UNIVERSIDAD DE MÁLAGA

ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Informática Industrial

Proyecto final: Garaje IoT

Grupo 7

Cambil Calderón, José Luis
Camúñez Rodríguez, David
Herrera López, Juan María
Morales Benítez, Francisco

Fecha: 03/02/2022

Índice

1. Introducción y objetivos

Este documento recoge la memoria del proyecto final de la asignatura de Informática Industrial del Grado en Ingeniería Electrónica, Robótica y Mecatrónica de la Universidad de Málaga, para el curso 2021-2022. El objetivo del proyecto es el de afianzar los conocimientos adquiridos durante el desarrollo de la asignatura en lo relacionado con el Internet de las Cosas, IoT, utilizando tecnologías como la programación de microcontroladores con capacidad de conexión WiFi, el uso de MQTT para el paso de mensajes, la utilización de NodeRED para desarrollar un sistema SCADA, el uso de bases de datos como MongoDB o la utilización de sensores como el DHT11. Para eso este grupo ha pensado en desarrollar el siguiente proyecto.

El proyecto trata de realizar una maqueta o prueba de concepto de un garaje. Este garaje contará con una entrada con barrera, modelada como un servomotor, controlada por una placa ESP8266 y accionada mediante un botón. El levantamiento de la barrera se realizará tras analizar la matrícula del vehículo a la entrada. Esto se hará captando una imagen con una ESP32-CAM, y enviando la imagen a un servicio de análisis de matrículas mediante redes neuronales, para lo cual se utilizará NodeRED.

También se controlará la ocupación de cada plaza de garaje mediante un sensor de distancia conectado a una placa ESP8266 que recogerá los datos del sensor. También a esta placa se le conectará un DHT11 para monitorizar temperatura y humedad. Todos estos datos se empaquetarán en una estructura y se enviarán por ESP-Now a otra placa ESP8266 que actuará como pasarela con MQTT, de forma que los datos puedan enviarse a NodeRED y guardarse en una base de datos.

Finalmente se desarrollarán dos interfaces de usuario. Una de ellas se hará en un Dashboard de NodeRED, y servirá como interfaz del sistema SCADA del garaje. Otra se hará también en NodeRED, pero de forma que sea accesible mediante un bot de Telegram. Ambas interfaces serán capaces de visualizar datos y enviar comandos al sistema.

2. Diseño Hardware y esquemas de conexionado

A continuación, se presenta un listado del hardware utilizado en el proyecto.

- Placas de desarrollo NodeMCU ESP8266, 3 unidades.
- Placa de desarrollo ESP32-CAM, 1 unidad.
- Sensor de humedad y temperatura DHT11, 1 unidad.
- Sensor de distancia de ultrasonidos HC-SR04, 1 unidad.
- Servomotor SG90, 1 unidad.

En las siguientes imágenes se pueden ver los esquemas de conexionad de todo el hardware utilizado en el proyecto, realizados con Fritzing.

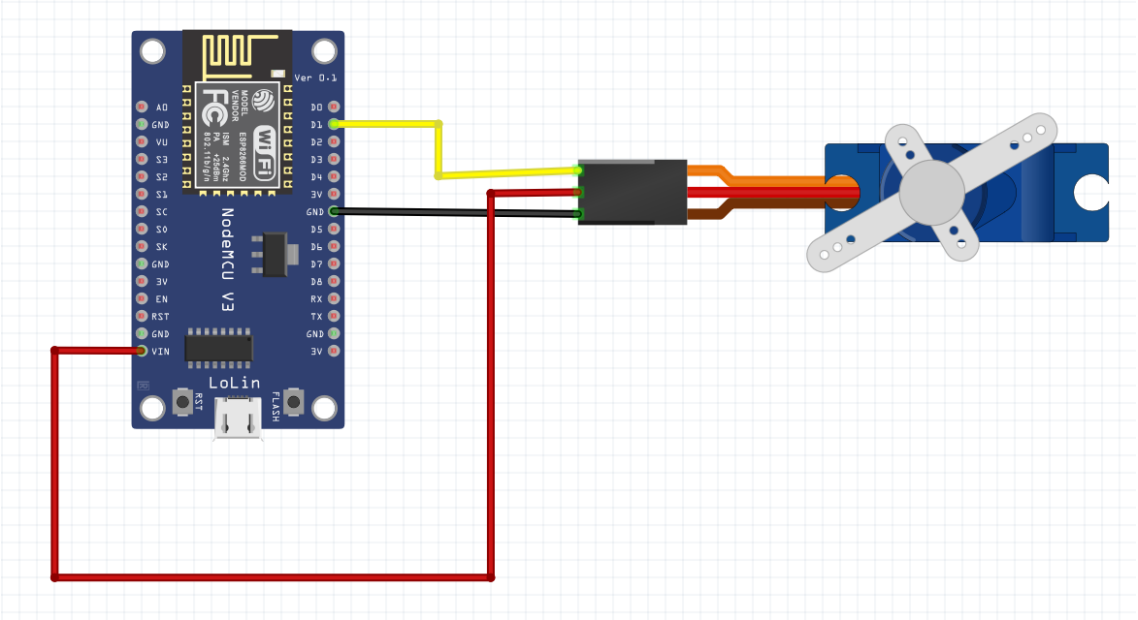

Ilustración 1. Placa de la entrada y servomotor de barrera

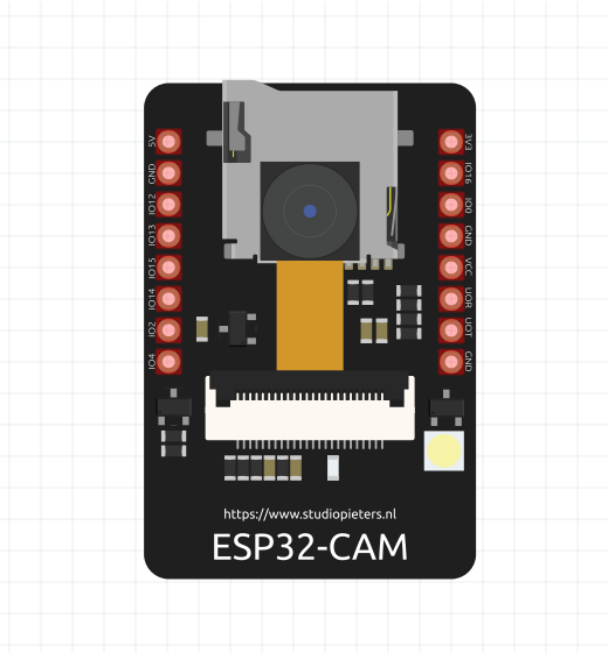

Ilustración 2. Cámara de la entrada

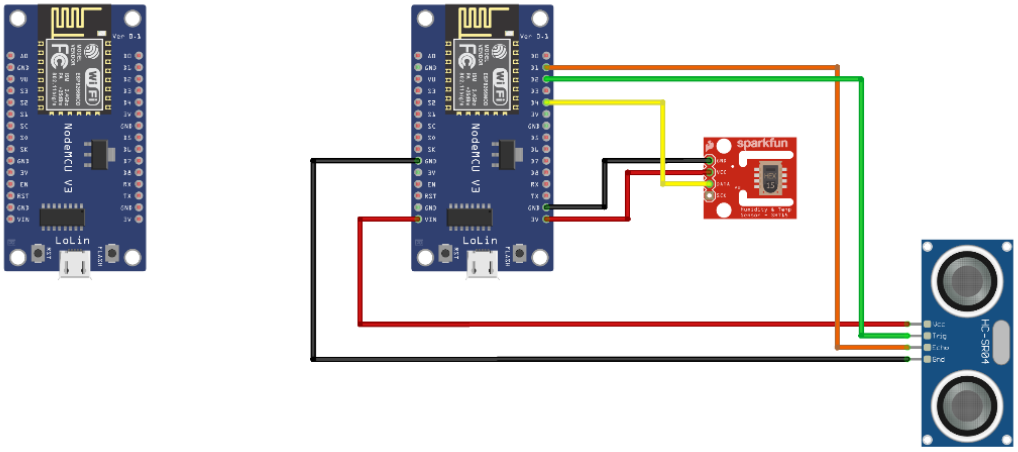

Ilustración 3. Placa de sensores y pasarela ESP-Now – MQTT

El conexionado de alimentación se ha obviado para todas las placas. Aparte de este, hay dos placas que no llevan ningún conexionado, como la de pasarela y la ESP32-CAM, mientras que la placa de sensores y la de control del motor de la entrada sí lo llevan. La placa de la entrada tiene conectada la alimentación del servomotor y la salida de datos. La de sensores lleva también la alimentación de dichos sensores, una conexión de datos para el DHT11, y dos para el sensor de distancia HC-SR04.

3. Diseño Software

3.1. Listado de topics MQTT

En el proyecto se ha decidido utilizar la siguiente estructura para los topics MQTT, utilizados para enviar mensajes entre dispositivos.

