UNIVERSIDAD
DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

ARQUITECTURA DE COMPUTADORES

ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

DISPOSITIVO INTELIGENTE PARA MONITORIZACIÓN INALÁMBRICA DE PILA DE COMPOSTAJE

SMART WIRELESS COMPOST PILE MONITORING SYSTEM

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Autor: **Javier Gil Palacios**

Tutor: **Andrés Rodriguez Moreno**

Cotutor: **Rubén Delgado Escaño**

MÁLAGA, SEPTIEMBRE de 2023

RESUMEN

Este proyecto surge por la necesidad de automatizar, monitorizar y controlar de forma eficiente todo el proceso de tratamiento y creación del compost de la estación ubicada en el huerto docente de la Universidad de Ciencias de Málaga.

Se ha fabricado un dispositivo portátil e inalámbrico, alimentado mediante baterías recargadas por el sol, este se introduce en una pila de compostaje a monitorizar, cada dispositivo consta de tres sensores de temperatura y un sensor de humedad. El objetivo es recopilar datos a distintos niveles de profundidad dentro de la pila de compost, con los datos de humedad se gestiona el aporte de agua para mantener las condiciones óptimas y con los datos de temperatura se podrá conocer cuando se ha alcanzado la fase de volteo de la pila, en la cual, se reactiva el proceso de compostaje al aportar oxígeno.

El dispositivo estará basado en microcontroladores con conexión WiFi ESP8266 y ESP32, los cuales usarán el protocolo de comunicación ESP-NOW para ser lo más eficiente posible en el consumo de energía.

La gestión del dispositivo y datos se integrarán al sistema existente en el huerto docente de Ciencias en el Campus. Esta solución permitirá un proceso de compostaje de forma eficiente y controlada.

PALABRAS CLAVES

Compostaje, Sostenibilidad, Microcontroladores, ESP-Now, MQTT, Monitorización, IoT.

ABSTRACT

This project arises from the need to automate, monitor, and efficiently control the entire treatment and composting process at the station located in the teaching garden of the University of Sciences of Malaga.

A portable and wireless device has been developed, powered by solar-recharged batteries. This device is inserted into a compost pile for monitoring. Each device consists of three temperature sensors and a humidity sensor. The goal is to gather data at different depths within the compost pile. The humidity data is used to manage the water input to maintain optimal conditions, and the temperature data will indicate when the turning phase of the pile has been reached. During this phase, the composting process is reactivated by introducing oxygen.

The device will be based on microcontrollers with WiFi connectivity, specifically ESP8266 and ESP32, which will utilize the ESP-NOW communication protocol for optimal energy consumption.

Device management and data will be integrated into the existing system in the Science teaching garden on campus. This solution will enable an efficient and controlled composting process.

KEYWORDS

Composting, Sustainability, Microcontrollers, ESP-Now, MQTT, Monitoring, IoT.

Contenido

Índice de figuras

Capítulo 1. Introducción

1.1 Antecedentes

En la era actual, el desarrollo de prácticas sostenibles se ha vuelto estrictamente necesario para hacer frente a los desafíos ambientales a los que se enfrenta nuestro planeta. La economía circular es esencial en este contexto, en ella se busca reducir al mínimo el desperdicio y maximizar la utilización de recursos, haciendo frente al modelo lineal tradicional de "tomar, hacer y desechar", la economía circular fomenta la reutilización, el reciclaje y la regeneración, con el objetivo de mantener los materiales en uso durante el mayor tiempo posible.

En este contexto, el compostaje emerge como un método valioso para cerrar el ciclo de vida de los materiales orgánicos. Transformar los residuos orgánicos en compost beneficia la fertilidad del suelo, reduce la acumulación de desechos y fortalece la producción de alimentos. La combinación de estos enfoques contribuye a la conservación de recursos, la reducción de la contaminación y la creación de sistemas más resistentes y equilibrados, asegurando un entorno más saludable y sostenible para el futuro.

1.2 Proceso de compost

El compostaje es un proceso natural de deterioro de materiales orgánicos, como restos de alimentos y residuos vegetales, en condiciones controladas. Se lleva a cabo mediante la acción de microorganismos, como bacterias y hongos, que descomponen los materiales en compuestos más simples y estables. El proceso de compostaje implica la creación de un entorno adecuado para que estos microorganismos prosperen, he aquí la importancia de llevar a cabo este proyecto. A continuación, se describen las fases del proceso de generación de compost.

1. Preparación: Se toman los materiales orgánicos adecuados, como restos de frutas, vegetales, hojas, recortes de césped y se mezclan

para lograr una combinación equilibrada de carbono y nitrógeno en la descomposición.

2. Descomposición: Crecen microorganismos, como bacterias y hongos, los cuales, comienzan a descomponer los materiales orgánicos, como resultado genera calor y libera dióxido de carbono y vapor de agua.
3. Mezcla y volteo: Es importante voltear o mezclar los materiales para asegurar que haya una distribución uniforme de oxígeno y humedad. Este proceso acelera la descomposición y evita la formación de malos olores.
4. Maduración: Con el tiempo, los materiales se descomponen en una sustancia oscura y rica en nutrientes llamada compost. Este proceso puede durar desde varias semanas a meses, dependiendo de las condiciones y los materiales utilizados.
5. Cribado: Una vez que el compost está listo, se puede cribar para eliminar objetos no deseados y obtener un producto uniforme y fino.

El resultado final del compostaje es un producto valioso que puede usarse para enriquecer el suelo en jardines, huertos y áreas verdes, además aumenta la capacidad de retención de agua, proporcionando nutrientes esenciales a las plantas.

1.3 Objetivos

El objetivo principal del proyecto es conocer a través de los índices de temperatura y humedad la fase en la que se encuentra el proceso de compostaje de la pila, con el fin de controlar de manera eficiente la generación del compost y estudiar las propiedades en las mezclas de los materiales orgánicos.

Capítulo 2. Fases del trabajo

El proyecto ha sido desarrollado siguiendo un proceso secuencial que involucra cinco fases distintas enumeradas y descritas a continuación:

2.1 Diseño, elección del Hardware y materiales

En primer lugar, se realiza un estudio de la zona donde serán alojados los equipos de medición y toma de datos, así mismo, se pretende encontrar la forma más adecuada atendiendo a la sostenibilidad y eficiencia para llevar a cabo el montaje. Por ello, se utilizan microcontroladores con pequeños consumos eléctricos con el fin de poder alimentarlos por medio de placas solares y baterías.

Respecto a la elección del material, se han seleccionado sensores de temperaturas y de humedad compuestos por materiales resistentes, con la finalidad de que perduren en el tiempo y sean de fácil integración en el sistema.

Pensando en la posibilidad de controlar y monitorizar múltiples pilas de compost, se diseña un sistema que podemos dividir en dos partes:

1. Estación fija principal: Este es el dispositivo principal encargado de recibir la información de varias estaciones móviles, actúa de pasarela con el servidor del huerto docente de la Universidad de Ciencias, comunicándose y transmitiendo los datos de todas las estaciones móviles para posteriormente monitorizarlos y almacenarlos en la base de datos. Está compuesto por un microcontrolador ESP32 alimentado por una fuente de tensión externa de 5v. Todo el tratamiento de datos se realiza por comunicación inalámbrica únicamente.
2. Estación móvil: Este dispositivo se encarga de tomar las medidas de los sensores y enviarlas a la estación fija. Por otro lado, se ha diseñado y fabricado el prototipo de una estación, puede verse en el *Anexo 1.1 Imagen de la estación móvil*. Estos equipos se introducen en las distintas pilas de compost alojadas en el huerto. La estación móvil está compuesta por:
    a. Pica de toma de datos: Esta consiste en una varilla compuesta por un sensor de humedad y tres sensores de temperatura, los cuales están situados a distinta distancia. La pica irá soterrada

en la pila de compost, puede verse en el *Anexo 1.2 Pica soterrada en pila de compost* y transmitirá los datos de todos los sensores por un único cable al módulo de control, puede verse en el *Anexo 1.3 Pica de toma de datos.*

b. Módulo de control: Consiste en una caja estanca compuesta por:
    - Microprocesador ESP8266 con antena WiFi: Se alimenta mediante una batería recargada con placas solares. El dispositivo se ha programado para comunicar y transmitir de forma inalámbrica los datos de los sensores a la estación fija.
    - Sistema de alimentación: Se encarga de alimentar tanto al ESP8266 como a los sensores. Se compone de:
        1. Placas solares de 5w.
        2. Batería de 1200 mAh, 4,44w y 3,7v.
        3. Circuito electrónico de control de carga de batería y estabilizador de tensión.
    - Pulsador (NO): Su función, al pulsar, es el envío de datos de temperatura y humedad de forma instantánea.

Puede verse en el *Anexo 1.4 Módulo de control de la estación móvil.*

"La información detallada sobre todos estos dispositivos mencionados se encuentra documentada en el Capítulo 2, apartado 2.2 Hardware".

2.2 Estudio de protocolos de comunicación de los distintos dispositivos

Los microcontroladores ESP32 y ESP8266 se caracterizan por su versatilidad a la hora de realizar comunicaciones y por admitir varios protocolos de comunicación. A continuación, se nombran y explican los protocolos de comunicación utilizados en el proyecto.

2.2.1 ESP-Now

Es un protocolo de comunicación inalámbrica desarrollado por “Espressif Systems”. El principal propósito es permitir la comunicación directa y eficiente entre dispositivos con microcontroladores ESP8266 o ESP32 en un rango cercano sin la necesidad de un enrutador WiFi. Como principal característica de funcionamiento, este protocolo ofrece una comunicación de baja latencia y bajo consumo de energía, gracias a que trabaja en la capa de enlace de datos, permitiendo que los dispositivos se comuniquen directamente entre sí, utilizando direcciones MAC únicas, esto hace que sea ideal para el proyecto. Sus características de funcionamiento en términos generales son:

- Creación de nodos: En una red ESP-Now, hay al menos dos tipos de dispositivos: el "nodo maestro" y los "nodos secundarios o esclavos". El nodo maestro es responsable de coordinar la comunicación (en nuestro caso es el ESP32), mientras que los nodos secundarios envían y reciben datos (en nuestro caso el ESP8266).
- Se establecen parejas de dispositivos: Los dispositivos ESP8266 o ESP32 deben ser emparejados antes de poder comunicarse entre sí utilizando ESP-Now. Esto implica que los nodos secundarios deben registrarse en el nodo maestro, estableciendo una relación de comunicación.
- Identificación con dirección MAC e ID: Cada dispositivo en la red ESP-Now tiene una dirección MAC única. El nodo maestro y los nodos secundarios utilizan estas direcciones para identificarse entre sí. Además, los nodos secundarios pueden tener un identificador único asignado por el usuario.
- Publicación y suscripción: ESP-Now utiliza un modelo de comunicación de "publicación-suscripción". Un nodo secundario puede publicar datos, y el nodo maestro reenviará esos datos a otros nodos secundarios que estén interesados en recibirlos. Los nodos secundarios pueden suscribirse a ciertos tipos de datos o dispositivos específicos.
- Transmisión de paquetes: Los datos se encapsulan en paquetes de datos y se envían mediante el protocolo ESP-Now. El nodo maestro tiene la tarea de reenviar los paquetes entre los nodos secundarios según sus

suscripciones. Permitiendo una comunicación eficiente y rápida entre los dispositivos.

- Manejo de errores y retransmisiones: ESP-Now ofrece confiabilidad en la transmisión de datos. Si un paquete se pierde o se corrompe durante la transmisión, se pueden implementar mecanismos de retransmisión para garantizar que los datos lleguen correctamente e identificar nodos que han dejado de funcionar.

La información sobre soluciones de bajo consumo de energía mediante la tecnología ESP-Now se encuentra disponible en [1].

2.2.2 MQTT

Fue desarrollado por “IBM” en la década de 1990, MQTT (Message Queuing Telemetry Transport) es un protocolo de comunicación eficiente y fiable, utilizado para enviar mensajes entre dispositivos en redes con ancho de banda limitado, es popularmente utilizado para las comunicaciones en el mundo del internet de las cosas (IoT).

Los tres elementos principales que forman este protocolo de comunicación son:

- Broker: Es un servidor central que actúa como intermediario entre los clientes. Gestiona la comunicación entre los clientes, recibiendo mensajes de los publicadores y reenviándolos a los suscriptores adecuados, además asegura la entrega según el nivel de calidad de servicio (QoS) especificado.
- Clientes MQTT: Son todos los dispositivos que se comunican a través de MQTT, tienen un identificador único denominado “Cliente ID”. Pueden ser:
    - Publicadores: Son los dispositivos que envía un mensaje a un topic específico en el Broker, este mensaje, puede tener diferentes niveles de calidad de servicio (QoS) para garantizar la entrega.
    - Suscriptores: Son los dispositivos que tras suscribirse en el Broker e indicar qué topics les interesan, reciben los mensajes e información contenida en dichos topics.

- Topics: Son cadenas de texto que se utilizan para organizar y enrutar los mensajes entre los dispositivos. Se organizan jerárquicamente, similar a una estructura de carpetas.

Para más información sobre el protocolo de comunicación MQTT puede consultar [2].

2.3 Diseño del esquema eléctrico y conexionado del sistema

En esta sección, se presenta el diseño detallado de la configuración eléctrica y las conexiones dentro del sistema. Empezamos analizando el conexionado de la pica de toma de datos *Figura 1*.

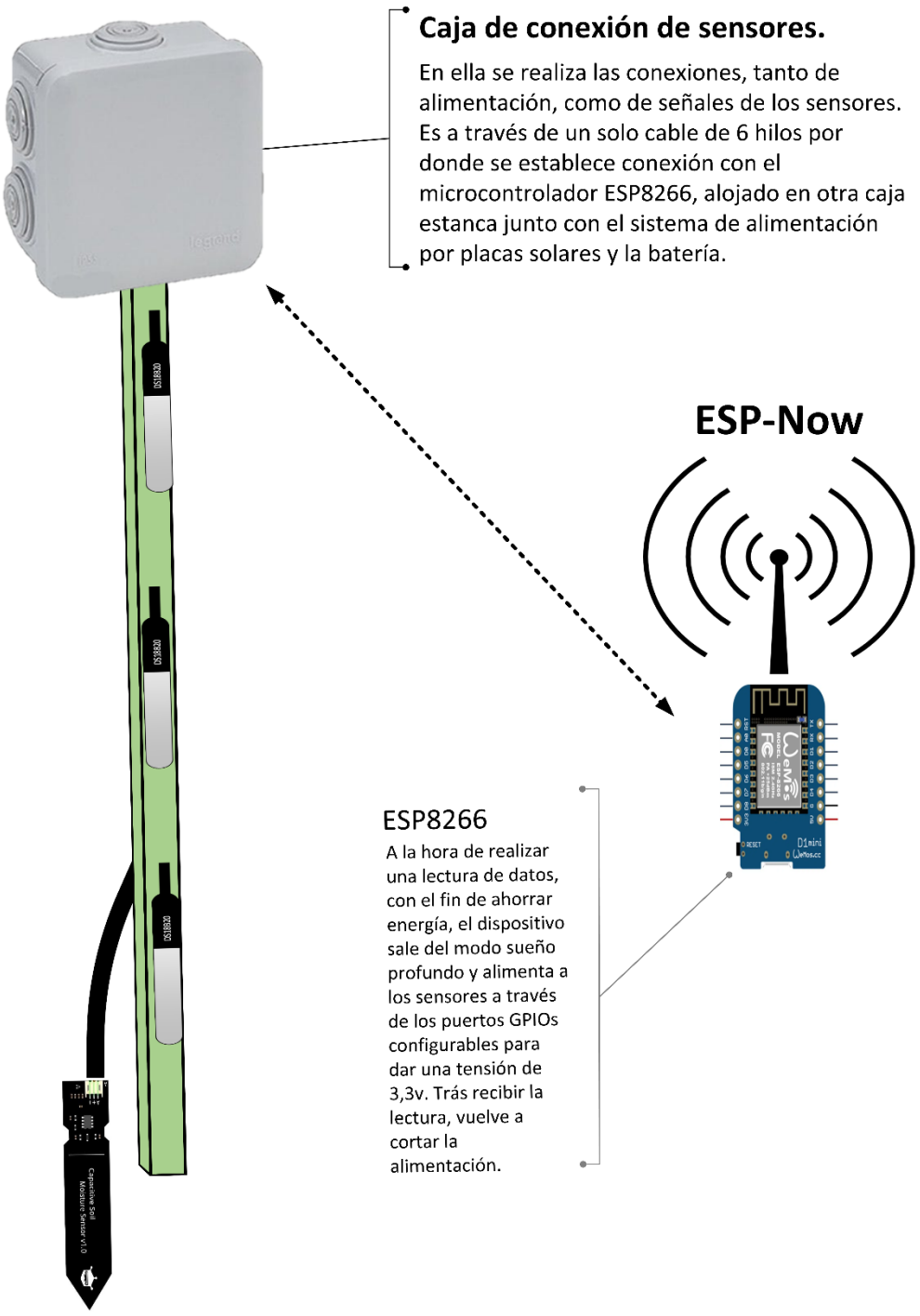

Figura 1.Diseño de La Pica de Toma de Datos. [Elaboración propia]

A continuación, se muestra en la *Figura 2* el esquema eléctrico del módulo de control de la estación móvil, este circuito, ha sido diseñado y fabricado por el alumno en una placa electrónica, la cual, se puede ver en el *Anexo 1.5 Placa electrónica ESP8266.*

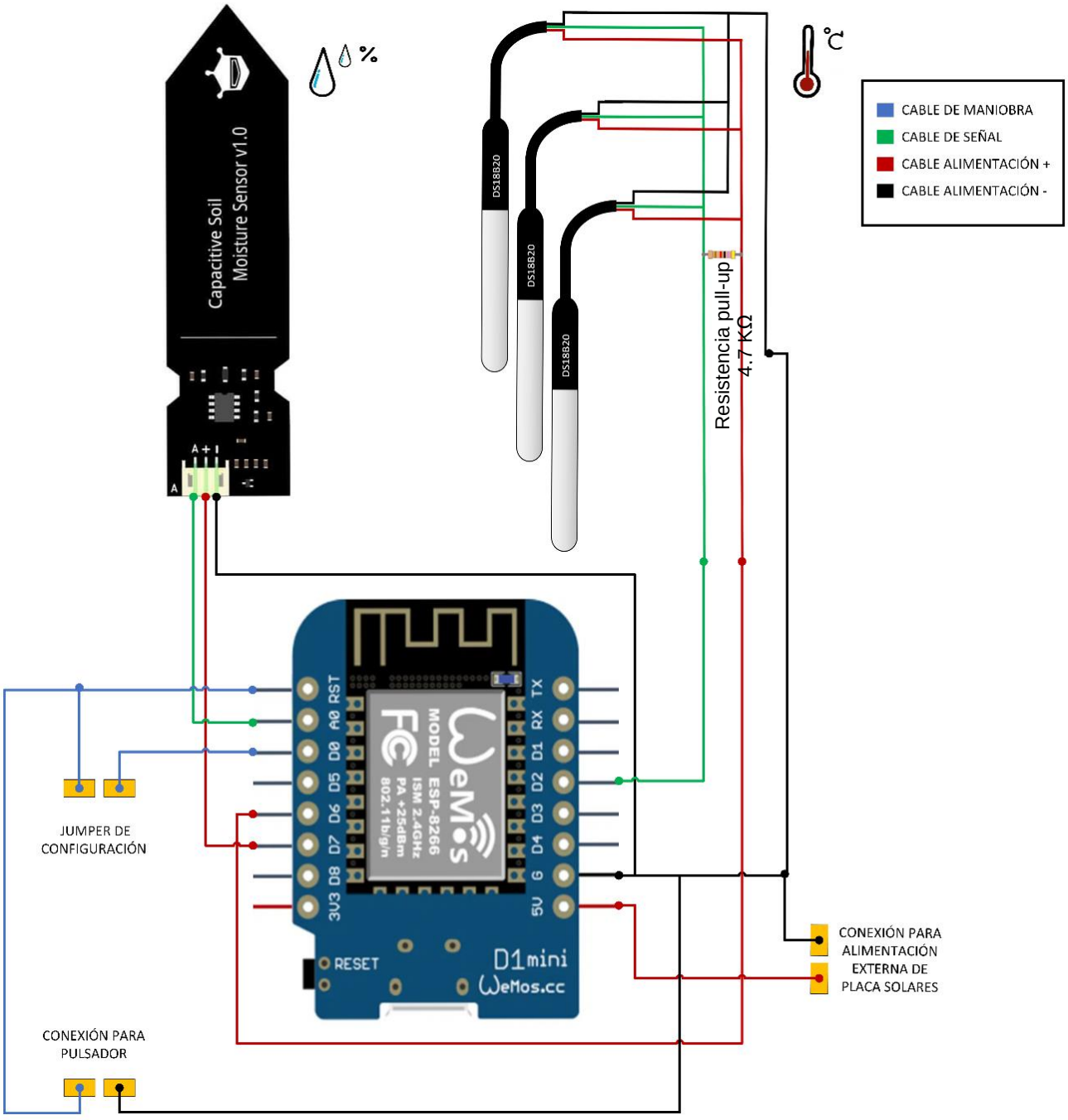

Figura 2. Conexionado de los Dispositivos de La Estación Móvil. [Elaboración propia]

2.4 Diseño del software y gestión de envío de datos

En este apartado, se analizará el diseño del software y la gestión de envío de datos, dividiendo el proyecto en dos partes. Se explorarán tanto la estructura del software como los protocolos de comunicación utilizados, abordando su implementación de forma detallada.

