ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Diseño, implementación e integración de un sistema embebido de telemetría basado en LoRa

(Design, implementation and integration of an embedded system for telemetry based on LoRa)

Para acceder al Título de

Graduada en

Ingeniería de Tecnologías de Telecomunicación

Autora: Esther Gutiérrez Conde

24 de Julio – 2024

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Esther Gutiérrez Conde

Director del TFG: Roberto Sanz Gil

Título: "Diseño, implementación e integración de un sistema embebido de telemetría basado en LoRa"

Title: "Design, implementation and integration of an embedded system for telemetry based on LoRa "

Presentado a examen el día: 24 de julio de 2024

para acceder al Título de

GRADUADA EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre): Cobo García, Adolfo

Secretario (Apellidos, Nombre): García Gutiérrez, Alberto Eloy

Vocal (Apellidos, Nombre): Lechuga Solaegui, Yolanda

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo: El Presidente

Fdo: El Secretario

Fdo: El Vocal

Fdo: El Director del TFG

(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº

(a asignar por Secretaría)

Agradecimientos

La realización de este Trabajo Fin de Grado no habría sido posible sin la ayuda y el apoyo de varias personas a las que quiero expresar mi agradecimiento.

En primer lugar, quiero agradecer a mi tutor, Roberto Sanz Gil, por su guía, paciencia y apoyo durante todo el proceso de desarrollo del trabajo. Su experiencia y comentarios han sido fundamentales para la realización y mejora continua del proyecto.

A mis padres y hermana, por su amor incondicional y su apoyo a lo largo del transcurso de la carrera como en el desarrollo de este trabajo. Por no dejar que me venga abajo ante los problemas sobrevenidos durante este periodo de mi vida y por recordarme que no debía distraerme, ni dejar las cosas para el último momento, sino que debía de ser constante.

A mis amigos, por su amistad, compañerismo y apoyo. Siempre han estado para apoyarme en los momentos difíciles, pero también por celebrar mis logros.

Y en especial a mis compañeros de clase, por todas las horas de estudio que hemos compartido, las discusiones, el apoyo y ayuda mutua que nos hemos dado durante estos cuatro años en los que hemos pasado de ser conocidos a ser amigos.

Resumen

Este Trabajo de Fin de Grado consiste en la realización del diseño, la implementación y la integración de un sistema de telemetría controlado por dispositivos embebidos. En concreto, se explicará cómo realizar un sistema de detección de movimiento y campo magnético complementado con varios sensores de los que se recabarán un conjunto de datos, como la medición de temperatura del lugar y la toma de decisiones en función de dichos datos.

Para ello, se utilizarán un sensor de detección de movimientos, un sensor de detección de campo magnético y un sensor de temperatura, junto con un LED para su posterior control. Estos dispositivos irán conectados a una placa transmisora LoRa cuyos datos serán enviados a otra placa receptora, mientras que el control del LED se hará a través de WhatsApp empleando una comunicación vía WiFi.

En la placa receptora se instalará un servidor web para mostrar al usuario los datos recibidos del transmisor de una forma más visual mediante un navegador web. Además, se enviarán notificaciones a través de Telegram cada vez que el sensor de movimiento o el sensor de campo magnético se activen.

Abstract

This Final Degree Project consists of carrying out the design, implementation and integration of a telemetry system controlled by embedded devices. Specifically, it will be explained how to create a motion and magnetic field detection system complemented by several sensors from which a set of data will be collected, such as measuring the temperature of the place and making decisions based on said data.

To do this, a motion detection sensor, a magnetic field detection sensor and a temperature sensor will be used, along with an LED for subsequent control. These devices will be connected to a LoRa transmitter board whose data will be sent to another receiving board, while the control of the LED will be done through WhatsApp using communication via WiFi.

A web server will be installed on the receiver board to display the data obtained from the transmitter in a more visual way through a web browser. Additionally, notifications will be sent via Telegram every time the motion sensor or the magnetic field sensor are activated.

Índice de contenidos

Lista de tablas

Lista de Figuras

Lista de acrónimos

AES	Advanced Encryption Standard
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Corrección de Redundancia Cíclica
CSS	Chirp Sleer Spectro
FHDR	Full High Dynamic Range
GUI	Graphical User Interface
IDE	Integrated Development Environment
IoT	Internet of Things
ISM	Industrial, Scientific y Medical
JSON	JavaScript Object Notation
LED	Light Emitting Diode
LoRa	Long Range Radio
LoS	Line of Sight
LPWAN	Low Power Wide Area Network
M2M	Machine to Machine
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
NFC	Near Field Comunication
OLED	Organic Light-Emitting Diode
OTA	Over the Air
PIR	Infrarrojo Pasivo
RAM	Random Access Memory
RF	Radio Frecuencia
RFID	Radio-frecuency Identification
SF	Spreading Factor
SPI	Serial Peripheral Interface
UIT	Unión Internacional de Telecomunicaciones
WiFi	Wireless Fidelity

1

Capítulo 1 Introducción

En este Trabajo de Fin de Grado se presenta el diseño y la implementación de un sistema de telemetría basado en sensores conectados y controlados por sistemas embebidos. El sistema monitorizará datos como detección de movimiento, de campo magnético y la temperatura del ambiente, transmitiendo los datos vía LoRa a una placa receptora y enviándolos a un servidor web. Se implementarán notificaciones por Telegram y control remoto de un LED vía WhatsApp usando WiFi.

1.1 Objetivos del proyecto

El objetivo de este Trabajo de Fin de Grado es realizar el diseño y posterior implementación de un sistema de telemetría basado en sensores conectados y controlados a través de dispositivos embebidos de propósito general, como son los microprocesadores de la familia ESP32, utilizando la tecnología LoRa para comunicarlos entre sí.

La elección de los sensores viene determinada por su funcionalidad, queriendo así monitorizar la detección de movimiento y campo magnético además de la medición de la temperatura del lugar. Estos se conectarán físicamente a la placa TTGO LORA32 ESP32, la cual leerá los datos y los transmitirá a otra placa receptora enviando dicha información a un servidor web y con la posibilidad de obtener notificaciones por Telegram. Además, en el dispositivo transmisor se conectará un LED, que emulará cualquier otro dispositivo sobre el que se desee actuar, el cual será controlado vía WhatsApp.

La comunicación entre los dispositivos será a través de la interfaz inalámbrica LoRa y la comunicación entre las diferentes plataformas como el servidor web, Telegram y WhatsApp, se hará a través de una conexión WiFi.

1.2 Motivación científica

En diferentes ámbitos, la monitorización y el control remoto de dispositivos es una necesidad creciente. Los sistemas tradicionales de telemetría que se basaban en cableado presentan grandes limitaciones tanto de alcance como de flexibilidad y escalabilidad. Debido a esto la tecnología LoRa, al ser inalámbrica, ofrece una gran alternativa que proporciona gran alcance combinado con un bajo consumo de energía.

En este Trabajo de Fin de Grado se propone el diseño e implementación de un sistema de telemetría empleando sensores conectados y controlados a través de sistemas embebidos, como LoRa y ESP32.

La motivación científica de este proyecto es la búsqueda de soluciones innovadoras para la monitorización y control remoto de sensores y otros dispositivos, empleando las ventajas de la tecnología inalámbrica y la IoT, así como de servidores web y plataformas de comunicación, como Telegram y WhatsApp, para la visualización y monitorización de los datos o el control remoto de dispositivos (LED) de forma eficiente y sencilla.

1.3 Motivación personal

En el plano más personal y académico, decidí escoger este tema para el Trabajo de Fin de Grado porque nunca había trabajado con placas como la TTGO LORA32 ESP32 ni con el entorno de Arduino y LoRa, y me parecía interesante aprender a utilizarlas. Creo que el uso de estos dispositivos es algo que se está utilizando bastante en los últimos años junto con el uso de sensores para la transmisión/recepción de información.

Además, tengo especial interés en los sistemas de telemetría que se utilizan en sectores como la automatización industrial, la domótica, la internet de las cosas (IoT) y los sistemas de control ambiental, lo cuales son muy relevantes hoy en día. De hecho, es una tarea interesante diseñar un sistema capaz de monitorizar estos parámetros físicos clave (movimiento, campo magnético y temperatura) y garantizar que el flujo de información se realice de manera efectiva. El uso de LoRa combinado con el microprocesador ESP32 como tecnologías también me ha motivado mucho. La capacidad de comunicación de largo alcance de LoRa con características de bajo consumo de energía más la potencia y la capacidad de procesamiento de ESP32 proporcionan una buena base para descubrir soluciones creativas con fines innovadores, un coste muy bajo y una curva de aprendizaje razonablemente rápida.

Por otro lado, combinar las notificaciones de Telegram con la gestión a través de WhatsApp no sólo añade funcionalidad práctica, sino que lo hace interesante. Agrega valor práctico al esfuerzo y también es un escenario donde se pueden asimilar los métodos de desarrollo de software junto con los protocolos de comunicaciones en la actualidad. Estas son habilidades vitales hoy en día, ya que estamos en un mundo que está más conectado que nunca, donde tener la capacidad de obtener información al instante y luego tomar decisiones al respecto puede tener un gran impacto en muchas situaciones.

En resumen, elegí este proyecto porque las tecnologías y los lenguajes que se usan son muy prácticos para todo el mundo y complementan la formación recibida a lo largo de la carrera. Además, me da la oportunidad de iniciarme en el ámbito de la IoT, que es algo que muchas personas ya tienen en sus casas, automóviles o empresas.

1.4 Estructura del documento

La organización del documento viene dada de la siguiente forma:

- **Capítulo 2**: se describe el estado del arte de la tecnología empleada en este proyecto comentando tanto la historia como su funcionamiento y estructura.
- **Capítulo 3**: se describen los elementos que han formado parte del hardware del sistema.
- **Capítulo 4**: se comentan las herramientas software empleadas, así como la manera de configurar y programar la placa TTGO LORA32 ESP32.
- **Capítulo 5**: se muestra la integración de las diferentes partes del sistema desarrollado junto con sus prestaciones, validación y pruebas de uso.
- **Capítulo 6**: se exponen las conclusiones y líneas futuras que puede ofrecer el sistema de telemetría desarrollado en este Trabajo de Fin de Grado.
- **Anexos**: al final de este documento se agregan varios anexos donde se incluyen los presupuestos junto con los códigos fuente utilizados para las diferentes implementaciones que se llevan a cabo en este trabajo.

2

Capítulo 2 Tecnología LoRa: Estado del arte

La tecnología LoRa ha revolucionado las redes inalámbricas de largo alcance y bajo consumo. Su arquitectura está basada en la modulación CSS que permite una comunicación bidireccional eficiente y un consumo muy bajo de energía. El protocolo LoRaWAN complementa la capa física de LoRa con capas de enlace de datos y aplicación, formando un sistema robusto y seguro para la transmisión de datos. Los dispositivos LoRaWAN se adaptan a diferentes necesidades de comunicación y consumo, mientras que el formato de trama LoRaWAN define la estructura de información intercambiada entre dispositivos y servidores.

2.1 Orígenes de LoRa

La tecnología LoRa fue desarrollada en 2009 por dos ingenieros franceses, Nicolas Sornin y Olivier Seller.

Ambos se propusieron crear una tecnología para transmisión de datos que, mediante el uso de una modulación de baja potencia, pudiera conseguir un largo alcance, de ahí su nombre, LoRa (Long Range) [1].

Inicialmente, el desarrollo de esta tecnología no fue sencillo, ya que se encontraron con bastantes dificultades, entre las que destacan las siguientes:

- **Superación de interferencias:** La banda de frecuencias que empleaban para el desarrollo de LoRa era la ISM de 868 MHz, la cual se encuentra sujeta a interferencias de diferentes fuentes. Para resolver esto, decidieron emplear técnicas de modulación de espectro ensanchado como Chirp Spread Spectrum (CSS), las cuales permiten transmitir la señal en un ancho de banda mayor para hacerla más resistente a las interferencias [1].
- **Uso de protocolos de comunicación eficientes:** Para obtener un máximo rendimiento de LoRa era necesario el diseño de protocolos de comunicación que, junto con técnicas de modulación y esquemas de codificación, optimizaran la transmisión de los datos.
- **Estandarización y adopción de la tecnología:** Para poder conseguir dicha estandarización y adopción era necesario conseguir la colaboración con organizaciones relevantes a nivel internacional [1].

Pero a pesar de esto, siguieron trabajando y consiguieron solventar todos los problemas iniciales.

A principios de 2010, decidieron trabajar junto a un nuevo integrante, François Sforza, y juntos fundaron la empresa que inicialmente denominaron Cycleo.

El principal mercado de negocio de Cycleo era la industria de la medición con el objetivo de añadir capacidades de comunicaciones inalámbricas al agua, así como medidores de gas y electricidad. Cycleo buscaba integrar capacidades de comunicaciones inalámbricas a estos medidores, especialmente a los de agua que se encuentran sumergidos en las redes de abastecimiento [1].

En 2012, la empresa de semiconductores Semtech se dio cuenta de las grandes capacidades de largo alcance y baja potencia que presentaba la tecnología de Cycleo y decidió comenzar a trabajar junto con Nicolas Sornin, Oliver Seller y François Sforza. Su principal objetivo era mejorar todavía más la tecnología de LoRa y finalizar la elaboración del hardware, principalmente los chips necesarios para los dispositivos finales, así como los gateways que permitían la conectividad de los diferentes dispositivos. Pero no solo se centraron en el hardware, también desarrollaron un protocolo de red que permite y controla el acceso al medio [1].

Inicialmente, este protocolo fue denominado LoRaMAC, pero en la actualidad se denomina LoRaWAN y se encarga de especificar los formatos de los paquetes de los mensajes a transmitir, así como de la seguridad para las comunicaciones dentro de la red [1].

Finalmente, en 2015 se fundó LoRa Alliance, una organización que reúne empresas, investigadores y desarrolladores para promover la utilización de LoRa. Actualmente sigue activa y distribuyendo los dispositivos de tecnología LoRa. Los principios de LoRa Alliance son apoyar y

promover el uso global del estándar LoRaWAN asegurando la interoperabilidad de dispositivos y tecnologías de distintos fabricantes [1].

2.2 LoRa

El término IoT (Internet of Things) deriva de un antiguo término denominado Máquina a Máquina. Máquina a Máquina (M2M) es un conjunto de tecnologías de redes cableadas e inalámbricas que permiten el intercambio automático de información entre sistemas sin intervención humana. El IoT es solo una visión más amplia del M2M, donde los dispositivos no sólo provienen del mundo industrial, sino también del uso doméstico.

El mercado IoT sigue aumentando significativamente en todo el mundo. La tecnología LoRa se encuentra dentro de las LPWAN (Low Power Wide Area Network) y cuenta con grandes características y propiedades para proyectos IoT.

El nombre de LoRa es un acrónimo de Long Range y trata del uso de una técnica de modulación de radiofrecuencia patentada por la empresa Semtech.

LoRa se encarga de aportar la capa física a los dispositivos de comunicación inalámbrica que presenta un bajo consumo de energía con un largo alcance de transmisión.

La tecnología LoRa presenta importantes características, entre las cuales cabe destacar su bajo coste, ya que tanto su implementación como los dispositivos presentan una baja inversión, bajo consumo y largo alcance ya que se pueden lograr varios kilómetros en enlaces con línea de vista. Se trata de una tecnología universal ya que emplea bandas de frecuencia sin licencias, que se encuentran disponibles a nivel mundial. Y, además, su transmisión es bidireccional, es decir puede tanto recibir datos como enviarlos [1].

2.3 Rango de frecuencias

LoRa emplea bandas de frecuencias que no necesitan licencias, es decir, no es necesario ni solicitarlas ni pagar por ellas para poder trabajar en dichos canales.

Este rango de frecuencias que no requiere licencias se ubica en las llamadas bandas ISM (Industrial, Scientific y Medical). Estas bandas dedicadas a usos industriales, científicos y médicos están definidas por el Reglamento de Radiocomunicaciones de la UIT, y son bandas reservadas internacionalmente para el uso de energía de radiofrecuencia (RF) para fines distintos de las telecomunicaciones [2].

Dependiendo del lugar del mundo en el que nos encontremos, las frecuencias asignadas a estas bandas cambian. LoRa en Europa emplea las bandas legales de los 433 MHz, 868 MHz o 2.4 GHz, mientras que en Estados Unidos usa la de los 915 MHz. Además, podrían emplear la banda de 2.4 GHz que es la misma que se utiliza para WiFi (para el uso de WiFi 5 se emplea la banda ISM de 5.8 GHz) y Bluetooth.

Dependiendo de la banda de frecuencias empleada varía la potencia de transmisión. LoRa, en la banda de los 868 MHz, puede tener una potencia máxima de 25 mW, mientras que en la banda de 915MHz en Estados Unidos la potencia máxima es de 1000 mW.

En la tabla 2.1 se muestran las tecnologías de comunicaciones asociadas con la banda ISM que utiliza.

BANDA	PROTOCOLOS
13.56 MHz	RFID, NFC
433 MHz	Walkie-talkie, mandos a distancia, LoRa
868 MHz	Sigfox, LoRa, mandos a distancia
2.4 GHz	WiFi, Bluetooth, Zigbee, LoRa
5 GHz	WiFi 5, WiFi 6

Tabla 2.1 Bandas de frecuencias ISM asignadas a tecnologías

2.4 Espectro ensanchado

La modulación LoRa se basa en procesado digital de señal y en el denominado ensanchado del espectro. Este modo de transmisión permite que los dispositivos finales puedan transmitir simultáneamente, en el mismo canal, pero con una estructura de señal especifica (con códigos diferentes) que permite al receptor recuperar la señal frente a niveles de ruido e interferencias muy altos [3].

En la figura 2.1 se puede observar cómo funciona el espectro ensanchado.