- II7
  - ESP32
    - conexion
    - config
    - FOTA
  - SensoresPlazas
    - conexion
    - config
    - FOTA
    - EstadoPlazas
  - Entrada
    - conexion
    - config
    - FOTA
    - BotonPulsado
    - BarreraEstado
    - BarreraCMD

La estructura utiliza una jerarquía que permite separar los topics en función de la placa a la que están destinados. Los que están bajo ESP32 son para la cámara, los que están bajo SensoresPlazas son para la pasarela MQTT – ESP-Now, que también conecta con las placas que monitoreen las plazas de aparcamiento, y los que estén bajo Entrada son para la placa que controla la barrera de entrada. Todas las placas tienen en común los topics acabados en /conexion, /config y /FOTA.

A continuación, se presenta el listado de los topics MQTT utilizados en el proyecto, junto con su descripción y ejemplos de uso.

- II7/ESP32/conexion
  - Indica el último estado de conexión del dispositivo. La placa es la que publica el mensaje.
  - Retenido: sí.
  - Ejemplo: {“CHIPID”: “ESP32”, “online”: “true”} o {“CHIPID”: “ESP32”, “online”: “false”}
- II7/ESP32/config
  - La placa se suscribe a este topic, en el que se pueden publicar mensajes con estructuras de datos para la configuración de parámetros en la placa. Principalmente se usa para especificar la frecuencia de comprobación de actualización FOTA.
  - Retenido: sí.
  - Ejemplo: {“frec_FOTA”: 20000}

- II7/ESP32/FOTA
    - La placa se suscribe a este topic. Cuando llega un mensaje, debe comprobar la disponibilidad de actualizaciones FOTA. El mensaje que se pase por el topic es indiferente.
    - Retenido: no.
    - Ejemplo: indiferente.
- II7/SensoresPlazas/conexion
    - Indica el último estado de conexión del dispositivo. La placa es la que publica el mensaje.
    - Retenido: sí.
    - Ejemplo: {“CHIPID”: “PSensores”, “online”: “true”} o {“CHIPID”: “PSensores”, “online”: “false”}
- II7/SensoresPlazas/config
    - La placa se suscribe a este topic, en el que se pueden publicar mensajes con estructuras de datos para la configuración de parámetros en la placa. Principalmente se usa para especificar la frecuencia de comprobación de actualización FOTA.
    - Retenido: sí.
    - Ejemplo: {“frec_FOTA”: 20000}
- II7/SensoresPlazas/FOTA
    - La placa se suscribe a este topic. Cuando llega un mensaje, debe comprobar la disponibilidad de actualizaciones FOTA. El mensaje que se pase por el topic es indiferente.
    - Retenido: no.
    - Ejemplo: indiferente.
- II7/SensoresPlazas/EstadoPlazas
    - La placa de pasarela MQTT – ESP-Now publica en este topic la estructura de datos que le llega por ESP-Now de la placa o placas de sensores.
    - Retenido: sí.
    - Ejemplo: {“chipId”: “ESP123456”, “numPlaza”: “1”, “ocupado”: “true”, “DHT11”: {“temperatura”: 23.4, “humedad”: 15.2}}
- II7/Entrada/conexion
    - Indica el último estado de conexión del dispositivo. La placa es la que publica el mensaje.
    - Retenido: sí.
    - Ejemplo: {“CHIPID”: “Entrada”, “online”: “true”} o {“CHIPID”: “Entrada”, “online”: “false”}
- II7/Entrada/config
    - La placa se suscribe a este topic, en el que se pueden publicar mensajes con estructuras de datos para la configuración de parámetros en la placa. Principalmente se usa para especificar la frecuencia de comprobación de actualización FOTA.
    - Retenido: sí.
    - Ejemplo: {“frec_FOTA”: 20000}

- II7/Entrada/FOTA
    - La placa se suscribe a este topic. Cuando llega un mensaje, debe comprobar la disponibilidad de actualizaciones FOTA. El mensaje que se pase por el topic es indiferente.
    - Retenido: no.
    - Ejemplo: indiferente.
- II7/Entrada/Pulsador
    - La placa de la entrada publica en este topic cada vez que se pulse su botón Flash (pulsación corta). No se utiliza el formato JSON porque el mensaje enviado es indiferente.
    - Retenido: no.
    - Ejemplo: indiferente.
- II7/Entrada/BarreraEstado
    - La placa de la entrada publica en este topic cada vez que se alce o se baje la barrera.
    - Retenido: sí.
    - Ejemplo: {“estado”: “subido”} o {“estado”: “bajado”}
- II7/Entrada/BarreraCMD
    - La placa de entrada se suscribe a este topic y sube o baja la barrera de entrada en función del contenido del mensaje.
    - Retenido: sí.
    - Ejemplo: {“comando”: “subir”} o {“comando”: “bajar”}

3.2. Listado de colecciones de la base de datos

En el proyecto se utiliza la base de datos MongoDB vista en clase, que permite su separación en colecciones. En concreto para este proyecto se han definido las siguientes colecciones.

- registro_matriculas_entrada
    - En esta colección se guardan como documentos los resultados de analizar las imágenes de las matrículas de los vehículos a la entrada con la API para NodeRED, tras ser estas recibidas de la cámara ESP32-CAM.
- registro_estado_plazas
    - En esta colección se guardan los documentos recibidos desde la pasarela MQTT – ESP-Now con el estado de ocupación y meteorológico de cada plaza de aparcamiento.
- registro_conexion
    - En esta colección se guardan los documentos recibidos por los topics acabados en /conexion.
- registro_barrera_entrada
    - En esta colección se guardan los documentos recibidos por el topic II7/Entrada/BarreraEstado.

En todas las colecciones, antes de guardar documentos en ellas, a cada documento se le adjunta también la fecha de guardado, como otro campo más del documento. Esto se hace bajo el campo payload. En nuestras bases de datos, además de guardar el payload de los mensajes

que les llegan por sus respectivos flujos, se guardan también el resto de campos, para tener un registro más completo de estos mensajes.

3.3. Camera Web Server

Esta parte del código se trata del proyecto Arduino que irá programado en la placa ESP32-Cam. Se encargará de montar un servidor http al que otros equipos conectados a la red podrán hacer peticiones de captura de imágenes. Además, la placa se conectará al sistema de intercambio de mensajes MQTT, principalmente para publicar sus datos de conexión y desconexión, además de para atender peticiones de otros dispositivos a través de la gestión de callbacks. Mediante diferentes topics MQTT se permitirá la configuración de parámetros relacionados con la actualización FOTA.

El código principal del proyecto viene del ejemplo CameraWebServer presente en el IDE de Arduino, que permite montar el servidor http de cámara. A los archivos de este proyecto se han añadido los archivos actualiza_dual.cpp y actualiza_dual.h, que son una modificación del proyecto actualiza_dual.ino provisto al grupo por el profesor Andrés Rodríguez Moreno, que contiene un ejemplo de código que permite realizar la actualización FOTA tanto para placas ESP32 como para ESP8266, con sentencias especiales para que en función de la placa para la que se vaya a compilar el proyecto, se incluyan unas librerías y funciones u otras.

Cabe destacar que el programa cargado en esta placa le asignará una IP a esta al conectarla a la red WiFi, y esta IP se escribirá por la consola del puerto serie tras el arranque del programa. Teniendo esta IP que, en las pruebas que hemos hecho, es siempre la misma para cada red a la que se conecta, podemos acceder al servidor http de la cámara para realizar peticiones, y también para acceder a su interfaz gráfica mediante un navegador.

Al archivo principal del proyecto, CameraWebServer.ino, también se le han añadido las funciones pertinentes para realizar la conexión con MQTT.

Se utilizan las librerías:

- PubSubClient
- WiFi
- ArduinoJson
- HTTPUpdate
- HTTPClient

El fichero actualiza_dual.h también incluye, aunque no se utiliza al compilar en la placa ESP32, las librerías:

- ESP8266WiFi
- ESP8266httpUpdate

Los demás ficheros del proyecto usan otras librerías que no se especificarán, pero que vienen todas en el ejemplo citado ya incorporadas.

El diagrama de flujo del software programado en la ESP32-Cam se muestra en la ilustración 4.

