UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Desarrollo de un sistema de control de un robot de exploración para zonas de baja cobertura.

Caso de estudio: Control remoto de robot utilizando LORAWaN y TTN

Autor

Francisco Javier Cañizares Cancho

Directores

Juan José Ramos Muñoz

Jorge Navarro Ortiz

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2025

Desarrollo de un sistema de control de un robot de exploración para zonas de baja cobertura

Caso de estudio: Control remoto de robot utilizando LORAWaN y TTN

Autor

Francisco Javier Cañizares Cancho

Directores

Juan José Ramos Muñoz
Jorge Navarro Ortiz

Desarrollo de un sistema de control de un robot de exploración para zonas de baja cobertura: Caso de estudio: Control remoto de robot utilizando LORAWaN y TTN

Francisco Javier Cañizares Cancho

Palabras clave: mBot, LORA, LORAWAN, TTN, mota, Gateway, Python

Resumen

En este trabajo de fin de grado se busca estudiar la posibilidad del control de un a gran distancia y para zonas de baja cobertura conectándole una mota LORA y un Gateway que se conectará a la aplicación de *The Things Network* (TTN) para la gestión de dispositivos y envío de instrucciones, y desarrollando una aplicación en Python que envíe las instrucciones del robot a TTN. Para ello, habrá que desarrollar el código del robot para que pueda recibir las instrucciones, estudiar la conexión del robot con la mota que se podrá hacer de distintas maneras de las cuales se hablará posteriormente, programar la mota para que sea capaz de enviar y recibir mensajes desde TTN y programar la aplicación en Python. El gateway se conectará a TTN, donde se mostrarán los mensajes enviados de la mota (para indicar la posición por ejemplo) y los mensajes que recibirá la mota (las instrucciones del robot). La aplicación para desarrollada en Python constará de una interfaz gráfica para elegir que comando enviar.

Development of an exploration robot control system for low coverage areas: Case study: Robot remote control using LORAWaN and TTN

Francisco Javier Cañizares Cancho

Keywords: mBot, LORA, LORAWAN, TTN, mote, Gateway, Python

Abstract

The aim of this final degree project is to study the possibility of controlling a robot from a great distance by connecting a LORAWAN mote and a Gateway that will be connected to The Things Network (TTN) application for managing devices and sending instructions and developing a Python application that sends the robot's instructions to TTN. To do this, we will have to develop the code of the robot so that it can receive the instructions, study the connection of the robot with the mote which can be done in different ways that we will explore along this document, write the code of the mote so it can send and receive messages from TTN and develop the application to send the instructions In Python. The gateway will be connected to TTN where the messages sent from the mote (to indicate the position for example) and the messages that the mote will receive (the robot instructions) will be displayed. The application to be developed in Python will consist of a graphical interface to choose which command to send.

Yo, **Francisco Javier Cañizares Cancho**, alumno de la titulación Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 26519110H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Javier Cañizares Cancho

Granada a Junio de 2025.

D. **Juan José Ramos Muñoz**, Profesor del área de Ingeniería Telemática del Departamento de Teoría de la señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Jorge Navarro Ortiz,** Profesor del área de Ingeniería Telemática del Departamento de Teoría de la señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Desarrollo de un sistema de control de un robot de exploración para zonas de baja cobertura, Caso de estudio: Control remoto de robot utilizando LoRaWAN y TTN***, ha sido realizado bajo su supervisión por **Francisco Javier Cañizares Cancho** y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a Junio de 2025.

Los directores:

Juan José Ramos Muñoz **Jorge Navarro Orti**

Agradecimientos

Doy gracias a todo el apoyo que he recibido por mi familia y amigos en todos los años de carrera y, sobre todo, en el último empujón que es el TFG.

Índice

Índice de figuras

Índice de tablas

Acrónimos

A | B | C | D | E | G | I | L | M | N | O | Q | R | S | T | U | W |

A

ABP *Activation By Personalization*

ADR *Adaptative Data Rate*

AFA *Adaptative Frequency Agility*

B

BW *BandWidth* (Ancho de banda)

BLE *Bluetooth Low Energy*

C

CSS *Chirp Spread Spectrum* (Espectro Ensanchado Chirp)

D

DL *DownLink* (Enlace Descendente)

E

ETSI *European Telecommunication Standards Institute*

G

GSM *Global System for Mobile communications* (Sistema Global para

comunicaciones Móviles)

I

IDE *Integrated Development Enviroment* (Entorno de desarrollo integrado)

IoT *Internet of Things* (Internet de las Cosas)