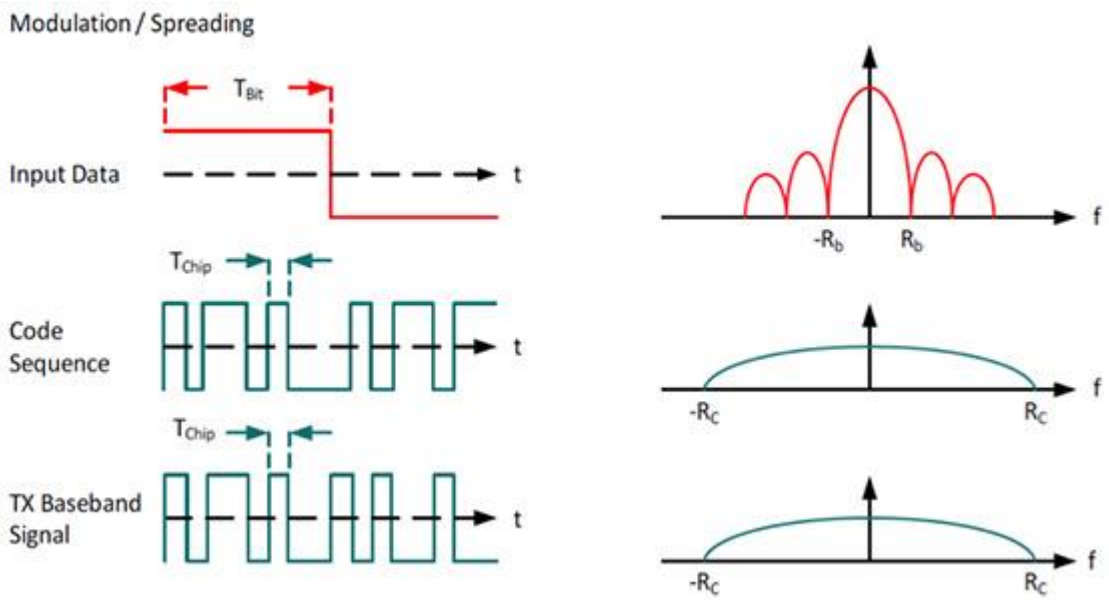

Figura 2.1 Sistema de espectro ensanchado

Este tipo de modulación presenta un parámetro fundamental llamado Factor de Ensanchamiento (Spreading Factor) que consiste en un código de ensanchamiento que define el número de bits usados para codificar un símbolo, que es la relación entre la tasa de símbolos y la tasa de Chirp. LoRa puede utilizar ocho SF (SF5, SF6, SF7, SF8, SF9, SF10, SF11 y SF12). La elección del SF tiene una relación con el rendimiento del enlace:

- **SF más bajo**: Mayor tasa de datos, menor alcance, menor robustez al ruido.
- **SF más alto**: Menor tasa de datos, mayor alcance, mayor robustez al ruido.

Es decir, un SF5 permite realizar transmisión de datos a mayor velocidad que un SF12, pero es mucho más susceptible al ruido en entornos con interferencias.

Junto con el ancho de banda y el tamaño del paquete de datos a transmitir, el Factor de Ensanchamiento determina el rendimiento del enlace respecto a la tasa de datos, alcance y robustez ante el ruido. La elección del SF adecuado es algo importante para poder optimizar el rendimiento de la red LoRa en función de las necesidades específicas que se quieran implementar [3].

En la tabla 2.2 se recopilan las especificaciones en Europa para el SF.

Spreading Factor	Ancho de Banda (kHz)	Tasa de Datos (bps)	Alcance Estimado (km)
12	125	250	14/17
11	125	440	11/14
10	125	980	8/11
9	125	1760	6/8
8	125	3125	5/6
7	125	5470	4/5
7	250	11000	4/5
6	125	22000	3/4
5	250	44000	2/3

Tabla 2.2 Especificaciones en Europa para el SF

2.5 Modulación LoRa

LoRa hace uso de la modulación Chirp Spread Spectrum (CSS) para la transmisión de datos. Sin embargo, en vez de emplear códigos para realizar la modulación emplea “Chirps”, de ahí viene su nombre.

Su objetivo sigue siendo tener varias transmisiones al mismo tiempo en el mismo canal, provocando una expansión del espectro en el ancho de banda utilizado [4].

La modulación CSS consiste en una técnica de modulación de espectro ensanchado que utiliza pulsos “Chirp” modulados en frecuencia lineal de banda ancha para codificar información, de manera que la señal se transmite en ráfagas, saltando entre las frecuencias de una secuencia pseudoaleatoria. Un Chirp es un tono en el cual la frecuencia se incrementa (up-chirp) o decrementa (down-chirp) con el tiempo. Su ancho de banda es equivalente al ancho de banda espectral de la señal [5].

En la figura 2.2 se muestra el funcionamiento de los pulsos “Chirp”.

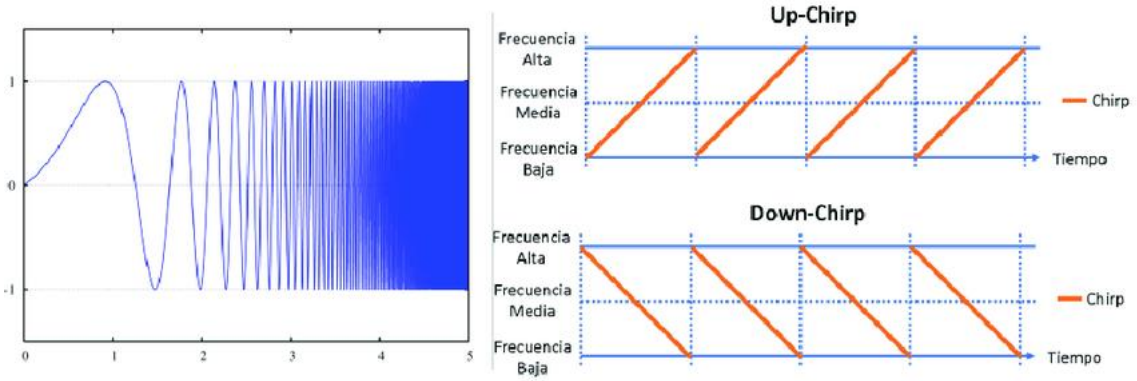

Figura 2.2 Modulación CSS

2.6 Tasa de datos

En LoRa, el tiempo de transmisión de cada símbolo depende del Factor de Ensanchamiento y el ancho de banda.

El Factor de Ensanchamiento especifica la relación entre la velocidad de transmisión y la robustez a las interferencias. Un valor de SF más alto implica un mayor ensanchamiento de la señal, lo que implica una transmisión más lenta, lo que dispersa la señal en un rango más amplio de frecuencias, pero más robusta al ruido y a las interferencias. Sin embargo, un SF más bajo permite una transmisión más rápida, que da lugar a una señal más concentrada en el espectro de frecuencias, pero con menor robustez ante las interferencias.

El ancho de banda define la cantidad de espectro radioeléctrico que se utiliza para la transmisión de datos. Un ancho de banda más grande permite una transmisión de datos más rápida, pero también disminuye la robustez ante las interferencias [4].

La elección del valor de Factor de Ensanchamiento como el del ancho de banda dependerá de las funciones que se quieran implementar y de las características del entorno de aplicación.

2.7 LoRaWAN

El término LoRa se refiere a la capa física de la tecnología LoRaWAN, mientras que LoRaWAN consta de una capa de enlace de datos y otra de red.

La capa física LoRa se encarga de la modulación y demodulación de las señales de radio, además define las características físicas de la comunicación inalámbrica. También se encarga de aportar las herramientas tanto de transmisión como de recepción de datos empleando la técnica de Chirp Spread Spectrum (CSS).

La capa de enlace de datos y de red LoRaWAN se encargan de la gestión de la red, el acceso al medio, la autenticación, el cifrado y el direccionamiento de los dispositivos. Además, proporciona un protocolo completo para la comunicación entre los dispositivos finales [6].

En la figura 2.3 se muestra la combinación de los protocolos LoRa y LoRaWAN.

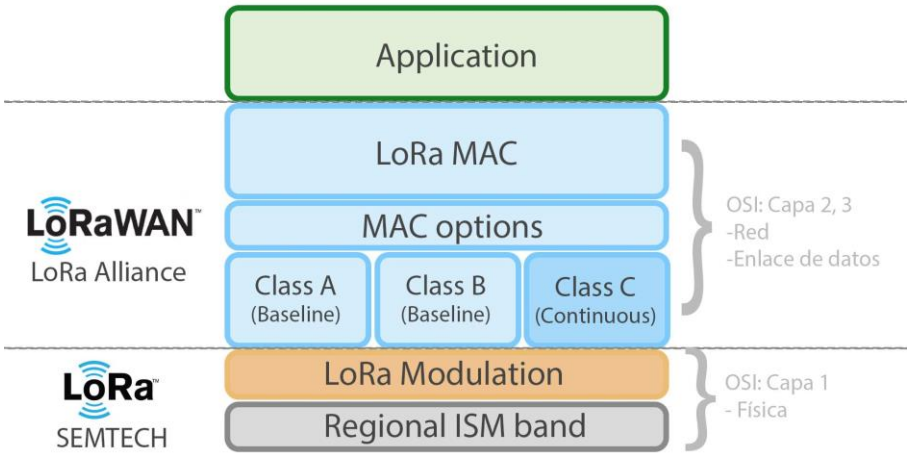

Figura 2.3 Protocolos LoRa y LoRaWAN

El protocolo LoRaWAN es un estándar global que ofrece comunicaciones bidireccionales de alcance de hasta 15 km. Por su parte, cuenta con un consumo de energía realmente bajo, lo que permite un funcionamiento de hasta diez años con la misma batería. Como toda especificación para redes de baja potencia y área amplia que operan en redes de alcance local, regional, nacional o global, esta tecnología se encarga de la trasmisión de pequeños paquetes de datos. Sin lugar a duda, LoRaWAN cuenta con un rol fundamental en el desarrollo de muchos proyectos relacionado con Internet de las Cosas [7].

2.8 Formato de trama LoRaWAN

LoRaWAN se trata del protocolo de capa de enlace y de red que le aporta, a la tecnología de capa física LoRa, una comunicación directa con la capa de aplicación. En la figura 2.4 se muestran las capas del protocolo LoRa.

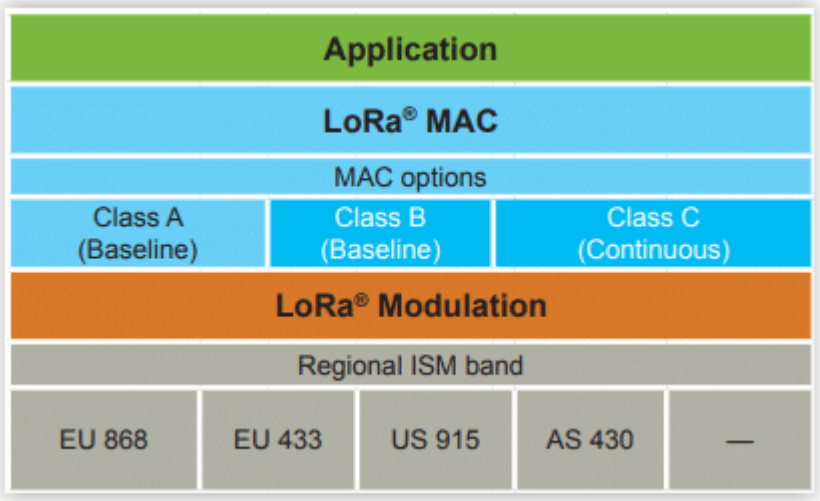

Figura 2.4 Capas de protocolos de LoRa

El formato de la trama de capa física de LoRa tiene la estructura que se muestra en la tabla 2.3.

Preámbulo	FHDR+PHDR_CRC	PHYPAYLOAD	CRC

Tabla 2.3 Trama física de LoRa

- **Preámbulo:** Consiste en una secuencia de 12 bits de longitud que permite que los dispositivos receptores se sincronicen con la señal transmitida, identificando el inicio de la trama y preparando la recepción de los datos posteriores.
- **FHDR+PHDR_CRC:** Se trata de una cabecera y un código de redundancia cíclica de 20 bits para detectar tramas malformadas.
- **PHYPAYLOAD:** Contiene la información que se enviará contenida dentro de una trama de la capa de enlace. El tamaño máximo es de 255 bytes, el cual se indica mediante un byte en la cabecera de la trama.
- **CRC:** Código de redundancia cíclica para detectar si la trama es correcta y puede ser procesada por los dispositivos de LoRa [8].

Respecto a la trama de capa de enlace de datos (MAC), que se encuentra dentro de PHYPAYLOAD, su estructura es la que aparece en la tabla 2.4.

MAC Header	MAC Payload	MIC

Tabla 2.4 Trama MAC de LoRa

- **MAC Header:** Cabecera donde se define el protocolo utilizado y el tipo de mensaje a transmitir. Dentro de esta cabecera se encuentran los siguientes campos:
    - **Flags:** Indica el tipo de mensaje.

    - **DevAddr**: Dirección de 32 bits que identifica el dispositivo final.
    - **FCnt**: Contador de tramas que incrementa con cada mensaje transmitido por el dispositivo final.
    - **FOpts:** Campos opcionales.
    - **MIC:** Código de integridad del mensaje.
- **MAC Payload:** Contiene la trama de la capa de aplicación.
- **MIC:** Como resultado de aplicar la clave de sesión de red (Network Key) a la cabecera y payload obtenemos el MIC que nos dará una capa de integridad con la finalidad de evitar el envío de mensajes desde nodos no autenticados o de suplantar ciertos nodos [8].

Y finalmente, tenemos la trama de capa de aplicación que tiene el formato que se muestra en la tabla 2.5.

Frame Header	Frame Port	Frame Payload

Tabla 2.5 Trama de aplicación de LoRa

- **Frame Header:** Permite identificar la red, contiene un contador de mensajes y la secuencia de estos.
- **Frame Port**: Se trata de un campo opcional para especificar el puerto de destino al que se deben enviar los datos de aplicación.
- **Frame Payload:** La información que transmite la aplicación, pero cifrada usando la calve de sesión de aplicación (App_Skey) usando el algoritmo AES 128 [8].

2.9 Clases de dispositivos

En la tecnología de LoRaWAN los dispositivos se pueden dividir en tres clases.

- **Clase A**: Estos dispositivos presentan mayor ahorro de energía ya que solo entra en modo escucha después de que haya enviado datos hacia el gateway u otro dispositivo para asegurarse de que el mensaje ha sido enviado. En el caso de que no escuche nada significa que el mensaje no ha sido enviado y lo reenvía, pero en otro canal y vuelve al modo de reposo [9].
- **Clase B**: Tienen ventanas de recepción periódicas, independientemente de si el dispositivo tiene que enviar o no mensajes. Dependiendo de los tiempos asignados para las ventanas de recepción, estos dispositivos emplearán baterías si los tiempos no son muy altos, o alimentaciones externas para tiempos más altos [9].
- **Clase C**: Estos dispositivos presentan un alto consumo de energía, ya que siempre se encuentran en modo escucha y solo cuando es necesario se ponen en modo transmisión, de esta forma estos dispositivos pueden recibir información de forma continua [9].

3

Capítulo 3 Hardware

La placa TTGO LORA32 ESP32 cuenta con un microprocesador ESP32, conectividad WiFi, pantalla OLED y módulo LoRa, actuando como centro de control y procesamiento de datos. A esta placa se conectan diversos sensores como el de detección de movimiento (PIR), el KY-001 para la medición de temperatura y el KY-025 para la detección de campo magnético, aportando toda la información necesaria para la monitorización del entorno.

3.1 Módulo TTGO LORA32 ESP32

El dispositivo TTGO LORA32 ESP32 consiste en un microprocesador ESP32 que incorpora un módulo WiFi con conectividad inalámbrica 802.11 b/g/n, un módulo LoRa y una pantalla OLED integrada. Muchas veces es necesario mostrar algún tipo de información al usuario como la de un sensor o el estado de la aplicación, por ejemplo. Para eso se suele cablear externamente una pantalla y eso hace más complejo e incómodo el montaje. La ESP32 OLED combina ambas cosas en una única y sencilla placa: el módulo ESP32 junto con una pantalla OLED. Además, dispone de un conector microUSB tanto para suministrarle alimentación como para programarlo directamente desde el IDE de Arduino.

El módulo LoRa permite transmitir de forma bidireccional datos a grandes distancias. En particular, este módulo es la variante que utiliza la banda de 900 MHz y por lo tanto se puede configurar por software para que opere tanto a 868 MHz como a 915 MHz. Además, incorpora una antena con una ganancia de 2 dBi.

A continuación, se presentan las principales características del módulo LoRa:

- Chip WiFi ESP32 de 80 MHz con conectividad inalámbrica 802.11 b/g/n
- Módulo LoRa 900 MHz (SX1276)
- Compatibilidad con IDE Arduino
- Pantalla OLED de 128x64 pixeles (SSD1306 dirección I2C 0x3C)
- Conexión microUSB tanto para alimentación como para programación vía puerto serie
- Antena de 2dBi con rosca SMA

Respecto a las características previamente mencionadas de la tecnología LoRa, este dispositivo trabaja con un factor de ensanchamiento de SF7, lo cual permite mayores velocidades de transmisión pudiendo llegar a alcanzar una tasa binaria de 5470 bps, aunque la distancia que alcanza sea menor. Además, se trata de un dispositivo de clase C, ya que a no ser que funcione exclusivamente como transmisor se encuentra permanentemente en modo escucha [10].

En la figura 3.1 se muestra la placa TTGO LORA32 ESP32.

Figura 3.1 TTGO LORA32 ESP32

En cuanto al ESP32, se trata de un microprocesador de bajo coste y consumo, que incluye comunicación inalámbrica WiFi y Bluetooth, así como interfaces para conectarse con diferentes periféricos. Se trata de un procesador con dos núcleos de procesamiento cuyas frecuencias operativas pueden controlarse entre 80 MHz y 240 MHz. Los periféricos del procesador permiten

la conexión de diferentes interfaces externas como, por ejemplo, la Interfaz periférica serie (SPI) [11].

El módulo TTGO LORA32 ESP32 utiliza el microprocesador ESP32-D0WDQ6, en la tabla 3.1 se muestran las especificaciones técnicas de este [12].

Interfaces inalámbricas	802.11b/g/n, BlueTooth 4.2
Frecuencia de operación	2.4 GHz
Tasa de datos máxima WiFi	150 Mbps
Potencia de salida WiFi	20 dBm
Sensibilidad WiFi	- 98 dBm
ROM	448 kB
SRAM	520 kB
Tensión suministro min/máx.	2.3 V / 3.6 V
Corriente recepción	100 mA
Corriente transmisión	240 mA
Anchura de bus de datos	32 bit
GPIO	34
Frecuencia máxima de reloj	240 MHz
Peso	97,5 mg

Tabla 3.1 Características técnicas del chip ESP32-D0WDQ6

3.2 Módulo sensor de presencia de movimiento PIR infrarrojo

El detector PIR se trata de un sensor modelo HC-SR501 capaz de detectar movimiento. Normalmente se emplea para detectar si una persona, un animal o un objeto se encuentra dentro o fuera del alcance del sensor.