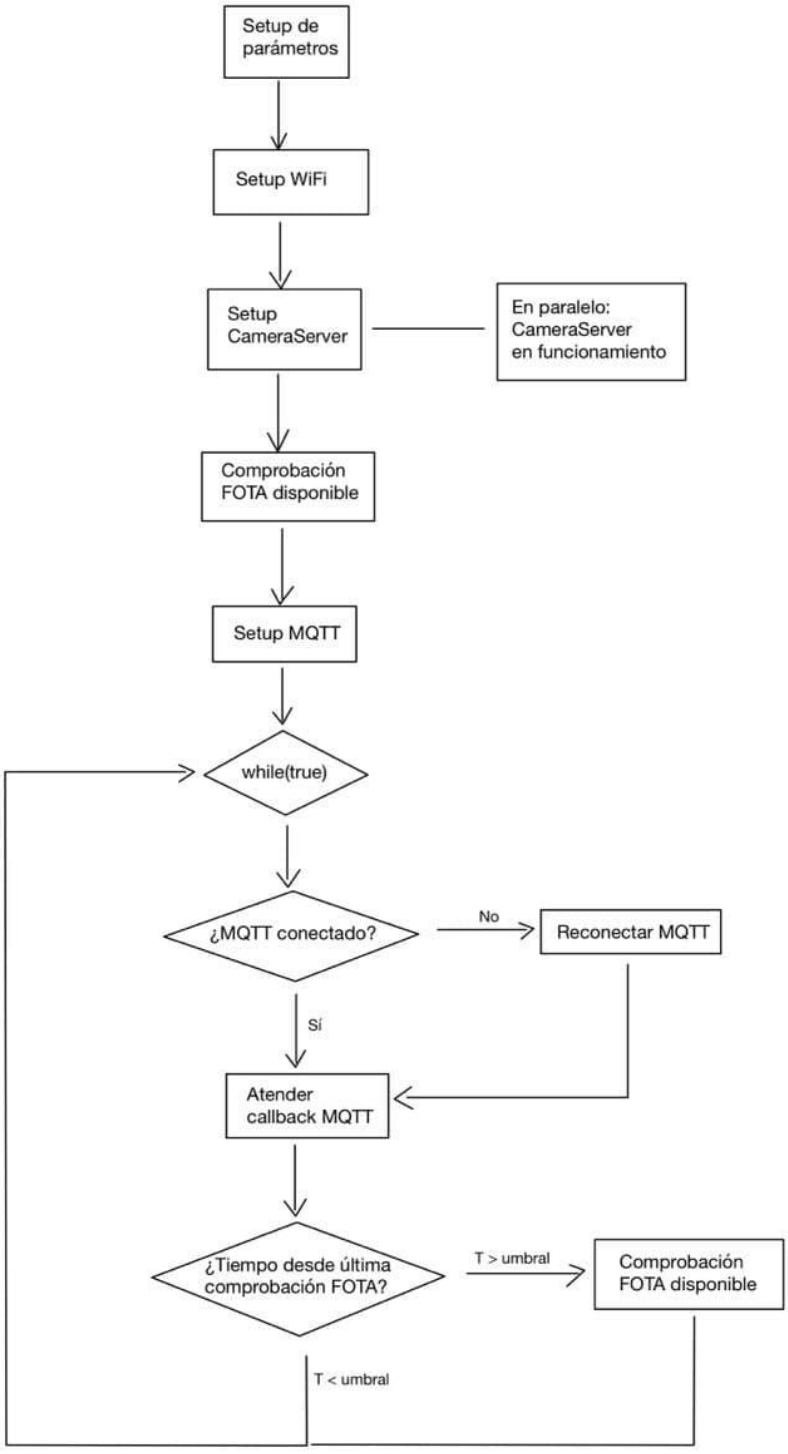

Ilustración 4. Diagrama de flujo de CameraWebServer

3.4. Entrada al garaje

Esta parte del código corresponde a la programación de una placa ESP8266 que tiene conectado un servo SG90, cuya propósito es comportarse como una hipotética barrera de garaje.

El código comienza con la inclusión de librerías y subcódigos. Aquí se incluyen:

- ESP8266WiFi.h : Permite la conexión de la placa a la red WiFi.
- PubSubClient.h : Permite el uso de MQTT.
- ArduinoJson.h : Facilita el formateo de mensajes en estructura JSON.
- Servo.h : Facilita el control de servos.
- "actualiza_dual.h" : Contiene las instrucciones necesarias para realizar la actualización FOTA.
- "pulsador_int.h" : Contiene las instrucciones necesarias para usar el botón Flash de la placa como un botón genérico.

Posteriormente, se definen e inicializan las variables y parámetros necesarios, desde las credenciales para WiFi o MQTT, hasta las relacionadas con el servo.

La función principal del botón (flash) consiste en realizar un intento de actualización FOTA cuando la duración de la pulsación es superior a 3 segundos o, en caso de durar menos, iniciar el proceso de solicitud de entrada al parking de un vehículo. Cuando el segundo caso se da, se envía un mensaje al servidor MQTT indicando que esta solicitud se ha realizado.

Cuando el sistema determina que el vehículo puede pasar, envía un mensaje que esta placa lee mediante suscripción a un topic, en el que indica que debe subirse la barrera y, posteriormente, otro para indicar que debe bajarse. Cuando la placa asociada a la barrera realiza las opciones designadas, indica a su vez al sistema del estado de la barrera. Los controles de la barrera también se pueden accionar sin necesidad de realizar una solicitud de entrada mediante los paneles de usuario creados en Node-RED (comentados más adelante).

Otras funciones incluidas son la posibilidad de realizar un intento de actualización FOTA tanto periódicamente como mediante la publicación en un topic determinado y la opción de reconfigurar la frecuencia de esta actualización de manera externa, también mediante la publicación en un topic.

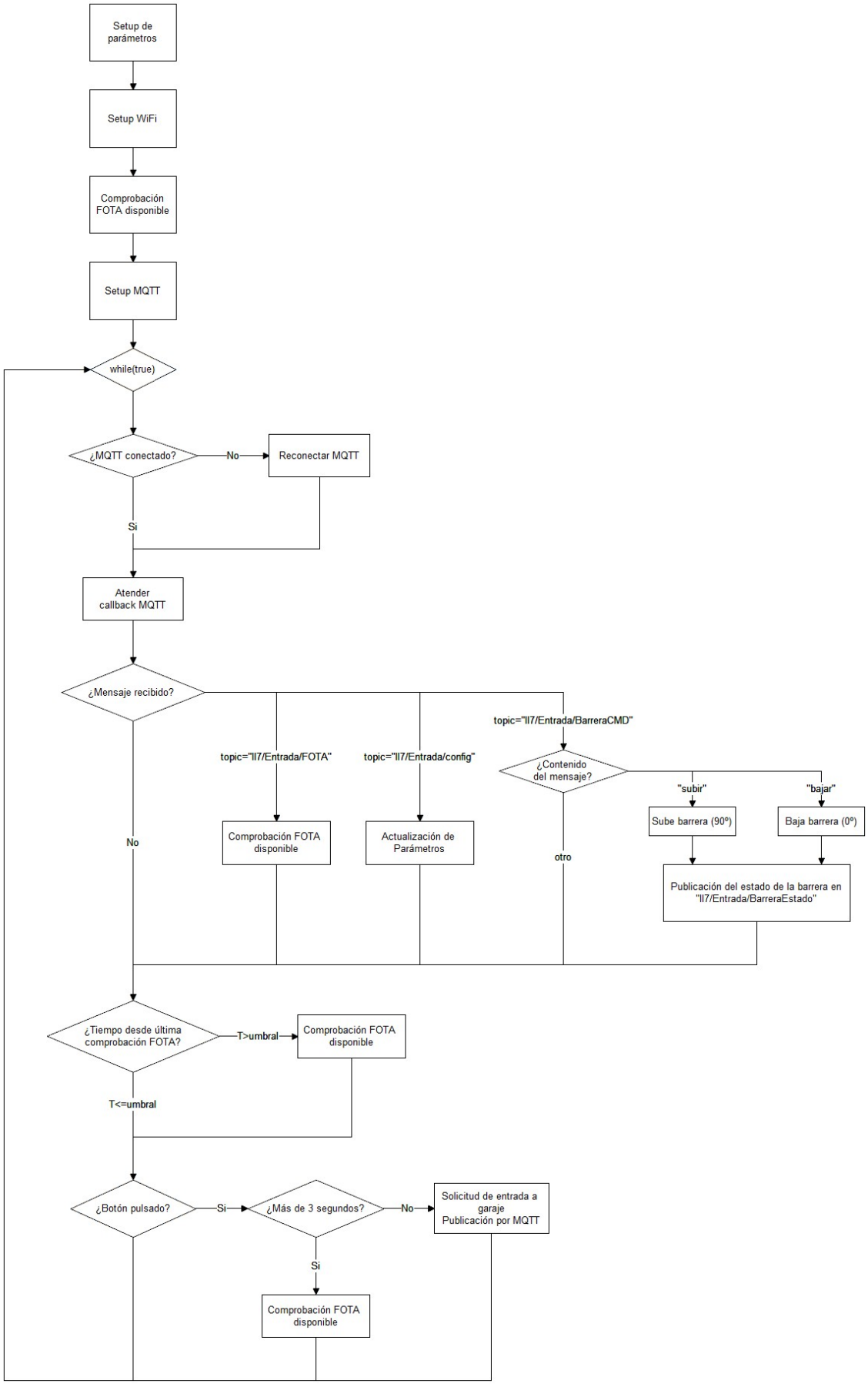

Ilustración 5. Diagrama de flujo de entrada (Servo)

3.5. Pasarela ESP-Now – MQTT

El código implementado en esta placa ESP8266 se caracteriza fundamentalmente por la recepción de mensajes mediante el protocolo ESP-NOW y su posterior procesado y publicación por MQTT.