2.4.1 Comunicación entre estaciones móviles ESP8266 y estación fija ESP32.

Desarrollando el proceso de comunicación tenemos dos etapas descritas a continuación.

Etapa 1. “Proceso de emparejamiento entre los dispositivos”.

La primera de las etapas consta de los siguientes pasos:

1. Inicialización del dispositivo emisor (ESP8266)
    - El dispositivo emisor inicia la librería "AUTOpairing" llamando al método “begin()”.
    - La librería establece el modo WiFi del dispositivo emisor en modo estación (WIFI_STA), lo que permite que el dispositivo se conecte a un punto de acceso WiFi.
    - Se inicia la comunicación ESP-NOW llamando a “esp_now_init()”.
2. Solicitud de emparejamiento
    - El dispositivo emisor envía un mensaje de solicitud de emparejamiento (tipo PAIRING) a través de ESP-NOW.
    - El mensaje contiene información sobre el dispositivo emisor, como su dirección MAC y el canal en el que se encuentra.
3. Recepción de solicitud de emparejamiento en el dispositivo receptor (ESP32)
    - El dispositivo receptor detecta el mensaje de solicitud de emparejamiento a través de la función de callback “OnDataRecv”.
    - El dispositivo receptor verifica que el mensaje sea del tipo PAIRING y extrae la información, como la dirección MAC y el canal, del mensaje.

4. Respuesta de emparejamiento
    - El dispositivo receptor crea un mensaje de respuesta de emparejamiento (tipo PAIRING) y lo envía de vuelta al dispositivo emisor.
    - El mensaje de respuesta contiene información sobre el dispositivo receptor, como su dirección MAC y el canal en el que se encuentra.
5. Confirmación de emparejamiento (Dispositivo Emisor)
    - El dispositivo emisor recibe el mensaje de respuesta de emparejamiento a través de la función de callback “OnDataRecv”.
    - El dispositivo emisor verifica que el mensaje sea del tipo PAIRING y extrae la información del mensaje.
    - La librería establece la configuración del emparejamiento en la memoria RTC para emparejamientos futuros.

Etapa 2. “Procedimiento de comunicación ESP-NOW”.

Una vez emparejado, para recibir y enviar datos, los dispositivos utilizan las funciones descritas en los siguientes puntos.

- Envío de datos (Dispositivo emisor)

El dispositivo emisor crea un mensaje de datos (tipo DATA), el cual, contiene el payload que se desea enviar y llama a “espnow_send” para enviar el mensaje a través de ESP-NOW al dispositivo receptor.

- Recepción de datos (Dispositivo receptor)

El dispositivo receptor recibe el mensaje de datos a través de la función de callback “OnDataRecv”. A continuación, verifica el tipo del mensaje (DATA) y procesa el payload del mensaje.

- Confirmación de envío (Dispositivo emisor)

El dispositivo emisor recibe la confirmación de envío a través de la función de callback “OnDataSent”, si el envío fue exitoso, el dispositivo emisor marcar el mensaje como enviado.

Para finalizar este apartado, se puede visualizar en la *Figura 3* un esquemático que resume el diseño de la red ESP-Now.

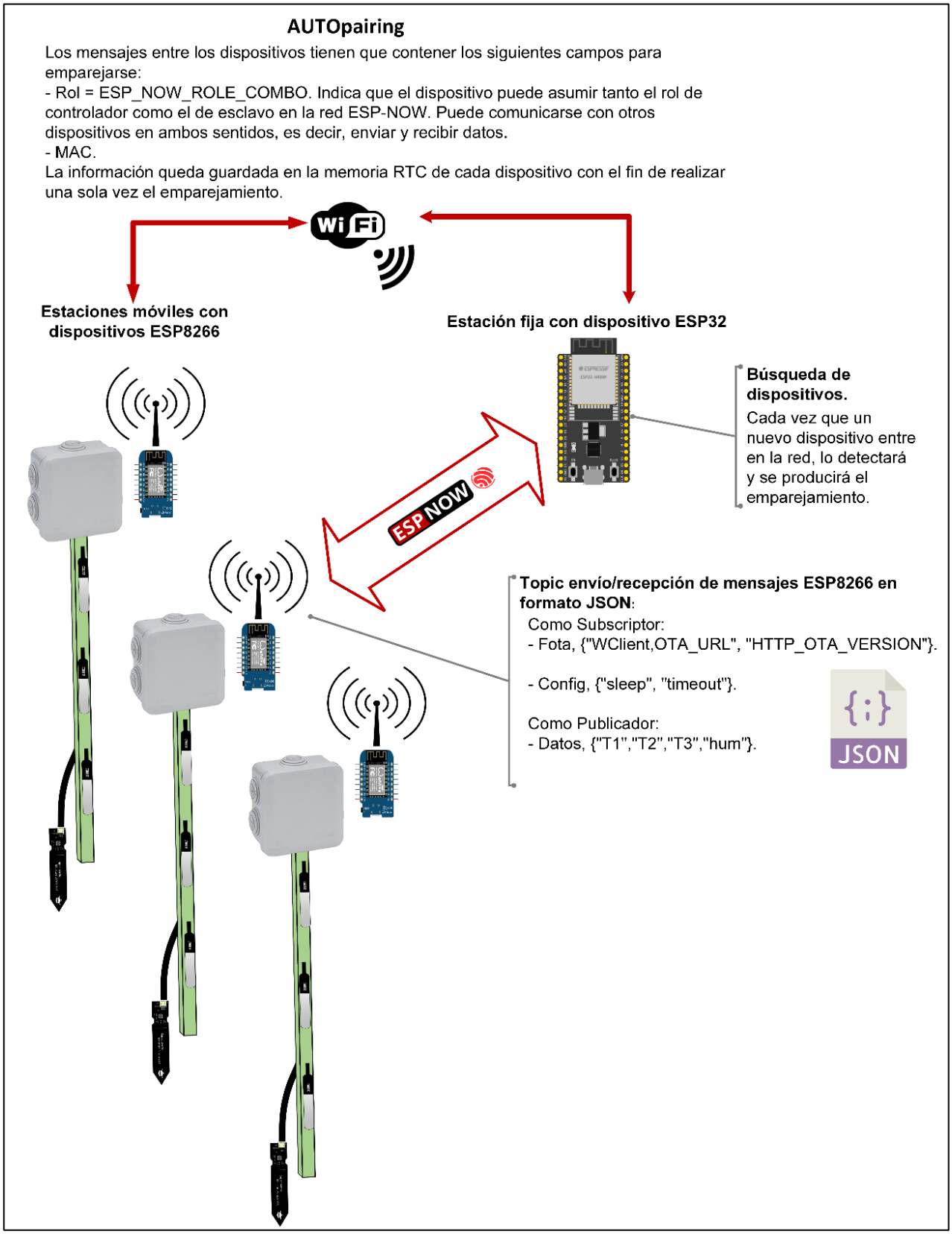

Figura 3. Diseño red ESP-Now. [Elaboración propia]

2.4.2 Comunicación entre el ESP32 y el servidor Web.

A continuación, se desglosa cómo el dispositivo ESP32 se comunica con el Broker MQTT.

1. Inicialización del cliente MQTT. En la función “setup()”, se crea una instancia del objeto “PubSubClient” con la dirección IP, el dominio del Broker MQTT y el número de puerto. En el proceso de iniciación se establece la función de callback “procesa_mensaje” que se ejecutará cada vez que llega un mensaje MQTT al dispositivo.
2. Conexión al Broker MQTT. En la función “conecta_mqtt()”, el dispositivo intenta conectarse al Broker MQTT.
    - Se llama a “mqtt_client.connect()” con el ID del dispositivo, el nombre de usuario y la contraseña.
    - Si la conexión es exitosa, el dispositivo se suscribe al tema "infind/espnow/" utilizando “mqtt_client.subscribe()” para recibir mensajes MQTT entrantes.
3. Publicación de mensajes. A la hora de publicar mensajes, se procesan los siguientes puntos.
    - En la función “loop()”, el dispositivo verifica si está conectado al Broker MQTT utilizando “mqtt_client.connected()”.
    - Si está conectado, puede publicar mensajes utilizando la función “mqtt_client.publish()”.
    - Los mensajes MQTT se publican en topic específicos, utilizando la dirección MAC del dispositivo y el payload del mensaje.
4. Recepción de mensajes. Cuando llega un mensaje MQTT al Broker en un topic al que el dispositivo está suscrito, la función de callback “procesa_mensaje()” se ejecuta. Dentro de esta función, se procesa el mensaje JSON recibido, se extraen las partes relevantes (como la dirección MAC, el topic y el payload) y se agregan a colas para su posterior procesamiento.
5. Procesamiento de mensajes. El dispositivo puede procesar los mensajes recibidos según sus necesidades. En este código, los mensajes MQTT se

convierten en mensajes ESP-NOW y se envían a dispositivos emparejados utilizando el protocolo ESP-NOW.

6. Desconexión. Cuando el dispositivo ya no necesita mantener la conexión, envía un mensaje DISCONNECT al Broker para cerrar la sesión de manera ordenada. Esto libera los recursos en el Broker y finaliza la comunicación.

En la *Figura 4* se muestra un esquema de conexión, el cual representa como se realiza la comunicación MQTT entre los dispositivos, los topic creados, así como, que dispositivo realiza la suscripción y cual envía la publicación en cada topic.

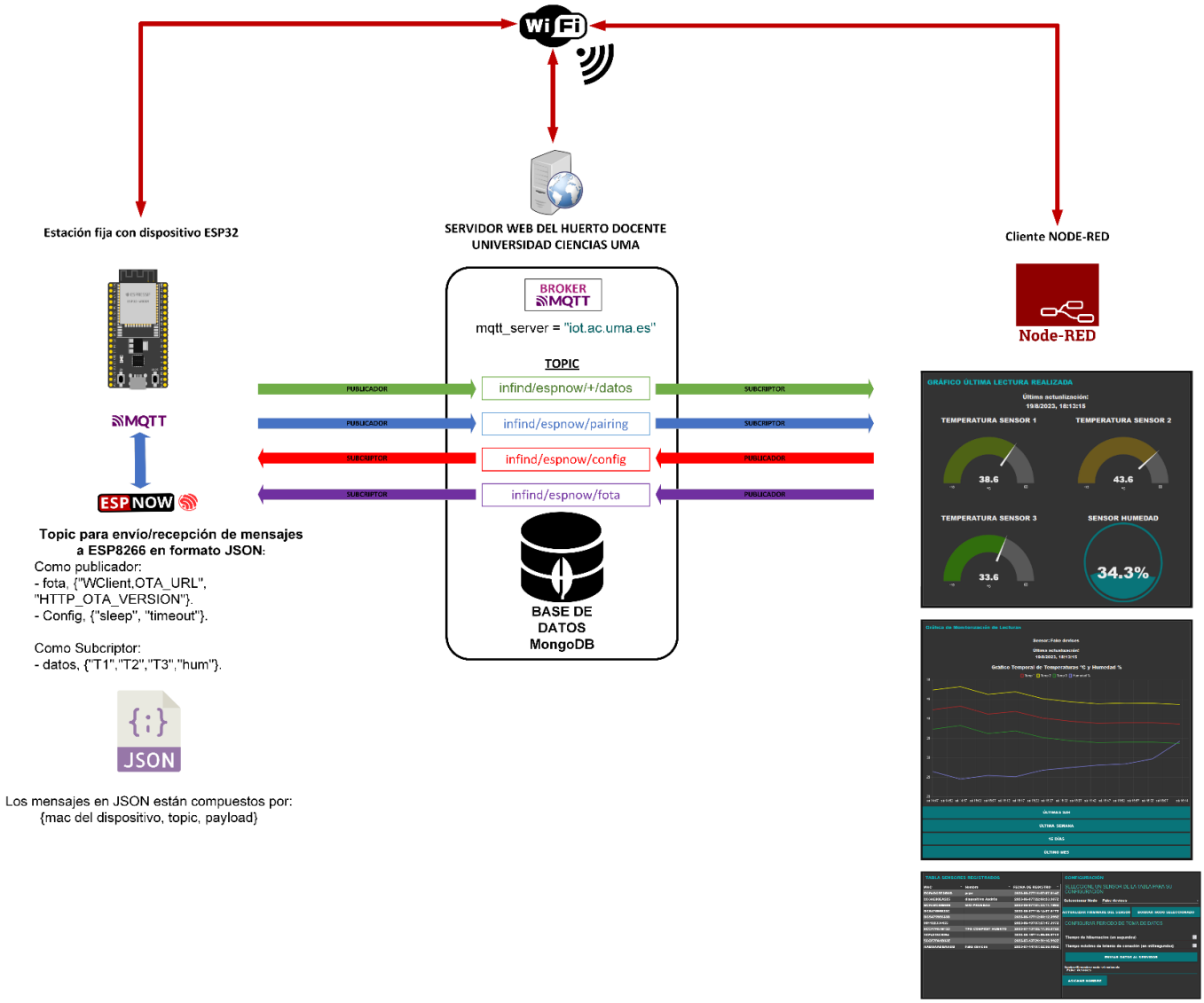

Figura 4. Comunicación MQTT entre ESP32 con servidor web. [Elaboración propia]

2.5 Diseño y programación de la Interfaz de monitorización

A través de Node-RED se ha creado un panel de control y monitorización en tiempo real, por el que podemos supervisar los datos adquiridos por los sensores, así como, ver los nodos activos y configurar el nodo que se desee. La interfaz se compone de dos pantallas. La primera de estas es la “Pantalla de configuración” (Ver *Figura 5*) y la segunda “Pantalla de histórico de datos” (Ver *Figura 6*).

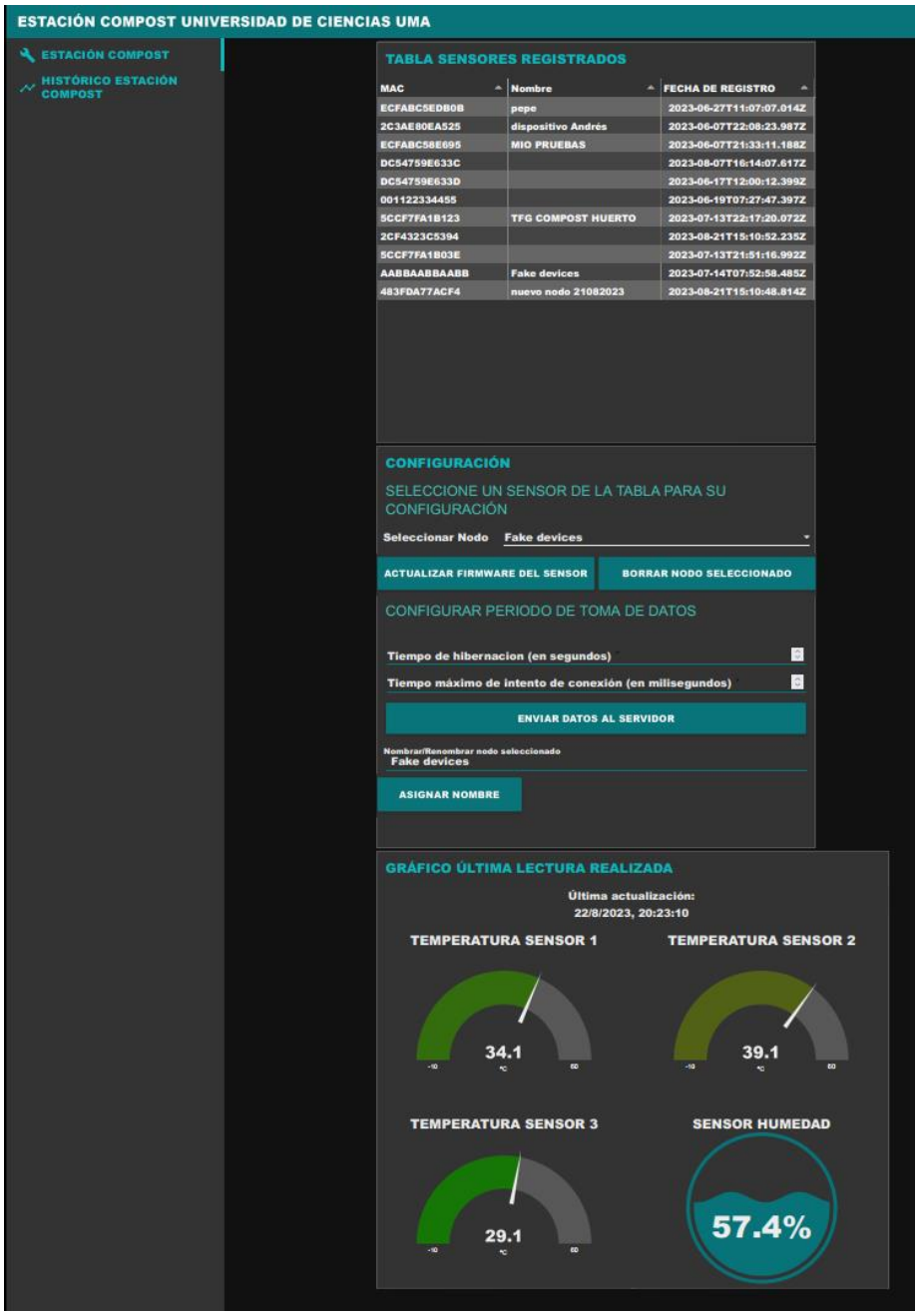

Figura 5. Pantalla configuración. [Elaboración propia]

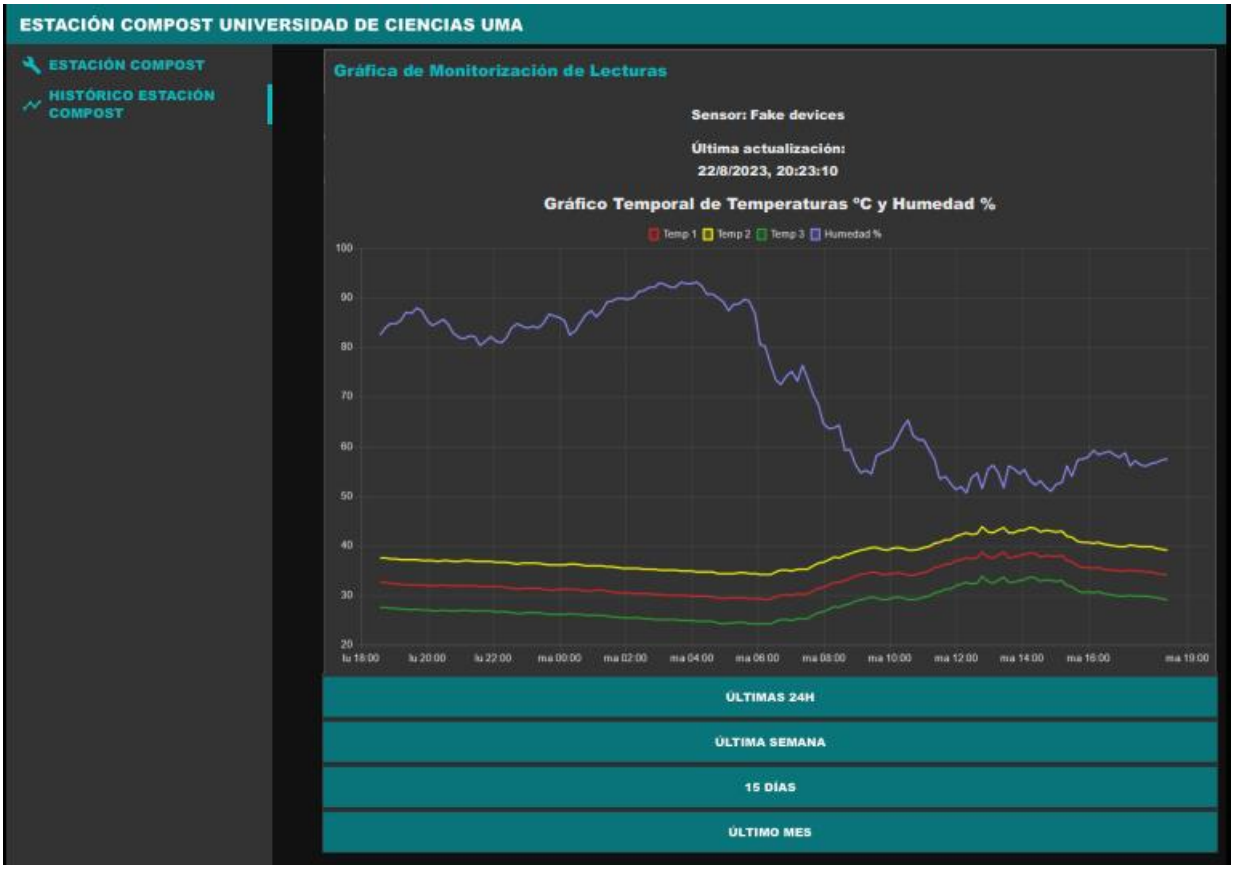

Figura 6. Pantalla de histórico de datos. [Elaboración propia]

1. Pestaña “ESTACIÓN COMPOST”, a través de la cual podemos acceder a la “Pantalla de configuración”, se puede ver en la *Figura 7.*

Figura 7. Pestaña ESTACIÓN COMPOST. [Elaboración propia]

Una vez dentro de esta pantalla, se muestran tres secciones.