Se trata de un sensor electrónico que utiliza la luz infrarroja irradiada por los objetos que se encuentran en su campo de visión. Este sensor es capaz de detectar la presencia de alguien a una distancia máxima de 6 metros. Para ello este módulo contiene un sensor piroeléctrico que busca la radiación infrarroja emitida de forma natural por la persona o animal que pasa por delante de su campo de acción y de esta forma se activa advirtiendo de una presencia. También puede ser utilizado con objetos siempre que emitan la suficiente radiación infrarroja.

Este módulo PIR cuenta con dos potenciómetros que permiten regular tanto la sensibilidad de detección como el tiempo entre mediciones [13].

El potenciómetro de sensibilidad regula la distancia a la que el sensor detectará movimiento. Para ajustarlo hay que girar el potenciómetro en sentido horario y, de esta manera, se aumenta la sensibilidad, lo que significa que el sensor detectará movimiento a mayor distancia. Por el contrario, girando el potenciómetro en sentido antihorario se disminuye la sensibilidad, reduciendo la distancia de detección. Si el sensor se emplea para detectar movimiento en una habitación pequeña, se puede ajustar la sensibilidad a un nivel bajo para evitar detecciones falsas provocadas por objetos en movimiento dentro de la habitación. Por otro lado, si se utiliza en un espacio grande, como pueden ser los pasillos de una casa, se puede aumentar la sensibilidad para asegurar una detección precisa [13].

El potenciómetro de tiempo entre mediciones regula el tiempo que la salida del sensor permanece activa después de detectar movimiento. Este parámetro se puede ajustar girando el potenciómetro en sentido horario para aumentar el tiempo entre mediciones o girarlo en sentido antihorario para disminuirlo. Por ejemplo, si empleamos este sensor para encender una luz se puede ajustar el tiempo entre mediciones para que la luz permanezca encendida durante un tiempo antes de que vuelva a realizar la medición y ya no detecte movimiento. Esto resulta de gran utilidad para evitar que la luz se encienda y se apague constantemente debido a breves movimientos [14].

En la figura 3.2 se puede ver el aspecto del detector de movimiento (PIR).

Figura 3.2 Módulo sensor de presencia de movimiento PIR

Las características técnicas del módulo sensor de presencia de movimiento PIR son las que se muestran en la tabla 3.2.

Modelo	HC-SR501
Voltaje de operación	4.5VDC – 20VDC
Consumo de corriente en reposo	<50uA
Voltaje de salida	3.3V (alto) / 0V (bajo)
Rango de detección	3 a 6 metros
Angulo de detección	<100º
Tiempo de retardo	5-200 segundos
Temperatura de trabajo	-20ºC hasta 80ºC
Dimensiones	3.2 cm x 2.4 cm x 1.8 cm

Tabla 3.2 Características técnicas del módulo sensor de presencia de movimiento PIR

Este sensor irá conectado a la placa TTGO LORA32 ESP32 utilizando el esquema que se muestra en la figura 3.3. El pin de datos digital del sensor se conecta al pin GPIO4 de la placa y se alimenta con una tensión de 5 voltios.

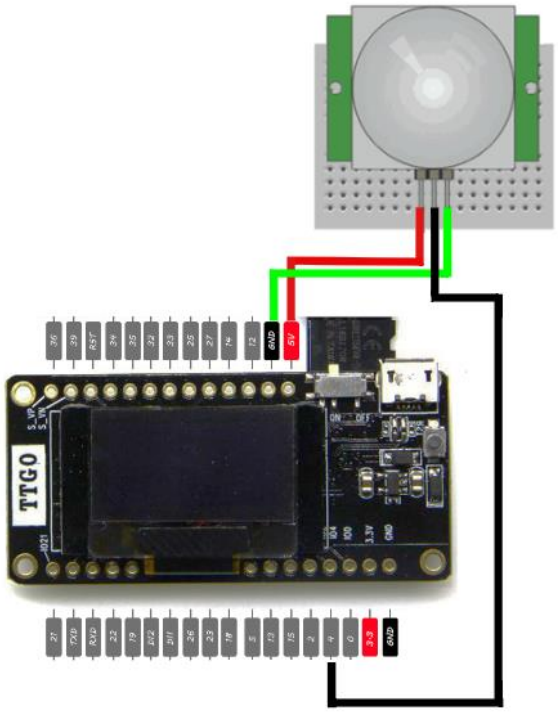

Figura 3.3 Conexión entre el módulo PIR y la placa TTGO LORA32 ESP32

El sensor PIR emite una señal de salida digital que solo indica si ha detectado o no movimiento, no proporciona información sobre la distancia o la intensidad del movimiento. Cuando el sensor detecta movimiento, la salida se pone en nivel alto transmitiendo una tensión de 3.3 voltios. La señal permanece en nivel alto durante un tiempo determinado fijado por el potenciómetro de tiempo entre mediciones. Cuando el sensor no detecta movimiento, la salida se pone en nivel bajo transmitiendo una tensión de 0 voltios. La salida permanecerá en nivel bajo tanto al encender el sensor como cuando no se detecte movimiento [13].

En este Trabajo Fin de Grado se empleará este módulo PIR inicialmente para monitorizar los datos de este a través de un servidor web. Además, se usará para activar notificaciones de alerta a través de Telegram cuando se detecte movimiento.

3.3 Módulo KY-001 sensor digital de temperatura

El módulo KY-001 consiste en un sensor Dallas OneWire DS18B20 que permite la medición de temperatura. La señal que transmite este módulo no es una señal analógica en la que la temperatura se extraiga en función del nivel de señal (amplitud) recibida, sino que se trata de una línea de comunicación OneWire de datos serie digital. Este sensor emplea para la comunicación un bus de un cable que solo requiere una línea de datos y una toma de tierra. Además, también proporciona lecturas de temperatura con resoluciones de 9 a 12 bits, las cuales son configurables, que indican la temperatura del dispositivo con diferentes grados de precisión. La medición de la temperatura se encuentra en unidades de grados Celsius, aunque mediante programación se puede realizar la conversión a grados Fahrenheit o Kelvin [15].

En la figura 3.4 se puede ver como es el sensor de temperatura KY-001 [16].

Figura 3.4 Módulo sensor de temperatura KY-001

Las características técnicas del módulo sensor de temperatura KY-001 son las que se muestran en la tabla 3.3 [16].

Modelo	KY-001 D518B20
Protocolo de comunicación	OneWire
Tensión de funcionamiento	3.0V a 5.5V
Rango de medición de temperatura	-55ºC a 125ºC
Rango de precisión de medición	$\pm$ 0.5 ^{0}C
Dimensiones	18.5 mm x 15 mm

Tabla 3.3 Características técnicas del módulo sensor de temperatura KY-001

Este sensor irá conectado al pin GPIO2 de la placa transmisora TTGO LORA32 ESP32 y al pin de alimentación de 5 voltios, como se muestra en la figura 3.5.

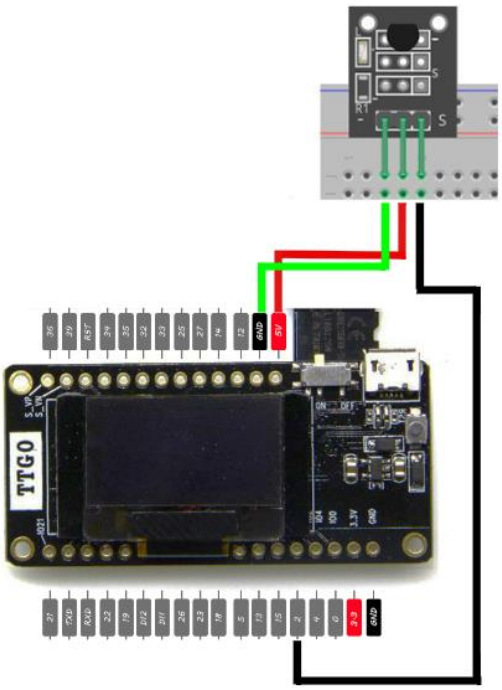

Figura 3.5 Conexión entre módulo KY-001 y la placa TTGO LORA32 ESP32

3.3.1 OneWire

OneWire es un protocolo de comunicaciones en serie, desarrollado por Dallas para interconexión de dispositivos y sensores. La principal diferencia con otros protocolos de comunicaciones en serie es que utiliza un solo cable de datos para transmitir información entre un dispositivo principal y varios dispositivos secundarios. En este proyecto el dispositivo principal es la placa transmisora TTGO LORA ESP32 y los dispositivos secundarios los sensores que cuentan con el protocolo OneWire [17].

Los dispositivos OneWire se conectan a un único bus de datos como se muestra en la figura 3.6, donde el dispositivo principal inicia la comunicación y los dispositivos secundarios le responden. El dispositivo principal controla la comunicación, seleccionando el dispositivo secundario con el que quieren comunicarse. Los dispositivos secundarios responden al principal enviando los datos requeridos. La comunicación entre el dispositivo principal y los secundarios es bidireccional permitiendo que ambos puedan intercambiar información entre ellos. El dispositivo principal puede saber si hay dispositivos secundarios conectados al bus mediante el envío de un pulso de reset enviado por el dispositivo secundario. Para que el dispositivo principal pueda diferenciar a cada dispositivo secundario, estos últimos poseen una dirección de 64 bits única establecida de fábrica, que lo identifica en el bus [17].

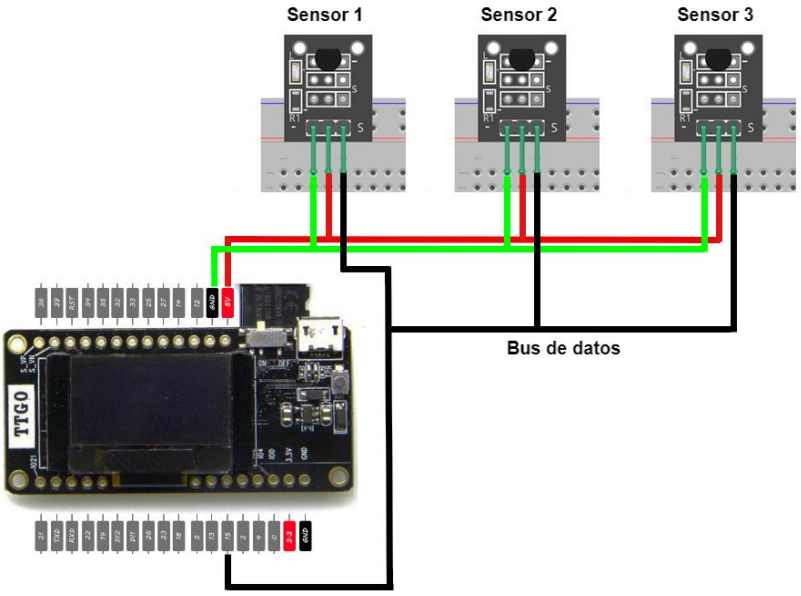

Figura 3.6 Conexión OneWire de varios sensores a un único bus de datos

Los dispositivos secundarios pueden alimentarse de varias formas:

- **Alimentación a través del pin de datos**: Los dispositivos secundarios internamente obtienen energía del pin de datos cuando este se encuentra en un estado alto y almacena carga en un condensador para cuando la línea de datos esté en un estado bajo. A esta forma de obtener energía se le llama Parasite Power y se usa cuando el dispositivo debe conectarse a grandes distancias o en donde el espacio físico es limitado, ya que de esta forma no es necesario la línea de VDD. Tanto el pin GND y VDD del dispositivo deben conectarse a GND como se muestra en la figura 3.7, esto es indispensable para que se active el Parasite Power [17].

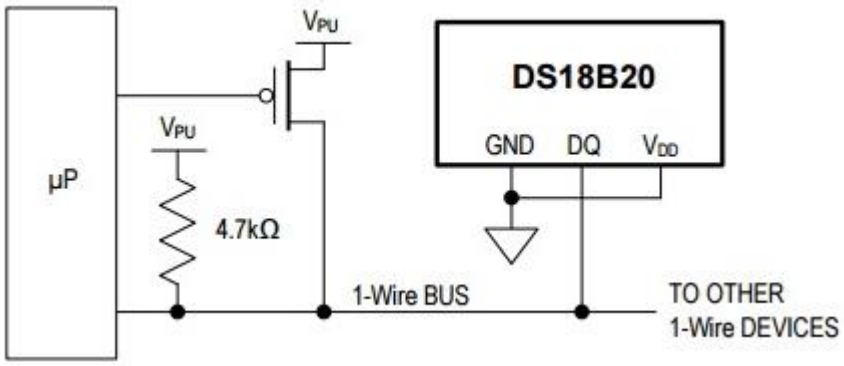

Figura 3.7 Alimentación a través del pin de datos OneWire

- **Alimentación usando una fuente externa**: Los dispositivos secundarios se alimentan a través del pin VDD como se muestra en la figura 3.8, de esta forma el voltaje es estable e independiente del tráfico del bus OneWire. Esta forma de alimentación es la más recomendada y la más empleada [17].

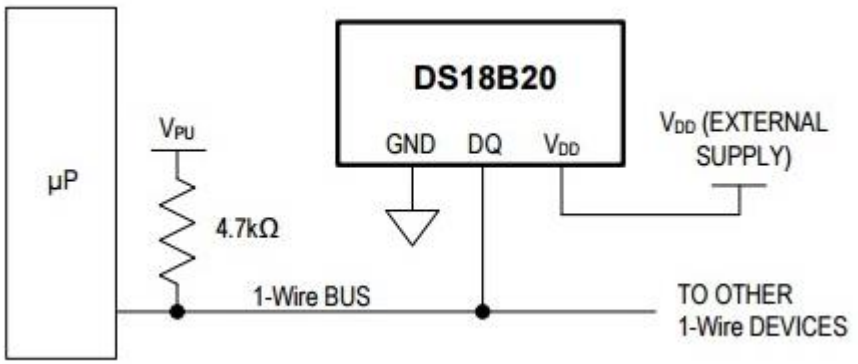

Figura 3.8 Alimentación a través de una fuente externa

Para realizar la lectura de los datos de varios dispositivos secundarios hay dos opciones:

- El primer método consiste en conectar cada sensor a un pin diferente del Arduino, de esta forma si hay 3 sensores, se emplearán 3 pines digitales del dispositivo principal. En este caso, el dispositivo principal empleará el pin al que se encuentran conectados para establecer la comunicación con los diferentes sensores [17].
- Otra forma es usar el mismo pin para todos los sensores, es decir todos los sensores se conectan al mismo bus OneWire donde cada dispositivo posee una identificación o dirección. De esta forma el dispositivo principal se comunicará con los dispositivos secundarios usando dicha dirección única. Para ello, es necesario que previamente se obtengan las direcciones de los diferentes dispositivos secundarios para, posteriormente, utilizarlas ya que estas direcciones vienen asignadas de fábrica [17].

3.4 Módulo KY-025 sensor interruptor por campo magnético

Se trata de un módulo de interruptor magnético que sirve para detectar un campo magnético y actuar en consecuencia. Además, se puede regular la sensibilidad del sensor mediante un potenciómetro. Cuando se detecta una determinada fuerza magnética próxima estará encendido y el módulo emite un nivel de tensión bajo. Cuando no hay fuerza magnética, está en un estado desconectado y emite un nivel de tensión alto. Para que el módulo pueda detectar correctamente el campo magnético es necesario que el imán se encuentre a una distancia menor de 1.5 cm de la lengüeta. Por el contrario, a mayor distancia el dispositivo no detecta el campo magnético.

Presenta dos pines de salida, uno analógico y otro digital. La interfaz digital envía una señal alta cuando se detecta campo magnético, mientras que la interfaz analógica devuelve un valor numérico alto cuando no se detecta campo magnético y bajará cuando se le acerque un imán, ya que el campo magnético del imán induce una corriente en la bobina de la lengüeta, lo que disminuye el voltaje de salida. Cuanto más cerca se encuentre el imán más bajo será el valor analógico. En este trabajo se emplea la salida D0 (digital) del módulo [18].

En la figura 3.9 se muestra el módulo KY-025.

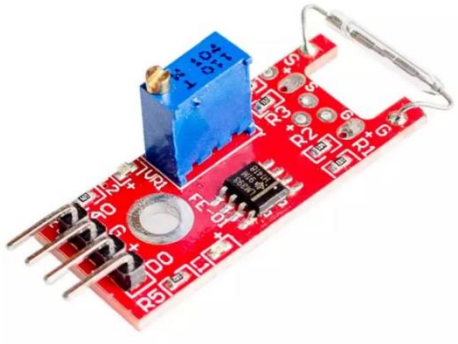

Figura 3.9 Módulo KY-025 sensor interruptor por campo magnético

Este sensor puede tener diferentes aplicaciones prácticas como, por ejemplo, la detección de la apertura de una puerta. Se instala un imán en el marco de la puerta y el sensor en la hoja de esta, de manera que al abrir la puerta el imán se aleja del sensor provocando que no se detecte campo magnético con lo que se podrá controlar la apertura de la puerta y actuar en consecuencia. También se puede emplear para la detección de nivel de líquidos. Instalando en la parte inferior del depósito de líquidos un sensor de campo magnético y dentro del depósito habrá un flotador con un imán. De esta forma si el depósito está lleno el sensor no detectará campo magnético, pero cuando se vacié el flotador bajará hasta la parte inferior donde se encuentra el sensor detectando, de esta manera, la presencia de campo magnético e indicando así un menor nivel de líquido [18].

Las características técnicas del módulo KY-025 sensor interruptor por campo magnético son las que se muestran en la tabla 3.4.

Modelo	KY-025
Voltaje de trabajo	3.3V a 5V
Tamaño del PCB	3.2 cm x 1.4 cm
Peso	3 g
Comparador de salida de corriente	16 mA
Salidas	Analógica y digital
Sensor magnético	De alta sensibilidad

Tabla 3.4 Características técnicas del módulo KY-025 sensor campo magnético

Este sensor irá conectado al pin GPIO15 y al pin de alimentación de 5 voltios de la placa transmisora TTGO LORA32 ESP32, como se muestra en la siguiente figura 3.10.