Esta placa toma el rol de COMBO en ESP-NOW, es decir, recibe mensajes de otra placa con rol de Master y a su vez puede mandar mensajes por ESP-NOW a otras placas (aunque en nuestro funcionamiento finalmente no realiza esta última acción).

En este caso la placa queda a la espera de la recepción de mensajes por ESP-NOW con la estructura definida previamente en sintonía con el Master, la cual es la siguiente:

- Identificador de la placa (chip ID).
- Estado de la plaza.
- DHT11
    - Temperatura.
    - Humedad.
- Número de plaza.
- Confirmador de mensaje.

Como se observa este dispositivo funciona como pasarela entre el dispositivo encargado de leer los sensores y procesar sus respuestas, y el servidor MQTT.

Una vez se confirma la llegada de un mensaje, se conecta el wifi en modo station y se conecta al servidor MQTT, tras esto formatea y publica por MQTT a un topic predefinido el mensaje recibido.

Por último, una vez el mensaje ha sido publicado, la placa entra en un bucle cuya duración es configurable mediante MQTT, en este bucle la placa se encuentra escuchando a los topics a los que se encuentra suscrita (La configuración del tiempo de Escucha de MQTT y la frecuencia de actualización por FOTA). A su vez, paralelamente a esto, en el mismo bucle se realiza la comprobación del pulsado del botón de la propia placa. Si se detecta una pulsación larga se realiza la actualización FOTA.

La gran mayoría del programa se realiza mediante funciones externas que facilitan el entendimiento, comprensión y escalabilidad de este.

El diagrama de flujo del programa es el siguiente.

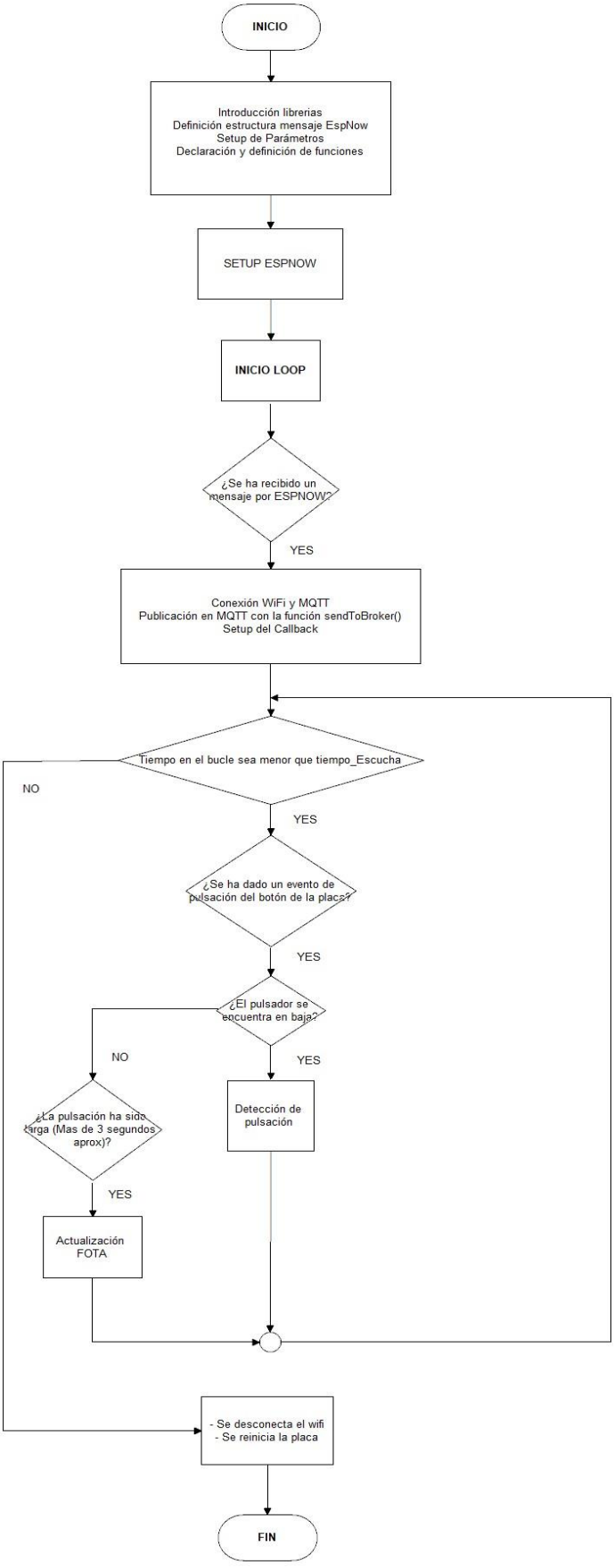

Ilustración 6. Diagrama de flujo de la pasarela

3.6. Sensores de aparcamiento

El código implementado en esta placa se caracteriza fundamentalmente por la lectura de sensores que monitorizan el estado de las plazas del parking para su posterior envío a través de la conexión ESP-NOW.

Esta placa toma el rol de Máster en ESP-NOW, es decir, es la encargada de enviar mensajes a otra placa con el rol de esclavo. En este caso en concreto se realiza un envio de mensajes cada 30 segundos. Los mensajes se encuentran en formato struct y contienen los siguientes campos:

- Identificador de la placa (chip ID).
- Estado de la plaza.
- DHT11
    - Temperatura.
    - Humedad.
- Número de plaza.
- Confirmador de mensaje.

La información relacionada con estos campos del mensaje viene dada por los sensores. El sensor DHT11 es el encargado de tomar valores de temperatura y humedad, mientras que el sensor de proximidad HCSR04 determina el estado de ocupación de las plazas y a su vez las mantiene identificadas mediante el número de plaza.

Modificación de robustez

Estudiando las posibles situaciones conflictivas que podían darse durante el uso de la placa se llegó a la conclusión de que era necesario preparar el programa para una escena en el que el sensor DHT11 deja de funcionar, o lo que es lo mismo, no proporcionara valores válidos. En caso de que esto ocurriera se ha tomado como solución el envío de los últimos valores correctos tanto de temperatura como de humedad.

El diagrama de flujo del programa es el siguiente.

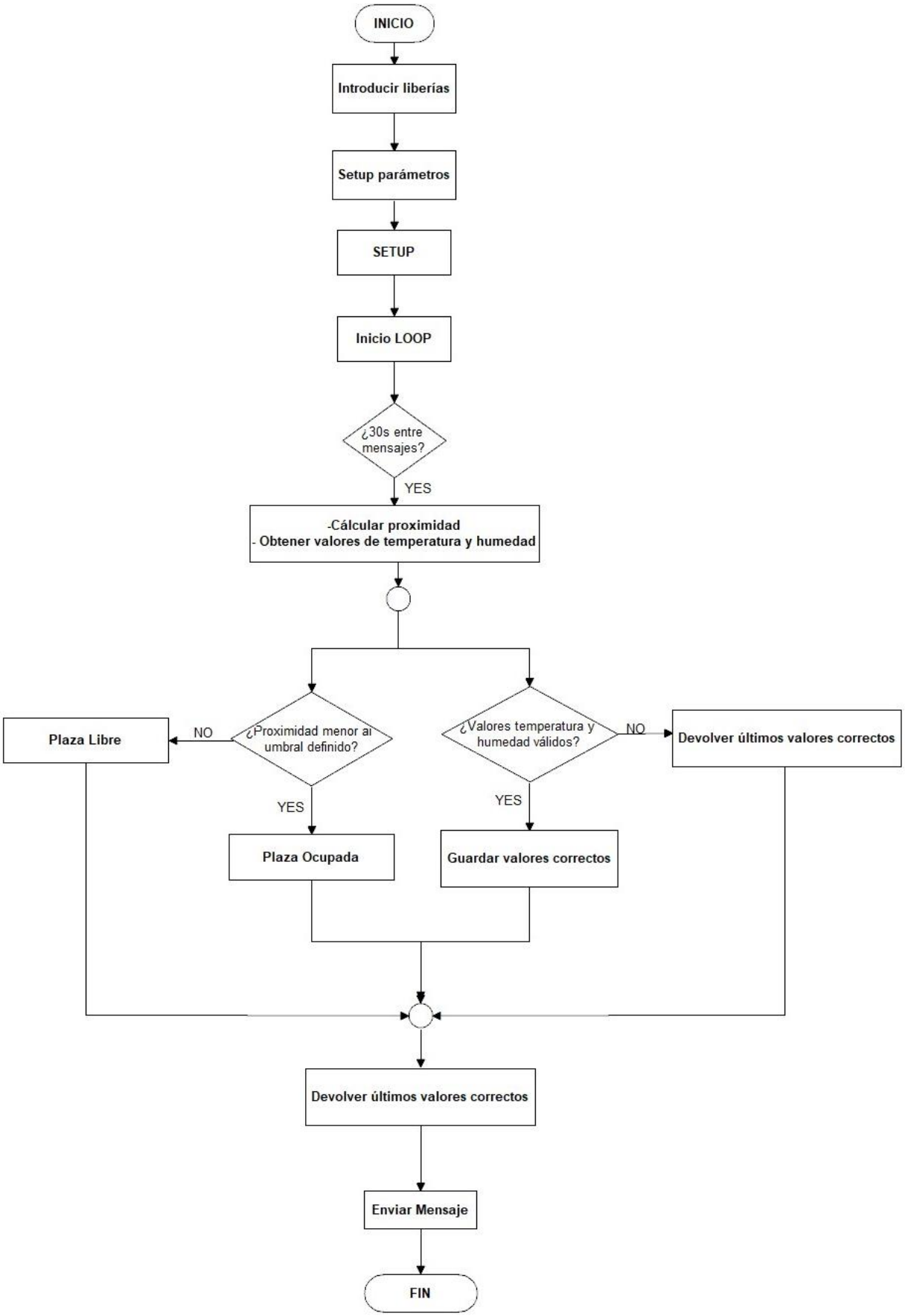

Ilustración 7. Diagrama de flujo de la placa de sensores

3.7. Flujo Node-RED: Cámara, entrada y mongoDB

El funcionamiento principal de este flujo parte de la suscripción a un topic MQTT del servidor "iot.ac.uma.es": "II7/Entrada/Pulsador".