1. “TABLA SENSORES REGISTRADOS”. Representada en la *Figura 8*, en la tabla aparecerán de forma automática y en tiempo real, todos los nodos que se emparejen con la estación fija.

Figura 8. Tabla de nodos registrados. [Elaboración propia]

2. "CONFIGURACIÓN". Sección visualizada en la *Figura 9,* seleccionando uno de los nodos de la tabla anterior, se autocompletan los datos en la sección de configuración y el usuario podrá:
    - Actualizar el Firmware del dispositivo.
    - Nombrar o renombrar el nodo como se desee.
    - Modificar el tiempo máximo de conexión del nodo con el servidor y actualizar el tiempo de adquisición y envío de datos.

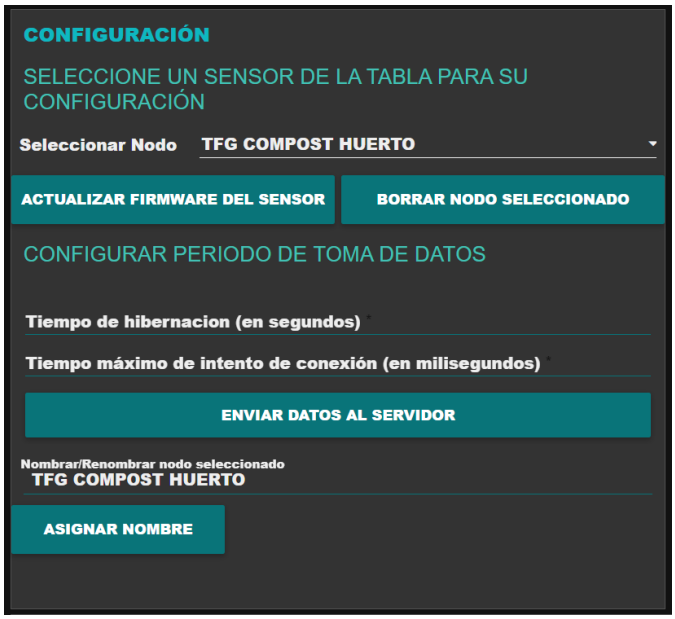

Figura 9.Configuración. [Elaboración propia]

3. “GRÁFICO ÚLTIMA LECTURA REALIZADA”. En esta última sección (Ver *Figura 10*), de esta pantalla, podemos ver los gráficos con la última lectura realizada de los sensores pertenecientes al nodo seleccionado.

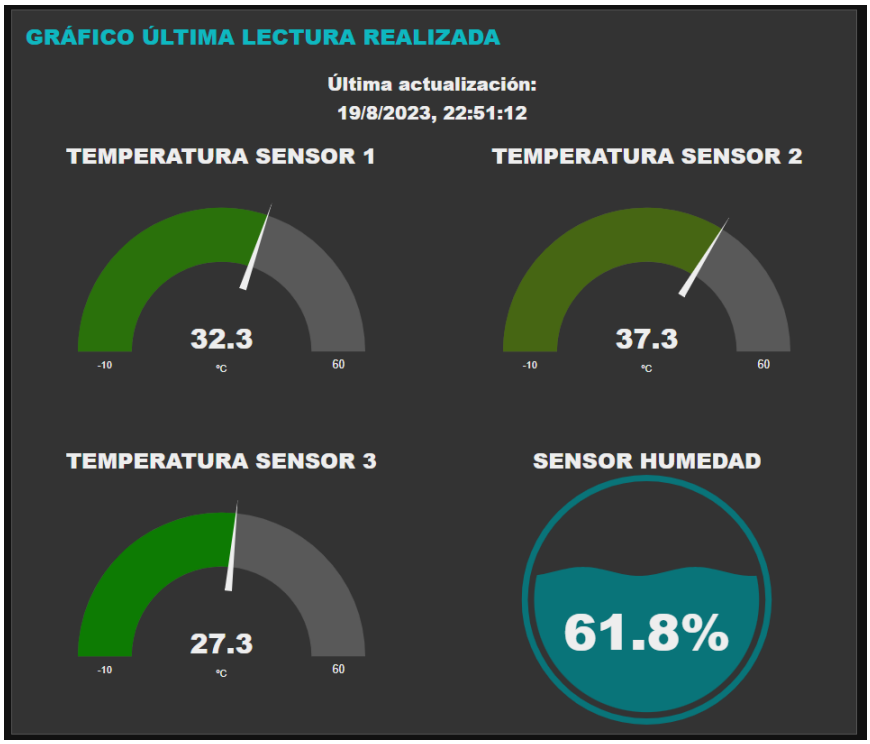

Figura 10.Gráfico de últimas lecturas realizadas. [Elaboración propia]

2. Pestaña “HISTÓRICO ESTACIÓN COMPOST”. Representada en la *Figura 11*, permite el acceso a la “Pantalla de histórico de datos”.

Figura 11.Pestaña HISTÓRICO ESTACIÓN COMPOST. [Elaboración propia]

En está pantalla se muestra un gráfico de líneas, perteneciente al sensor seleccionado en la pestaña de configuración, donde aparecen las lecturas de los sensores de temperatura y de humedad. El usuario puede seleccionar el periodo de tiempo a graficar (últimas 24 horas, última semana, últimos 15 días o último mes) gracias a los botones que aparecen en la zona inferior del gráfico (Ver *Figura 12*).

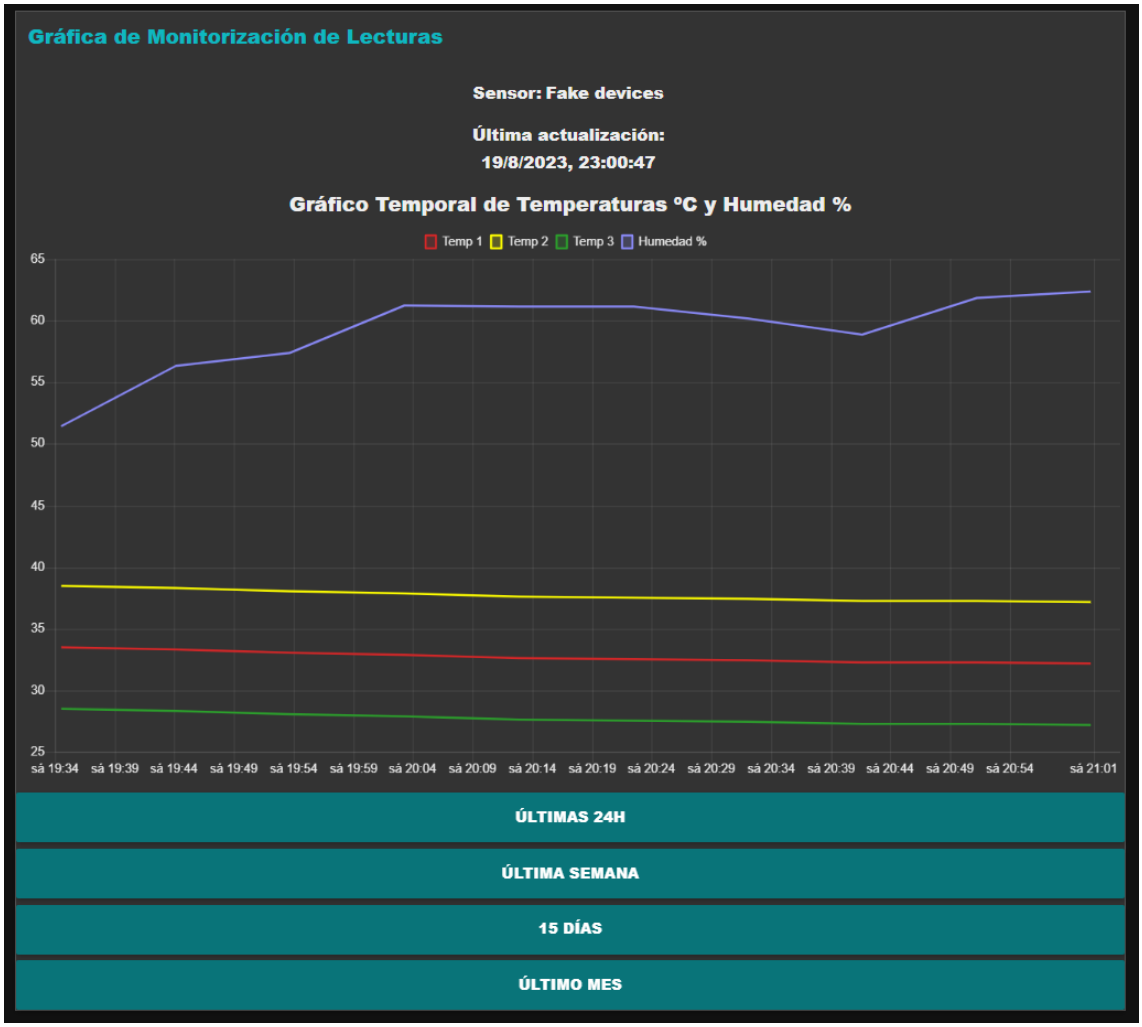

Figura 12.GRÁFICO HISTÓRICO ESTACIÓN COMPOST. [Elaboración propia]

Capítulo 3. Software y hardware empleado

En esta sección, detallaremos el software y hardware seleccionados para alcanzar los requisitos y objetivos del proyecto, garantizando un rendimiento óptimo.

3.1 Software

En la realización del proyecto se han utilizado dos softwares diferentes y un sistema de gestión de bases de datos gestionado a través de Node-RED, llamado MongoDB.

3.1.1 Arduino

El software de Arduino es una herramienta que ha tomado gran importancia en el mundo de la electrónica y la programación, ya que además de ser open source, ha sido diseñado para ser accesible y amigable, con una interfaz intuitiva, permite escribir, cargar y depurar código en microcontroladores de manera eficiente. Brinda la flexibilidad necesaria para crear proyectos complejos de automatización y robótica.

Para más información el lector puede dirigirse a la siguiente web de Arduino [3].

Figura 13. Logo Arduino. [4]

3.1.2 Node-RED

Node-RED es una plataforma gráfica de programación, cuya interfaz de usuario corre a través de un navegador. Los usuarios pueden arrastrar y soltar nodos prediseñados con funciones específicas para construir flujos que pueden proporcionar soluciones complejas, tanto en la recopilación de datos, como en la

toma de decisiones y ejecución de acciones. Node-RED se ha convertido en una gran herramienta para la creación rápida de prototipos y la implementación de proyectos en el ámbito del internet de las cosas (IoT).

Los factores que han llevado a utilizar esta herramienta han sido:

- Interfaz visual intuitiva: Su entorno gráfico facilita la creación y gestión de flujos de datos y lógica.
- Rápido prototipado: Facilita la creación rápida de prototipos y pruebas de concepto para proyectos IoT y automatización.
- Integración con plataformas: Puede conectarse a diversas plataformas y servicios populares, mejorando la interoperabilidad.
- Código abierto: Al ser de código abierto, es gratis, accesible y adaptable a diferentes entornos y requisitos.

Para más información pueden acceder a la página web del desarrollador [5].

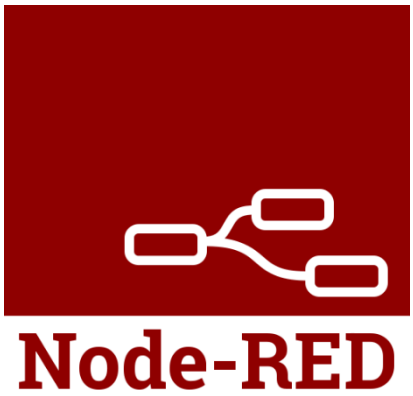

Figura 14. Logo Node-RED. [16]

3.2 Hardware

La realización del proyecto se ha llevado a cabo principalmente con los dispositivos expuestos a continuación.

3.2.1 ESP8266

El ESP8266 es un microcontrolador que ha evolucionado hasta convertirse en una plataforma completa de desarrollo. Entre sus características principales, destacamos:

- Conectividad WiFi integrada: Incluye un módulo WiFi que permite la conexión a redes inalámbricas, facilitando la comunicación y la transmisión de datos.
- Procesador de bajo consumo: Tiene un tamaño compacto y cuenta con un procesador eficiente que equilibra el rendimiento y el consumo de energía.
- Amplia gama de pines: Ofrece una variedad de pines GPIO (Entrada/Salida de Propósito General) que permiten la conexión y control de diversos componentes electrónicos.
- Memoria flash integrada: Tiene memoria flash incorporada que se utiliza para almacenar el firmware, programas y datos.
- Modos de suspensión: Ofrece modos de suspensión para ahorrar energía cuando no se requiere una operación activa.
- Comunicación serial: Admite comunicación serial (UART) para la interacción con otros dispositivos.
- Interfaz I2C y SPI: Ofrece interfaces I2C y SPI que facilitan la conexión a sensores y otros dispositivos.
- Soporte para servidor web: Puede funcionar como un servidor web, lo que permite crear interfaces de usuario y aplicaciones en línea.
- Programación versátil: Se puede programar utilizando lenguajes y entornos como el Arduino.

Todo ello, ha hecho que nos hayamos decantado por este dispositivo para crear el sistema de toma de datos portátil que recoge la información de los sensores y se comunica de forma inalámbrica a través de la pasarela para transmitir los datos con el ESP32. En la siguiente web podrán consultar todo tipo de información del dispositivo, así como ejemplos de diversos proyectos [7].

3.2.2 ESP32

El ESP32 es la evolución del microcontrolador ESP8266, ha sido diseñado para ofrecer un mayor rendimiento y capacidades de conectividad más amplias. Este dispositivo integra un potente procesador de doble núcleo, junto con

conectividad WiFi y Bluetooth, lo que lo convierte en una herramienta ideal para programar y gestionar aplicaciones de Internet de las cosas (IoT) y proyectos electrónicos. Las principales características que lo diferencian del ESP8266 son:

- Doble núcleo: Ofrece un procesador de doble núcleo que permite realizar múltiples tareas de manera eficiente.
- WiFi y Bluetooth: Integra conectividad WiFi y Bluetooth clásica y de baja energía (BLE), lo que facilita la comunicación inalámbrica.
- Potencia de procesamiento: Su procesador más rápido permite un mejor rendimiento en comparación con el ESP8266.
- Memoria Flash y RAM: Cuenta con mayor capacidad de memoria flash y RAM para almacenamiento y ejecución de programas.
- Soporte para pantallas y sensores: Puede manejar pantallas y sensores de manera eficiente, lo que lo hace adecuado para aplicaciones multimedia y de detección.
- Seguridad: Ofrece capacidades de seguridad mejoradas, incluido el cifrado de datos y las funciones de seguridad de hardware.

Toda la información acerca del dispositivo se puede encontrar en la web [8].

3.2.3 Sensores

El sistema se compone de tres sensores de temperatura DS18B20 y un sensor de humedad capacitivo, las características principales se analizan a continuación.

3.2.3.1 Sensor digital de temperatura DS18B20

Es un sensor de temperatura altamente preciso, fabricado por “Maxim Integrated”, utiliza la tecnología de medición digital de temperatura para proporcionar lecturas confiables y exactas. La comunicación la realiza a través del protocolo 1-Wire, este protocolo necesita solo un pin de datos para comunicarse y permite conectar más de un sensor en el mismo bus. Sus características son las siguientes:

- Medición digital: Proporciona lecturas digitales precisas de temperatura, lo que elimina la necesidad de tener que realizar conversiones de medidas.
- Comunicación por un cable: Utiliza un protocolo de comunicación por un cable, lo que facilita la integración en proyectos con limitaciones de cableado.
- Precisión: Ofrece una alta precisión en las mediciones de temperatura, lo que lo hace adecuado para aplicaciones sensibles a cambios pequeños en la temperatura.
- Rango amplio de temperatura: Puede medir temperaturas en un rango amplio, esto la hace idóneo para diversas condiciones ambientales.
- Calibración interna: Incluye una función de calibración interna que mejora la precisión de las mediciones.
- Alimentación por el Bus: Puede ser alimentado directamente a través del cable de comunicación, simplificando el diseño del circuito.
- Amplia compatibilidad: Soporta diversas plataformas y microcontroladores, lo que facilita su integración en proyectos.
- Robusto y duradero: Presenta un encapsulado resistente con gran durabilidad y confiabilidad.

A continuación, en la *Figura 15* se muestra una imagen de la sonda.

Figura 15. Sonda sensor de temperatura DS18B20. [9]

3.2.3.2 Sensor de humedad capacitivo soil moistureV1.0

Es un dispositivo diseñado para medir la humedad evaluando con alta sensibilidad los cambios en la capacitancia causados por la presencia de agua en el suelo. Presenta una sonda de detección que se inserta en el suelo, diseñada para soportar condiciones ambientales variables y entornos de uso al aire libre. Es un sensor a tres hilos, que trabaja con una tensión que va desde 3,3 v a 5,5 v, la comunicación se realiza mediante una salida analógica.

Por estas características y su fácil integración a la hora de programación y conexión se ha elegido para de toma de medidas de humedad del compost.

En la siguiente imagen *Figura 16* podemos ver la sonda capacitiva.

Figura 16. Sonda sensor de humedad Capacitive Soil Moisture v1.0. [10]

Capítulo 4. Desarrollo del software

4.1 Software ESP32

El microcontrolador ESP32 se ha programado para que actúe como nodo intermediador entre los dispositivos ESP8266 y el servidor web del huerto docente. El código consiste en una pasarela ESP-Now a MQTT con autoemparejamiento entre dispositivos, contenido en el *Anexo 2.2.1 ESP32_auto_pairing.ino*. A continuación, se explica de forma detallada las partes principales del código y las acciones llevadas a cabo en cada una de ellas.

- Función principal de configuración “void setup()”

Esta función se ejecuta solamente una vez al inicio del dispositivo. Las acciones que implementa son:

- Inicializa la comunicación en serie. Se inicia la comunicación con el monitor serie a una velocidad de 115200 baudios para la depuración y visualización de mensajes.
- Establece la conexión WiFi. Antes de realizar la conexión, ejecuta la función “sprintf(ID_PLACA, "ESP_%d", ESP.getEfuseMac())”, con el fin de adquirir la dirección MAC del dispositivo y almacenarla en la variable “ID_PLACA” para utilizarla en el protocolo de comunicación ESP-Now. A continuación, Se llama a la función “conecta_wifi” para establecer la conexión WiFi utilizando las credenciales proporcionadas (SSID y contraseña).
- Realiza tareas de configuración. En este punto, se configura:
  - Configuración del Canal WiFi: Se obtiene el número del canal WiFi en el que se encuentra el dispositivo ESP32 y se almacena en la variable “myChannel”.
  - Configuración del Cliente MQTT: Se configura el cliente MQTT (mqtt_client) para que se conecte al servidor MQTT en la dirección y puerto especificados. También se establece el tamaño del búfer de mensajes a 512 bytes y se registra la función “procesa_mensaje()” como callback para manejar los mensajes MQTT entrantes.

- Configura ESP-NOW. Se llama a la función “initESP_NOW()” para inicializar la comunicación utilizando el protocolo ESP-NOW. En ella se definen otras funciones y callbacks relacionados con ESP-NOW, como “addPeer” para agregar dispositivos emparejados, “OnDataSent” para manejar la confirmación de envío y “OnDataRecv” para procesar los datos recibidos.

- Función “encuentra_mensaje()”

Se ejecuta antes de la función principal “void loop()”. Busca un mensaje en la cola de mensajes MQTT para un dispositivo específico. Para ello se ha declarado como un iterador “std::list<TmensajeMQTT>::iterator”, esto indica que la función “encuentra_mensaje()”,recorre la lista de mensajes MQTT (“TmensajeMQTT”), buscando y captando mensajes procedente del dispositivo cuya MAC se le pase como argumento(uint8_t *_mac) para poder procesarlos posteriormente.

- Función principal del programa “void loop()”

Es el bucle principal del programa, en esta función se implementa la pasarela ESP-Now a MQTT. Se encarga de mantener en funcionamiento las comunicaciones MQTT y ESP-NOW, gestionando los dispositivos listos para ser emparejados o los ya emparejados y activos, para enviar o revivir mensajes pendientes a través de ambos protocolos de comunicación.

4.2 Software ESP8266

En este punto se analiza el código del programa diseñado para el dispositivo ESP8266, contenido en el *Anexo 2.1.1 Estacion_compos_esp8266_emisor.ino*.

Es una implementación que recopila datos de temperatura y humedad de sensores y los envía a través del protocolo de comunicación ESP-NOW. Además, permite la actualización de firmware y la configuración remota a través de mensajes recibidos.

El programa desarrollado se divide y analiza en seis secciones:

1. Declaración de librerías

En esta sección se incluyen las librerías necesarias para el programa. Estas librerías proporcionarán las funciones y clases necesarias para la comunicación con los sensores, la conexión a redes WiFi y la actualización del firmware a través de OTA (Over-the-Air).