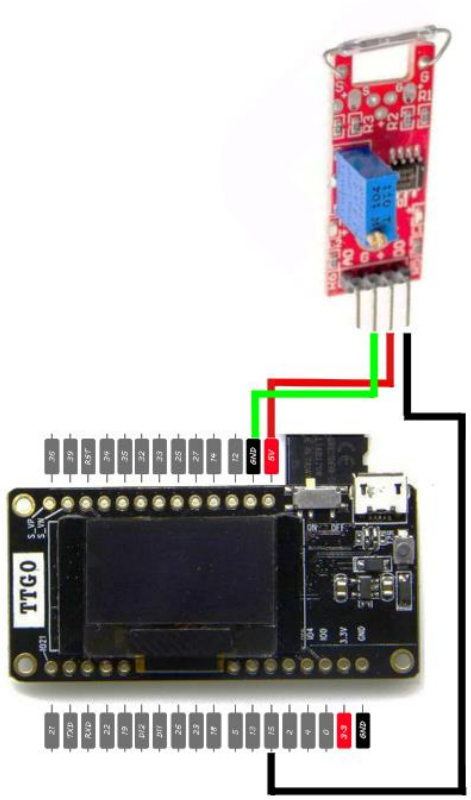

Figura 3.10 Conexión entre el sensor KY-025 y la placa TTGO LORA32 ESP32

4

Capítulo 4 Software

En este capítulo se realiza el desarrollo software de la programación de todos los componentes que conforman el sistema de telemetría final. Se explican las partes más relevantes del código fuente de cada uno: placas LoRa transmisora y receptora, servidor web y su integración con las plataformas de notificaciones y mensajería instantánea.

4.1 Arduino IDE

Para la programación de los dispositivos que vamos a utilizar y poder implementar las funcionalidades que se desean se va a usar el entorno de desarrollo Arduino IDE. Este entorno de desarrollo integrado, llamado IDE (Integrated Development Environment), se trata de un programa informático compuesto por un conjunto de herramientas de programación principalmente un compilador y un linkador, entre otras funcionalidades complementarias. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios, como Arduino, C, C++, Python y JavaScript.

Un IDE es un entorno de programación que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Además, en el caso de Arduino incorpora las herramientas para cargar el programa ya compilado en la memoria flash del hardware. [19]

En la figura 4.1 se muestra la pantalla de inicio de Arduino IDE.

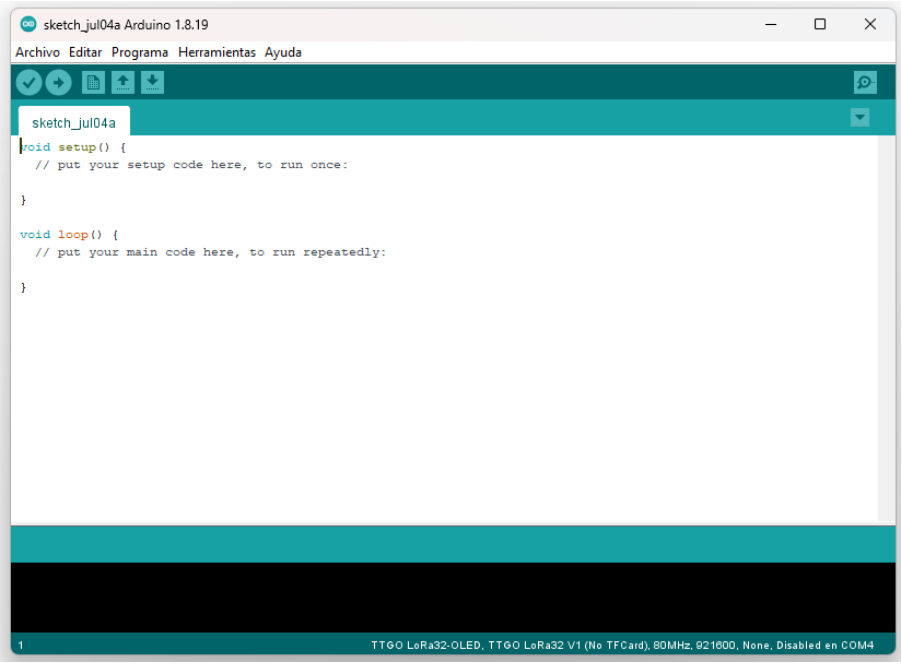

Figura 4.1 Programa Arduino IDE

Arduino IDE es un entorno bastante sencillo de utilizar en el que hay un menú superior con las opciones del IDE, un conjunto de botones de acceso rápido, el área para el editor de código y por último el área de mensajes con la consola.

En este IDE se puede escribir el código para programar las placas y ofrece algunas ayudas como el uso de colores para sintaxis correctamente escrita. Una vez creado el código, este debe ser enviado a la memoria flash del dispositivo final para su ejecución.

El IDE se encargará de validar el código, compilarlo y cargarlo en el microcontrolador del dispositivo simplemente usando uno de los botones de acceso rápido que proporciona la interfaz gráfica del IDE.

En caso de identificar algún error durante el proceso, aparecerá un mensaje en el área inferior dedicado a ello.

Arduino IDE proporciona una utilidad que sirve para monitorizar el puerto serie, esto es, la interfaz de entrada y salida para la placa. Permite introducir datos por dicho puerto, así como mostrar mensajes enviados por la placa.

El lenguaje de programación Arduino se basa en C/C++, y se simplifica con el uso de la biblioteca Arduino que incorpora el IDE. Además, tiene la capacidad de incluir diversas librerías que nos facilitarán la programación de nuestra placa [20].

Dentro del programa de Arduino IDE cabe destacar los menús *Programa* y *Herramientas*. El menú *Programa* ofrece las opciones necesarias para la gestión, compilación y ejecución de los programas en Arduino. Los apartados que presenta el menú *Programa* de la versión Arduino IDE 1.8.19 son los siguientes:

- **Verificar/Compilar**: Compila el código y verifica que no hay errores de sintaxis. Si la compilación es correcta, aparece un mensaje de *compilación finalizada* y, si hay errores, se muestran los errores detectados.
- **Subir**: Carga el código compilado en la placa Arduino conectada al ordenador.
- **Subir usando Programador**: Carga el código compilado en la placa Arduino utilizando un programador externo. Para ello es necesario tener un programador externo compatible con la placa y realizar correctamente la configuración de este.
- **Exportar Binarios Compilados**: Exporta el código compilado en un archivo binario que puede utilizarse para cargar el programa en la placa sin utilizar el IDE.
- **Mostrar carpeta de programa**: Abre la carpeta donde se encuentra el programa guardado.
- **Incluir librería**: Permite agregar una librería externa al proyecto para utilizar sus funciones y variables.
- **Añadir fichero**: Permite agregar un archivo adicional al proyecto, como una imagen, un archivo de texto o de datos.

El menú *Herramientas* aporta opciones para configurar, gestionar y depurar los programas de Arduino. Los apartados que presenta el menú *Herramientas* de la versión Arduino IDE 1.8.19 son los siguientes:

- **Auto formato**: Aplica un formato automático al código fuente del programa.
- **Archivo de programa**: Abre una ventana emergente para seleccionar el archivo del programa que se quiere abrir.
- **Reparar codificación & Recargar**: Analiza el código fuente en busca de errores de sintaxis. Si encuentra errores los muestra en una lista.
- **Administrar Bibliotecas**: Abre el administrado de bibliotecas, donde se pueden instalar, desinstalar, actualizar y gestionar las bibliotecas que se utilizarán en el programa.
- **Monitor Serie**: Abre una ventana de monitor serie, donde se pueden ver los mensajes de depuración y la comunicación que se envían desde el programa a través del puerto serie.
- **Serial plotter**: Muestra una gráfica en tiempo real de los datos recibidos por el puerto serie.

- **WiFi101 / WiFiNINA Fimware/Certificates Updater**: Actualiza el firmware y los certificados de los módulos WiFi101 y WiFiNINA utilizados en algunos programas con conectividad WiFi.
- **Placa**: Permite seleccionar la placa con la que se va a trabajar.
- **Upload Speed**: Selecciona la velocidad binaria de carga del programa a la placa a través del puerto microUSB.
- **Flash Frecuency**: Configura la frecuencia de reloj del microcontrolador de la placa para cargar el programa.
- **Core Debug Level**: Establece el nivel de depuración del lenguaje. Un nivel más alto proporciona más información de depuración, pero puede afectar al rendimiento del programa.
- **Board Versión**: Muestra la versión de la placa conectada. Algunas placas tienen diferentes versiones con características o compatibilidades específicas.
- **Puerto**: Permite seleccionar el puerto serie al que está conectada la placa.
- **Obtén Información de la placa**: Lee y muestra la información sobre la placa conectada, como su modelo, número de serie y versión del firmware.
- **Programador**: Selecciona el programador que se utilizará para cargar el programa en la placa.
- **Quemador Bootloader**: Graba el gestor de arranque en el microcontrolador de la placa, lo que permite ser reconocida por el IDE y cargar programas.

4.2 Programación de los dispositivos LoRa

Para la programación del TTGO LORA32 ESP32 se empleará el entorno de Arduino IDE. Inicialmente se programará el dispositivo para que pueda realizar la lectura de los datos de los sensores que tenga conectados. Además, se programará para que los datos sean transmitidos a otro dispositivo. Para la transmisión se empleará la tecnología LoRa que nos permite una comunicación entre diferentes dispositivos con un alcance de varios kilómetros.

Antes de comenzar con el código, hay que realizar unos pasos previos. Lo primero de todo es instalar la definición de la placa TTGO LORA32 ESP32 para poder compilar el código a desarrollar como se muestra en la figura 4.2, ya que Arduino IDE no la tiene por defecto.

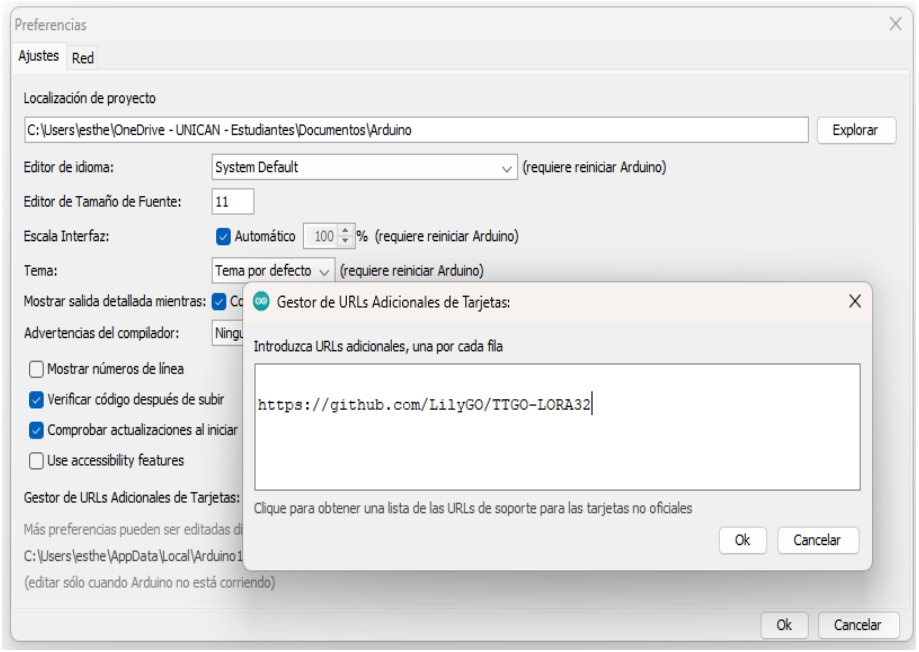

Figura 4.2 Instalación de la definición de la placa TTGO LORA32 ESP32

Además, es necesaria la instalación previa de varias librerías. Para poder usar el protocolo LoRa hay que instalar la librería "LoRa by Sandeep Mistry". En cuanto al uso de la pantalla OLED del dispositivo es necesario la instalación de la librería "Adafruit SSD1306 by Adafruit" como se muestra en la figura 4.3.

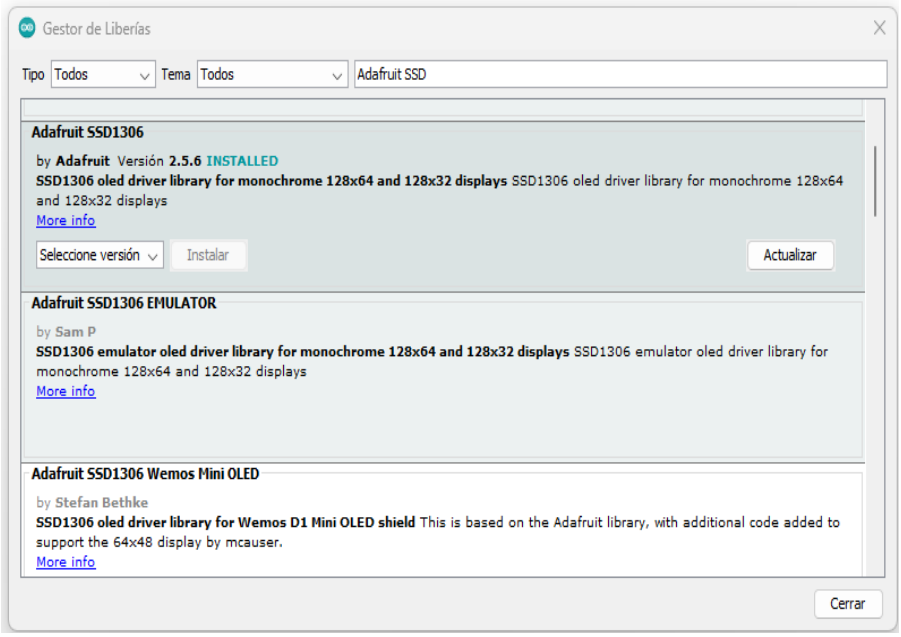

Figura 4.3 Instalación de las librerías

Cuando se entre en el entorno de Arduino IDE hay que comprobar que en Herramientas -> Placa se encuentra seleccionada la TTGO Lora32-OLED. También hay que comprobar que el puerto

serie seleccionado sea el correcto. En este trabajo la placa receptora se encuentra conectada al puerto COM3 y la placa transmisora al puerto COM4 como se muestra en la figura 4.4.

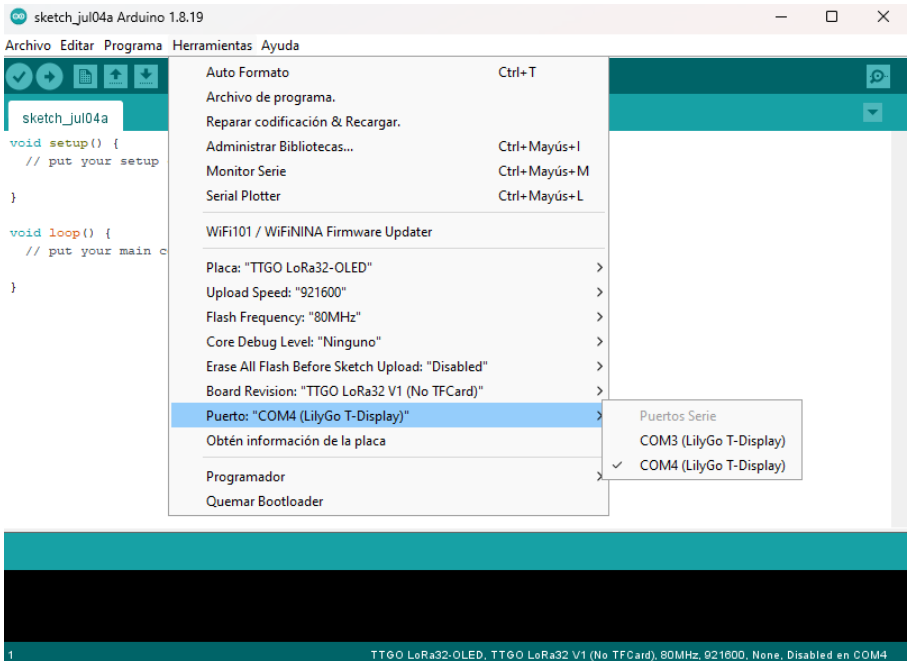

Figura 4.4 Selección del puerto

4.2.1 Dispositivo transmisor

En el Anexo II se muestra el código completo del programa "TransmisorDatosSensor.ino". Este código se estructura en dos funciones principales: *setup* y *loop*. En la función *setup* inicialmente se realiza la inicialización de la comunicación serie a 115200 baudios, lo que permite mostrar mensajes en el monitor serie para posteriormente realizar la depuración y seguimiento del programa, esto se consigue con la siguiente línea;

```
Serial.begin(115200);
```

A continuación, se inicializa la comunicación SPI con los pines SCK, MISO, MOSI y SS, y se configuran los pines de LoRa SS, RST y DIO0, para la comunicación entre Arduino y el módulo LoRa.

```
SPI.begin(SCK, MISO, MOSI, SS);
LoRa.setPins(SS, RST, DIO0);
```

Y por último se realiza la inicialización del módulo LoRa con la frecuencia definida en la constante *BAND* y para verificar la correcta inicialización se mostrará por el monitor serie el estado de la inicialización.

```
if (!LoRa.begin(BAND)) {
  Serial.println("¡Fallo al iniciar LoRa!");
  while (1);
}
```

```
  Serial.println("LoRa inicializado correctamente!");

  }
```

En la función *loop* se comienza realizando la lectura del estado del sensor y se guarda en la variable *SensorState*.

```
  int SensorState = digitalRead(DETECTION_PIN);
```

Finalmente, se procede a la creación y envío del paquete de datos LoRa, el cual contendrá el valor del sensor.

```
  LoRa.beginPacket();
  LoRa.print("Estado sensor: ");
  LoRa.print(SensorState);
  LoRa.endPacket();
```

4.2.2 Dispositivo receptor

En el Anexo III se encuentra el código del programa “ReceptorDatosSensor.ino”. Este programa se estructura en tres funciones principales: *setup, updateLEDs* y *loop*. La función *setup* es igual que la del código anterior a excepción de que se realiza la configuración de los pines a los que están conectados los LEDs como salidas, permitiendo el control de su estado.

```
  pinMode(LED_ACTIVE_PIN, OUTPUT);
  pinMode(LED_INACTIVE_PIN, OUTPUT);
```

En la función *updateLEDs* se comienza apagando los LEDs y a continuación dependiendo del valor del sensor se encenderá el LED correspondiente mientras que el otro se mantendrá apagado.

```
  digitalWrite(LED_ACTIVE_PIN, LOW);
  digitalWrite(LED_INACTIVE_PIN, LOW);
  if (LoRaData == "0") {
    digitalWrite(LED_ACTIVE_PIN, HIGH);
  } else if (LoRaData == "1") {
    digitalWrite(LED_INACTIVE_PIN, HIGH);
  }
```

Y en la función *loop* se empieza comprobando que se han recibido paquetes LoRa, y almacenando el tamaño del paquete LoRa en la variable *packetSize*.

```
  int packetSize = LoRa.parsePacket();
```

A continuación, se realiza la lectura de los datos, si se ha recibido un paquete el tamaño de este será mayor que 0, entonces se lee el contenido del paquete carácter a carácter y se almacenan en la variable *LoRaData*. Una vez leído el paquete completo se llama a la función *updateLEDs* para la actualización del estado de los LEDs.

```
  if (packetSize > 0) {
    while (LoRa.available()) {
      LoRaData = LoRa.readString();
    }
    updateLEDs();
  }
```

En resumen, estos programas definen que cada segundo el transmisor realizará la lectura del sensor de movimiento PIR, localizado en el pin 4 y almacenará el dato en un paquete LoRa que se enviará al receptor cada segundo. El receptor tendrá conectados dos LEDs a sus pines 2 y 4. Este recibirá el paquete LoRa del transmisor y, dependiendo de su valor, encenderá un LED u otro. Si el valor es "1" se activará el LED del pin 4, y si es "0" se activará el LED del pin 2.