Quien publica en este topic es la placa ESP8266 que tiene conectado un servomotor, es decir, la placa de la barrera. Cuando un vehículo solicita entrar (pulsación del botón de la placa), se envía un mensaje con valor "1" al topic indicado.

Cuando esto ocurre, el flujo de Node-RED recibe el mensaje y envía una solicitud http a la URL "http://192.168.127.108/capture". Esta URL puede cambiar en función de la red a la que la cámara esté conectada. Esto provoca que la placa ESP32 con cámara tome una fotografía, que es recibida por el flujo. En caso de que la ESP32 se conecte a una nueva red y cambie su IP, habrá que modificar esta en el flujo para que coincida con la nueva.

Posteriormente, esta imagen es enviada a un servicio de reconocimiento de matrículas (platerecognizer.com) que proporciona, entre otros datos, el código de matrícula en formato string.

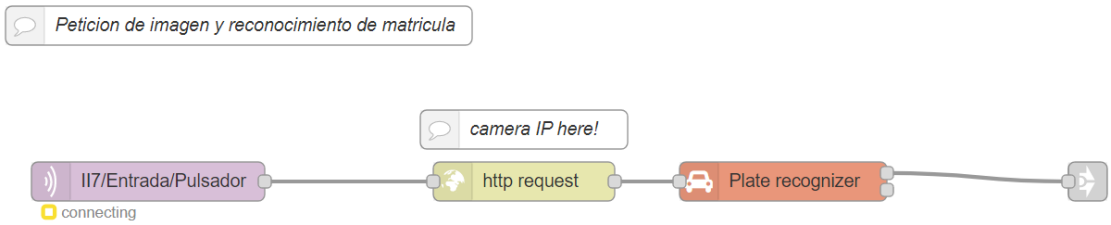

Ilustración 8. Flujo Entrada - Nodos 1

Aquí el programa se divide en dos ramas:

1. Por un lado, genera un mensaje que contiene el código de matrícula y la fecha de cuando ha sido reconocida. Este mensaje se guarda en una base de datos mongodb llamada "registro_matriculas_entrada".

2. Por otro lado, envía un mensaje al topic "II7/Entrada/BarreraCMD" con el valor "subir". Cinco segundos después, gracias al uso de un nodo "delay", envía otro con el valor "bajar". A este topic se encuentra suscrita la placa de la barrera, que emplea estos mensajes como indicadores para moverla.

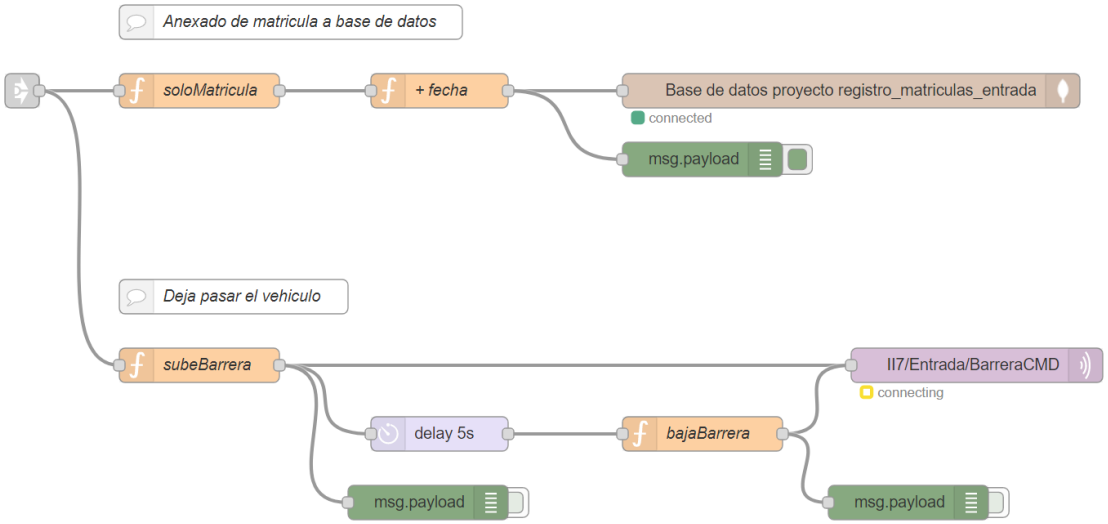

Ilustración 9. Flujo Entrada - Nodos 2

En paralelo con esto, pero dentro del mismo flujo también se realizan las acciones de indexado de mensajes en las bases de datos correspondientes con el registro de conexión de los dispositivos, el registro de mensajes provenientes de las plazas de aparcamiento, y el registro de subida y bajada de la barrera de entrada.

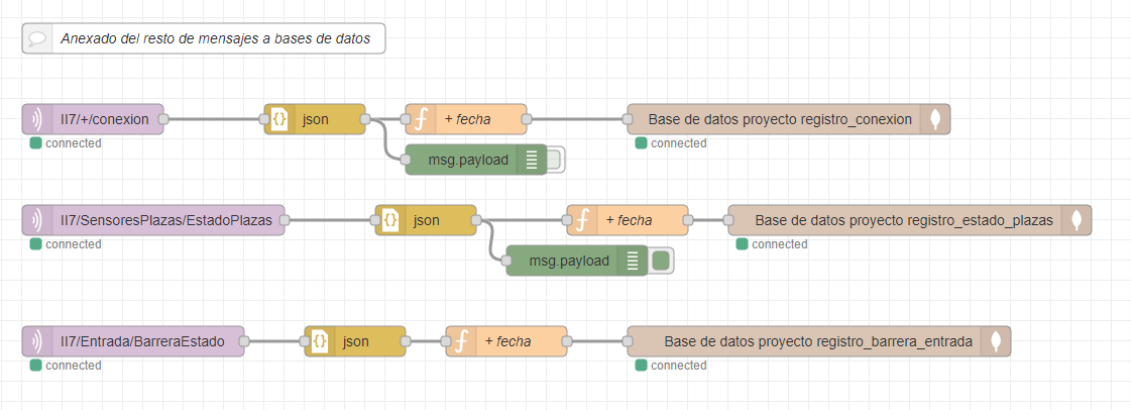

Ilustración 10. Flujo Entrada - Nodos 3

3.8. Flujo Node-RED: Telegram

Este flujo de Node-RED se encarga de gestionar el comportamiento del bot de Telegram diseñado como interfaz de control del sistema. Los mensajes se envían tanto por el chat con el bot como por un canal (en ciertos casos).

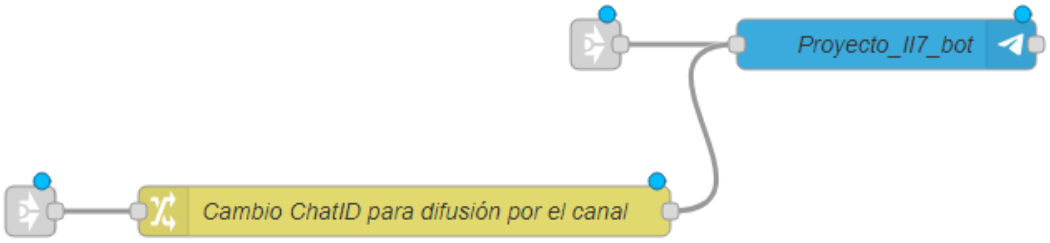

Ilustración 11. Flujo Telegram - Nodos 1

Para facilitar su análisis, se comentará en función de los comandos de usuario a los que reacciona:

- **/start:** Proporciona el mensaje de bienvenida, que facilita además la lista de comandos.