2. Declaración de pines de alimentación de los sensores, variables, estructura y objeto

En primer lugar, se definen los pines para controlar la alimentación de los sensores de temperatura y humedad. A continuación, se declaran variables para almacenar los valores de humedad y de temperatura (las variables “S1_T”, “S2_T” y “S3_T” representan las temperaturas individuales de cada sensor, mientras que el array “temp” se utiliza para almacenar los valores de temperatura de los tres sensores).

Se instancia un objeto de la clase “AUTOpairing_t”, definido en el código del *Anexo 2.1.1 AUTOpairing.h,* llamado “clienteAP” para manejar la comunicación a través de ESP-NOW.

Se define una estructura llamada “mi_configuración” que tiene dos campos "sleep” y “timeout”, ambos de tipo uint32_t. La estructura se utiliza para almacenar la configuración del programa, definida por el tiempo de sueño profundo y el tiempo máximo de espera en realizar la conexión.

3. Declaración de variables para la actualización vía FOTA (Firmware Over-the-Air)

Aquí se definen variables relacionadas con la actualización del firmware a través de FOTA. Se define el nombre de la red WiFi (ssid) y la contraseña (password) para la conexión, la URL para la actualización del firmware (OTA_URL) y la versión del firmware (HTTP_OTA_VERSION). Además, se crea un objeto WiFiClientSecure llamado “WClient” para la conexión segura con el servidor.

4. Declaración de la función “procesa mensaje”

Es una función compleja, la cual se encarga de procesar los mensajes recibidos a través de ESP-NOW. Está compuesta por dos argumentos:

- Topic: Es el primer campo del mensaje recibido y nos indica cual es el tema.
- Payload: Es la parte del mensaje que contiene los datos.

La función “procesa mensaje” dependiendo del topic del mensaje, se realiza dos acciones:

- La actualización del firmware vía FOTA.
- Configuración y establecimiento de los parámetros de tiempo de sueño profundo y tiempo máximo de establecimiento de conexión.

A continuación, para una mayor comprensión del funcionamiento y del código de la función, se desglosan y analizan los pasos que se ejecutan en la transición del código de forma secuencial. Estos son:

- Imprime información sobre el mensaje recibido, el tema (topic) y el mensaje (payload).
- Comprueba el tema del mensaje, utilizando una estructura de control if-else. Hay dos posibles temas:
  a. Si el tema es "fota", significa que se ha recibido un mensaje para realizar una actualización de firmware. En este caso, corre el código de actualización FOTA.

- Realiza la conexión WiFi utilizando la librería “<ESP8266WiFi.h>”. Si no conecta transcurrido el tiempo establecido, el nodo pasa a estado de sueño profundo, si conecta, a través de la clase WClient realiza la actualización llamando a la función “ESPhttpUpdate.update(WClient,OTA_URL, HTTP_OTA_VERSION)”.
- Si no realiza la conexión el nodo entra en sueño profundo.

b. Si el tema es "config", significa que se recibió un mensaje para actualizar la configuración del dispositivo. En este caso, ejecuta el código correspondiente para realizar la configuración.
   - Se crea un StaticJsonDocument con un tamaño adecuado para cargar el mensaje JSON recibido.
   - Se descompone el mensaje JSON usando “deserializeJson” para convertir el mensaje en un documento JSON manejable.
   - Se comprueba si la deserialización fue exitosa.
   - Se verifica si el documento JSON contiene los campos "sleep" y "timeout". Si están presentes, significa que se recibió una configuración válida.
   - Se extraen los valores de "sleep" y "timeout" del documento JSON y se almacenan en la estructura de configuración “mi_configuracion”.
   - Finalmente, se llama al método “set_config” del objeto “clienteAP” para actualizar la configuración con los nuevos valores.

5. Entrada en el bucle de Setup

El código de este bucle se ejecuta en el arranque e inicialización del dispositivo. A continuación, se desarrolla su funcionamiento de forma secuencial ejecutando los siguientes pasos:

- Se configura la alimentación de los sensores de temperatura y humedad mediante los pines “Power_s_temp” y “Power_s_hum”. Se establece su modo como salida y se encienden inicialmente.

- Se inicia la comunicación serie (puerto serie) con una velocidad de transmisión de 115200 baudios.
- Se imprimen mensajes en la consola para indicar que la configuración está comenzando.
- Se inicializa el tamaño de la configuración del objeto “clienteAP” para que coincida con el tamaño de la estructura “mi_configuracion”.
- Se verifica si hay una configuración previamente guardada utilizando el método “get_config” de la clase “clienteAP”. Si no hay configuración guardada, se establecen valores por defecto para “sleep” y “timeout” en la estructura “mi_configuracion”.
- Se imprimen en la consola los valores de “sleep” y “timeout” obtenidos o establecidos.
- Se configuran parámetros de la clase “clienteAP” utilizando los métodos:
  a. “set_timeOut”, establece el tiempo máximo de conexión.
  b. “set_deepSleep”, establece el tiempo en sueño profundo (en segundos).
  c. “set_channel, set_debug”, establece el canal donde empieza el escaneo de dispositivos en la red WIFI para el autoemparejamiento.
  d. “set_callback”, llama a la función “procesa_mensajes”.
- Se inicia la comunicación a través de ESP-NOW llamando al método “begin” de “clienteAP”. Este método, se corresponde con la función definida como “void begin()”en el código contenido en el *Anexo 2.1.2 AUTOpairing.h.*
- Se realiza una adquisición de las direcciones de los sensores de temperatura llamando a la función “adquisicion_direcciones_temp()”. Dicha función se encuentra definida en el código del *Anexo 2.1.3 Funciones.cpp.*

6. Entrada en el bucle principal.

En el bucle, se retoma la conexión cada vez que el nodo despierta y se comprueba si hay mensajes disponibles de actualización o configuración del dispositivo. A continuación, obtienen los datos de temperaturas y humedad de los sensores, una vez recibidos deja de alimentar a los sensores y se crea un mensaje con los datos en formato JSON. Luego se utiliza en método

“espnow_send_check()” de la clase “clienteAP” para enviar el mensaje a través de ESP-NOW. Después de enviar los datos, el dispositivo entra en modo sueño profundo (deepsleep) hasta que se active nuevamente transcurrido en el tiempo fijado por configuración.

4.3 Node-RED

En este apartado se analizan cada uno de los flujos creados en Nore-RED.

1. Flujo para realizar el registro de dispositivos emparejados (Ver *Figura 17*).

   Este flujo en Node-RED se utiliza para administrar emparejamientos de dispositivos utilizando direcciones MAC, actualizando la base de datos MongoDB según sea necesario, y también permite la comunicación a través de un Broker MQTT("iot.ac.uma.es"). El nodo se suscribe al tema específico "infind/espnowpairing" para recibir mensajes.

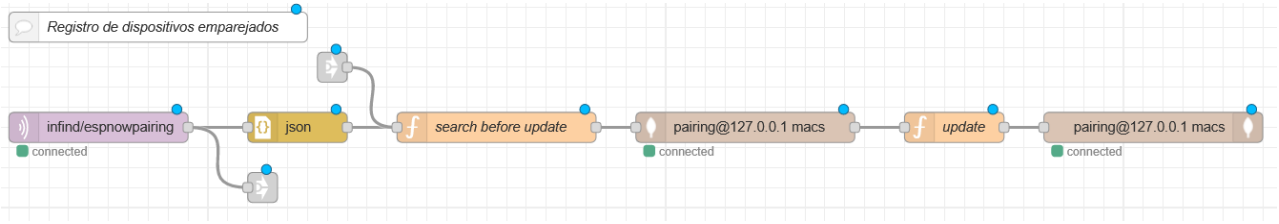

Figura 17. Flujo registro de dispositivos emparejados. [Elaboración propia]

2. Flujo para configurar a los dispositivos (Ver *Figura 18*).

Este flujo permite interactuar con nodos registrados en una base de datos, configurar parámetros de tiempo a través de una interfaz de usuario, y asignar nombres a nodos específicos utilizando direcciones MAC. También envía mensajes a través de MQTT para configurar el tiempo de sueño profundo y tiempo máximo de conexión.

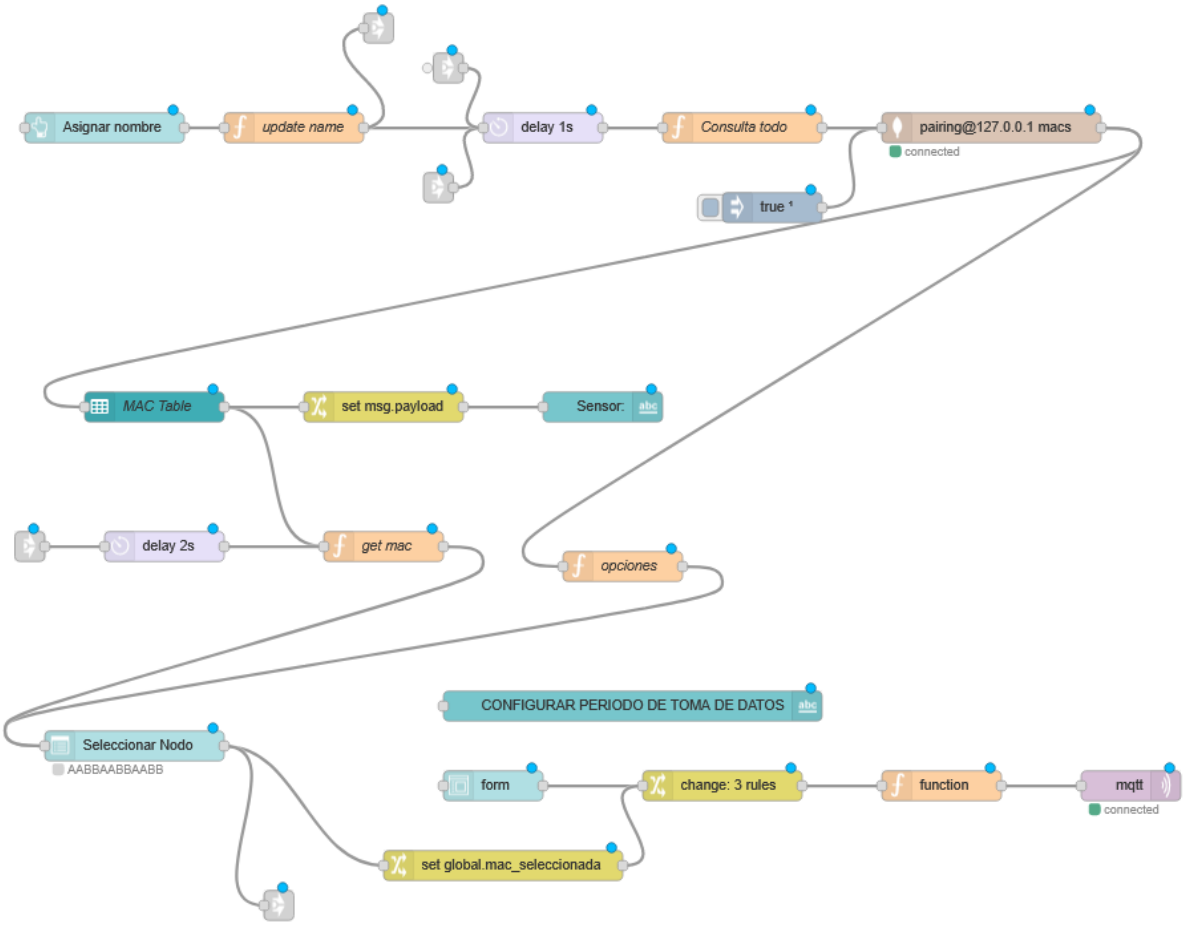

Figura 18. Flujo configuración de dispositivos. [Elaboración propia]

3. Flujo para borrar nodos emparejados (Ver *Figura 19*).

Permite a los usuarios borrar nodos específicos de la base de datos MongoDB utilizando el botón “Borrar nodo seleccionado” en la interfaz de usuario.

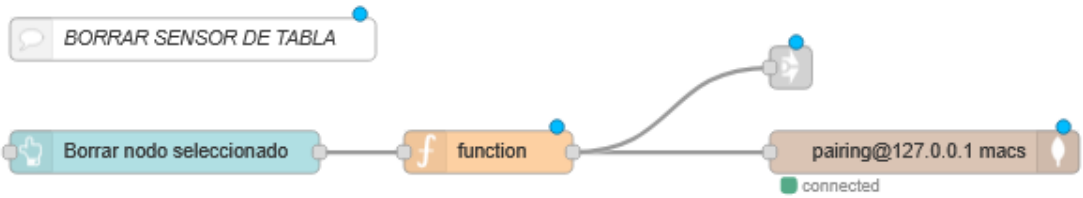

Figura 19. Flujo borrar sensor de base de datos. [Elaboración propia]

4. Flujo actualización FOTA (Ver *Figura 20*).

   El flujo muestra un botón en la interfaz de usuario con el texto "Actualizar firmware del sensor". Al hacer clic en este botón, se envía la orden FOTA al sensor a través del tema MQTT "infind/espnowdevice".

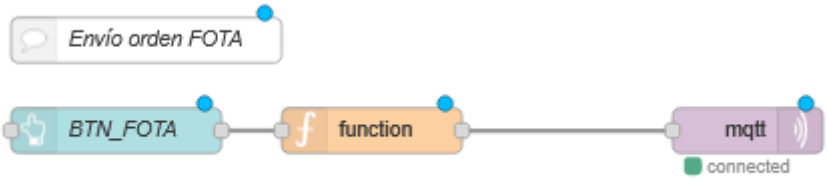

Figura 20. Flujo envío orden actualización FOTA. [Elaboración propia]

5. Flujo para actualizar nombre del nodo seleccionado (Ver *Figura 21*).

   El flujo proporciona un campo de entrada de texto en la interfaz de usuario donde los usuarios pueden ingresar el nombre para el nodo seleccionado, toma el valor ingresado en el campo de entrada de texto y lo almacena en una variable global llamada "n_nombre" para posteriormente actualizar o nombrar al nodo en la base de datos.

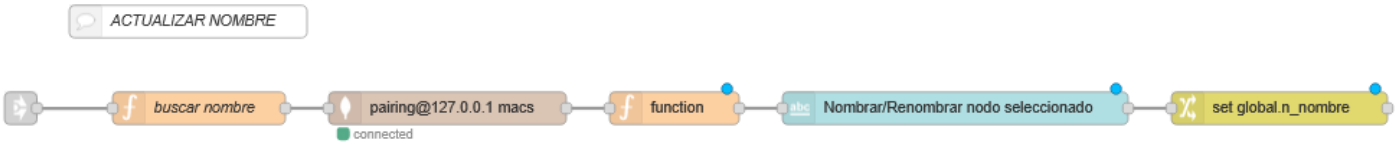

Figura 21. Flujo actualizar/nombrar nombre de nodo seleccionado. [Elaboración propia]

6. Flujo de recepción de datos de los dispositivos (Ver *Figura 22*).

   El flujo recibe mensajes MQTT con el tema "infind/espnow/+/datos". Estos mensajes contienen los datos del dispositivo, a continuación, se le agrega la fecha actual al mensaje para luego almacenarlos en la base de datos. Por último, se realiza una consulta con el fin de adquirir la última entrada de datos y graficarlos en tiempo real.

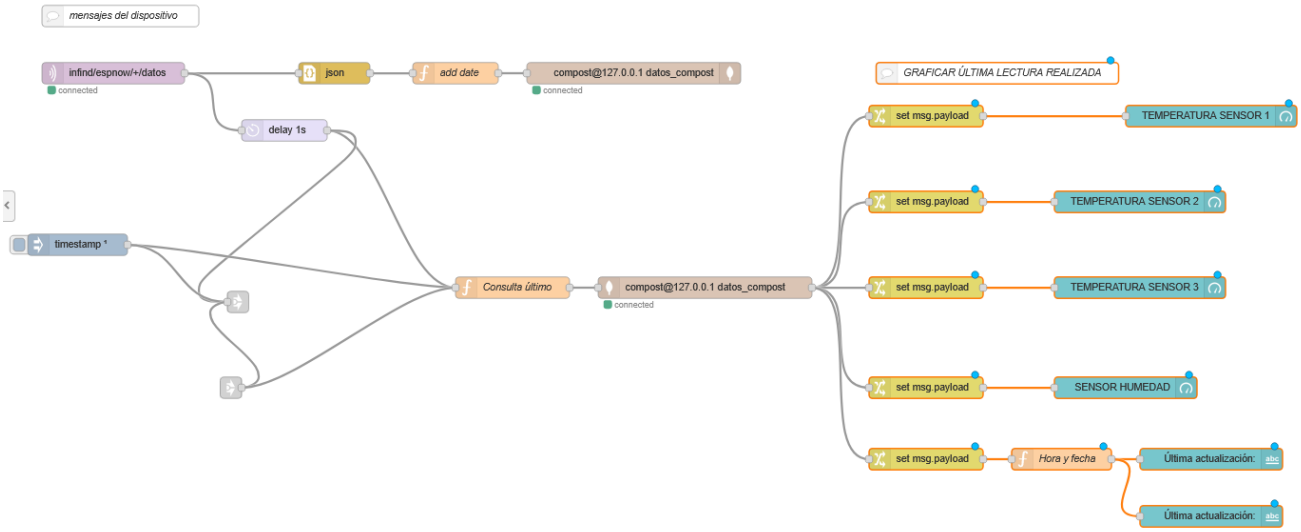

Figura 22. Flujo de recepción y almacenamiento de datos en MongoDB. [Elaboración propia]

7. Flujo para graficar los datos del periodo de tiempo deseado (Ver *Figura 23*).

Este flujo está diseñado para permitir a los usuarios seleccionar diferentes rangos de tiempo (por ejemplo, las últimas 24 horas, la semana pasada) y visualizar lecturas de temperatura y humedad de la estación de compost utilizando una base de datos MongoDB.

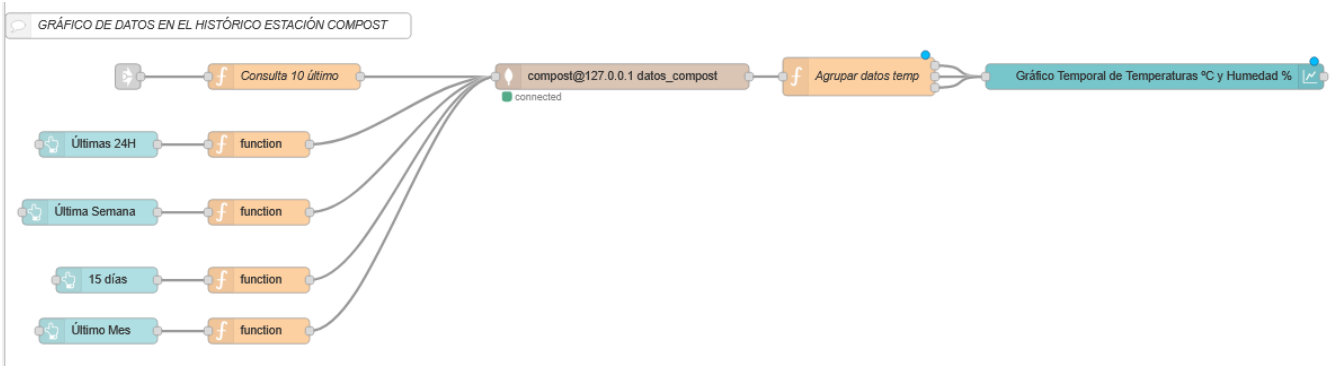

Figura 23. Flujo para graficar datos. [Elaboración propia]

Capítulo 5. Resultados y pruebas realizadas

A continuación, se muestran y analizan capturas de pantallas tomadas del puerto serie, tanto del dispositivo ESP8266, como del dispositivo ESP32. El objetivo es realizar el testeo y comprobación del correcto funcionamiento del código, para ello conectamos un nuevo dispositivo ESP8266, nombrado como “Dispositivo para capturas”, cuya dirección MAC es 2CF4323C5394.

Se procede realizando cuatro pruebas.

PRUEBA 1. “Autoemparejamiento de dispositivos”

Inicialmente, tras conectar el dispositivo ESP8266 inicia el autoemparejamiento buscando y enviando un mensaje secuencialmente en los canales WIFI, a la espera de que algún dispositivo cuya configuración sea de servidor le conteste y envía los datos necesarios para realizar el emparejamiento.