4.3 Servidor WEB

Para tener acceso a los datos de los sensores que se encuentran en el dispositivo transmisor, se instalará en el receptor un servidor web donde se almacenen todos los datos de dichos sensores. Para la transmisión de los datos entre el dispositivo transmisor y el receptor se seguirá empleando LoRa.

En cuanto al código del transmisor es necesario realizar cambios en el código del Anexo II, ya que a partir de ahora se van a conectar todos los sensores y hay que almacenar de forma correcta los datos de estos para una correcta transmisión y lectura. Para ello se va a emplear el formato de datos JSON, ya que se trata de un formato de datos ligero y fácil de leer, y además emplea una estructura de pares clave-valor. Esta estructura permite que los elementos que forman parte del documento JSON estén formados por una clave y un valor. La clave es una cadena de caracteres que identifica al valor, el cual es el dato asociado a dicha clave. De esta forma es más fácil identificar los valores de los diferentes sensores. Para poder emplear JSON es necesario que previamente se instale la biblioteca "Arduino_JSON by Arduino".

El código completo para está implementación se encuentra en el Anexo IV. Este código se divide en dos funciones: *setup* y *loop*. Lo primero que hay que implementar es la inicialización de la comunicación OneWire, ya que se va a emplear un sensor de temperatura que emplea está tecnología. Para esto es necesario añadir el siguiente comando, con el que se inicia la comunicación OneWire en el pin definido en la constante *oneWireBus,* en este caso el pin 2.

```
OneWire oneWire(oneWireBus);
```

A continuación, se inicializa la lectura del sensor de temperatura con el siguiente comando:

```
sensors.begin(); //Inicializa la lectura del sensor de temperatura
```

En la función *loop* se realiza la lectura de los datos de los sensores y se guardan en diferentes variables que posteriormente se almacenarán en un documento JSON, el cual facilitará diferenciar los datos de los diferentes sensores cuando los reciba el transmisor. Antes del envío del fichero JSON se convierte en una cadena de caracteres, ya que el protocolo LoRa está diseñado para la transmisión de datos binarios, por lo que es necesario enviar los datos JSON convertidos en una secuencia de bits que el protocolo pueda interpretar y transmitir sin problemas.

```
int detectionState = digitalRead(DETECTION_PINTION_PIN);
int magneticState = digitalRead(MAGNETIC_PIND);
float temperaturaState = sensors.getTempCByIndex(0);

SensorData sensor1Data = {detectionState, magneticState, temperaturaState};

StaticJsonDocument<128> doc1;
doc1["data1"] = sensor1Data.data1;
```

```
doc1["data2"] = sensor1Data.data2;
doc1["data3"] = sensor1Data.data3;

char jsonBuffer1[128];
serializeJson(doc1, jsonBuffer1);
```

En el código del receptor se tiene que implementar el servidor web. Para ello es necesario instalar la librería “ESPAsyncWebServer.h” para crear un servidor web en el ESP32 del dispositivo TTGO LORA32 ESP32. Además, hay que añadir la biblioteca “WiFi.h”, ya que el acceso al servidor web se realizará vía WiFi.

```
WiFi.mode(WIFI_STA);
WiFi.begin("iPhone de Esther", "**********");
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
}
```

También se hará uso de la pantalla OLED, en la que se mostrará la dirección IP del servidor web donde se encontrarán todos los datos de los sensores. Para ello se incluyen las bibliotecas “Adafruit_GFX.h” y “Adafruit_SSD1306.h” las cuales permiten el uso de dicha pantalla.

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0, 0);
display.println("Dirección IP WiFi:");
display.println(WiFi.localIP());
display.display();
```

A continuación, se inicializa el servidor web y se definen los manejadores de eventos, que se originarán cuando se reciban los datos de los sensores enviados por el transmisor, y enviará la página HTML almacenada en la variable *index_html* al cliente web con el código de estado HTTP 200.

```
server.addHandler(&events);
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html);
});
```

Y por último, en la función *loop* se extraen los datos del paquete LoRa recibido y se convierten a cadena de texto para posteriormente enviarlos a la página web y mostrarlos en los elementos HTML correspondientes a través de la siguiente línea de código:

```
String eventData = "{\"data1\": " + String(data1) + ", \"data2\": " +
                   String(data2) + ", \"data3\": " + String(data3) + "}";
events.send(eventData.c_str(), "new_data", millis());
```

El código fuente completo empleado para el receptor es el que se muestra en el Anexo V.

4.4 Notificaciones vía Telegram

De cara al usuario, se ha implementado el envío de notificaciones cuando los valores de los sensores sean modificados. Es decir, si el valor del sensor de presencia de movimiento PIR infrarrojo o el sensor interruptor por campo magnético es 1 (existe la presencia de los datos que miden), el dispositivo TTGO LORA32 ESP32 receptor enviara un mensaje a través de la aplicación de Telegram informando al usuario de la detección de movimiento o de campo magnético.

Para esta implementación solo hay que modificar el código del dispositivo receptor. En este caso necesitaremos el uso de la biblioteca "HTTPClient.h" para poder realizar solicitudes HTTP en el dispositivo TTGO LORA32 ESP32 receptor. El envío de los mensajes se hará haciendo uso de una solicitud HTTP GET de la URL de Telegram para enviar mensajes a través de su API. Para ello, en dicha URL hay que incluir el token del bot, el ID del chat de Telegram que se va a usar para la recepción de los datos y el mensaje a transmitir.

Para poder obtener tanto el token del bot como el ID del chat de Telegram es necesario realizar los siguientes pasos. Primero se debe acceder en Telegram al chat del BotFather al cual se le escribe el comando /token y a continuación el nombre del bot del que se quiere conocer su token como se muestra en la figura 4.5.

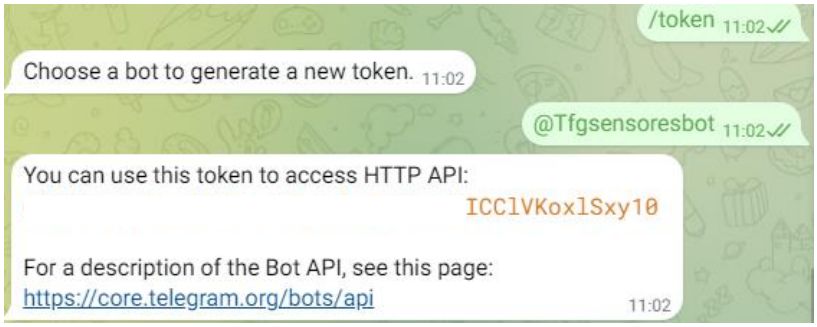

Figura 4.5 Obtención del token del bot de Telegram a través del bot BotFather

Respecto al chat ID de Telegram se tiene que entrar en Telegram al chat de userinfobot, en él se debe de escribir el comando /start, como se muestra en la figura 4.6, y la respuesta será tanto el chat ID como otra información que en este caso no es de interés.

Figura 4.6 Obtención del chat ID de Telegram a través del bot userinfobot

El código completo del receptor se encuentra en el Anexo VI. Inicialmente es necesario instalar la librería *HTTPClient.h* para poder realizar solicitudes HTTP a la API de Telegram para realizar el envío de notificaciones.

```
#include <HTTPClient.h>
```

Cabe destacar que en la función loop hay que comprobar la correcta conexión WiFi. Si se encuentra correctamente conectado a la red WiFi se crea un objeto *HTTPClient* para realizar las peticiones HTTP a la API de Telegram. A continuación, se comprobarán los datos recibidos de los sensores de movimiento y campo magnético. En caso de que el valor de estos sea igual a 1, es decir que se haya detectado movimiento o de campo magnético, se crea un mensaje informativo y se procede a la elaboración de la URL de la API de Telegram, ya que es necesario crear una URL específica para enviar el mensaje al chat de Telegram. Una vez se tiene la URL se envía una solicitud HTTP GET a la URL que realizará el envío del mensaje a Telegram.

```
void loop() {

  if (WiFi.status() == WL_CONNECTED) {

    HTTPClient http;

    if(data1 == "1"){

      mensaje1 = "Se ha detectado movimiento";
      String url1 =
"https://api.telegram.org/***/sendMessage?chat_id=***&text=" + mensaje1;
      http.begin(url1);
      http.GET();
      http.end();

    }

    if(data2 == "1"){

      mensaje2 = "Se ha detectado campo magnético";
      String url2 =
"https://api.telegram.org/***/sendMessage?chat_id=***&text=" + mensaje2;
      http.begin(url2);
      http.GET();
      http.end();