Ilustración 12. Flujo Telegram - Nodos 2

- **/ultentrada:** Envía un mensaje al usuario con la matrícula y fecha del último vehículo registrado. Para ello hace una consulta en la base de datos mongodb "registro_matriculas_entrada". La información se envía también al canal.
- **/ultregsensores:** Envía un mensaje al usuario con la temperatura y humedad de la plaza de garaje, si se encuentra ocupada o vacía y la fecha del último sensado. Para ello hace una consulta en la base de datos mongodb "registro_estado_plazas".
- **/subirbarrera:** Envía un mensaje con la palabra "subir" al topic "II7/Entrada/BarreraCMD", lo que provoca que suba la barrera. La información se envía también al canal.
- **/bajarbarrera:** Envía un mensaje con la palabra "bajar" al topic "II7/Entrada/BarreraCMD", lo que provoca que baje la barrera. La información se envía también al canal.

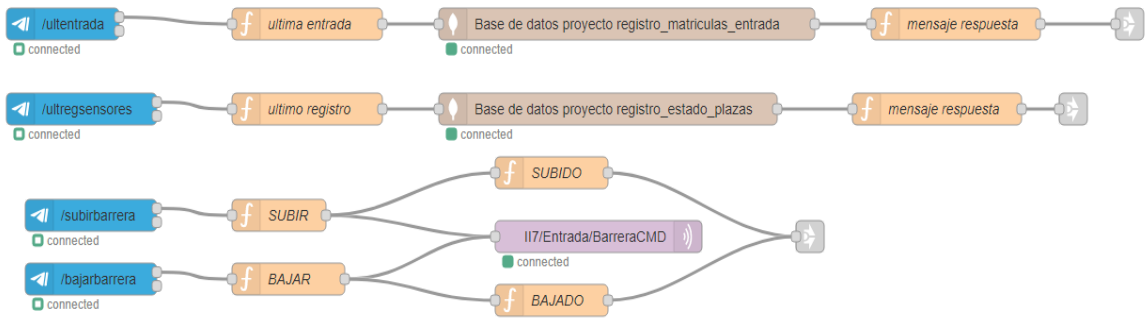

Ilustración 13. Flujo Telegram - Nodos 3

- **/plazasOcupadas:** Envía un mensaje al usuario con el número de plazas ocupadas. Para ello hace una consulta en la base de datos mongodb "registro_estado_plazas". La información se envía también al canal.
- **/plazasLibres:** Envía un mensaje al usuario con el número de plazas libres. Para ello hace una consulta en la base de datos mongodb "registro_estado_plazas". La información se envía también al canal.
- **/temp_hum_Parking:** Envía un mensaje al usuario con la temperatura y humedad relativa medias del parking. Para ello hace una consulta en la base de datos mongodb "registro_estado_plazas". La información se envía también al canal.

Ilustración 14. Flujo Telegram - Nodos 4

- **/frecFOTA:** Proporciona al usuario cuatro respuestas predefinidas a elegir. Estas corresponden con distintas frecuencias de actualización FOTA. Una vez el usuario selecciona una de estas opciones, se envía un mensaje a los topics "II7/+/config", para configurar todas las placas. La información se envía también al canal.

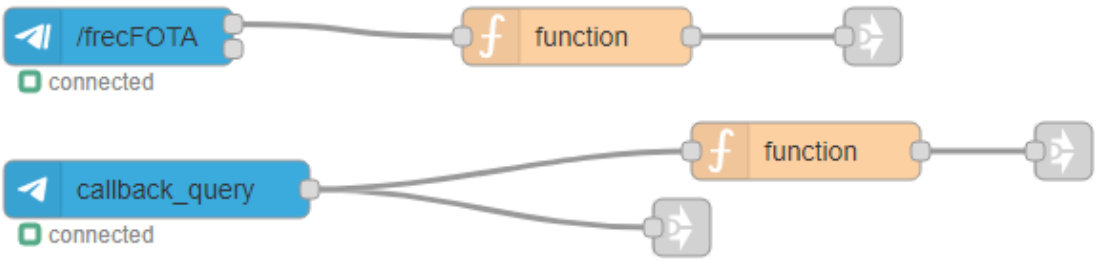

Ilustración 15. Flujo Telegram - Nodos 5

- **/camara_ahora:** Solicita una imagen a la placa ESP32 con cámara y la envía en un mensaje al usuario, junto a un pie de foto. La información se envía también al canal.

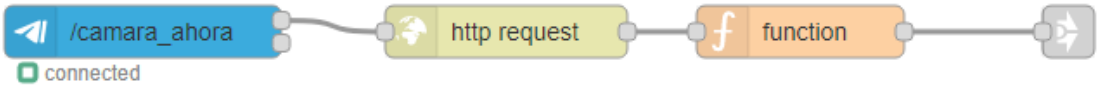

Ilustración 16. Flujo Telegram - Nodos 6

Punto problemático: agregaciones

A la hora de desarrollar el proyecto, en este flujo nos hemos encontrado con bastantes dificultades a la hora de definir las agregaciones necesarias para obtener el número de plazas libres y el número de plazas ocupadas. Lo hemos solucionado de la siguiente manera.

```
var ms_per_min= 60000;
var now_ms= new Date().getTime();
var fiveminsago = new Date(now_ms - 5*ms_per_min);
msg.chatId = msg.payload.chatId;
msg.payload=
[   { "$match": { "date": {"$gt": fiveminsago } }},
    { "$sort": {"date":-1}}, //ordena por fecha
    { "$group": {            //agrupa por
        "_id": "$NumPlaza",
        "Ocupacion":{"$first":"$Ocupado"},
        "Fecha":{"$first":"$date"}
    }},
    { "$match": { "$and": [{ "Ocupacion": true } ] } },
    { "$group": {
        "_id":0,
        "numPlazasLibres":{"$sum":1},
    }},
];
return msg;
```

Ilustración 17. Agregación de número de plazas ocupadas

Este código se trata de la definición de parámetros para la agregación que nos dará como resultado el número de plazas ocupadas. Lo primero que se hace es ordenar los documentos de la base de datos por fecha, siendo los primeros los de fecha más reciente. Luego se hace una agrupación de documentos en función de su idenficador, NumPlaza, para que haya solo un documento por plaza, y a cada documento se le asocia un campo Ocupación, al que se le da el valor del último valor conocido de ocupación de todos los documentos que había con ese identificador. Luego se busca qué documentos tienen el campo Ocupación a true, y calcula el número de documentos encontrados.

Sin embargo, nos podemos encontrar con un problema, y es que si se da el caso en el que todas las plazas están desocupadas, no se encontrará ningún documento con el campo Ocupación a true, y no se podrá realizar la suma. Como resultado, el nodo de la base de datos devolverá un mensaje con el campo payload vacío. Para lidiar con este problema se ha desarrollado el siguiente código, que también es el encargado de construir el mensaje a enviar.

```
var plazas;
if (msg.payload==''){
    plazas=0;
}
else{
    plazas=msg.payload[0].numPlazasOcupadas;
}
msg.plazas=plazas;
msg.payload={};
msg.payload.chatId=msg.chatId;
msg.payload.content="El número de Plazas Ocupadas en el Parking es de  ("+plazas+") plazas.
msg.payload.type = "message";
return msg;
```

Ilustración 18. Construcción del mensaje de número de plazas ocupadas

Como se puede ver, en caso de que el payload que entra al nodo esté vacío, el número de plazas ocupadas se pone a 0 de forma manual. En caso de que no esté vacío, se accede al campo correspondiente para extraer este número. Finalmente se construye el mensaje y se definen el resto de parámetros para el envío de este.

3.9. Flujo Node-RED: Dashboard

Este flujo se encarga de gestionar la interfaz de usuario web. Divide el dashboard en 4 secciones y un menú principal, que permite acceder a estas mediante 4 botones. Por otro lado, existe un botón de vuelta al menú en cada una de estas secciones, las cuales son: entrada, plazas, registros y configuración.

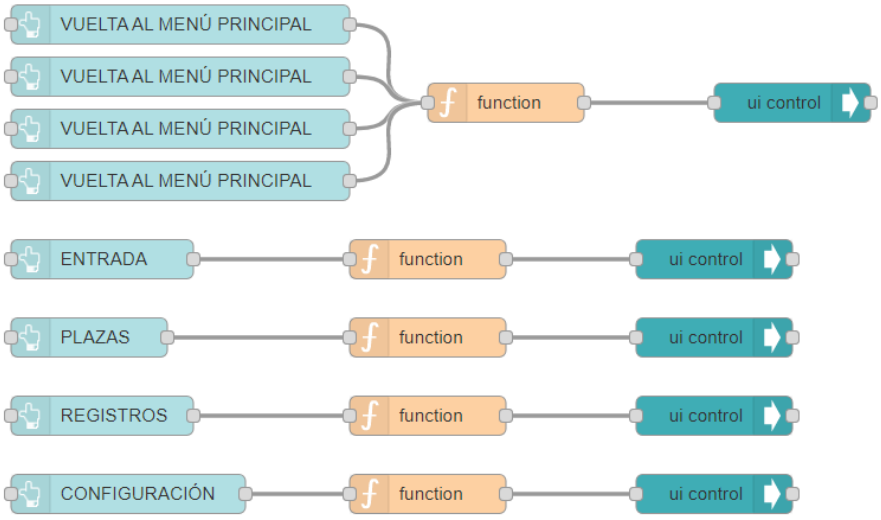

Ilustración 19. Flujo Dashboard - Nodos 1

Entrada

Mediante una búsqueda en la base de datos “registro_barrera_entrada”, obtiene el último estado de la barrera, y muestra una imagen de una barrera subida o bajada en función del resultado.