El dispositivo ESP8266, encuentra al servidor en el canal WIFI 9, recibe la dirección MAC del servidor y almacena los datos de conexión en la memoria RTC para futuras conexiones, esta secuencia se muestra en la *Figura 24*.

```
13:35:15.515 ->
13:35:15.515 -> Comienza AUTOpairing...
13:35:15.515 -> Magic code en RTC MEM: 0 - esperando: A5A5
13:35:15.515 -> Localización de dispositivos...encontrados 0 Sensores de temperatura.
13:35:16.266 ->  Solicitud de emparejamiento en el canal 6
13:35:16.407 -> Estado de envío del último paquete a ff:ff:ff:ff:ff:ff >> Éxito de entrega
13:35:16.485 ->  Solicitud de emparejamiento en el canal 7
13:35:16.532 -> Estado de envío del último paquete a ff:ff:ff:ff:ff:ff >> Éxito de entrega
13:35:16.609 ->  Solicitud de emparejamiento en el canal 8
13:35:16.655 -> Estado de envío del último paquete a ff:ff:ff:ff:ff:ff >> Éxito de entrega
13:35:16.734 ->  Solicitud de emparejamiento en el canal 9
13:35:16.734 -> Estado de envío del último paquete a ff:ff:ff:ff:ff:ff >> Éxito de entrega
13:35:16.778 -> tipo de mensaje recibido = 0 EMPAREJAMIENTO PAN:0
13:35:16.778 -> Tamaño del mensaje : 12 de 08:3a:f2:aa:ca:48
13:35:16.778 -> Emparejamiento hecho para 08:3a:f2:aa:ca:49 en el canal 9 en 1248ms
13:35:16.778 -> Escribimos emparejamiento en RTC MEM
13:35:16.778 -> Emparejado en milisegundo = 1313
```

Figura 24. AUTOpairing ESP8266. [Elaboración propia]

El dispositivo ESP32, se ha configurado como servidor, en primer lugar, establece conexión con el Broker MQTT, a continuación, recibe la solicitud de emparejamiento del dispositivo ESP8266, la cual acepta y envía el mensaje de confirmación y emparejado realizado. Por último, envía los datos del nuevo dispositivo a través de MQTT al Broker, este, se encarga de almacenar y registrar

al dispositivo ESP8266 en la base de datos. Se puede ver el ensayo en la *Figura 25*.

```
13:35:02.549 -> Connecting to infind:
13:35:02.861 -> ..
13:35:03.034 -> WiFi connected, IP address: 192.168.8.254
13:35:03.034 -> Server MAC Address:  08:3A:F2:AA:CA:48
13:35:03.078 -> Server SOFT AP MAC Address:  08:3A:F2:AA:CA:49
13:35:03.078 -> Station IP Address: 192.168.8.254
13:35:03.078 -> Wi-Fi Channel: 9
13:35:03.078 -> Identificador placa: ESP_-1426966008
13:35:03.078 -> Attempting MQTT connection... conectado a broker: iot.ac.uma.es
13:35:16.734 ->
13:35:16.734 -> received message type = 0 EMPAREJAMIENTO PAN:0
13:35:16.734 -> ESPNOW: 12 bytes of data received from : 2c:f4:32:3c:53:94
13:35:16.778 -> 0
13:35:16.778 -> 1
13:35:16.778 -> Pairing request from: 2c:f4:32:3c:53:94
13:35:16.778 -> send response
13:35:16.778 -> Pair success
13:35:16.778 -> New device paired
13:35:16.778 -> Enviando mensaje MQTT notificación pairing 2CF4323C5394
13:35:16.778 -> Last Packet Send Status: Delivery Success to 2c:f4:32:3c:53:94
13:35:17.682 ->
```

Figura 25. AUTOpairing ESP32. [Elaboración propia]

PRUEBA 2. “Envío y recepción de datos desde el ESP8266 al ESP32”

El dispositivo ESP8266, al ser el primer envío de datos y no haber recibido ninguna configuración del periodo de envío de mensajes, recupera de la memoria los parámetros de configuración y los parámetros del emparejamiento con el servidor, estableciendo un envío cada 10 segundos con un tiempo máximo de conexión de 3000 milisegundos. Por último, envía los datos adquiridos de los sensores al servidor y entra en modo sueño profundo. Se puede comprobar el funcionamiento de la prueba realizada en la *Figura 26*.

```
13:40:51.331 -> SETUP...
13:40:51.331 -> Magic code en FLASH: 3361 - esperando: C7C7
13:40:51.331 -> Sin configuración en FLASH
13:40:51.331 ->  > DeepSleep : 10
13:40:51.331 ->  > TimeOut   : 3000
13:40:51.331 ->
13:40:51.331 -> Comienza AUTOpairing...
13:40:51.331 -> Magic code en RTC MEM: A5A5 - esperando: A5A5
13:40:51.378 -> Emparejamiento recuperado de la memoria RTC del usuario 08:3a:f2:aa:ca:49 en el canal 9 en 12ms
13:40:51.457 ->  MI DIRECCIÓN MAC: 2C:F4:32:3C:53:94
13:40:51.457 -> Localización de dispositivos...encontrados 0 Sensores de temperatura.
13:40:53.128 -> sensor humedad RH% = 161.82
13:40:53.128 -> sending message type = 10000101 DATOS PAN:1 & SOLICITA MENSAJES
13:40:53.128 -> Longitud del mensaje: 64
13:40:53.128 -> mensaje: {"topic":"datos","T1":31.40,"T2":42.50,"T3":35.60, "hum":64.70 }
13:40:53.128 -> Estado de envío del último paquete a 08:3a:f2:aa:ca:49 >> Éxito de entrega
13:40:53.160 -> tipo de mensaje recibido = 11 SIN_DATOS PAN:0
13:40:53.160 -> Tamaño del mensaje : 1 de 08:3a:f2:aa:ca:48
13:40:53.160 -> No hay mensajes MQTT
13:40:53.160 -> Apaga y vamonos��
```

Figura 26. envió de datos ESP8266. [Elaboración propia]

El dispositivo ESP32, recibe la solicitud de un nuevo envío de datos del dispositivo ESP8266 a través de ESP-Now, lo aprueba y recibe los datos para enviarlos al Broker MQTT. A continuación, se muestra la secuencia en la *Figura 27*.

```
13:40:53.127 ->
13:40:53.127 -> received message type = 10000101 DATOS PAN:1 & SOLICITA MENSAJES
13:40:53.127 -> ESPNOW: 65 bytes of data received from : 2c:f4:32:3c:53:94
13:40:53.127 -> Device wants to get messages (CHECK==1)...
13:40:53.128 -> Buscando mensajes para dispositivo 2CF4323C5394
13:40:53.159 -> Pair success
13:40:53.159 -> New device paired
13:40:53.159 ->  > Sin mensajes pendientes
13:40:53.159 -> Enviando mensaje MQTT notificación pairing 2CF4323C5394
13:40:53.159 -> Last Packet Send Status: Delivery Success to 2c:f4:32:3c:53:94
13:40:53.159 -> Mensaje len: 64
13:40:53.159 ->  payload: {"topic":"datos","T1":31.40,"T2":42.50,"T3":35.60, "hum":64.70 }
13:40:53.159 -> Mensaje OK, topic = datos
13:40:53.159 -> Mensaje from: 2CF4323C5394, publicado en MQTT
13:40:53.159 ->        topic: infind/espnow/2CF4323C5394/datos
13:40:53.203 ->      payload: {"mac":"2CF4323C5394","payload":{"topic":"datos","T1":31.40,"T2":42.50,"T3":35.60, "hum":64.70 }}
13:40:53.203 ->
```

Figura 27. Recepción de datos ESP32. [Elaboración propia]

PRUEBA 3. “Configuración del dispositivo desde la interfaz”

Esta prueba consiste en enviar al dispositivo ESP8266 desde la interfaz de usuario un nuevo mensaje de configuración y envío de datos.

Se establece un tiempo de envío de datos de 300 segundos con un tiempo de conexión máximo entre dispositivos de 4000 milisegundos. Se puede observar el ejercicio en la *Figura 28.*

Figura 28. Configuración desde la interfaz. [Elaboración propia]

Como se puede comprobar en la *Figura 29*, el mensaje es recibido por el dispositivo ESP32, este, proceda el mensaje MQTT del Broker, busca al destinatario y lo envía a través de ESP-Now al dispositivo ESP8266.

```
13:47:04.443 -> Mensaje recibido [infind/espnowdevice] {"mac":"2CF4323C5394","topic":"config","payload":{"sleep":300,"timeout":4000}}
13:47:04.443 -> Mensaje OK, mac = 2CF4323C5394
13:47:04.443 -> compacto = config|{"sleep":300,"timeout":4000}
13:47:26.932 ->
13:47:26.932 -> received message type = 10000101 DATOS PAN:1 & SOLICITA MENSAJES
13:47:26.932 -> ESPNOW: 65 bytes of data received from : 2c:f4:32:3c:53:94
13:47:26.932 -> Device wants to get messages (CHECK==1)...
13:47:26.932 -> Buscando mensajes para dispositivo 2CF4323C5394
13:47:26.932 -> Pair success
13:47:26.932 -> New device paired
13:47:26.932 ->  > Un mensaje encontrado
13:47:26.932 -> Enviando mensaje MQTT notificación pairing 2CF4323C5394
13:47:26.964 -> Last Packet Send Status: Delivery Success to 2c:f4:32:3c:53:94
13:47:26.964 -> Mensaje len: 64
13:47:26.964 ->  payload: {"topic":"datos","T1":31.40,"T2":42.50,"T3":35.60, "hum":64.70 }
13:47:26.964 -> Mensaje OK, topic = datos
13:47:26.964 -> Mensaje from: 2CF4323C5394, publicado en MQTT
13:47:26.964 ->        topic: infind/espnow/2CF4323C5394/datos
13:47:26.964 ->      payload: {"mac":"2CF4323C5394","payload":{"topic":"datos","T1":31.40,"T2":42.50,"T3":35.60, "hum":64.70 }}
13:47:27.009 ->
13:47:27.009 -> Buscando mensajes para dispositivo 2CF4323C5394
13:47:27.009 ->  > Sin mensajes pendientes
13:47:27.009 -> Last Packet Send Status: Delivery Success to 2c:f4:32:3c:53:94
```

Figura 29. Recepción nueva configuración ESP32. [Elaboración propia]

Para finalizar, como observamos en la *Figura 30*, el mensaje es recibido por el dispositivo ESP8266 tras despertar del estado de sueño profundo, realiza un envío de los datos adquiridos por los sensores y a continuación guarda la nueva configuración para futura conexiones antes de entrar en el estado de sueño profundo.

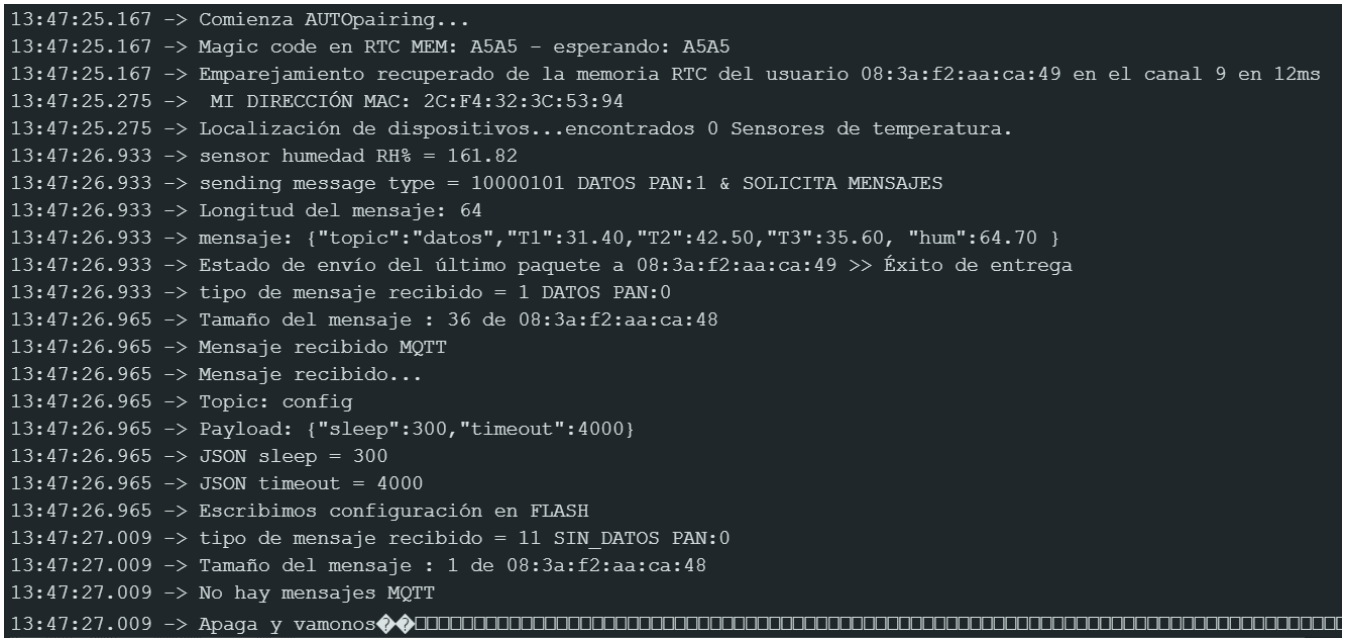

Figura 30.Recepción nueva configuración ESP8266. [Elaboración propia]

PRUEBA 4.” Envío actualización FOTA”

En esta última prueba realizada, desde la interfaz de usuario enviamos una petición de actualización del Firmware del dispositivo ESP8266.

El mensaje es recibido y procesado por el ESP32 y tras enviarlo al dispositivo ESP8266, intenta realizar la actualización, es este caso, se puede ver, como no se actualiza el Firmware, ya que tiene la última versión, se ilustra en la *Figura 31*.

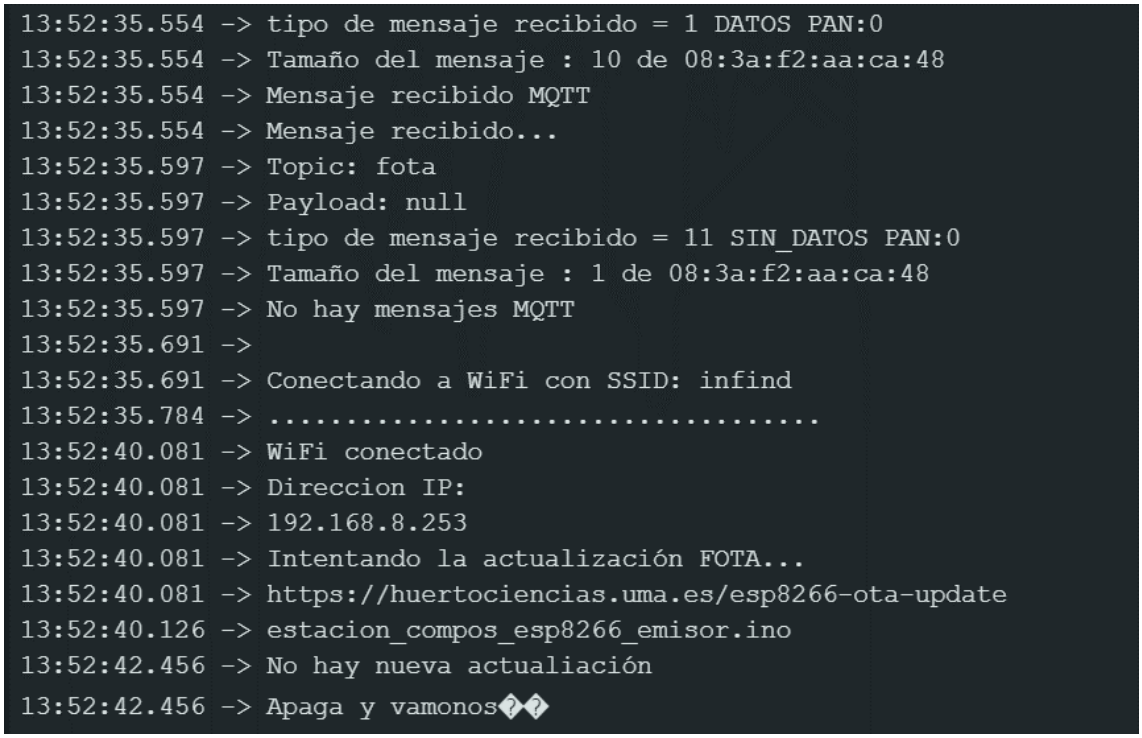

Figura 31. Actualización FOTA del ESP8266. [Elaboración propia]

Capítulo 6. Conclusiones y líneas futuras

Después de finalizar la ejecución y la puesta en marcha de este proyecto, se derivarán conclusiones de la experiencia obtenida durante su desarrollo. Asimismo, se destacarán oportunidades de mejoras que puedan impulsar futuros trabajos basados en los fundamentos establecidos en este proyecto, con el propósito de perfeccionarlo o complementarlo.

Partiendo principalmente del objetivo del proyecto, puedo asegurar que se ha cumplido de forma satisfactoria. La monitorización se lleva a cabo de forma eficiente, en tiempo real y desde cualquier dispositivo que tenga acceso a internet mediante un navegador, ofreciendo una gran consistencia multiplataforma entre diferentes dispositivos.

En la realización del proyecto, en cuanto a formación y manejo o utilización de las diferentes herramientas y programas mencionados, he de decir que, gracias a los conocimientos adquiridos en las diferentes asignaturas del Grado de “Ingeniería Electrónica, Robótica y Mecatrónica”, no me he encontrado grandes dificultades, la puesta en marcha, una vez planteado y organizado el trabajo, ha sido satisfactoria.

El principal hándicap ha sido el estudio a fondo de las fases, jerarquías y funciones que constituyen los protocolos de comunicación utilizados, con el fin de conseguir que la implementación del código, en los distintos dispositivos, cumpla con los siguientes puntos:

- Eficiencia temporal. El flujo de un código optimizado debe minimizar la cantidad de tiempo necesaria para completar las operaciones.
- Eficiente de la memoria y recursos del sistema. Un código óptimo utiliza la menor cantidad de memoria posible y evita perdidas de memoria.
- Legibilidad. El código tiene que ser fácil de leer y comprender. Utilizar nombres de variables y funciones descriptivos, seguir patrones en la implementación del código y evita la redundancia innecesaria.
- Complejidad. Utilizar estructuras de control y algoritmos simples en lugar de soluciones excesivamente complicadas. Además de la reutilización de

funciones y componentes para no reescribir el mismo código en diferentes partes del programa.

- Manejo de errores. El flujo de un código optimizado incluye manejo adecuado de errores y excepciones para garantizar que el programa no falle inesperadamente.

Para ello se han tenido que crear librerías e incluido funciones, métodos, clases y estructuras, que han permitido cumplir con estos puntos.

A base de ensayo prueba y error, se completa y aprueba el buen funcionamiento entre los dispositivos.

Hay que destacar que, tras la instalación de los dispositivos en el huerto y puesta en marcha, después de la realización de pruebas para verificar el correcto funcionamiento de todos los elementos, lamentablemente, se constató un fallo en el aspecto de la alimentación. Este contratiempo posiblemente surge debido a la utilización de componentes reciclados (batería, control de carga y paneles fotovoltaicos) provenientes de un proyecto previo, los cuales podrían presentar algún problema funcional o de confiabilidad. Aunque se evaluaron con éxito luego de un ensamblaje inicial, su rendimiento decayó poco después de su despliegue en el huerto.

Para abordar esta situación, se resolvió este inconveniente mediante la creación de un dispositivo virtual capaz de transmitir datos en intervalos de 10 minutos. Esta solución permitió culminar el desarrollo y la prueba de la aplicación de la interfaz, la cual, recopila, almacena y muestra la información obtenida. El flujo del código se puede ver en el *Anexo 2.3 Flujo del dispositivo virtual*.

Con el inicio del nuevo curso académico, se tiene previsto adquirir los componentes necesarios para garantizar la correcta alimentación y el funcionamiento definitivo del dispositivo. Este paso próximo será esencial para llevar a cabo la implementación final exitosa y cumplir con los objetivos propuestos en el proyecto.

Todo el contenido del presente trabajo se encuentra disponible en el repositorio [12].

Para finalizar, se mencionan a continuación posibles ideas de mejoras y creación de líneas futuras basadas en este proyecto.

Como bien indica el título del proyecto “DISPOSITIVO INTELIGENTE PARA MONITORIZACIÓN INALÁMBRICA DE PILA DE COMPOSTAJE”, se ha llevado a cabo un proyecto sólo y exclusivamente con el objetivo de monitorizar. Como líneas futuras, el proyecto puede crecer incorporando la automatización del proceso de compostaje, tras analizar la etapa en la que se encuentra la materia, se puede diseñar un sistema de volteo de la pila e incorporar un sistema de alarmas en función de temperaturas y humedad, con el fin de, actuar sobre elementos que permitan las condiciones óptimas para el proceso, tales como, control de riego, de temperatura o aireación.

Otra idea de mejora para líneas futuras, la cual, no se ha llevado a cabo por falta de tiempo a la hora de finalizar el proyecto para la entrega, es implementar un flujo en Node-RED que permita descargar en un archivo Excel con los datos de los sensores almacenados en la base de datos del servidor. Esto facilitará la integración de los datos en otras bases de datos distintas a MongoDB si se requiere, además, permitirá al personal de gestión y control del huerto realizar el procesamiento y análisis de los datos con las diferentes herramientas que nos brinda Microsoft en su programa Excel.