    }
  }
}
```

4.5 Interacción vía WhatsApp

Para aportar más interacción por parte del usuario se ha implementado la funcionalidad de encender/apagar un LED situado en el dispositivo TTGO LORA32 ESP32 transmisor. Para ello solo hay que modificar el código del dispositivo transmisor. En este Trabajo de Fin de Grado se ha decidido emplear un LED, pero se podría emplear cualquier otro elemento tanto de baja tensión como de alta, para estos últimos sería necesario el uso de relés.

En esta implementación para controlar un LED a través de WhatsApp será necesario emplear las plataformas de Twilio y ThingESP, que nos permiten la comunicación entre el TTGO LORA32 ESP32 transmisor y la plataforma de mensajería instantánea. Twilio es un servicio de mensajería que mediante la integración de su API permite enviar y recibir mensajes de WhatsApp. Pero para ello es necesario la implementación de ThingESP ya que se comportará como el intermediario que realiza la configuración y la gestión de la comunicación entre el TTGO LORA32 ESP32 transmisor y Twilio. Es decir, como se refleja en la figura 4.7, Twilio se empleará como la interfaz con el usuario mediante WhatsApp y ThingESP permite la comunicación entre nuestro transmisor y Twilio para el control del LED.

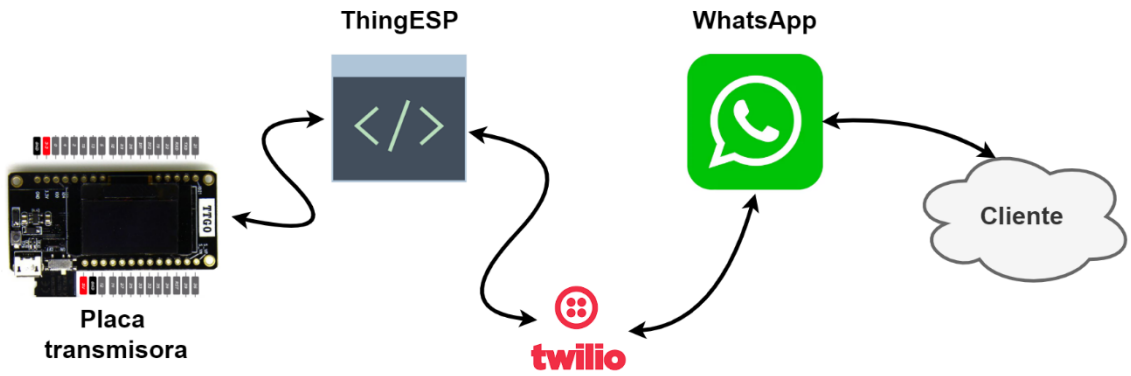

Figura 4.7 Interfaces para el control de un LED vía WhatsApp

A continuación, se va a explicar el desarrollo que se ha llevado a cabo para el control de un LED usando WhatsApp mediante el microcontrolador ESP32.

Lo primero que hay que hacer es acceder a la página de ThingESP y crear una cuenta de sesión. Una vez creada hay que ir a la pestaña de *Projects* y una vez dentro hay que darle a *Add New Project.* Aquí se deben introducir tanto el nombre del proyecto, que en este caso será *Prueba*, como las credenciales del proyecto, que en este caso se ha puesto *Control* como se muestra en la figura 4.8, estos datos son solo descriptivos, pero más adelante se emplearán para realizar la configuración del control del LED.

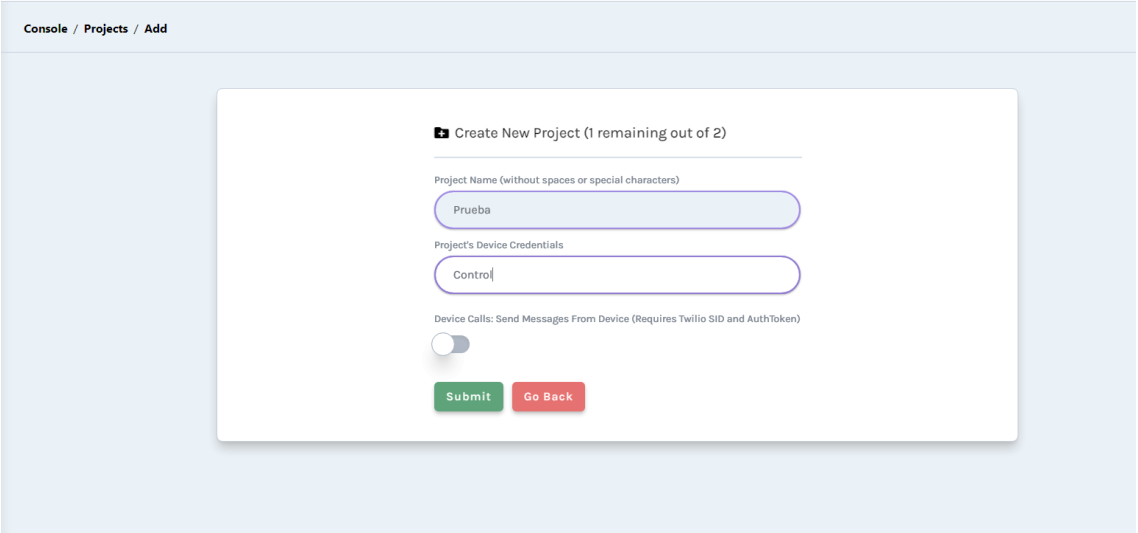

Figura 4.8 Como crear un nuevo proyecto en ThingESP

Una vez introducidos los datos se guardan y ya se tendría el proyecto creado. Una vez dentro del proyecto se puede observar que todo está fuera de línea y sin ninguna dirección IP asignada como se muestra en la figura 4.9. Dentro del proyecto se debe guardar la URL del Endpoint para Twilio ya que será necesaria posteriormente para vincular la plataforma Twilio con la de ThingESP.

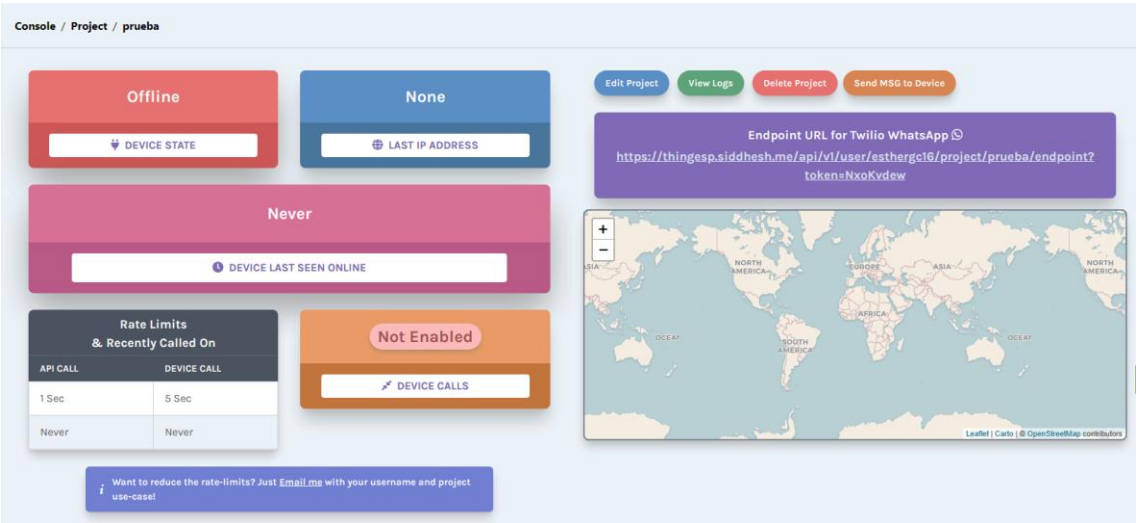

Figura 4.9 Información inicial sobre el proyecto de Twilio

El siguiente paso es acceder a la página de Twilio y crear una cuenta de sesión. Una vez dentro de la plataforma hay que ir a *Create New Account*, donde se pide introducir un nombre, aquí da igual que se ponga, en este caso se ha puesto *Sensores*. A continuación, hay que realizar la verificación del número de teléfono que se va a emplear para el control del LED, se debe introducir el número y una vez hecho enviarán un código de verificación y con esto se validará el número de teléfono. Después aparecerá el panel de configuración que se muestra en la figura

4.10. En el campo donde se pregunta el producto de Twilio que se va a usar hay que poner *WhatsApp*, en el apartado de para qué vamos a utilizar Twilio hay que seleccionar la opción de *Other*, en el apartado de descripción de la organización se pondrá la opción de *Hobbyist or Student,* en cómo se quiere trabajar con Twilio se elegirá la opción de *With no code at all* y finalmente en la opción de cuál es la meta se pondrá la opción de *Something else*. Una vez completado se da al botón de *Get Started with Twilio* y de esta forma ya está creada la cuenta (Account) para el control del LED.

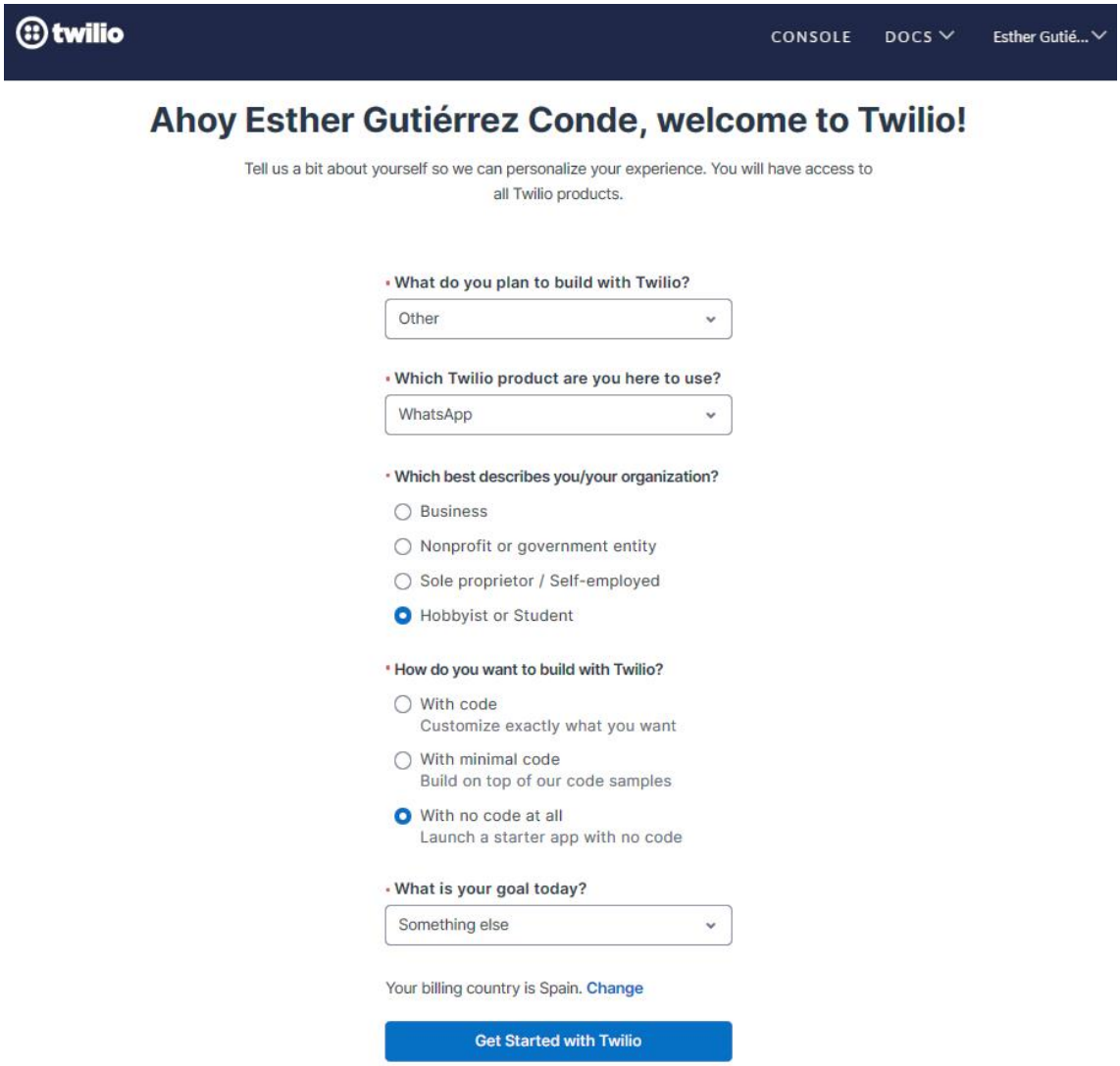

Figura 4.10 Formulario para la creación del proyecto Twilio

A continuación, hay que ir a la pestaña *Messaging > Try it out > Send a WhatsApp message,* una vez dentro se debe guardar el número de teléfono, ya que será con el que se realizará la comunicación a través de WhatsApp para el control del LED, y el código que nos proporcionan como podemos ver en la figura 4.11.

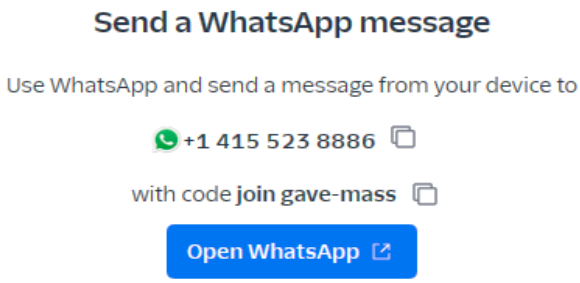

Figura 4.11 Teléfono de Twilio y código de verificación

Una vez se ha guardado el teléfono proporcionado como un contacto nuevo, se debe acceder al chat de WhatsApp de dicho número y enviar el código anterior, de esta forma Twilio responderá con un mensaje donde se indica que la comunicación ya está establecida y se pueden realizar el intercambio de mensajes desde WhatsApp. A continuación, en la plataforma de Twilio se deben de guardar los tokens de autenticación que se muestran en la figura 4.12. Para obtener los tokens hay que acceder a la página de inicio de nuestro Account y en la pestaña *Account Info* se muestran los tokens que usaremos posteriormente para la configuración del control del LED.

Figura 4.12 Token de autenticación de Twilio

Y para finalizar con la configuración de Twilio hay que acceder a *Sandbox settings* y en el campo *When a message comes in* se debe de poner la URL del Endpoint de Twilio, que anteriormente habíamos obtenido en la página de ThingESP, y en el campo *Method* hay que poner POST como se muestra en la figura 4.13. Una vez hecho todo esto ya estaría finalizada la configuración de Twilio.

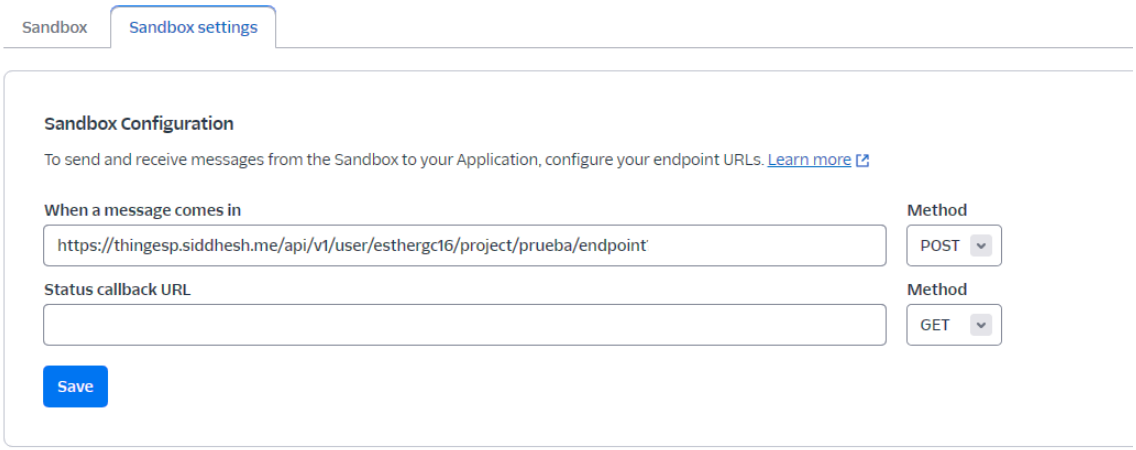

Figura 4.13 Configuración del Sandbox de Twilio

Seguidamente hay que volver a la página de ThingESP, donde se accede a *Edit Project* y se activa la opción *Device Calls*, en el campo de *Twilio SID* se debe de introducir el token de *account_sid y* en el campo *Twilio AuthToken h*ay que meter el *auth_token* que se obtuvieron anteriormente durante la configuración de Twilio como se muestra en la figura 4.14.

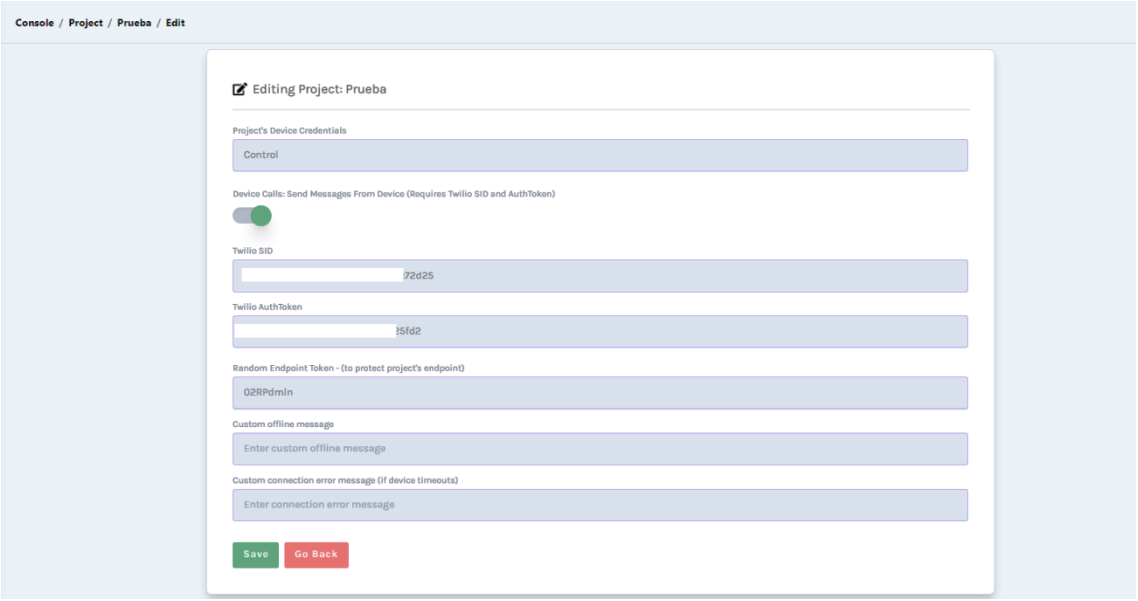

Figura 4.14 Vinculación de ThingESP y Twilio

Una vez realizados todos los pasos anteriores solo desarrollar realizar el código que implementará el control del LED a través de WhatsApp.
El código que hay que implementar es el que se muestra a continuación:

```
#include <WiFi.h>
#include <ThingESP.h>
ThingESP32 thing("esthergc16", "Prueba", "Control");
```

