897fb9b38cf4a742e80aa4afe05a32e09cc113
  => => exporting attestation manifest sha256:db3a35f2cdac3e8e14d4f1cc7e2e3af296c2099abab28a6a309ac5955d580e
  => => exporting manifest list sha256:f98574a3abc304f304e20f8a1907d3723e04a3a2f084b51d5a59a5c39e0dcc62
  => => pushing layers
  => => pushing manifest for docker.io/javifortu/idcwioterminal:arm@sha256:f98574a3abc304f304e20f8a1907d3723e04a3a2f084b51d5a59a5c39e0dcc62
  => [auth] javifortu/idcwioterminal:pull,push token for registry-1.docker.io

Build multi-platform images faster with Docker Build Cloud: https://docs.docker.com/go/docker-build-cloud
PS D:\Documentos\Arduino\ProyectoV2\WioTerminal_TinyMLWeatherStation-main\docker_project>
```

Al igual que la anterior imagen, esta nueva versión está disponible desde cualquier máquina, tan solo hay que instalarla mediante **docker run javifortu/idcwioterminal:arm**.

Lamentablemente no logramos la compatibilidad total de la imagen con la Raspberry pi, por lo que terminamos creando el servicio pasando los scripts manualmente y configurandolos apropiadamente como se explica en el punto sobre la Raspberry Pi.
Esto es lo máximo que logramos con la imagen Docker, se ejecutaba pero no lograba inferir correctamente.

```
javifortu@raspberrypi:/home/sshp1 $ sudo docker run javifortu/idcwioterminal:armv2
/app/main.py:88: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
| client = mqtt.Client(client_id="")
| ^CTraceback (most recent call last):
|   File "/app/main.py", line 109, in <module>
|     time.sleep(900)
KeyboardInterrupt
[28.00, '28.00', '28.00', '28.00', '28.00']
[6, 6, 6, 6, 6]
[32.00, '32.00', '32.00', '32.00', '32.00']
Últimos 6 datos del campo 1: ['28.00', '28.00', '28.00', '28.00', '28.00', '28.00']
Últimos 6 datos del campo 2: ['32.00', '32.00', '32.00', '32.00', '32.00', '32.00']
Últimos 6 datos del campo 3: [6, 6, 6, 6, 6, 6]
18
[28.00, '32.00', 6, '28.00', '32.00', 6, '28.00', '32.00', 6, '28.00', '32.00', 6]
Error al realizar la predicción: Error when deserializing class 'InputLayer' using config={'batch_shape': [None, 18], 'dtype': 'float32', 'sparse': False, 'name': 'input_layer'}.

Exception encountered: Unrecognized keyword arguments: ['batch_shape']
Falló la predicción.
msg published (mid=1)
```

11. Bibliografía

- [Seed Studio Get Started With Wio Terminal](#)
- [Wio Terminal Wi-Fi Connectivity](#)
- [Wio Terminal Weather Prediction](#)
- [Wio Terminal Forum](#)
- [Wio Terminal Classroom Video Series by Seeed Studio](#)
- [Build a TinyML Smart Weather Station with Wio Terminal](#)
- [GitHub POST to ThingSpeak Example by lakshanthad](#)
- [Tensorflow Documentation](#)
- [ThingSpeak Documentation](#)
- [How to send an Email Alert in ThingSpeak](#)
- [Dockerdocs Build, tag, and publish an image](#)
- [Raspberry Pi 4 Getting Started, Youtube Video](#)
- [Raspberry Pi Documentation](#)
- [From Python to Daemon: Turn Your Python App into a Linux Service Controlled by Systemd](#)
- PDFs de clase
- [ChatGPT](#)