ISM *Industrial, Scientific, Medical*

I2C *Inter-Integrated Circuit*

L

LPWAN *Low Power Wide Area Network* (Red de área amplia de baja potencia)

LoRaWAN *LoRa Wide Area Network* (Protocolo de comunicación sobre LoRa)

LTE *Long-Term Evolution* (Evolución a Largo Plazo)

LBT *Listen Before Talk*

M

MQTT *Message Queuing Telemetry Transport* (Protocolo usado para comunicación máquina a máquina)

N

NTP *Network Time Protocol*

O

OSI *Open Systems Interconnection*

OTAA *Over-The-Air-Activation*

Q

QoS *Quality of Service*

R

RFID *Radio-Frequency Identification*

RSSI *Received Signal Strength Indicator*

S

SF *Spreading Factor*

SCL *System Clock*

SDA *System Data*

SNR *Signal-to-Noise Ratio*

T

TTN *The Things Network* (Plataforma para red LoRaWAN)

U

UL *UpLink* (Enlace Ascendente)

UART *Universal Asynchronous Receiver-Transmitter*

W

WiFi *Wireless Fidelity*

Capítulo 1

Introducción

1.1. Resumen del Proyecto

En este Trabajo de Fin de Grado se pretende diseñar e implementar un sistema de control de un robot para explorar zonas con baja o nula cobertura.

Para lograr este objetivo se va a crear una red LPWAN, más en concreto LoRaWAN, cuyas ventajas son que tiene una gran cobertura a la vez que posee consumo energético muy bajo en los dispositivos que la utilizan, y a esta red se conectará un robot que realice las instrucciones que reciba. El robot que se va a utilizar es el mBot, robot educativo que permite una fácil programación y modificación de elementos *hardware*, y para crear esta LoRaWAN se utilizará la plataforma de *The Things Network* (TTN) en la que podríamos añadir fácilmente los dispositivos y pasarela que se conectarán a esta red. La pasarela que se utilizará será una Pycom Pygate y la mota que recibirá las instrucciones y transmitirá al mBot es una Lilygo T-Beam. Además, para facilitar el uso del mBot, se creará una aplicación con interfaz gráfica en Python donde se podrá elegir que acción realizará el mBot, y esta aplicación enviará la instrucción a TTN utilizando el protocolo MQTT.

1.2. Objetivos a cumplir

Para obtener resultados satisfactorios en este proyecto se han propuesto los siguientes objetivos a cumplir:

- Analizar el mBot, tanto los componentes del *hardware* que posee como el *software* de Makeblock para programarlo, estudiar las funcionalidades que puede realizar y también realizar una comparación con robots similares.
- Analizar la plataforma *The Things Network* para saber las capacidades que tiene y que ventajas tiene respecto a plataformas que cumplen el mismo propósito.

- Crear la red LoRaWAN, programar la placa Lilygo T-Beam y la pasarela, comprobar su correcto funcionamiento y el de los dispositivos con pruebas sencillas.
- Diseñar e implementar la aplicación en Python que, mediante una interfaz gráfica, permite enviar instrucciones a TTN.
- Estudiar la conexión entre la placa Lilygo T-Beam y el mBot y programar el mBot para interpretar las instrucciones que reciba.
- Realizar pruebas para obtener resultados de alcance, latencia y fiabilidad, tanto en entornos urbanos como rurales.

1.3. Contexto

En la actualidad tenemos una gran red de infraestructuras para redes móviles como 4G o 5G que permiten la interconexión una gran multitud de dispositivos, ya no solo de *smartphones* sino también de dispositivos de dispositivos del Internet de las Cosas (IoT). Aun así, existen muchos entornos donde estas redes están limitadas como pueden ser zonas rurales o remotas, teniendo que buscar otras soluciones que permitan mantener las comunicaciones y supervisar los dispositivos desplegados en dichos entornos.

Para ello se han desarrollado las tecnologías LPWAN (*Low Power Wide Area Network*) que son redes diseñadas para los dispositivos de IoT porque disponen de una gran cobertura, con un consumo mínimo de energía y bajos costes [1]. De LPWAN han surgido varias tecnologías como son *Sigfox, NB-IoT* o *LoRaWAN*. Dichas tecnologías, presentan diferencias técnicas entre sí que pueden ver a continuación en la siguiente tabla (1.1) y una comparación entre ellas en la figura (1.1):