```
4
5       int LED = 21;
6
7       unsigned long previousMillis = 0;
8       const long INTERVAL = 6000;
9
10      void setup()
11      {
12              Serial.begin(115200);
13              pinMode(LED, OUTPUT);
14              thing.SetWiFi("iPhone de Esther", "***********");
15              thing.initDevice();
16      }
17
18      String HandleResponse(String consulta)
19      {
20              if (consulta == "prendido") {
21                    digitalWrite(LED, 1);
22                    return "REALIZADO LUZ ENCENDIDA";
23              }
24
25              else if (consulta == "apagado") {
26                    digitalWrite(LED, 0);
27                    return "REALIZADO LUZ APAGADA";
28              }
29
30              else if (consulta == "estatus")
31                    return digitalRead(LED)?"ledencendida":"ledapagada";
32              else return "invalido";
33      }
34
35      void loop()
36      {
37              thing.Handle();
38      }
```

Inicialmente hay que añadir las bibliotecas de WiFi y la de ThingESP. En la línea 3 se conecta el microcontrolador ESP32 con la plataforma de ThingESP, para ello hay que introducir el usuario, este dato lo encontramos en la URL del Endpoint de Twilio como se muestra en la figura 4.15, el nombre y las credenciales del proyecto que se muestran en la figura 4.8.

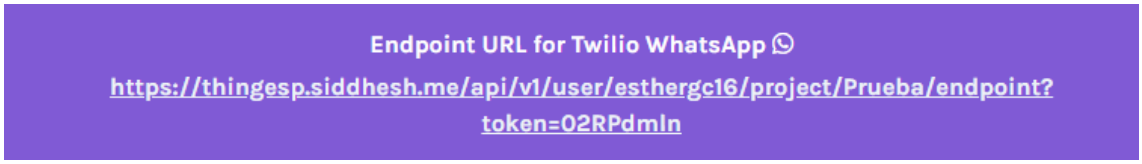

Figura 4.15 URL del Endpoint de Twilio

En la línea 14 se realiza la conexión del ESP32 a la red WiFi, para ello hay que introducir tanto el SSID como la contraseña de la red WiFi. En la línea 18 se encuentra la función *HandleResponse* cuya funcionalidad es gestionar la respuesta de la llamada, es decir, si la llamada se realiza correctamente analiza la respuesta que se va a enviar. Por ejemplo, si tenemos configurado que al enviar por WhatsApp la palabra *Hola* Twilio responda con otro *Adiós* esta función lo que hace es comprobar que se ha enviado la palabra *Hola* y a continuación mirar qué es lo que debe hacer. En este caso, comprobará si el usuario ha mandado “prendido”, “apagado” o “estatus”, para encender, apagar o comprobar el estado del LED. Y por último, en la línea 37 se encuentra el

thing.Handle que se encarga de realizar la llamada al cliente para iniciar la transferencia de datos a través de WiFi.

El código completo empleado para está implementación se muestra en el Anexo VII.

5

Capítulo 5 Integración y validación del sistema

En este capítulo, se presenta el diseño final del sistema de telemetría y las prestaciones que ofrece al usuario. Desatacando su flexibilidad ya que los sensores pueden ser ubicados en cualquier sitio, su facilidad de uso y su accesibilidad al poder acceder a los datos de los sensores a través del servidor web, recibir notificaciones vía Telegram e interactuar con el sistema a través de la plataforma WhatsApp.

5.1 Integración de los componentes

Aunque en este proyecto se han colocado todos los sensores en la placa TTGO LORA32 ESP32 transmisora, como se puede ver en la figura 5.1, el usuario podrá colocar cada sensor donde quiera simplemente usando cables con la longitud suficiente para llegar a la localización donde quiera instalarlos. Se podría conectar alguno en la placa receptora modificando el código fuente de las placas LoRa.

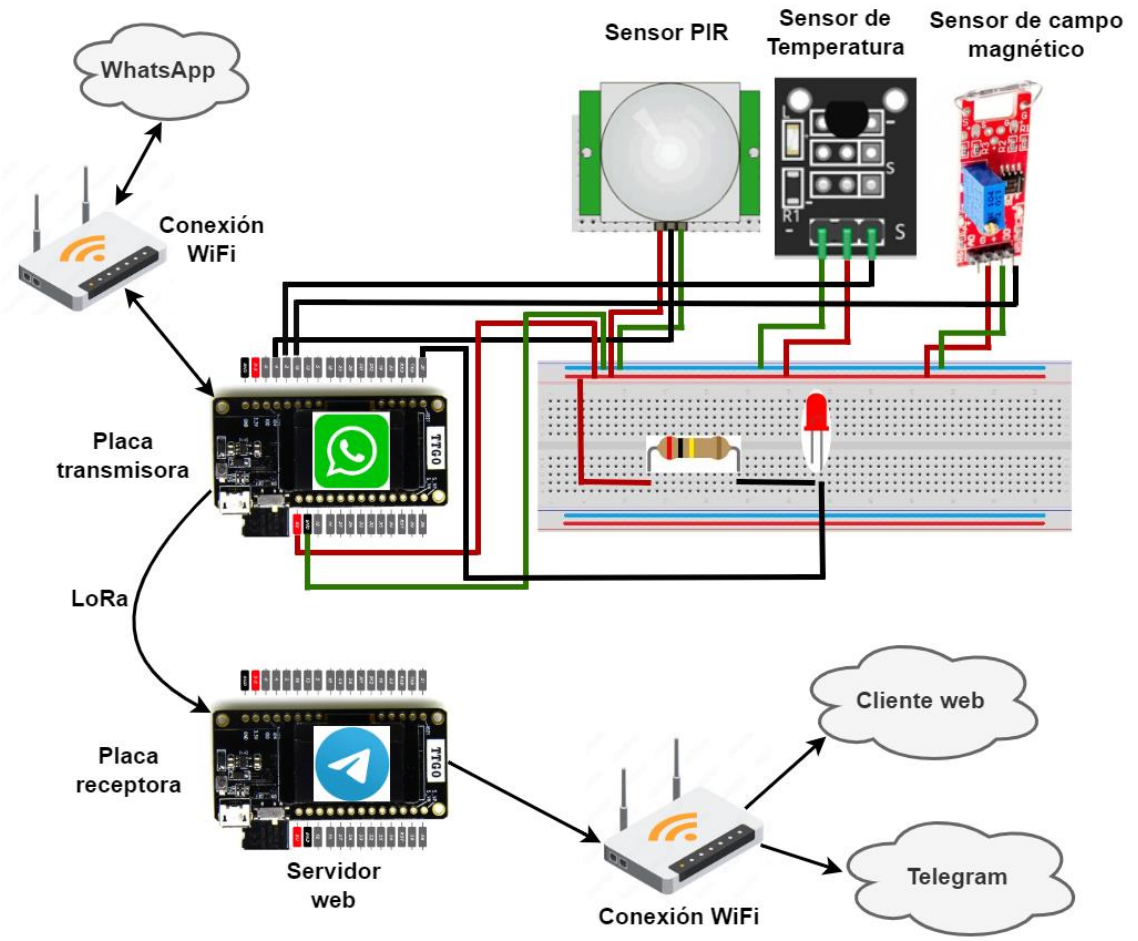

Figura 5.1 Esquema completo de la conexión entre los sensores y la placa TTGO LORA32 ESP32

La placa TTGO LORA32 ESP32 transmisora lee los datos de los sensores y los envía a la placa receptora. La TTGO LORA32 ESP32 receptora enviará estos datos a un servidor web interno, el cual los mostrará en una página web con un formato amigable como se muestra en la figura 5.2.

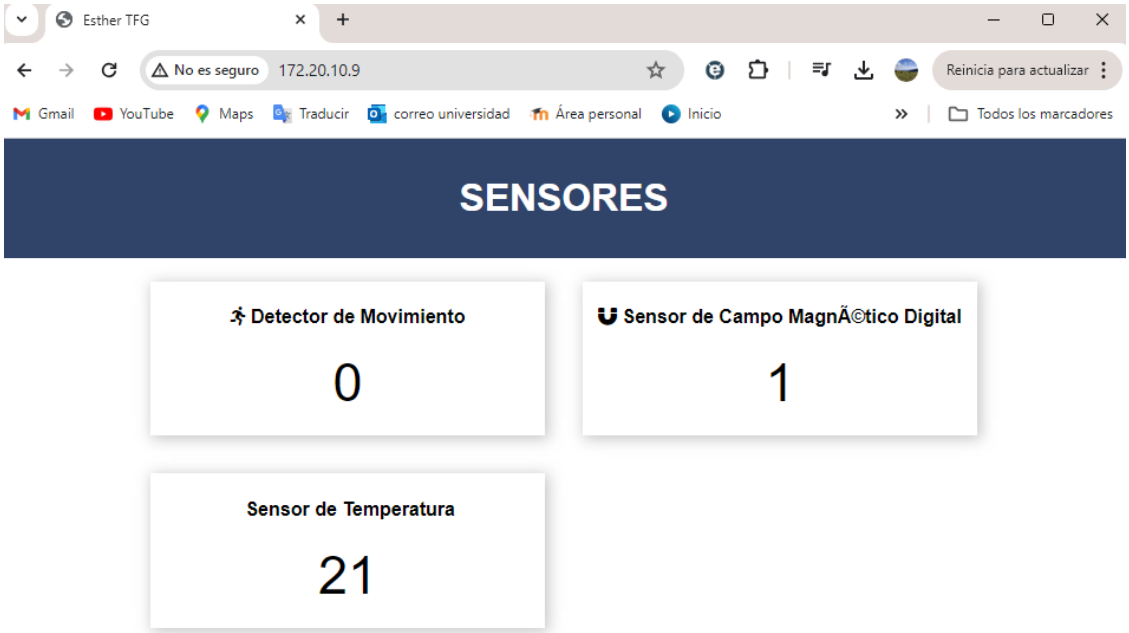

Figura 5.2 Página Web donde se muestran los datos de los sensores

El usuario podrá acceder a la página web conectándose a la misma red WiFi que el dispositivo, y accediendo en cualquier navegador a la dirección IP que muestra el dispositivo receptor en la pantalla OLED como se puede ver en la figura 5.3.

Figura 5.3 Dirección IP de la página Web mostrada en la pantalla OLED del receptor

Respecto a la interacción con Telegram, cada vez que los sensores detecten movimiento o campo magnético se enviarán notificaciones al bot especificado en el código, indicando la detección de estos parámetros como se muestra en la figura 5.4.

Figura 5.2 Notificaciones de los sensores vía Telegram

Y por último, para el control del LED a través de WhatsApp se debe acceder al chat de Twilio y enviar “Prendido”, “Apagado” para modificar el estado del LED, además Twilio devolverá un mensaje indicando que se ha realizado el cambio de estado. También se puede consultar el estado en el que se encuentra el LED enviando el mensaje “Estatus”. En la figura 5.5 se muestra la comunicación con el chat de Twilio.

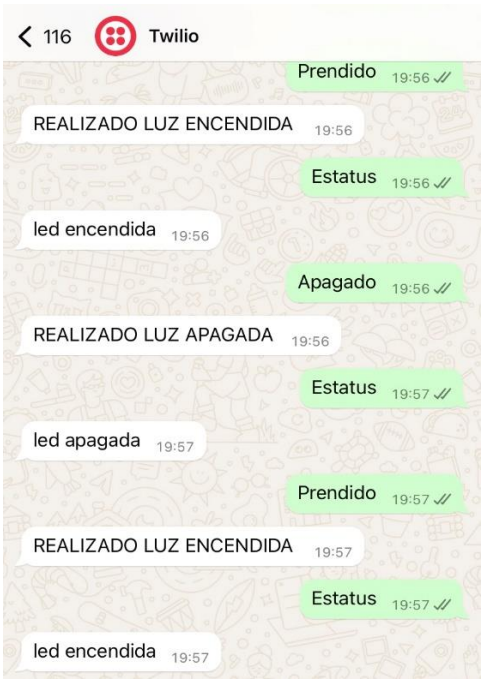

Figura 5.5 Chat de Twilio para el control remoto del LED

6

Capítulo 6 Conclusiones y líneas futuras

En este capítulo, se van a recopilar todas las conclusiones a las que se ha llegado durante la realización de este proyecto, destacando el correcto funcionamiento del sistema de telemetría LoRa para la monitorización de diferentes sensores.

De cara al futuro, se proponen diferentes mejoras como ampliar la funcionalidad con más sensores y control remoto, mejoras en la comunicación con acceso web universal, e investigar nuevas aplicaciones como el control de incendios o inundaciones.

6.1 Conclusiones

Actualmente, la automatización en la obtención, inserción y análisis de datos es muy útil e importante, consiguiendo ahorrar retardos y optimizar los resultados obtenidos.

En particular, los sensores que se han elegido para ser utilizados en este trabajo han sido: el sensor de detección de movimiento, el sensor de detección de campo magnético y el sensor de medición de temperatura para su posterior procesado y visualización de datos.

La placa transmisora TTGO LORA32 ESP32 a través de su interfaz LoRa, le transmite los datos obtenidos de los sensores a la otra placa receptora, la cual se encargará de mostrar los datos en la página web generada por el servidor web que se encuentra en el dispositivo receptor. De esta forma, el usuario podrá consultar los datos de manera más intuitiva y cómoda. Además, se enviarán notificaciones a través de Telegram cuando se detecte movimiento o campo magnético para un conocimiento más inmediato de los datos. Y, por último, se podrá controlar el encendido o apagado de un LED, así como obtener su estado a través de un chat de WhatsApp. El hecho de poder actuar sobre un LED es únicamente una prueba de concepto para demostrar que se puede actuar sobre cualquier elemento tanto de baja tensión (LED) como de alta a través de uno o varios relés.

Al inicio de este trabajo se exponían como objetivos; realizar el diseño y la implementación de un sistema de telemetría basado en sensores conectados y controlados por los microprocesadores ESP32 empleando la tecnología LoRa para la transmisión de los datos entre los dispositivos TTGO LORA32 ESP32, la notificación de diferentes eventos a través de Telegram, y el control de un LED a través de una conversación de WhatsApp. Finalmente, se han hecho las correspondientes pruebas de validación del prototipo para comprobar que, tanto los sensores, como la programación de las dos placas (transmisor y receptor) funcionaran correctamente, dando así por alcanzados los objetivos previamente establecidos.

6.2 Líneas futuras

Una posible mejora, sería añadir más sensores de este o de otros tipos, ampliando así la zona de cobertura de monitorización frente a diferentes campos como el control de incendios, inundaciones u otros tipos de peligros. Esto llevaría, a su vez, a una posible ampliación del número de placas TTGO LORA32 ESP32 desplegadas no sólo en la misma zona, sino en ubicaciones alejadas y estratégicamente situadas, ya que la tecnología permite la conexión entre diferentes dispositivos separados varios kilómetros.

Otro aspecto que se podría estudiar es el alcance (LoS o con obstáculos) entre las placas TTGO LORA32 ESP32 transmisora y receptora. Esto permitiría saber con exactitud la distancia a la que se pueden situar ambos dispositivos sin que muestren retardos de transmisión excesivos, errores en los datos recibidos o, en el peor caso, la perdida de la comunicación.

Por otro lado, respecto a la parte software del sistema, se podría configurar el sistema para que se pudieran cargar los programas a la placa TTGO LORA32 ESP32 de manera inalámbrica (OTA, Over the Air), de esta forma sería posible modificar o actualizar el código independientemente de la ubicación.

También, sería interesante integrar el sistema desarrollado en la plataforma Home Assistant que es un sistema operativo diseñado exclusivamente para integración, control y monitorización de dispositivos IoT, con múltiples posibilidades de configuración y visualización de parámetros mediante paneles de control personalizables. Esto permitiría utilizar un LoRa Gateway combinando la pila de protocolos LoRaWAN en el lado de los sensores con el protocolo MQTT en el backend del sistema.

Por último, respecto al servidor web, se podrían realizar las modificaciones oportunas en la red para que el acceso a dicho servidor fuera posible desde el exterior de la WLAN local y, de esta manera, los datos de los sensores se pudieran mostrar de manera remota en cualquier dispositivo de Internet.

Referencias

Todas las referencias web han sido visitadas y están online a fecha de 15 de mayo de 2024

[1] A. Bassi, «Goto IoT,» [En línea]. Available: https://www.gotoiot.com/pages/articles/lora_intro/index.html

[2] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Banda_ISM

[3] P. Pickering, «DigiKey,» 29 06 2017. [En línea]. Available: https://www.digikey.com/es/articles/develop-lora-for-low-rate-long-range-iot-applications

[4] E. B. Cadena, 06 06 2022. [En línea]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=8652915

[5] Calderón, «ResearchGate,» 2019. [En línea]. Available: https://www.researchgate.net/figure/Figura-1-Modulacion-CSS-Fuente-Calderon-2019_fig1_368351768

[6] «Becolve Digital,» 14 06 2022. [En línea]. Available: https://becolve.com/blog/conceptos-tecnicos-basicos-que-te-ayudaran-a-entender-lora-y-lorawan-low-power-wide-area-network-en-pocos-minutos/

[7] «Alai Secure,» [En línea]. Available: https://alaisecure.es/glosario/lorawan-que-es-para-que-sirve-y-como-funciona/

[8] «El bosque de Silicio,» [En línea]. Available: https://elbosquedesilicio.es/lorawan3/

[9] Sabas, «Medium,» 2 10 2017. [En línea]. Available: https://medium.com/beelan/haciendo-iot-con-lora-capitulo-2-tipos-y-clases-de-nodos-3856aba0e5be

[10] «BricoGeek,» [En línea]. Available: https://tienda.bricogeek.com/lora/1122-ttgo-lora32-esp32-con-oled-900-mhz.html

[11] «DigiKet,» 21 Enero 2020. [En línea]. Available: https://www.digikey.com/es/articles/how-to-select-and-use-the-right-esp32-wi-fi-bluetooth-module

[12] «DigiKey,» [En línea]. Available: https://www.digikey.es/es/products/detail/espressif-systems/ESP32-D0WDQ6/8028404

[13] «Naylamp Mechatronics,» [En línea]. Available: https://naylampmechatronics.com/sensores-proximidad/55-modulo-sensor-pir-hc-sr501.html

[14] «Robotlandia,» [En línea]. Available: https://robotlandia.es/movimiento-y-distancia/853-sensor-movimiento-pir.html

[15] «Unit Electronics,» [En línea]. Available: https://uelectronics.com/producto/modulo-ky-001-sensor-de-temperatura/

[16] «Leantec,» [En línea]. Available: https://leantec.es/tienda/modulo-sensor-de-temperatura-ky-001/

[17] «naylamp mechatronics,» [En línea]. Available: https://naylampmechatronics.com/blog/46_tutorial-sensor-digital-de-temperatura-ds18b20.html

[18] «Leantec,» [En línea]. Available: https://leantec.es/tienda/ky-025-modulo-sensor-interruptor-por-campo-magnetico-reed-grande/

[19] «IDE Arduino,» [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/

[20] «CEAC» [En línea]. Available: https://www.ceac.es/blog/que-es-el-ide-de-arduino-en-robotica

[21] «Brico Geek» [En línea]. Available: https://tienda.bricogeek.com/lora/1122-ttgo-lora32-esp32-con-oled-900-mhz.html

[22] «Brico Geek» [En línea]. Available: https://tienda.bricogeek.com/sensores-distancia/1270-modulo-sensor-pir-hc-sr501.html

[23] «Brico Geek» [En línea]. Available: https://tienda.bricogeek.com/sensores/41-sensor-de-temperatura-ds18b20-one-wire.html

[24] «Brico Geek» [En línea]. Available: https://tienda.bricogeek.com/otros-sensores/514-sensor-de-efecto-hall-us1881.html

[25] «Aliexpress, LILYGO®TTGO-módulo inalámbrico LoRa32 V2.1, Versión 1,6, 433/868/915Mhz, ESP32 LoRa OLED, tarjeta SD de 0,96 pulgadas, Bluetooth, WIFI, ESP-32 SMA» [En línea]. Available: https://es.aliexpress.com/item/1005005771488627.html

[26] «Aliexpress, HC-SR501 HC-SR505 AM312 ajuste IR piroeléctrico infrarrojo Mini PIR Módulo Sensor de movimiento módulo Detector soporte para Arduino» [En línea]. Available: https://es.aliexpress.com/item/1005005140420875.html

[27] «Aliexpress, Módulo de Sensor de temperatura DS18B20 de 3 pines, KY-001 KY001 para Arduino, Kit DIY, módulo de KY-001» [En línea]. Available: https://es.aliexpress.com/item/1005003130161420.html

[28] «Aliexpress, Módulo de sensor de caña de KY-025, MagSwitch para Arduino» [En línea]. Available: https://es.aliexpress.com/item/1005002986946586.html

Anexo I. Presupuesto

Presupuesto tienda electrónica		
Producto	**Precio**	**Con IVA**
TTGO LORA 32 ESP32 [21] x 2	55,80 €	67,52 €
Sensor de movimiento PIR HC-SR501 [22]	1,85 €	2,24 €
Sensor de temperatura DS18B20 One-Wire [23]	1,50 €	1,82 €
Sensor de efecto Hall US1881 [24]	1,10 €	1,33 €
Total	**60,25 €**	**72,91**€

Presupuesto low cost	
Producto	**Precio (IVA incluido)**
Tarjeta TTGO con Bluetooth, módulo WIFI, ESP32-Paxcounter Lora32 V2.1, versión 1.6.1, Lora ESP-32, OLED, SD de 0,96 pulgadas [25] x 2	41,48 €
Sensor de movimiento infrarrojo para Arduino, con ajuste de Sensor piroeléctrico HC-SR501 [26]	0.94 €
Módulo de KY-001 de Sensor de temperatura [27]	0,94 €
Módulo de Sensor de lengüeta de KY-025, módulo de magnetrón, interruptor de lengüeta [28]	0,79 €
Total	**44,15 €**

Anexo II. Código del programa para el transmisor de los datos de sensores

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <Wire.h> //Librería que permite la comunicación I2C

#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define DETECTION_PIN 4 // Pin donde está conectado el sensor PIR

void setup() {

  Serial.begin(115200); //Inicialica la comunicación serial a 115200 baudios
  SPI.begin(SCK, MISO, MOSI, SS); //Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0); //Configura los pines para LoRa

  if (!LoRa.begin(BAND)) {
    Serial.println("¡Fallo al iniciar LoRa!");
    while (1);
  } //Comprobación de que se ha inicializado correctamente LoRa

  Serial.println("LoRa inicializado correctamente!");

}

void loop() {

  int SensorState = digitalRead(DETECTION_PIN); //Lee el valor del sensor

  LoRa.beginPacket(); //Inicialización del paquete LoRa
  LoRa.print("Estado sensor: ");
  LoRa.print(SensorState);
  LoRa.endPacket(); //Finalización y envío del paquete


  delay(1000); //Se realiza el bucle cada un segundo
}
```

Anexo III. Código del receptor de los datos de sensores

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <Wire.h> //Librería que permite la comunicación I2C

#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define LED_ACTIVE_PIN 2 // Pin para el LED rojo
#define LED_INACTIVE_PIN 4 // Pin para el LED verde

String LoRaData; //Variable que almacena los datos recibidos por LoRa
void setup() {
  Serial.begin(115200); //Inicializa la comunicación serial a 115200 baudios
  SPI.begin(SCK, MISO, MOSI, SS); //Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0); //Configura los pines para LoRa
  if (!LoRa.begin(BAND)) {
    Serial.println("¡Fallo al iniciar LoRa!");
    while (1);
  } //Comprueba que se ha inicializado correctamente LoRa
  pinMode(LED_ACTIVE_PIN, OUTPUT);
  pinMode(LED_INACTIVE_PIN, OUTPUT); //Configuracion de los pines como salidas
}

void updateLEDs() {
  digitalWrite(LED_ACTIVE_PIN, LOW);
  digitalWrite(LED_INACTIVE_PIN, LOW); //Apagamos los LEDs
  if (LoRaData == "0") {
    digitalWrite(LED_ACTIVE_PIN, HIGH); //Si el valor es cero se enciende el
LED del pin 2
  } else if (LoRaData == "1") {
    digitalWrite(LED_INACTIVE_PIN, HIGH); //Si el valor es uno se enciende el
LED del pin 4
  }
}

void loop() {
  int packetSize = LoRa.parsePacket(); //Se comprueba que se han recibido
paquetes LoRa
  if (packetSize) {
    while (LoRa.available()) {
      LoRaData = LoRa.readString(); //Lee los datos recibidos y se guardan en
la variable LoRaData
    }
    updateLEDs(); //Actualiza el estado de los leds segun el valor almacenado
en LoRaData
  }
}
```

Anexo IV. Código implementación servidor Web del transmisor

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <Wire.h> //Librería que permite la comunicación I2C
#include <ArduinoJson.h> //Librería que permite el uso de JSON
#include <OneWire.h> //Librería que permite la comunicación OneWire
#include <DallasTemperature.h> //Librería para el uso del sensor de
temperatura

#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define DETECTION_PIN 4 // Pin sensor PIR
#define MAGNETIC_PIND 17 // Pin sensor campo magnético
#define TEMPERATURA 2 // Pin sensor temperatura

int oneWireBus = TEMPERATURA; //Pin del bus OneWire

OneWire oneWire(oneWireBus); //Inicializa la comunicación OneWire
DallasTemperature sensors(&oneWire); //Inicializa el sensor de temperatura

typedef struct {
  int data1;
  int data2;
  int data3;
} SensorData; //Estructura de datos para la transmisión de los valores de los
sensores


void setup()
{
  Serial.begin(115200); //Inicializa la comunicación serial a 115200 baudios
  SPI.begin(SCK, MISO, MOSI, SS); //Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0); //Configura los pines para LoRa

  if (!LoRa.begin(BAND)) {
    Serial.println("¡Fallo al iniciar LoRa!");
    while (1);
  } //Comprobación de que se ha inicializado correctamente LoRa

  sensors.begin(); //Inicializa la lectura del sensor de temperatura

}


void loop()
{

  sensors.requestTemperatures(); //Inicializa la lectura del sensor de
temperatura
  int detectionState = digitalRead(DETECTION_PINTION_PIN); //Guarda el valor
del sensor de movimiento
  int magneticState = digitalRead(MAGNETIC_PIND); // Guarda el estado del
sensor de campo magnético
```

```
  float temperaturaState = sensors.getTempCByIndex(0); //Guarda el valor del
sensor de temperatura

  SensorData sensor1Data = {detectionState, magneticState, temperaturaState};
//Crea la estructura de mensaje para almacenar los valores de los sensores en
sensor1Data

  StaticJsonDocument<128> doc1; // Crea un documento JSON para guardar los
valores de los sensores
  doc1["data1"] = sensor1Data.data1;
  doc1["data2"] = sensor1Data.data2;
  doc1["data3"] = sensor1Data.data3; //Asigna los valores a diferentes campos
del documento JSON

  char jsonBuffer1[128]; //Define un buffer de 128 caracteres
  serializeJson(doc1, jsonBuffer1); // Convierte doc1 en una cadena de
caracteres y la guarda en el jsonBuffer1
  LoRa.beginPacket();//Inicialización del paquete LoRa
  LoRa.print(jsonBuffer1);
  LoRa.endPacket();//Finalización y envío del paquete


  delay(1000);//Se realiza el bucle cada un segundo
}
```