Referencias

[1] Espressif Systems. “ESP-NOW Wireless Communication Protocol | Espressif Systems”. Wi-Fi & Bluetooth MCUs and AIoT Solutions I Espressif Systems. Accedido el 30 de agosto de 2023. [En línea]. Disponible: https://www.espressif.com/en/solutions/low-power-solutions/esp-now

[2] “MQTT - The Standard for IoT Messaging”. MQTT - The Standard for IoT Messaging. Accedido el 1 de junio de 2023. [En línea]. Disponible: https://mqtt.org/

[3] “Arduino - Home”. Arduino - Home. Accedido el 1 de junio de 2023. [En línea]. Disponible: https://www.arduino.cc/

[4] Logo Arduino. Accedido el 3 de julio de 2023. [Imagen]. Con licencia bajo GNU General Public License. Disponible: https://icon-icons.com/icon/arduino-official-logo/167833

[5] “Node-RED”. Node-RED. Accedido el 7 de junio de 2023. [En línea]. Disponible: https://nodered.org/

[6] OpenJS Foundation. Logo Node-RED. Accedido el 23 de junio de 2023. [Imagen]. Disponible: https://nodered.org/about/resources/media/node-red-icon-2.png

[7] Random Nerd Tutorials. “150+ ESP8266 NodeMCU Projects, Tutorials and Guides with Arduino IDE | Random Nerd Tutorials”. Random Nerd Tutorials. Accedido el 10 de julio de 2023. [En línea]. Disponible: https://randomnerdtutorials.com/projects-esp8266/

[8] Espressif Systems. “ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems”. Wi-Fi & Bluetooth MCUs and AIoT Solutions I Espressif Systems. Accedido el 6 de junio de 2023. [En línea]. Disponible: https://www.espressif.com/en/products/socs/esp32

[9] Cetronic. Sonda sensor de temperatura DS18B20. Accedido el 6 de julio de 2023. [Imagen]. Disponible: https://www.cetronic.es/sqlcommerce/ficheros/dk_93/productos/999334005-4.jpg

[10] Sonda sensor de humedad Capacitive Soil Moisture v1.0. Accedido el 30 de junio de 2023. [Imagen]. Disponible: https://www.geekfactory.mx/wp-content/uploads/2018/10/Capacitive_soil_moisture_sensor.jpg

[11] R. Santos. “GitHub - Servayejc/esp_now_web_server”. GitHub. Accedido el 3 de mayo de 2023. [En línea]. Disponible: https://github.com/Servayejc/esp_now_web_server/

[12] J. Gil Palacios. “GitHub - JaviGP91/TFG-JAVIER-GIL-PALACIOS-ESTACION-DE-COMPOST: TFG JAVIER GIL "Dispositivo inteligente para monitorización inalámbrica de pila de compostaje"”. GitHub. Accedido el 1 de septiembre de 2023. [En línea]. Disponible: https://github.com/JaviGP91/TFG-JAVIER-GIL-PALACIOS-ESTACION-DE-COMPOST.git

Anexos

Anexo 1 Imágenes

Anexo 1.1 Imagen de la estación móvil

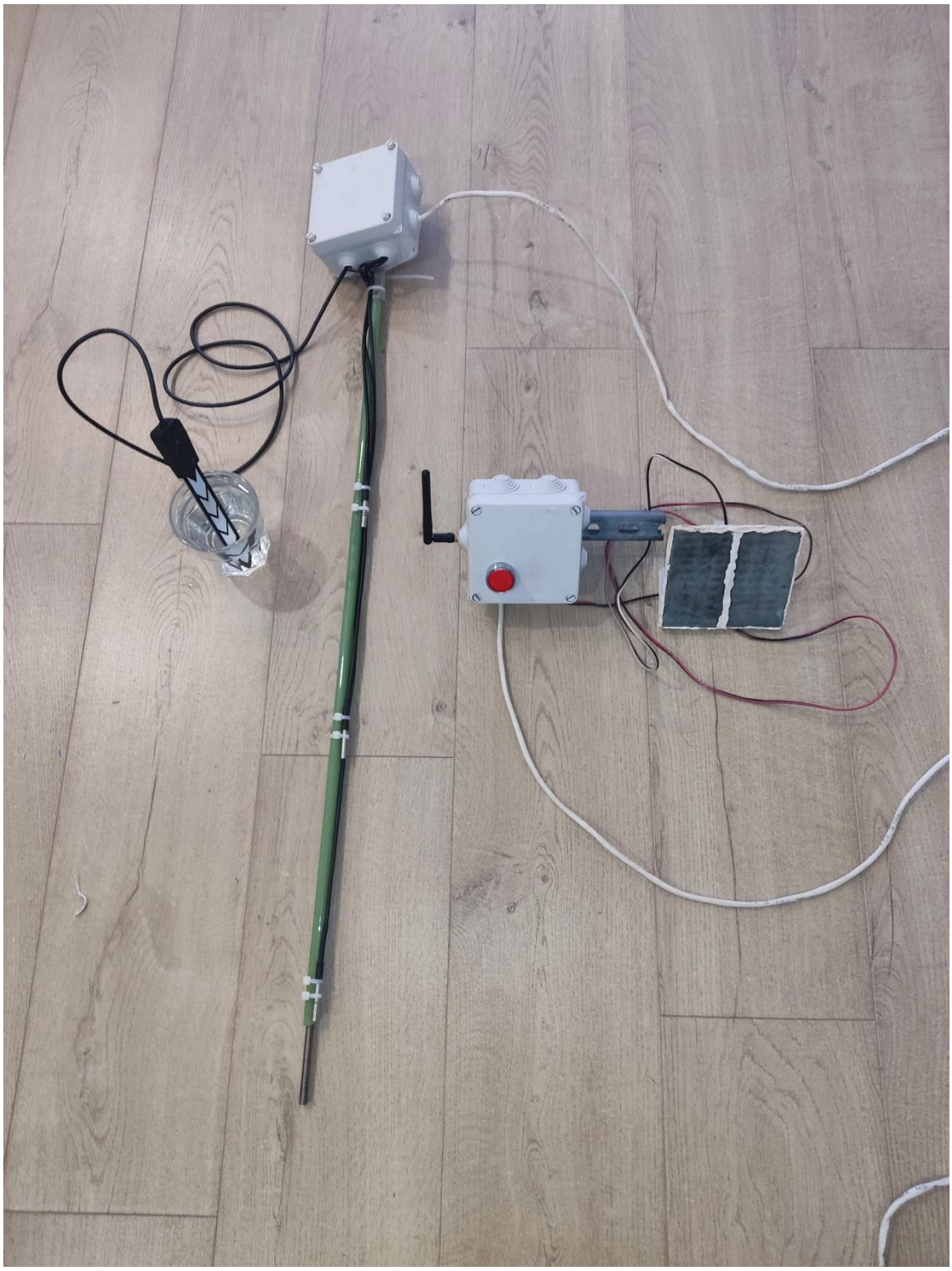

Figura 32. Estación móvil. [Elaboración propia]

Anexo 1.2 Pica soterrada en pila de compost

Figura 33. Pica soterrada en pila de compost. [Elaboración propia]

Anexo 1.3 Pica de toma de datos

Figura 34. Pica de toma de datos. [Elaboración propia]

Anexo 1.4 Módulo de control de la estación móvil.

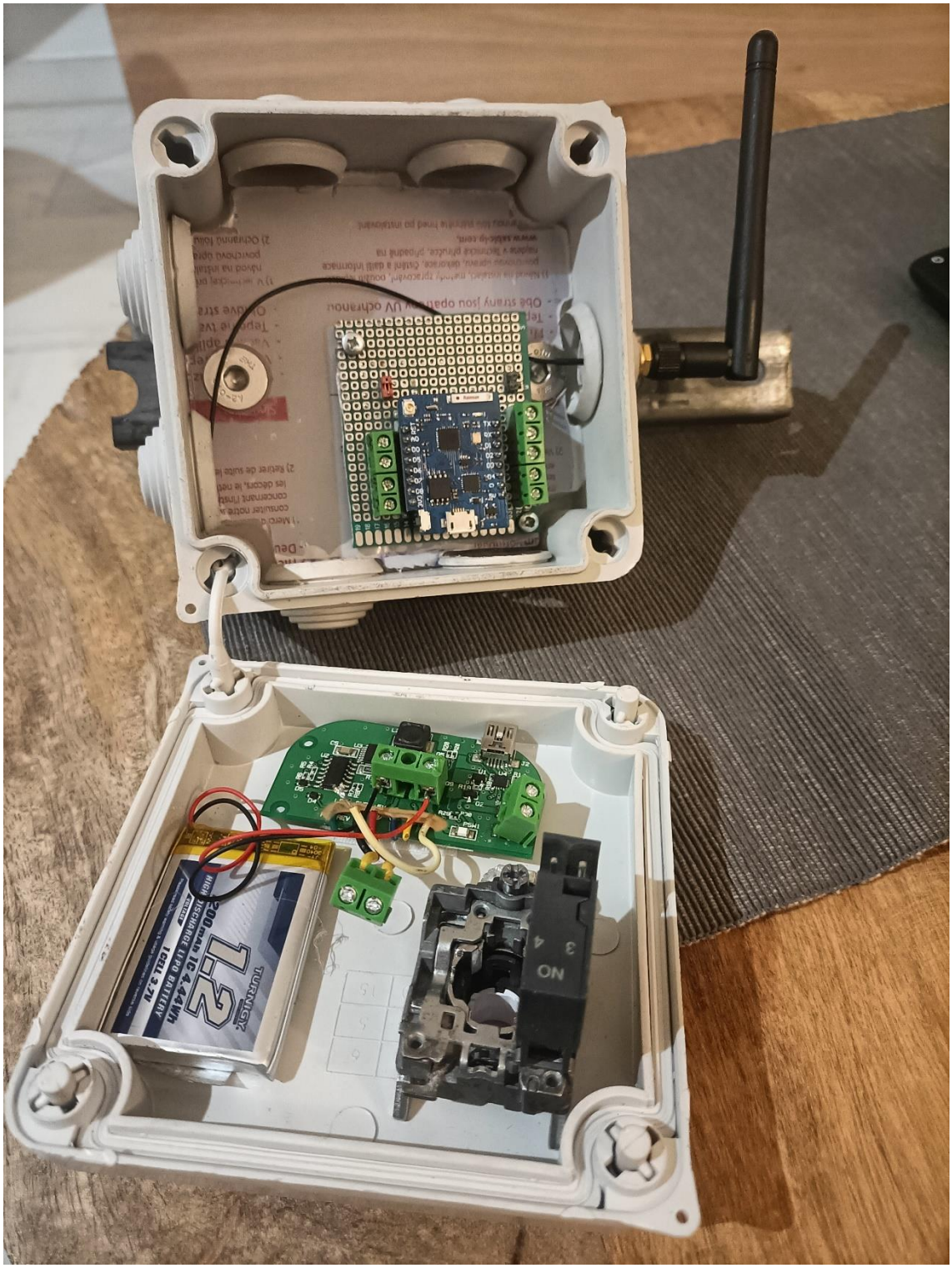

Figura 35. Módulo de control estación móvil. [Elaboración propia]

Anexo 1.5 Placa electrónica ESP8266

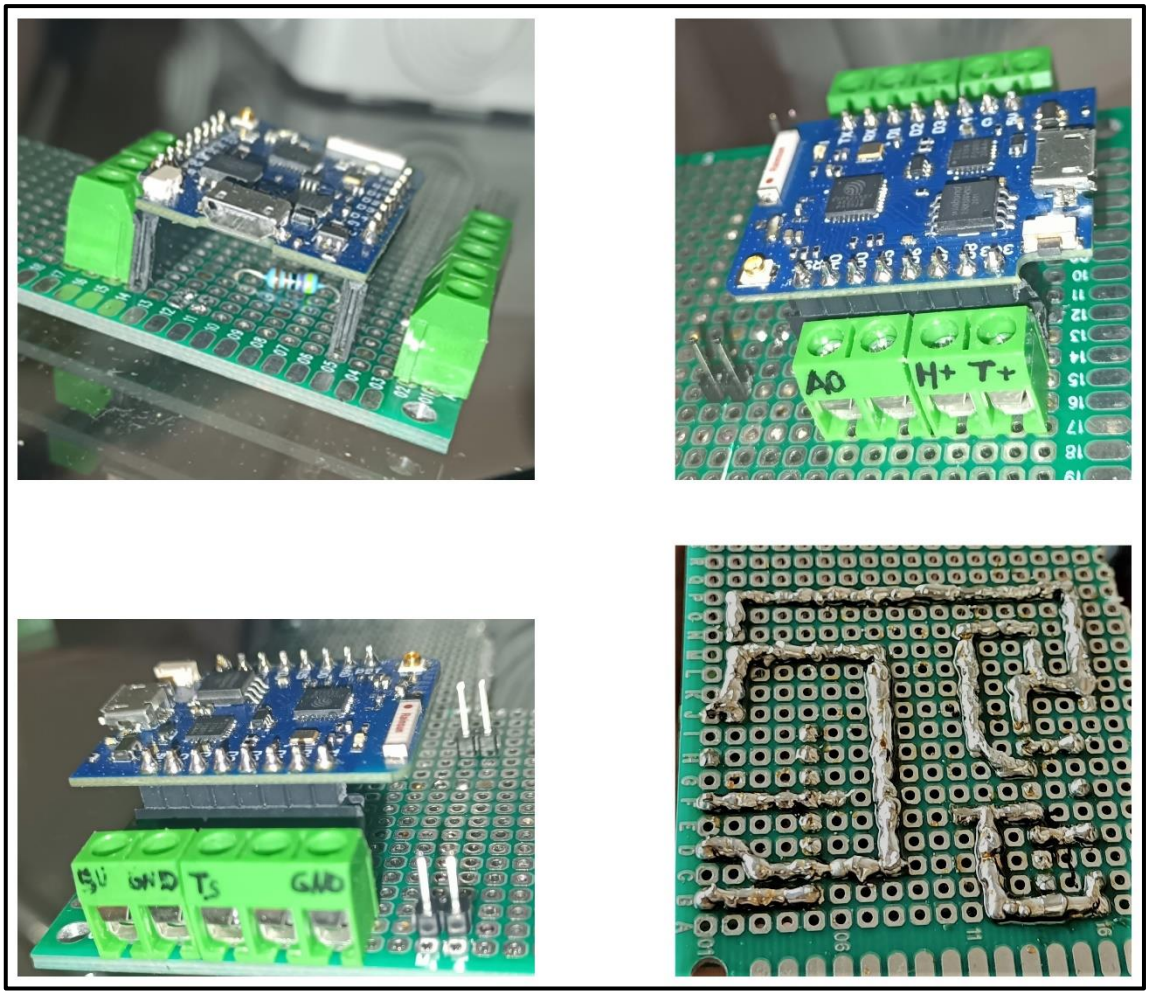

Figura 36. Placa electrónica ESP8266. [Elaboración propia]

Anexo 2 Códigos

Este apartado se desglosa en dos secciones principales. En la primera sección se muestra los códigos correspondientes al dispositivo ESP8266. En la segunda sección se presenta los códigos asociados al dispositivo ESP32.

Anexo 2.1 Código del dispositivo ESP8266

Anexo 2.1.1 Estacion_compos_esp8266_emisor.ino

```
/*LIBRERIAS*/
#include "AUTOpairing.h"
#include <ArduinoJson.h>
#include <ESP8266httpUpdate.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include "funciones.h" // fichero del proyecto

/*PINES*/
const int Power_s_temp = 12; // Pin GPIO para alimentar los sensores de
temperatura
const int Power_s_hum = 13; // Pin GPIO para alimentar sensor de humedad


/*VARIABLES*/
float S1_T=0, S2_T=0, S3_T=0;
float temp[3];
float humedad;

// Objeto de la clase AUTOpairing_t para la comunicación por ESP-NOW.
AUTOpairing_t clienteAP;

// Estructura utilizada para almacenar la configuración del programa,
(tiempo de sueño profundo y el tiempo de espera de conexión).
struct configuracion
{
  uint32_t sleep;
  uint32_t timeout;
} mi_configuracion;

//---------------------- ACTUALIZACION FOTA ----------------------

//const char* ssid = "huerticawifi";
//const char* password = "4cc3sshu3rt1c4";
```

```
const char* ssid = "sagemcom67E0_EXT";
const char* password = "UWZKFTHRWZZTYY";
#define OTA_URL    "https://huertociencias.uma.es/esp8266-ota-update"
#define
HTTP_OTA_VERSION   String(__FILE__).substring(String(__FILE__).lastIndexO
f('\\')+1) + ".d1_mini_pro"

// IDE > 2.0 Mac
//#define
HTTP_OTA_VERSION   String(__FILE__).substring(String(__FILE__).lastIndexO
f('/')+1)

WiFiClientSecure WClient;

void procesa_mensajes (String topic, String payload)
{
  Serial.println("Mensaje recibido...");
  Serial.print("Topic: ");
  Serial.println(topic);
  Serial.print("Payload: ");
  Serial.println(payload);
  if(topic=="fota")
  {
    delay(100);
    Serial.println();
    Serial.print("Conectando a WiFi con SSID: ");
    Serial.println(ssid);
    //wifi_promiscuous_enable(0);
    WiFi.disconnect();
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    unsigned long ahora= millis();
    while (WiFi.status() != WL_CONNECTED && millis()-ahora<15000) {
      delay(100);
      Serial.print(".");
    }
    if(WiFi.status() != WL_CONNECTED)
    {
      Serial.println("WiFi NO conectado");
      clienteAP.gotoSleep();
    }
    Serial.println("");
    Serial.println("WiFi conectado");
    Serial.println("Direccion IP: ");
    Serial.println(WiFi.localIP());
    WClient.setTimeout(12); // timeout argument is defined in seconds for
setTimeout
```

```
    WClient.setInsecure();  // no chequea certificado...
    Serial.println( "Intentando la actualización FOTA..." );
    Serial.println(OTA_URL);
    Serial.println(HTTP_OTA_VERSION);

    switch(ESPhttpUpdate.update(WClient,OTA_URL, HTTP_OTA_VERSION)) {
    case HTTP_UPDATE_FAILED:
      Serial.printf("Error en actualización HTTP: Error #(%d): %s\n",
ESPhttpUpdate.getLastError(),
ESPhttpUpdate.getLastErrorString().c_str());
      break;
    case HTTP_UPDATE_NO_UPDATES:
      Serial.println(F("No hay nueva actualiación"));
      break;
    case HTTP_UPDATE_OK:
      Serial.println(F("Actualiación OK"));
      break;
    }
    clienteAP.gotoSleep();
  } // end FOTA

  if(topic=="config") // el mesaje debe tener topic = config y payload =
{"sleep": # , "timeout": # }
  {
    StaticJsonDocument<512> doc; // el tamaño tiene que ser adecuado para
el mensaje
    // Deserialize the JSON document
    DeserializationError error = deserializeJson(doc,
payload.c_str(),payload.length());

    // Compruebo si no hubo error
    if (error) {
      Serial.print("Error fallo en deserializeJson(): ");
      Serial.println(error.c_str());
    }
    else
    if(doc.containsKey("sleep") && doc.containsKey("timeout") )  //
comprobar si existe el campo/clave que estamos buscando
    {
     int valor = doc["sleep"];
     Serial.print("JSON sleep = ");
     Serial.println(valor);
     mi_configuracion.sleep=valor;
     valor = doc["timeout"];
     Serial.print("JSON timeout = ");
     Serial.println(valor);
     mi_configuracion.timeout=valor;
```

```
     clienteAP.set_config((uint8_t*)&mi_configuracion);
    }
    else
    {
      Serial.println("ERROR: claves \"sleep\" & \"timeout\" no aparecen
en JSON");
    }
  } // end config
}

//---------------------------------------------------------------------
//-----------------------   SETUP   ---------------------------------
void setup() {
  // Iniciar alimentación de sensores
  pinMode(Power_s_temp, OUTPUT);
  pinMode(Power_s_hum, OUTPUT);
  digitalWrite(Power_s_temp, HIGH);
  digitalWrite(Power_s_hum, HIGH);

  // Iniciar puerto serie
  Serial.begin(115200);
  Serial.println();
  Serial.println("SETUP...");
  clienteAP.init_config_size(sizeof(mi_configuracion));

  if (clienteAP.get_config((uint8_t*)&mi_configuracion)==false)
  { // si no hay configuración guardada la pongo por defecto
    mi_configuracion.sleep=10;
    mi_configuracion.timeout=3000;
  }
  Serial.printf(" > DeepSleep : %d\n",mi_configuracion.sleep );
  Serial.printf(" > TimeOut   : %d\n",mi_configuracion.timeout );
  clienteAP.set_timeOut(mi_configuracion.timeout,true); // tiempo máximo
  clienteAP.set_deepSleep(mi_configuracion.sleep);  //tiempo dormido en
segundos
  clienteAP.set_channel(6);  // canal donde empieza el scaneo
  clienteAP.set_debug(true);   // depuración, inicializar Serial antes
  clienteAP.set_callback(procesa_mensajes);  //por defecto a NULL -> no
se llama a ninguna función
  clienteAP.begin();
  adquisicion_direcciones_temp();
}

//---------------------------------------------------------------------
//-----------------------   LOOP    ---------------------------------
void loop() {
  //digitalWrite(Power_s_temp, HIGH);
```

```
  //digitalWrite(Power_s_hum, HIGH);
  clienteAP.mantener_conexion();

  if (clienteAP.envio_disponible()) {
      char mensaje[256];

      pedir_temperaturas(temp);
      digitalWrite(Power_s_temp, LOW); // apagamos los sensores de
temperatura
      humedad=pedir_humedad();
      digitalWrite(Power_s_hum, LOW);  // apagamos sensor de humedad
      sprintf(mensaje,
"{\"topic\":\"datos\",\"T1\":%4.2f,\"T2\":%4.2f,\"T3\":%4.2f,
\"hum\":%4.2f }", temp[0], temp[1], temp[2], humedad);
      clienteAP.espnow_send_check(mensaje); // hará deepsleep por defecto
  }

  //digitalWrite(Power_s_temp, LOW);
  //digitalWrite(Power_s_hum, LOW);
}
```