Permite también controlar la posición de la barrera con dos botones que publican en el topic “II7/Entrada/BarreraCMD”.

Por último, muestra la matrícula y fecha de entrada del último vehículo registrado en la base de datos “registro_matriculas_entrada”.

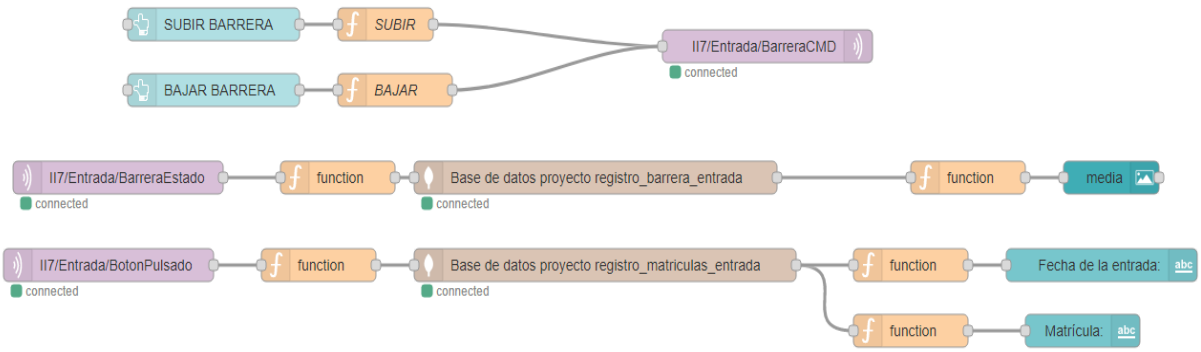

Ilustración 20. Flujo Dashboard - Nodos 2

Plazas

Esta pestaña muestra la información relativa a la placa situada junto a la plaza del parking.

Por un lado, muestra dos gráficos de tipo velocímetro o manómetro y dos gráficas de líneas con las últimas medidas de temperatura y humedad relativa tomadas por el sensor DHT-11. Por otro lado, indica el número de plaza asociado (en este caso solo hay una), y si la plaza se encuentra ocupada o no, mediante un indicador tipo “led”. Esta información se extrae de la base de datos “registro_estado_plazas”.

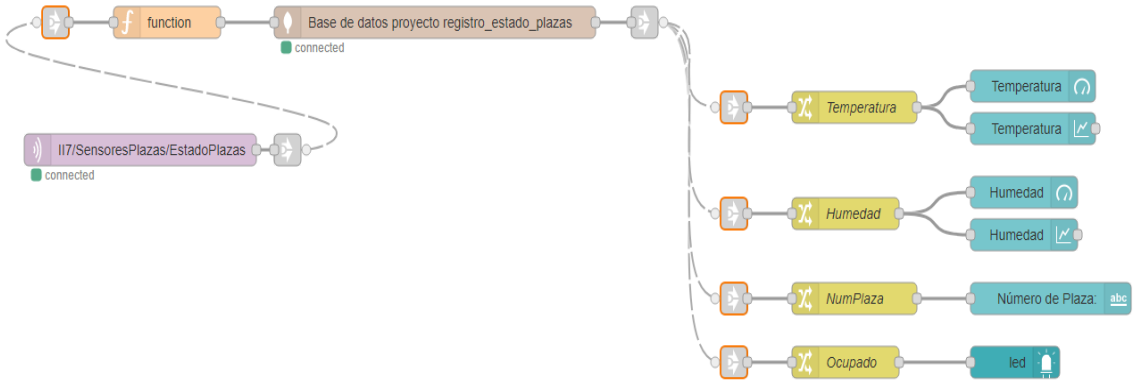

Ilustración 21. Flujo Dashboard - Nodos 3

Registros

Esta pestaña permite observar las entradas de la base de datos “registro_estado_plazas”. Por un lado, se muestran los datos relativos a la última entrada; a su derecha, una lista con todos los registros; y, a la derecha de este, una lista con los últimos datos filtrados por día, semana o mes gracias a tres botones.

También permite la opción de descargar la base de datos en formato csv, eligiendo previamente el rango de fechas y el tipo de separador de decimales (punto o coma) mediante un switch.

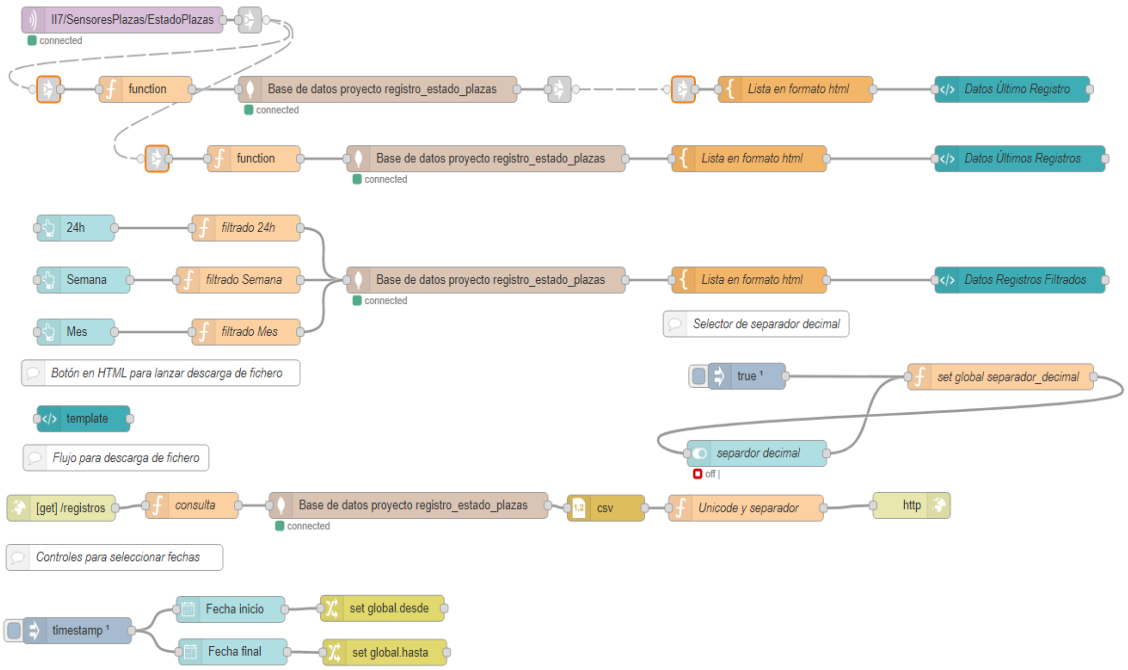

Ilustración 22. Flujo Dashboard - Nodos 4

Configuración

En este apartado se puede configurar el tiempo de escucha de los sensores y la frecuencia de actualización FOTA de las placas mediante dos sliders. Esto se logra mediante el envío de un mensaje en formato JSON a la sección “config” de los topics asociados a cada placa.

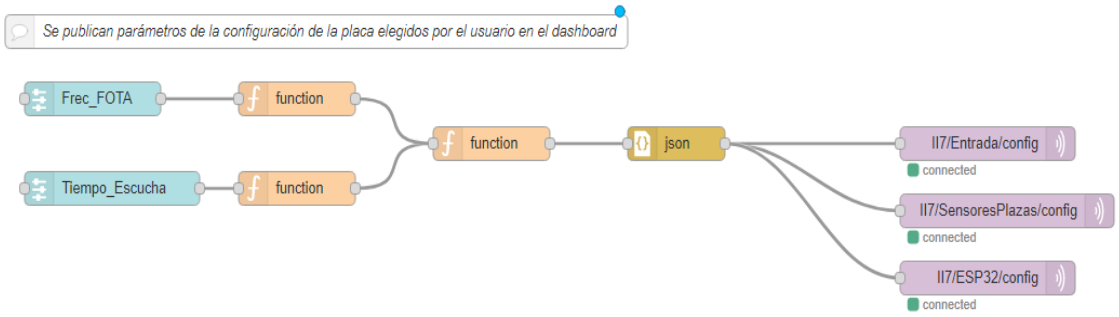

Ilustración 23. Flujo Dashboard - Nodos 5

3.10. Estudio de consumo

El estudio de consumo se ha hecho para dos de las placas. La primera es la ESP32-Cam. El estudio de autonomía para esta placa no es fiable porque, aunque teóricamente el montaje de medición del laboratorio debía alimentar la placa a 5 V, en la práctica, midiendo con un voltímetro, solo se estaba alimentando con 4.2 V la placa, por lo que no era capaz de funcionar correctamente y no llegaba a arrancar el programa de forma adecuada, no respondiendo a las peticiones de captura de imagen, aunque sí se conectaba a la red WiFi. Se intentó solucionar alimentando el montaje con una fuente de alimentación de PC, pero el problema persistió y la alimentación que llegaba a la placa no llegaba a superar los 4.2 V. De todas formas, se midió una corriente de 129 mA.