Anexo V. Código implementación servidor Web del receptor

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <WiFi.h> //Librería que permite la conexión WiFi
#include "ESPAsyncWebServer.h" //Librería que permite crear un servidor web
#include <Adafruit_GFX.h> //Librería para el uso de gráficos
#include <Adafruit_SSD1306.h> //Librería que permite el uso de la pantalla
OLED
#include <ArduinoJson.h> //Librería que permite el uso de JSON

#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define OLED_SDA 21 //Define el pin SDA de la pantalla OLED
#define OLED_SCL 22 //Define el pin SCL de la pantalla OLED
#define OLED_RST 16 //Define el pin reset de la pantalla OLED
#define SCREEN_WIDTH 128 //Define el ancho de la pantalla OLED
#define SCREEN_HEIGHT 64 //Define el largo de la pantalla OLED


 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RST);//Inicializa la pantalla OLED
 AsyncWebServer server(80); //Permite el manejo de solicitudes HTTP
 AsyncEventSource events("/events"); //Permite el uso de eventos asincronos

 String LoRaData; //Crea una variable

//Código HTML para la página WEB
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
  <title>Esther TFG</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
  <link rel="icon" href="data:,">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    p {  font-size: 1.2rem;}
    body {  margin: 0;}
    .topnav { overflow: hidden; background-color: #2f4468; color: white; font-
size: 1.7rem; }
    .content { padding: 20px; }
    .card { background-color: white; box-shadow: 2px 2px 12px 1px
rgba(140,140,140,.5); }
    .cards { max-width: 700px; margin: 0 auto; display: grid; grid-gap: 2rem;
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); }
    .reading { font-size: 2.8rem; }
    .packet { color: #bebebe; }
    .card.temperature { color: #fd7e14; }
    .card.humidity { color: #1b78e2; }
```

```
  </style>
</head>
<body>
  <div class="topnav">
    <h3>SENSORES</h3>
  </div>
  <div class="content">
    <div class="cards">
      <div class="card" id="sensor1">
        <h4><i class="fas fa-running"></i> Detector de Movimiento </h4>
        <p><span class="reading" id="loradata1">Esperando los
datos...</span></p>
      </div>
      <div class="card" id="sensor2">
        <h4><i class="fas fa-magnet"></i> Sensor de Campo Magnético
Digital</h4>
        <p><span class="reading" id="loradata2">Esperando los
datos...</span></p>
      </div>
      <div class="card" id="sensor3">
        <h4><i class="fas fa-temperature"></i> Sensor de Temperatura</h4>
        <p><span class="reading" id="loradata3">Esperando los
datos...</span></p>
      </div>
    </div>
  </div>
<script>

if (!!window.EventSource) {
 var source = new EventSource('/events');

 source.addEventListener('open', function(e) {
  console.log("Events Connected");
 }, false);
 source.addEventListener('error', function(e) {
  if (e.target.readyState != EventSource.OPEN) {
    console.log("Events Disconnected");
  }
 }, false);

 source.addEventListener('message', function(e) {
  console.log("message", e.data);
 }, false);

 source.addEventListener('new_data', function(e) {
  console.log("new_data", e.data);
  var jsonData = JSON.parse(e.data);
  var data1 = jsonData.data1;
  var data2 = jsonData.data2;
  var data3 = jsonData.data3;

  document.getElementById("loradata1").innerHTML = data1;
  document.getElementById("loradata2").innerHTML = data2;
  document.getElementById("loradata3").innerHTML = data3;


 }, false);
}

</script>
</body>
</html>)rawliteral";

void setup() {
  Serial.begin(115200);//Inicializa la comunicación serial a 115200 baudios
```

```
  // Inicializar LoRa
  SPI.begin();//Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0);//Configura los pines para LoRa
  if (!LoRa.begin(BAND)) {
    while (1);
  }//Comprueba que se ha inicializado correctamente LoRa
  // Inicializar WiFi
  WiFi.mode(WIFI_STA);//Inicializa el modo WiFi
  WiFi.begin("iPhone de Esther", "estherguapa");//Conexión a la red WiFi
definida
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }//Espera a que se establezca correctamente la conexión

  //Inicializa la pantalla OLED
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("Dirección IP WiFi:");
  display.println(WiFi.localIP()); //Muestra la dirección IP de la página web
  display.display();

  //Inicializa el servidor web
  server.addHandler(&events);//Recibe los eventos cada vez que llegan los
datos de los sensores
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);}); //Muestra la página HTML
cuando se accede al servidor

  server.begin();//Inicializa el servidor web

}



void loop() {

  //Recepción del paquete LoRa
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    while (LoRa.available()) {
      LoRaData = LoRa.readString();
    }
  //Lee los datos de LoRa y extraer ID del sensor y dato
    StaticJsonDocument<128> doc;
    deserializeJson(doc, LoRaData);
    String data1 = doc["data1"];
    String data2 = doc["data2"];
    String data3 = doc["data3"];
  //Enviar datos de LoRa a través de EventSource
    String eventData = "{\"data1\": " + String(data1) + ", \"data2\": " +
                       String(data2) + ", \"data3\": " + String(data3) + "}";
    events.send(eventData.c_str(), "new_data", millis());

    }

  delay(1);

}
```

Anexo VI. Código implementación notificaciones vía Telegram

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <WiFi.h> //Librería que permite la conexión WiFi
#include "ESPAsyncWebServer.h" //Librería que permite crear un servidor web
#include <Adafruit_GFX.h> //Librería para el uso de gráficos
#include <Adafruit_SSD1306.h> //Librería que permite el uso de la pantalla
OLED
#include <ArduinoJson.h> //Librería que permite el uso de JSON
#include <HTTPClient.h> //Librería para realizar solicitudes HTTP


#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define OLED_SDA 21 //Define el pin SDA de la pantalla OLED
#define OLED_SCL 22 //Define el pin SCL de la pantalla OLED
#define OLED_RST 16 //Define el pin reset de la pantalla OLED
#define SCREEN_WIDTH 128 //Define el ancho de la pantalla OLED
#define SCREEN_HEIGHT 64 //Define el largo de la pantalla OLED


 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RST);//Inicializa la pantalla OLED
 AsyncWebServer server(80); //Permite el manejo de solicitudes HTTP
 AsyncEventSource events("/events"); //Permite el uso de eventos asincronos

 String LoRaData; //Crea una variable

//Código HTML para la página WEB
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
  <title>Esther TFG</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
  <link rel="icon" href="data:,">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    p {  font-size: 1.2rem;}
    body {  margin: 0;}
    .topnav { overflow: hidden; background-color: #2f4468; color: white; font-
size: 1.7rem; }
    .content { padding: 20px; }
    .card { background-color: white; box-shadow: 2px 2px 12px 1px
rgba(140,140,140,.5); }
    .cards { max-width: 700px; margin: 0 auto; display: grid; grid-gap: 2rem;
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); }
    .reading { font-size: 2.8rem; }
    .packet { color: #bebebe; }
```

```
    .card.temperature { color: #fd7e14; }
    .card.humidity { color: #1b78e2; }
  </style>
</head>
<body>
  <div class="topnav">
    <h3>SENSORES</h3>
  </div>
  <div class="content">
    <div class="cards">
      <div class="card" id="sensor1">
        <h4><i class="fas fa-running"></i> Detector de Movimiento </h4>
        <p><span class="reading" id="loradata1">Esperando los
datos...</span></p>
      </div>
      <div class="card" id="sensor2">
        <h4><i class="fas fa-magnet"></i> Sensor de Campo Magnético
Digital</h4>
        <p><span class="reading" id="loradata2">Esperando los
datos...</span></p>
      </div>
      <div class="card" id="sensor3">
        <h4><i class="fas fa-temperature"></i> Sensor de Temperatura</h4>
        <p><span class="reading" id="loradata3">Esperando los
datos...</span></p>
      </div>
    </div>
  </div>
<script>

if (!!window.EventSource) {
 var source = new EventSource('/events');

 source.addEventListener('open', function(e) {
  console.log("Events Connected");
 }, false);
 source.addEventListener('error', function(e) {
 if (e.target.readyState != EventSource.OPEN) {
    console.log("Events Disconnected");
  }
 }, false);

 source.addEventListener('message', function(e) {
  console.log("message", e.data);
 }, false);

 source.addEventListener('new_data', function(e) {
  console.log("new_data", e.data);
  var jsonData = JSON.parse(e.data);
  var data1 = jsonData.data1;
  var data2 = jsonData.data2;
  var data3 = jsonData.data3;

  document.getElementById("loradata1").innerHTML = data1;
  document.getElementById("loradata2").innerHTML = data2;
  document.getElementById("loradata3").innerHTML = data3;


 }, false);
}

</script>
</body>
</html>)rawliteral";
```

```
void setup() {
  Serial.begin(115200);//Inicializa la comunicación serial a 115200 baudios
  // Inicializar LoRa
  SPI.begin();//Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0);//Configura los pines para LoRa
  if (!LoRa.begin(BAND)) {
    while (1);
  }//Comprueba que se ha inicializado correctamente LoRa
  // Inicializar WiFi
  WiFi.mode(WIFI_STA);//Inicializa el modo WiFi
  WiFi.begin("iPhone de Esther", "estherguapa");//Conexión a la red WiFi
definida
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }//Espera a que se establezca correctamente la conexión

  //Inicializa la pantalla OLED
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("Dirección IP WiFi:");
  display.println(WiFi.localIP()); //Muestra la dirección IP de la página web
  display.display();

  //Inicializa el servidor web
  server.addHandler(&events);//Recibe los eventos cada vez que llegan los
datos de los sensores
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  }); //Muestra la página HTML cuando se accede al servidor

  server.begin();//Inicializa el servidor web

}



void loop() {

  //Recepción del paquete LoRa
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    while (LoRa.available()) {
      LoRaData = LoRa.readString();
    }
  //Lee los datos de LoRa y extraer ID del sensor y dato
    StaticJsonDocument<128> doc;
    deserializeJson(doc, LoRaData);
    String data1 = doc["data1"];
    String data2 = doc["data2"];
    String data3 = doc["data3"];
  //Enviar datos de LoRa a través de EventSource
    String eventData = "{\"data1\": " + String(data1) + ", \"data2\": " +
                       String(data2) + ", \"data3\": " + String(data3) + "}";
    events.send(eventData.c_str(), "new_data", millis());

    String mensaje1; //Variables para guardar los mensajes de Telegram
    String mensaje2;

    if (WiFi.status() == WL_CONNECTED) { //Comprueba si está conectado a una
red WiFi
      HTTPClient http; //Objeto para realizar solicitudes HTTP

     if(data1 == "1"){ //Si se detecta movimiento inicia el proceso de envío
de mensaje
        mensaje1 = "Se ha detectado movimiento"; //Mensaje a enviar
```

```
        // URL de la API de Telegram para enviar mensajes
        String url1 = "https://api.telegram.org/bot6423366563:AAG3Rmn3Qx2xh-
ZiKCCNKICClVKoxlSxy10/sendMessage?chat_id=6307998350&text=" + mensaje1;
        http.begin(url1); //Inicializa la solicitud HTTP GET de la url2
        http.GET(); //Realiza la solicitud HTTP GET
        http.end(); //Finaliza la solicitud HTTP GET

     }

     if(data2 == "1"){ //Si se detecta campo magnético inicia el proceso de
envío de mensaje
        mensaje2 = "Se ha detectado campo magnético"; //Mensaje a enviar
             // URL de la API de Telegram para enviar mensajes
        String url2 = "https://api.telegram.org/bot6423366563:AAG3Rmn3Qx2xh-
ZiKCCNKICClVKoxlSxy10/sendMessage?chat_id=6307998350&text=" + mensaje2;
        http.begin(url2); //Inicializa la solicitud HTTP de la url2
        http.GET(); //Realiza la solicitud HTTP GET
        http.end(); //Finaliza la solicitud HTTP GET

     }

    }
  }

  delay(1);

}
```

Anexo VII. Código implementación control LED vía WhatsApp

```
#include <SPI.h> //Librería que permite comunicación SPI
#include <LoRa.h> //Librería que permite la comunicación LoRa
#include <Wire.h> //Librería que permite la comunicación I2C
#include <ArduinoJson.h> //Librería que permite el uso de JSON
#include <OneWire.h> //Librería que permite la comunicación OneWire
#include <ThingESP.h> //Librería que permite el uso de ThingESP
#include <DallasTemperature.h> //Librería para el uso del sensor de
temperatura
#include <WiFi.h> //Librería que permite el uso de WiFi

#define SCK 5 //Pin SCK
#define MISO 19 //Pin MISO
#define MOSI 27 //Pin MOSI
#define SS 18 //Pin Slave Select
#define RST 14 //Pin Reset
#define DIO0 26 //Pin DIOO

#define BAND 866E6 //Frecuencia a la que trabaja LoRa

#define DETECTION_PIN 4 // Pin sensor PIR
#define MAGNETIC_PIND 17 // Pin sensor campo magnético
#define TEMPERATURA 2 // Pin sensor temperatura

ThingESP32 thing("esthergc16", "Prueba", "Control"); //Conecta el dispositivo
con la plataforma ThingESP

int LED = 21; //Pin del LED
int oneWireBus = TEMPERATURA; //Pin del bus OneWire

OneWire oneWire(oneWireBus); //Inicializa la comunicación OneWire
DallasTemperature sensors(&oneWire); //Inicializa el sensor de temperatura

typedef struct {
  int data1;
  int data2;
  int data3;
} SensorData; //Estructura de datos para la transmisión de los valores de los
sensores


void setup()
{
  Serial.begin(115200); //Inicializa la comunicación serial a 115200 baudios
  SPI.begin(SCK, MISO, MOSI, SS); //Inicializa la comunicación SPI
  LoRa.setPins(SS, RST, DIO0); //Configura los pines para LoRa

  if (!LoRa.begin(BAND)) {
    Serial.println("¡Fallo al iniciar LoRa!");
    while (1);
  } //Comprobación de que se ha inicializado correctamente LoRa

  sensors.begin(); //Inicializa la lectura del sensor de temperatura

  pinMode(LED, OUTPUT); //Se asegura de que el LED está apagado
  thing.SetWiFi("iPhone de Esther", "estherguapa"); //Se conecta a la red WiFi
para el funcionamiento de ThingESP
  thing.initDevice(); //Inicializa las conexiones y la puesta en marcha de los
servicios necesarios

}
```

```
String HandleResponse(String consulta) {//Gestiona la respuesta de la llamada
y si la llamada se realiza correctamente, analiza la consulta
  if (consulta == "prendido") {
    digitalWrite(LED, 1);
    return "REALIZADO LUZ ENCENDIDA";
  }

  else if (consulta == "apagado") {
    digitalWrite(LED, 0);
    return "REALIZADO LUZ APAGADA";
  }
  else if (consulta == "estatus")
    return digitalRead(LED) ? "led encendida" : "led apagada";
  else return "invalido";
}



void loop()
{

  sensors.requestTemperatures(); //Inicializa la lectura del sensor de
temperatura
  int detectionState = digitalRead(DETECTION_PINTION_PIN); //Guarda el valor
del sensor de movimiento
  int magneticState = digitalRead(MAGNETIC_PIND); // Guarda el estado del
sensor de campo magnético
  float temperaturaState = sensors.getTempCByIndex(0); //Guarda el valor del
sensor de temperatura

  SensorData sensor1Data = {detectionState, magneticState, temperaturaState};
//Crea la estructura de mensaje para almacenar los valores de los sensores en
sensor1Data

  StaticJsonDocument<128> doc1; // Crea un documento JSON para guardar los
valores de los sensores
  doc1["data1"] = sensor1Data.data1;
  doc1["data2"] = sensor1Data.data2;
  doc1["data3"] = sensor1Data.data3; //Asigna los valores a diferentes campos
del documento JSON

  char jsonBuffer1[128]; //Define un buffer de 128 caracteres
  serializeJson(doc1, jsonBuffer1); // Convierte doc1 en una cadena de
caracteres y la guarda en el jsonBuffer1
  LoRa.beginPacket();//Inicialización del paquete LoRa
  LoRa.print(jsonBuffer1);
  LoRa.endPacket();//Finalización y envío del paquete

  thing.Handle(); //Llama al cliente para la transferencia de datos

  delay(1000);//Se realiza el bucle cada un segundo
}
```