Anexo 2.1.2 AUTOpairing.h

La siguiente implementación de código se basa en el proyecto inicialmente propuesto por "Rui Santos", cuya referencia completa se encuentra en su repositorio [11].

```
/*
 * Librería para comunicación ESPNOW<->MQTT a través de pasarela con
auto-emparejamiento
 * Recuerda la configuración de emparejamiento usando FLASH o memoria RTC
 * Recuerda configuración del usuario usando FLASH
 *
 * Distribuido bajo licencia: CC BY-NC-SA 4.0
http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es
 * Autor: Andrés Rodríguez (andres@uma.es)
 *
 * Basado en el trabajo de:
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com/?s=esp-now
  Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files.
  The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
```

```
  Based on JC Servaye example:
https://https://github.com/Servayejc/esp8266_espnow
*/

#ifndef AUTOpairing_H
#define AUTOpairing_H
#include <ESP8266WiFi.h>
#include <espnow.h>
#include <EEPROM.h>
#include <string>
#include <queue>
#include "Arduino.h"
#include "AUTOpairing_common.h"

class mensajeMQTT_t
{
  public:
  String topic;
  String payload;
  mensajeMQTT_t ( String t, String p)
   {
    topic=t;
    payload=p;
   }
};

// cola de mensajes mqtt recibidos
std::queue<mensajeMQTT_t> cola_mensajes;

uint8_t broadcastAddressX[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

#define MAGIC_CODE1 0xA5A5
#define MAGIC_CODE2 0xC7C7
#define INVALID_CODE 0

#ifndef MAX_CONFIG_SIZE
#define MAX_CONFIG_SIZE 64
#endif

class AUTOpairing_t
{
  static AUTOpairing_t *este_objeto;

  enum PairingStatus {PAIR_REQUEST, PAIR_REQUESTED, PAIR_PAIRED, };

  struct struct_rtc{
    uint16_t code1;
```

```
    uint16_t code2;
    struct_pairing data;
    uint8_t config[MAX_CONFIG_SIZE]; // max config size
  } ;

   int config_size;
   int mensajes_sent;
   PairingStatus pairingStatus;
   struct_rtc rtcData;
   struct_pairing pairingData;
   bool mensaje_enviado; // para saber cuando hay que dejar de enviar
porque ya se hizo y estamos esperando confirmación
   bool terminar; // para saber cuando hay que dejar de enviar porque ya
se hizo y estamos esperando confirmación
   unsigned long previousMillis_scanChannel;    // will store last time
channel was scanned
   unsigned long start_time;  // para controlar el tiempo de escaneo
   bool esperando;  // esperando mensajes
   bool EEPROM_init;

   void (*user_callback)(String, String);

//------------------------------------------------------------
 void printMAC(const uint8_t * mac_addr){
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3],
mac_addr[4], mac_addr[5]);
  if(debug) Serial.print(macStr);
}

// para modificar por el usuario usando funciones

   unsigned long timeOut;
   bool debug;
   bool timeOutEnabled;
   bool usar_FLASH;
   uint8_t channel;  // canal para empezar a escanear
   int segundos_en_deepSleep;   // tiempo dormido en segundos
   int panAddress;

public:

 AUTOpairing_t()
  {
    este_objeto = this;
    pairingStatus = PAIR_REQUEST;
```

```
    mensaje_enviado=false; // para saber cuando hay que dejar de enviar
porque ya se hizo y estamos esperando confirmación
    terminar=false; // para saber cuando hay que dejar de enviar porque
ya se hizo y estamos esperando confirmación
    esperando=false;
    timeOutEnabled=true;
    usar_FLASH=false;
    EEPROM_init=false;
    segundos_en_deepSleep = 10;   // tiempo dormido en segundos
    panAddress = 1;
    config_size = MAX_CONFIG_SIZE;
    start_time=0;  // para controlar el tiempo de escaneo
    previousMillis_scanChannel=0;
    timeOut=3000;
    debug=true;
    channel = 6;  // canal para empezar a escanear
    mensajes_sent=0;
  }
//------------------------------------------------------------
 void set_pan(uint8_t _pan=1)
{
  panAddress=_pan;
}
//------------------------------------------------------------
 int get_pan()
{
  return panAddress;
}
//------------------------------------------------------------
 void set_timeOut(unsigned long _timeOut=3000, bool _enable=true)
{
  timeOut=_timeOut;
  timeOutEnabled=_enable;
}

//------------------------------------------------------------
 void set_channel(uint8_t _channel=6)
{
  channel=_channel;
}

//------------------------------------------------------------
 void set_FLASH(bool _usar_FLASH=false)
{
  usar_FLASH=_usar_FLASH;
}
```

```
//-------------------------------------------------------------
 void set_debug(bool _debug=true)
{
  debug=_debug;
}

//-------------------------------------------------------------
 void set_deepSleep(int _segundos_en_deepSleep=10){
  segundos_en_deepSleep=_segundos_en_deepSleep;
}

//-------------------------------------------------------------
 bool init_config_size(uint8_t size)
{
  config_size = size;
  if(!EEPROM_init) { EEPROM.begin(sizeof(rtcData)); EEPROM_init=true; }
  if (size>MAX_CONFIG_SIZE)
  {
    Serial.printf("Espacio reservado en FLASH demasiado pequeño: %d\n Por
favor incremente el valor MAX_CONFIG_SIZE",MAX_CONFIG_SIZE );
    return false;
  }
  else
  return true;
}

//-------------------------------------------------------------
 bool get_config(uint8_t* config)
{
  //init check FLASH

  EEPROM.get(0, rtcData);
  if(debug) Serial.print("Magic code en FLASH: ");
  if(debug) Serial.print(rtcData.code2,HEX);
  if(debug) Serial.print(" - esperando: ");
  if(debug) Serial.println((unsigned long)MAGIC_CODE2,HEX);

  if(rtcData.code2==MAGIC_CODE2)
  {
    memcpy(config, &(rtcData.config), config_size);
    if(debug) Serial.println("Configuración leída de FLASH");
    return true;
  }
  else
  {
    if(debug) Serial.println("Sin configuración en FLASH");
    return false;
```

```
  }
}

//-------------------------------------------------------------
 void set_config(uint8_t* config)
{
  rtcData.code2=MAGIC_CODE2;
  memcpy(&(rtcData.config), config, config_size);
  if(debug) Serial.println("Escribimos configuración en FLASH");
  EEPROM.put(0, rtcData);
  EEPROM.commit();
}

//-------------------------------------------------------------
 void begin()
  {
    if(debug) Serial.println("\nComienza AUTOpairing...");
    previousMillis_scanChannel=0;
    start_time=millis();

    // Inicialización de la memoria EEPROM o RTC MEM según la
configuración
    // usamos rtc memory
    if(usar_FLASH)
    {
      if(!EEPROM_init) { EEPROM.begin(sizeof(rtcData)); EEPROM_init=true;
}
      EEPROM.get(0, rtcData);
      if(debug) Serial.print("Magic code en FLASH: ");
    }
    else
    {
      ESP.rtcUserMemoryRead(0,(uint32_t *)&rtcData, sizeof(rtcData));
      if(debug) Serial.print("Magic code en RTC MEM: ");
    }
    if(debug) Serial.print(rtcData.code1,HEX);
    if(debug) Serial.print(" - esperando: ");
    if(debug) Serial.println((unsigned long)MAGIC_CODE1,HEX);

    if(rtcData.code1==MAGIC_CODE1) //Si coinciden, significa que hay
datos de emparejamiento almacenados en la memoria y se procede a
recuperar esos datos.
    {
      memcpy(&pairingData, &(rtcData.data), sizeof(pairingData));
      if(debug) Serial.print("Emparejamiento recuperado de la memoria RTC
del usuario ");
      printMAC(pairingData.macAddr);
```

```
      if(debug) Serial.print(" en el canal " );
      if(debug) Serial.print(pairingData.channel);    // channel used by
the server
      // Imprime el tiempo transcurrido desde el inicio del proceso de
emparejamiento
      if(debug) Serial.print(" en ");
      if(debug) Serial.print(millis()-start_time);
      if(debug) Serial.println("ms");

    // Set device as a Wi-Fi Station
      WiFi.mode(WIFI_STA);
      if(debug) Serial.print(" MI DIRECCIÓN MAC: ");
      if(debug) Serial.println(WiFi.macAddress());
      wifi_promiscuous_enable(1);
      wifi_set_channel(pairingData.channel);
      wifi_promiscuous_enable(0);
      WiFi.disconnect();

      // Init ESP-NOW
      if (esp_now_init() != 0) {
        if(debug) Serial.println("Error al inicializar ESP-NOW");
        return;
      }

      // Set ESP-NOW Role
      esp_now_set_self_role(ESP_NOW_ROLE_COMBO);

      // Register for a callback function that will be called when data
is received
      esp_now_register_recv_cb(AUTOpairing_t::OnDataRecv);
      esp_now_register_send_cb(AUTOpairing_t::OnDataSent);

      esp_now_add_peer(pairingData.macAddr, ESP_NOW_ROLE_COMBO,
pairingData.channel, NULL, 0); // add the server to the peer list
      pairingStatus = PAIR_PAIRED ;             // set the pairing status
    }
}

static void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  este_objeto->_OnDataSent(mac_addr,sendStatus);
}

//-------------------------------------------------------------
// Callback when data is sent
 void _OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  if(debug) Serial.print("Estado de envío del último paquete a ");
  printMAC(mac_addr);
```

```
  if (sendStatus == 0){
    if(debug) Serial.println(" >> Éxito de entrega");
    if(pairingStatus == PAIR_PAIRED && mensaje_enviado)
    {
       mensaje_enviado=false;
       if (terminar && !esperando) gotoSleep();
    }
  }
  else{
    if(debug) Serial.println(" >> Error de entrega");
    if(pairingStatus == PAIR_PAIRED && mensaje_enviado)
    {
      //no hemos conseguido hablar con la pasarela emparejada...
      // invalidamos config en flash;
      rtcData.code1=INVALID_CODE;
      rtcData.data.channel= 2;
      if(usar_FLASH) {
            EEPROM.put(0, rtcData);
            EEPROM.commit();
      } else {
            ESP.rtcUserMemoryWrite(0,(uint32_t *)&rtcData,
sizeof(rtcData));
      } // end if usar_FLASH
      if(debug) Serial.println(" INFO de emparejamiento invalidada");
      pairingStatus = PAIR_REQUEST; // volvemos a intentarlo?
      mensaje_enviado=false;
      terminar=false;
      //delay(100);
      //gotoSleep();
    }
  }

}

static void OnDataRecv(uint8_t * mac, uint8_t *incommingData, uint8_t
len)
{
  este_objeto->_OnDataRecv(mac,incommingData,len);
}
//-----------------------------------------------------------
// Callback when data is received
 void _OnDataRecv(uint8_t * mac, uint8_t *incommingData, uint8_t len) {
  uint8_t type = incommingData[0];
  if(debug) Serial.print("tipo de mensaje recibido = ");
  if(debug) Serial.print(type,BIN);
  if(debug) Serial.println(messType2String(type));
  if(debug) Serial.print("Tamaño del mensaje : ");
```

```
  if(debug) Serial.print(len);
  if(debug) Serial.print(" de ");
  printMAC(mac);
  if(debug) Serial.println();
  uint8_t i;
  String topic;
  String payload;

  switch (type & MASK_MSG_TYPE) {

  case NODATA:
        esperando = false;
        if(debug) Serial.println("No hay mensajes MQTT");
        if (terminar && cola_mensajes.empty()) gotoSleep();
    break;

  case DATA:
        // recibimos mensaje mqtt, trasladarlo al callback()
        if(debug) Serial.println("Mensaje recibido MQTT");
        for(i=0; i<len; i++) if(incommingData[i]=='|') break;
        topic=String(std::string((char*)incommingData+1,i-1).c_str());
        payload = String(std::string((char*)(incommingData+i+1),len-i-
1).c_str());
        if(user_callback!=NULL) cola_mensajes.push(mensajeMQTT_t(topic,
payload));
        //user_callback(topic,payload);
    break;

  case PAIRING:
    memcpy(&pairingData, incommingData, sizeof(pairingData));
    if (pairingData.id == GATEWAY) {                // the message comes
from server
        if(debug) Serial.print("Emparejamiento hecho para ");
        printMAC(pairingData.macAddr);
        if(debug) Serial.print(" en el canal " );
        if(debug) Serial.print(pairingData.channel);    // channel used
by the server
        if(debug) Serial.print(" en ");
        if(debug) Serial.print(millis()-start_time);
        if(debug) Serial.println("ms");
      //esp_now_del_peer(pairingData.macAddr);
      //esp_now_del_peer(mac);
      esp_now_add_peer(pairingData.macAddr, ESP_NOW_ROLE_COMBO,
pairingData.channel, NULL, 0); // add the server to the peer list
      pairingStatus = PAIR_PAIRED ;             // set the pairing status
      //guardar en FLASH
      rtcData.code1=MAGIC_CODE1;
```

```
      memcpy(&(rtcData.data), &pairingData, sizeof(pairingData));
      if(usar_FLASH) {
            if(debug) Serial.println("Escribimos emparejamiento en
FLASH");
            EEPROM.put(0, rtcData);
            EEPROM.commit();
      } else {
            if(debug) Serial.println("Escribimos emparejamiento en RTC
MEM");
            ESP.rtcUserMemoryWrite(0,(uint32_t *)&rtcData,
sizeof(rtcData));
      } // end if usar_FLASH
       if(debug) Serial.println("Emparejado en milisegundo =
"+String(millis()));
    }
    break;
  }
}

//void func ( void (*f)(int) );
//-----------------------------------------------------------
 void set_callback( void (*_user_callback)(String, String) )
{
  user_callback=_user_callback;
}

//-----------------------------------------------------------
 void check_messages()
  {
    mensaje_enviado=true;
    esperando=true;
    timeOut+=500;
    uint8_t mensaje_esp;
    mensaje_esp=CHECK;
    if(debug) Serial.print("Enviando petición de compronación de
mensajes... ");
    esp_now_send(pairingData.macAddr, (uint8_t *) &mensaje_esp, 1);
  }

//-----------------------------------------------------------
 bool espnow_send_check(char * mensaje, bool fin=true, uint8_t
_msgType=DATA)
  {
    esperando=true;
    timeOut+=500;
    return espnow_send(mensaje, fin, _msgType | CHECK);
  }
```

```
//------------------------------------------------------------
 bool espnow_send(char * mensaje, bool fin=true, uint8_t _msgType=DATA)
  {
    _msgType = _msgType | ((panAddress << PAN_OFFSET) & MASK_PAN);
    if(debug) Serial.print("sending message type = ");
    if(debug) Serial.print(_msgType,BIN);
    if(debug) Serial.println(messType2String(_msgType));
    mensaje_enviado=true;
    terminar=fin;
    mensajes_sent++;
    int size = strlen(mensaje);
    if (size> 249)
    {
      if(debug) Serial.print("Longitud del mensaje demasiado larga: ");
      if(debug) Serial.println(size);
      return false;
    }
    struct_espnow mensaje_esp;
    mensaje_esp.msgType=_msgType;
    memcpy(mensaje_esp.payload, mensaje, size);
    if(debug) Serial.print("Longitud del mensaje: ");
    if(debug) Serial.println(size);
    if(debug) Serial.print("mensaje: ");
    if(debug) Serial.println(mensaje);
    esp_now_send(pairingData.macAddr, (uint8_t *) &mensaje_esp, size+1);
    return true;
  }

//------------------------------------------------------------
 int mensajes_enviados()
{
  return mensajes_sent;
}

//------------------------------------------------------------
 bool emparejado()
{
  return (pairingStatus==PAIR_PAIRED) ;
}

//------------------------------------------------------------
 bool envio_disponible()
{
  return (pairingStatus==PAIR_PAIRED && mensaje_enviado==false &&
terminar==false) ;
}
```

```
//-------------------------------------------------------------
 bool mantener_conexion()
  {
  unsigned long currentMillis;

  if(!cola_mensajes.empty())
  {
    mensajeMQTT_t mensaje = cola_mensajes.front();
    user_callback(mensaje.topic,mensaje.payload);
    cola_mensajes.pop();
  }
  else if( terminar && !esperando) gotoSleep();

  if(millis()-start_time > timeOut && timeOutEnabled )
  {
    if(debug) Serial.println("SE PASO EL TIEMPO SIN EMPAREJAR o SIN
ENVIAR");
    if(debug) Serial.println("millis = "+String(millis())+" limite:
"+String(timeOut));
    gotoSleep();
  }

  switch(pairingStatus) {
  case PAIR_REQUEST:
    if(debug) Serial.print(" Solicitud de emparejamiento en el canal
"  );
    if(debug) Serial.println(channel);

    // clean esp now
    esp_now_deinit();
    WiFi.mode(WIFI_STA);
    // set WiFi channel
    wifi_promiscuous_enable(1);
    wifi_set_channel(channel);
    wifi_promiscuous_enable(0);
    //WiFi.printDiag(Serial);
    WiFi.disconnect();

    // Init ESP-NOW
    if (esp_now_init() != 0) {
      if(debug) Serial.println("Error al inicializar ESP-NOW");
    }
    esp_now_set_self_role(ESP_NOW_ROLE_COMBO);
    // set callback routines
    esp_now_register_send_cb(AUTOpairing_t::OnDataSent);
    esp_now_register_recv_cb(AUTOpairing_t::OnDataRecv);
```

```
    // set pairing data to send to the server
    pairingData.msgType = PAIRING;
    pairingData.id = ESPNOW_DEVICE;
    previousMillis_scanChannel = millis();
    // send request
    esp_now_send(broadcastAddressX, (uint8_t *) &pairingData,
sizeof(pairingData));
    pairingStatus = PAIR_REQUESTED;
    break;

  case PAIR_REQUESTED:
    // time out to allow receiving response from server
    currentMillis = millis();
    if(currentMillis - previousMillis_scanChannel > 100) {
      previousMillis_scanChannel = currentMillis;
      // time out expired,  try next channel
      channel ++;
      if (channel > 11) {
        channel = 0;
      }
      pairingStatus = PAIR_REQUEST;
    }
    break;

  case PAIR_PAIRED:
    //if(debug) Serial.println("Paired!");
    break;
  }
  return (pairingStatus==PAIR_PAIRED) ;
}


//------------------------------------------------------------
 void gotoSleep() {
  // add some randomness to avoid collisions with multiple devices
  if(debug) Serial.println("Apaga y vamonos");
  ESP.deepSleepInstant(segundos_en_deepSleep * 1000000, RF_NO_CAL);
}

// end of class
};

//------------------------------------------------------------
//------------------------------------------------------------

// statics:
 AUTOpairing_t *AUTOpairing_t::este_objeto=NULL;
```

```
#endif
```