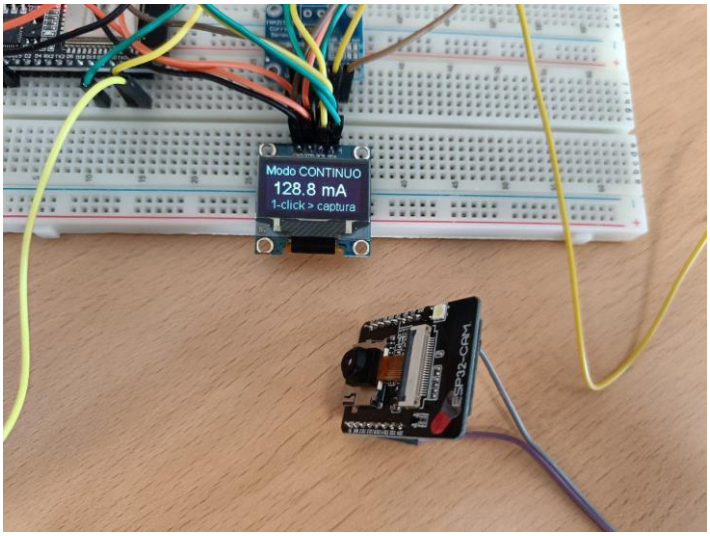

Ilustración 24. Estudio de consumo. ESP32-Cam

La otra placa para la que se realizó la medición de consumo fue la ESP8266 con los sensores de la plaza de aparcamiento (DHT11 y sensor de distancia de ultrasonidos HC-SR04). Esta placa sí arrancó correctamente y estableció la conexión ESP-Now con la placa pasarela. Se observó una corriente requerida de 87 mA.

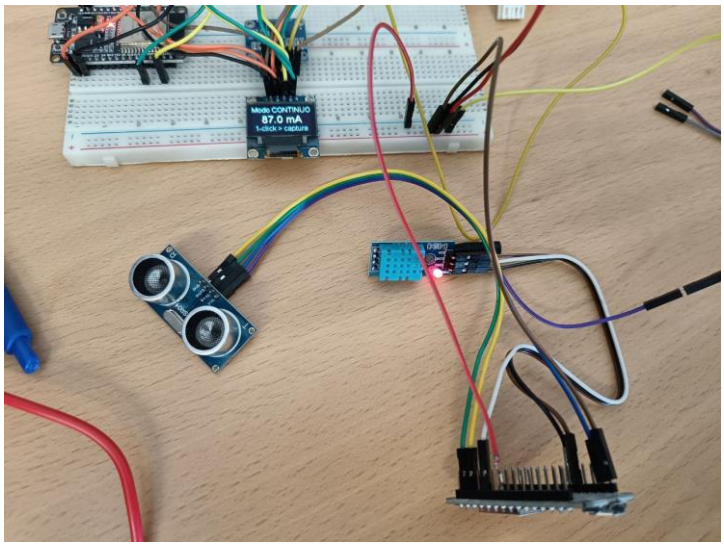

Ilustración 25. Estudio de consumo. ESP8266 con sensores DHT11 y HC-SR04

Al haber planteado como proyecto un parking, con fácil acceso a corriente, no se ha planteado un estudio de duración de batería exhaustivo, pero sí que se puede realizar la siguiente consideración. Si junto a cada placa tuviéramos una batería para emergencias de 1200 mAh, con los datos calculados, la ESP32-Cam podría funcionar con esa batería durante 9.3 h, y la placa de sensores, durante 13.8 h. Estos resultados son más que suficientes para resolver una emergencia en la alimentación normal de las placas. Sin embargo, como ya se ha discutido más arriba, los datos para la ESP32-Cam no son del todo fiables.

4. Resultados y conclusiones

Una vez finalizado el proyecto, se puede concluir que se han cumplido todos los objetivos que se marcaron al plantear el mismo, consiguiendo poner en marcha una prueba de concepto de garaje con dispositivos IoT. Como consecuencia de esto, el grupo ha afianzado los conocimientos obtenidos en la asignatura de Informática Industrial sobre programación de microcontroladores con Arduino, el uso de MQTT para paso de mensajes, el uso de las bases de datos con MongoDB y la interconexión de todos estos componentes con NodeRED. También se ha conseguido hacer una interfaz de usuario, tanto en Telegram como en el Dashboard de NodeRED, que permite abstraer el funcionamiento del sistema a nivel de programación para poder utilizar el proyecto sin preocuparse de entender lo que hay por debajo de la interfaz. Aun así, con la documentación recogida en esta memoria y en los comentarios de los códigos, es perfectamente factible replicar el proyecto y seguir construyendo sobre lo que se ha hecho.

Como objetivos extra, el grupo ha podido profundizar su conocimiento sobre el funcionamiento de ESP-Now, y sobre cómo establecer una pasarela entre esta herramienta y MQTT, lo cual es realmente útil en un proyecto como el nuestro, al tratarse un garaje con muchas plazas normalmente de un recinto bastante extenso.

También se ha aprendido a utilizar sensores nuevos como el sensor de distancia HC-SR04, y actuadores como el servomotor de la barrera de entrada, cuyo funcionamiento se facilita mucho gracias al uso de librerías diseñadas para ello. Además, también se ha aprendido el funcionamiento de APIs que permiten conectar nuestro código con otras aplicaciones que proveen servicio online como platerecognizer.com, cuyo servicio está integrado con NodeRED mediante la librería node-red-contrib-plate-recognizer.

Otro elemento aprendido ha sido el desarrollo del código para una placa distinta a la que veníamos usando durante toda la asignatura, que es la ESP32, aprendiendo sobre las diferentes compatibilidades e incompatibilidades a la hora de usar librerías específicas para cada placa. Esto se pone principalmente de manifiesto en el uso del código para la actualización FOTA, que sirve tanto para la ESP32 como para la ESP8266. Mediante el estudio de este código también se ha aprendido sobre el funcionamiento de las directivas de pre-procesadores que han permitido que el código sea compatible con los dos tipos de microcontrolador sin cambio alguno.

Finalmente, el grupo también ha aprendido a utilizar repositorios mediante Git y Github, lo cual ha ayudado mucho en el proceso de desarrollar un proyecto de código en grupo, en el que se ha intentado desarrollar todo el código de la forma más modular posible, distribuyendo las partes reutilizables en ficheros que luego pudieran usarse en los proyectos para otras placas. Esto se ha hecho por ejemplo con los ficheros de actualización FOTA, que precisamente por ser válidos tanto para ESP32 como ESP8266, se han podido reutilizar para todas las placas que lo necesitaran.

Para acabar, ya que el proyecto se ha realizado teniendo en cuenta todo lo posible la escalabilidad del mismo, nos habría gustado hacer una prueba de esta escalabilidad con más placas para los sensores de cada plaza de garaje, así como con la posibilidad de tener salida con barrera motorizada en el mismo. Esto, en un futuro, nos permitiría probar si se ha planteado bien la escalabilidad del proyecto.

Anexo 1. Lista y descripción de ficheros entregados

- Codigo Arduino
  - CameraWebServer
    - CameraWebServer.ino: código principal de la placa ESP32.
    - actualiza_dual.cpp: código con las funciones necesarias para la actualización FOTA.
    - actualiza_dual.h: código auxiliar de “actualiza_dual.cpp”.
    - app_httpd.cpp, camera_index.h, camera_pins.h: archivos del ejemplo con el que se ha desarrollado el código. Permiten configurar el tipo de cámara y el servidor http.
  - ESPNOW
    - ESPNOW.ino: código principal de la placa ESP8266 que hace de pasarela ESPNOW a MQTT.
    - src/actualiza_dual.cpp y src/actualiza_dual.h : Archivos con el código para realizar la actualización FOTA de ESP32 o ESP8266.
    - src/pulsador_int.cpp: código con las funciones necesarias para usar el botón flash.
    - src/pulsador_int.h: código auxiliar de “pulsador_int.cpp”.
  - SENSORES
    - SENSORES.ino: envía por ESPNOW los datos recogidos a la pasarela.
  - Servo
    - Servo.ino: código principal de la placa ESP8266 con servomotor.
    - actualiza_dual.cpp
    - actualiza_dual.h
    - pulsador_int.cpp
    - pulsador_int.h
- Flows NodeRED
  - Camara_Entrada_MongoDB.json: flujo asociado al control de la barrera, cámara y bases de datos.
  - DASHBOARD.json: flujo asociado a la interfaz de usuario vía web.
  - Telegram_II7.json: flujo asociado a la interfaz de usuario vía Telegram.

Anexo 2. Manual de usuario