Anexo 2.1.3 Funciones.cpp

Dentro de este apéndice, se presenta el código correspondiente a una biblioteca (creación propia) declarada como “funciones.h”, en la cual, se han implementado diversas funciones que desempeñan un papel fundamental en el proyecto, centrándose en la captura de datos provenientes de sensores y en el registro de las mediciones efectuadas por estos dispositivos.

```
//librerias

#include "funciones.h"
#include<OneWire.h>
#include<DallasTemperature.h>

//variables
/* DECLARACIÓN DE PINES */
#define ONE_WIRE_BUS 4 // DECLARACIÓN DE GPIO 4 (D2) COMO BUS DE
COMUNICACIÓN CON LOS 3 SENSORES DE TEMPERATURA
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

/* DECLARACIÓN DE VARIABLES */
int numberOfDevices;
/*adquisición de las direcciones de los sensores conectados al bus*/
DeviceAddress tempDeviceAddress;
/* LLAMADA A FUNCIÓN DE ADQUISICIÓN DE DIRECCIONES DE SENSORES CONECTADOS
AL BUS DE TEMPERATURA*/


// FUNCIÓN PARA IMPRIMIR LAS DIRECCIONES DE LOS SENSORES ENCONTRADOS
void printAddress(DeviceAddress deviceAddress){
    for (uint8_t i = 0; i < 8; i++){
      if (deviceAddress[i] < 16) Serial.print("0");
      Serial.print(deviceAddress[i], HEX);
    }
  }
    // FUNCIÓN DE ADQUISICIÓN DE DIRECCIONES DE SENSORES CONECTADOS AL
BUS DE TEMPERATURA
  void adquisicion_direcciones_temp(){
    sensors.begin();
    sensors.setWaitForConversion(true);
    // Variable para almacenar cuantos dispositivos tengo conectados al
bus
    numberOfDevices = sensors.getDeviceCount();
```

```
    // Encontrar dispositivos conectados al bus
    Serial.print("Localización de dispositivos...");
    Serial.print("encontrados ");
    Serial.print(numberOfDevices, DEC);
    Serial.println(" Sensores de temperatura.");

    // Recorrer cada dispositivo y obteber sus direcciónes
    for(int i=0;i<numberOfDevices; i++){

         if(sensors.getAddress(tempDeviceAddress, i)){
          Serial.print("Sensor  ");
          Serial.print(i, DEC);
          Serial.print(" con dirección: ");
          printAddress(tempDeviceAddress);
          Serial.println();
          } else {
          Serial.print("Dispositivos sin dirección ");
          Serial.print(i, DEC);
          Serial.print(" pero no pudo detectar la dirección. Comprobar la
alimentación y el cableado");
          }
         }
    sensors.requestTemperatures();
  } //b


  void pedir_temperaturas(float temp[]){

    /*ENVIO DE PETICIÓN PARA ADQUIRIR TEMPERATURAS*/

    for(int i=0;i<numberOfDevices; i++){
    // Search the wire for address
      if(1 || sensors.getAddress(tempDeviceAddress, i)){
      // Output the device ID
      Serial.print("Temperatura sensor: ");
      Serial.println(i,DEC);
      // Print the data
      float tempC = sensors.getTempCByIndex(i);
      temp[i]=tempC;
      Serial.print("Temp C: ");
      Serial.print(tempC);
      Serial.print(" Temp F: ");
      Serial.println(DallasTemperature::toFahrenheit(tempC)); // Converts
tempC to Fahrenheit
    }
  }
```

```
  //delay(1000);
}

float  pedir_humedad()
{
  //declaración de pines
  const int analogInPin = A0;  // ESP8266 Analog Pin ADC0 = A0

  //inicialización de pines
  float sensorValue = 0;  // Inicializar valor de lectura analógica


  /*calibración sensor de humedad*/
  float ValorHumedadMinima=1080; // Registre el valor del sensor cuando
la sonda esté expuesta al aire como "ValorHumedadMinima". Este es el
valor límite del suelo seco "Humedad: 0% HR"
  float ValorHumedadMaxima=420; // Registre el valor del sensor cuando la
sonda esté expuesta al agua como "ValorHumedadMaxima". Este es el valor
límite del suelo húmedo "Humedad: 100% HR"
  float HUM_RANGO (ValorHumedadMinima - ValorHumedadMaxima);

 delay(900);
  // lectura de sensor humedad
  sensorValue = analogRead(analogInPin);
  float porcentaje_humedad = 100 * (1 - (sensorValue -
ValorHumedadMaxima) / HUM_RANGO);

  // Imprimir medida por el monitor de l puerto serie
  //  Serial.print("sensor humedad = ");
  //  Serial.print(sensorValue);
  //  Serial.print('\n');
  Serial.print("sensor humedad RH% = ");
  Serial.print(porcentaje_humedad);
  Serial.print('\n');
  return porcentaje_humedad;

}
```

Anexo 2.2 Código del dispositivo ESP32

Anexo 2.2.1 ESP32_auto_pairing.ino

El código que se presenta a continuación es una adaptación desarrollada a partir del proyecto inicialmente creado por "Rui Santos". El origen de este proyecto se encuentra documentado en su repositorio [11].

```
/*   Pasarela ESP-NOW <-> MQTT con auto emparejamiento
 *    El dispositivo envía mensajes en JSON por ESPNOW (más un primer
byte de control)
 *    Los mensajes se publican en infind/espnow/<MAC>[/<TOPIC>]
 *    Siendo MAC la dirección mac del dispositivo emisor y TOPIC
opcionamente puede establecerse con la clave "topic" en el JSON
 *
 *   Se pueden enviar mensajes a los dispositivos a través del topic
infind/espnowdevice
 *   El mensaje de debe contener una cadena JSON, con las claves: mac,
topic, payload
 *   Siendo mac la dirección mac del dispositivo destino, topic una
cadena que identifique el tipo de mensaje y payload el mensaje.
 *
 *   Para arquitectura ESP32, y así poder comunicarse simultaneamente por
ESPNOW y WiFi (mqtt).
 *
 * Distribuido bajo licencia: CC BY-NC-SA 4.0
http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es
 * Autor: Andrés Rodríguez (andres@uma.es)
 *
 * Basado en el trabajo de:
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com/?s=esp-now
  Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files.
  The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
  Based on JC Servaye example:
https://https://github.com/Servayejc/esp8266_espnow
*/

#include <esp_now.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <queue>
#include <list>
```

```
#include <string>

#include "AUTOpairing_common.h"

//---------------------------------------------------------------
// clase para manejar un mensaje esp-now
class TmensajeESP
{
  public:
  uint8_t mac[6];
  uint8_t data[250];
  uint8_t len;
  TmensajeESP ( uint8_t *_mac, uint8_t *_data, uint8_t _len)
   {
    memcpy(mac,_mac,6);
    if(_len>250) _len=250;
    memcpy(data,_data,_len);
    len=_len;
   }
};

//---------------------------------------------------------------
// clase para manejar un mensaje MQTT
class TmensajeMQTT
{
  public:
  uint8_t mac[6];
  uint8_t data[250];
  uint8_t len;
  TmensajeMQTT ( uint8_t *_mac, uint8_t *_data, uint8_t _len)
   {
    memcpy(mac,_mac,6);
    if(_len>250) _len=250;
    memcpy(data,_data,_len);
    len=_len;
   }
};


// cola de mensajes esp-now recibidos
std::queue<TmensajeESP> cola_mensajes;
// cola de mensajes MQTT recibidos
std::list<TmensajeMQTT> mensajes_MQTT;
// cola de dispositivos (mac) esperando por sus mensajes
std::queue<String> readyMACs;
std::queue<String> pairMACs;
```

```
WiFiClient wClient;
PubSubClient mqtt_client(wClient);

// cadenas para topics e ID
char ID_PLACA[16];
char topic_PUB[256];
char mensaje_mqtt[512];
char JSON_serie[257];  // 6 bytes de la MAC + 1 byte del tamaño del
mensaje esp-now + 250 bytes para el mensaje = 257
char JSON_serie_bak[257];
String deviceMac;

const char* mqtt_server = "iot.ac.uma.es";
const char* mqtt_user = "infind";
const char* mqtt_pass = "zancudo";

// Replace with your network credentials (STATION)
//const char* ssid = "infind";
//const char* password = "1518wifi";
//const char* ssid = "tfgshuerta";
//const char* password = "946CXQ2d*WHm";
const char* ssid = "sagemcom67E0_EXT";
const char* password = "UWZKFTHRWZZTYY";

esp_now_peer_info_t slave;
int myChannel;


int counter = 0;

//-----------------------------------------------------
void conecta_wifi() {
  Serial.printf("\nConnecting to %s:\n", ssid);

  WiFi.mode(WIFI_AP_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
    Serial.print(".");
  }
  Serial.printf("\nWiFi connected, IP address: %s\n",
WiFi.localIP().toString().c_str());
}

//-----------------------------------------------------
void conecta_mqtt() {
```

```
  // Loop until we're reconnected
  while (!mqtt_client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (mqtt_client.connect(ID_PLACA, mqtt_user, mqtt_pass)) {
      Serial.printf(" conectado a broker: %s\n",mqtt_server);
      mqtt_client.subscribe("infind/espnowdevice");
    } else {
      Serial.printf("failed, rc=%d  try again in 5s\n",
mqtt_client.state());
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}


// ---------------------------- esp_ now -------------------------
void printMAC(const uint8_t * mac_addr){
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3],
mac_addr[4], mac_addr[5]);
  Serial.print(macStr);
}

//-------------------------------------------------------------
bool addPeer(const uint8_t *peer_addr) {      // add pairing
  memset(&slave, 0, sizeof(slave));
  const esp_now_peer_info_t *peer = &slave;
  memcpy(slave.peer_addr, peer_addr, 6);

  slave.channel = myChannel; // pick a channel
  slave.encrypt = 0; // no encryption
  // check if the peer exists
  bool exists = esp_now_is_peer_exist(slave.peer_addr);
  if (exists) {
    // Slave already paired.
    Serial.println("Already Paired");
    return false;
  }
  else {
    esp_err_t addStatus = esp_now_add_peer(peer);
    if (addStatus == ESP_OK) {
      // Pair success
      Serial.println("Pair success");
      return true;
```

```
    }
    else
    {
      Serial.println("Pair failed");
      return false;
    }
  }
}

//-------------------------------------------------------------
// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("Last Packet Send Status: ");
  Serial.print(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success to " :
"Delivery Fail to ");
  printMAC(mac_addr);
  Serial.println();
}

//-------------------------------------------------------------
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData,
int len) {
  uint8_t type = incomingData[0];       // first message byte is the type
of message
  Serial.println();
  Serial.print("received message type = ");
  Serial.print(type,BIN);
  Serial.println(messType2String(type));Serial.print("ESPNOW: ");
  Serial.print(len);
  Serial.print(" bytes of data received from : ");
  printMAC(mac_addr);
  Serial.println();

  if(type & CHECK)
  {
   Serial.println("Device wants to get messages (CHECK==1)...");
   readyMACs.push(byte2HEX((uint8_t*)mac_addr));
  }

  switch (type & MASK_MSG_TYPE) {  // dos bits menos significativos

  case DATA :                              // the message is data type
     // encola el mensaje para su envío por serie en loop()
     cola_mensajes.push(TmensajeESP((uint8_t *)mac_addr,(uint8_t
*)incomingData+1, len-1));
     if( addPeer(mac_addr))
        {
```

```
          Serial.println("New device paired");
          pairMACs.push(byte2HEX((uint8_t*)mac_addr));
        }
     // TEST respuesta
     /*    struct_espnow mensaje_esp;
       mensaje_esp.msgType=DATA;
       memcpy(mensaje_esp.payload, "topic/dos|{\"otro\":12}",21);
       esp_now_send(mac_addr, (uint8_t *) &mensaje_esp, 22);
       */
    break;

  case PAIRING:                             // the message is a pairing
request
    struct_pairing pairingData;
    memcpy(&pairingData, incomingData, sizeof(pairingData));
    Serial.println(pairingData.msgType);
    Serial.println(pairingData.id);
    Serial.print("Pairing request from: ");
    printMAC(mac_addr);
    Serial.println();
    if (pairingData.id != GATEWAY) {     // do not replay to server
itself
      if (pairingData.msgType == PAIRING) {
        pairingData.id = GATEWAY;
        // Server is in AP_STA mode: peers need to send data to server
soft AP MAC address
        WiFi.softAPmacAddress(pairingData.macAddr);   // send my mac
Address to pairing device
        pairingData.channel = myChannel;
        Serial.println("send response");
        if( addPeer(mac_addr))
        {
          Serial.println("New device paired");
          pairMACs.push(byte2HEX((uint8_t*)mac_addr));
        }

        esp_err_t result = esp_now_send(mac_addr, (uint8_t *)
&pairingData, sizeof(pairingData));

      }
    }
    break;
  }
}

//-------------------------------------------------------------
void initESP_NOW(){
```

```
    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
      Serial.println("Error initializing ESP-NOW");
      return;
    }
    esp_now_register_send_cb(OnDataSent);
    esp_now_register_recv_cb(OnDataRecv);
}

//-----------------------------------------------------
//RECIBIR MQTT
void procesa_mensaje(char* topic, byte* payload, unsigned int length) {
  String mensaje=String(std::string((char*) payload,length).c_str());
  Serial.println("Mensaje recibido ["+ String(topic) +"] "+
mensaje);//LEVEL CON LO DE PROCESS
  // compruebo el topic
  if(String(topic) == "infind/espnowdevice")
  {
    StaticJsonDocument<512> json; // el tamaño tiene que ser adecuado
para el mensaje
    // Deserialize the JSON document
    DeserializationError error = deserializeJson(json, mensaje.c_str());

    // Compruebo si no hubo error
    if (error) {
      Serial.print("Error funcion deserializeJson(): ");
      Serial.println(error.c_str());
    }
    else
    if(!json.containsKey("mac") && !json.containsKey("topic") &&
!json.containsKey("payload"))  // comprobar si existe el campo/clave que
estamos buscando
    {
      Serial.print("Error : ");
      Serial.println("campos mac, topic o payload no encontrados");
    }
    else
    {
     String mac =json["mac"];
     String topic =json["topic"];
     String output;
     serializeJson(json["payload"], output);

     String compacto = topic + "|" + output;

     Serial.print("Mensaje OK, mac = ");
     Serial.println(mac);//es lo que hay que sacar
```

```
      uint8_t mac_addr[6];
      HEX2byte(mac_addr, (char*) mac.c_str() );
      mensajes_MQTT.push_back(TmensajeMQTT(mac_addr, (uint8_t
*)compacto.c_str(), compacto.length()));
      Serial.print("compacto = ");
      Serial.println(compacto);//es lo que hay que sacar

    }
  } // if topic
  else
  {
    Serial.println("Error: Topic desconocido");
  }

}

//--------------------------------------------------------------------------
------
//--------------------------------------------------------------------------
------
void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  Serial.println();

  sprintf(ID_PLACA, "ESP_%d", ESP.getEfuseMac());
  conecta_wifi();

  Serial.print("Server MAC Address:  ");
  Serial.println(WiFi.macAddress());

  // Set the device as a Station and Soft Access Point simultaneously

  Serial.print("Server SOFT AP MAC Address:  ");
  Serial.println(WiFi.softAPmacAddress());

  mqtt_client.setServer(mqtt_server, 1883);
  mqtt_client.setBufferSize(512); // para poder enviar mensajes de hasta
X bytes
  mqtt_client.setCallback(procesa_mensaje);

  myChannel = WiFi.channel();
  Serial.print("Station IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Wi-Fi Channel: ");
```

```
  Serial.println(WiFi.channel());
  Serial.printf("Identificador placa: %s\n", ID_PLACA );

  initESP_NOW();

  }

//------------------------------------------------------------
//std::list<TmensajeMQTT> mensajes_MQTT;
std::list<TmensajeMQTT>::iterator encuentra_mensaje(uint8_t *_mac)
{
  for (std::list<TmensajeMQTT>::iterator it=mensajes_MQTT.begin() ; it !=
mensajes_MQTT.end(); it++ ) {
    if(igualMAC(it->mac, _mac)) return it;
  }
  return mensajes_MQTT.end();
}

//------------------------------------------------------------
//------------------------------------------------------------------------
------
void loop() {
  static unsigned long lastEventTime = millis();
  if (!mqtt_client.connected()) conecta_mqtt();
  mqtt_client.loop(); // esta llamada para que la librería recupere el
control

  if (!readyMACs.empty())
  {
    uint8_t mac_addr[6];
    uint8_t size;
    String mac = readyMACs.front();
    struct_espnow mensaje_esp;
    //
    Serial.println("Buscando mensajes para dispositivo "+ mac);
    HEX2byte(mac_addr, (char*)mac.c_str());
    std::list<TmensajeMQTT>::iterator it= encuentra_mensaje(mac_addr);
    if(it==mensajes_MQTT.end())
    {
      mensaje_esp.msgType=NODATA;
      size=1;
      readyMACs.pop();
      Serial.println(" > Sin mensajes pendientes");
    }
    else
    {
      mensaje_esp.msgType=DATA;
```

```
      memcpy(mensaje_esp.payload, it->data, it->len);
      size=it->len+1;
      mensajes_MQTT.erase(it);
      Serial.println(" > Un mensaje encontrado");
    }
    esp_err_t result = esp_now_send(mac_addr, (uint8_t *) &mensaje_esp,
size);
  }

  if (!pairMACs.empty())
  {
    uint8_t mac_addr[6];
    uint8_t size;
    String mac = pairMACs.front();
    pairMACs.pop();
    Serial.println("Enviando mensaje MQTT notificación pairing "+ mac);
    sprintf(mensaje_mqtt, "{\"mac\":\"%s\"}",mac.c_str());
    mqtt_client.publish("infind/espnowpairing", mensaje_mqtt);
  }

  // envía todo lo que haya en la cola de mensajes pendientes
  if (!cola_mensajes.empty())
  {
    TmensajeESP mensaje = cola_mensajes.front();
    cola_mensajes.pop();

    deviceMac = byte2HEX(mensaje.mac);

    byte len =  mensaje.len;
    // recibimos el mensaje
    for(int i = 0; i<len ; i++)
    {
       JSON_serie[i]=mensaje.data[i];
    }
    JSON_serie[len]='\0'; //fin de cadena de caracteres de C

    Serial.printf("Mensaje len: %d \n payload: %s\n", len, JSON_serie);
    // parece que la deserialización altera la cadena de caracteres
original, así que la copiamos antes
    strcpy(JSON_serie_bak,JSON_serie);

    //comprobamos si hay campo topic en el mensaje, para añadirlo al
topic de envío
    StaticJsonDocument<512> doc;
    // Deserialize the JSON document
    DeserializationError error = deserializeJson(doc, JSON_serie, len);
```

```
    sprintf(topic_PUB, "infind/espnow/%s", deviceMac.c_str());

    // Compruebo si no hubo error
    if (error) {
      Serial.print("Error sintaxis JSON; deserializeJson() failed: ");
      Serial.println(error.c_str());
      mqtt_client.publish(topic_PUB, "{\"error\":\"Error sintaxis JSON;
deserializeJson() failed\"}");
    }
    else
    if(doc.containsKey("topic") && doc["topic"].is<const char*>())  //
comprobar si existe el campo/clave que estamos buscando
    {
     String valor = doc["topic"];
     Serial.print("Mensaje OK, topic = ");
     Serial.println(valor);
     sprintf(topic_PUB, "infind/espnow/%s/%s", deviceMac.c_str(),
valor.c_str() );
    }
    else // no existe el campo topic
    {
      Serial.println("\"topic\" key not found in JSON");
    }

    sprintf(mensaje_mqtt,
"{\"mac\":\"%s\",\"payload\":%s}",deviceMac.c_str(),JSON_serie_bak);
    // publica el mensaje recibido
    mqtt_client.publish(topic_PUB, mensaje_mqtt);

    Serial.printf("Mensaje from: %s, publicado en MQTT\n",
deviceMac.c_str());
    Serial.printf("       topic: %s\n", topic_PUB);
    Serial.printf("     payload: %s\n\n", mensaje_mqtt);
}

}
```

Anexo 2.3 Flujo del dispositivo virtual

En este último apartado se muestra y analiza el flujo creado en Node-RED, para la simulación de un dispositivo que envía datos de humedad y temperaturas, como si fuera un dispositivo instalado en el huerto y perteneciente a nuestro proyecto.

Para ello, se ha creado un nodo de entrada MQTT. Este nodo recibe mensajes MQTT con el tema "huerta/ESP12_13064636/datos". Los mensajes entrantes provienen de un dispositivo ESP8266 que tiene instalado un solo sensor de temperatura y un sensor de humedad. Por ello, se crea una función que permite obtener tres datos de temperatura a partir de la medida realizada, sumando de forma consecutiva 5°c y 10°c para enviar un mensaje con la información necesaria a través de MQTT.

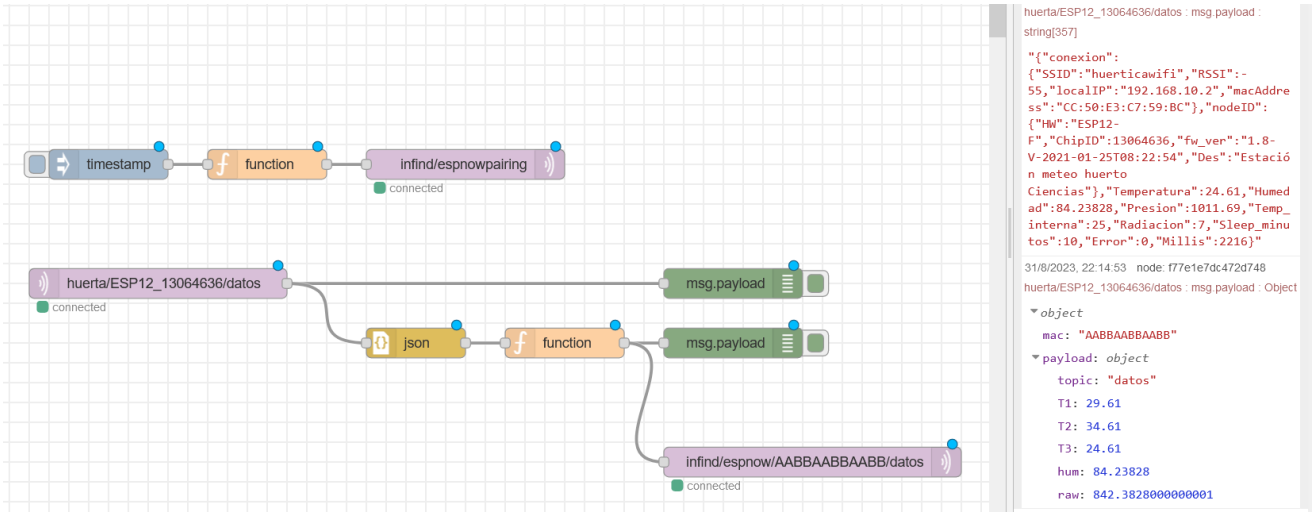

Figura 37. Simulación de un nodo para toma de datos. [Elaboración propia]