	Sigfox	LoRaWAN	NB-IoT
Modulación	BPSK	CSS	QPSK
Frecuencia	Bandas ISM sin licencia (868 MHz en Europa, 915 MHz en América y 433 MHz en Asia)	Bandas ISM sin licencia (868 MHz en Europa, 915 MHz en América y 433 MHz en Asia)	Bandas LTE con licencia
Ancho de banda	100 Hz	250 KHz y 125 KHz	200 KHz
Data rate máximo	100 bps	50 kbps	200 kbps
Mensajes máximos diarios	140 (UL), 4 (DL)	Sin límites	Sin límites
Tamaño máximo del payload	12 bytes (UL), 8 bytes (DL)	243 bytes	1600 bytes
Rango	10 Km (zonas urbanas), 40 Km (zonas rurales)	5 Km (zonas urbanas), 20 Km (zonas rurales)	1 Km (zonas urbanas), 10 Km (zonas rurales)
Inmunidad a interferencias	Muy alta	Muy alta	Baja
Handover	Los dispositivos finales no se conectan a una sola estación base	Los dispositivos finales no se conectan a una sola estación base	Los dispositivos finales se conectan a una sola estación base
Estandarización	Sigfox colaborando con ETSI	LoRa-Alliance	3GPP

Tabla 1.1: Características las tecnologías *Sigfox, LoRaWAN y NB-IoT* [1].

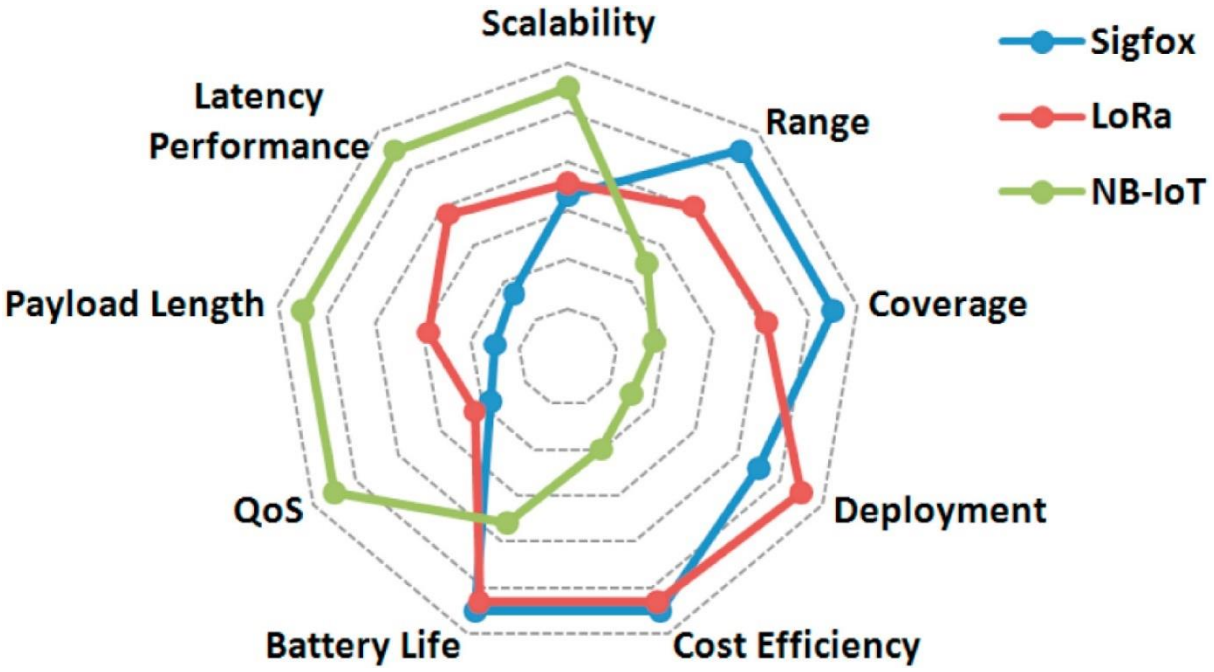


Figura 1.1: Comparación de las características de *Sigfox*, *LoRa* y *NB-IoT* [1].

En este proyecto utilizaremos LoRaWAN ya que, además de las características que se han mencionado en la tabla 1.1, existen otros factores que hacen que LoRaWAN sea el más apropiado para este proyecto. Por ejemplo, NB-IoT requiere de la infraestructura de las redes móviles de LTE ya que necesita de la cobertura que ofrecen, y además NB-IoT ocupa parte del ancho de banda de LTE [2]. Como hemos enfocado el proyecto para zonas sin cobertura de otro tipos de redes descartamos esta opción. Por otro lado, Sigfox también requiere de una red desplegada por un proveedor específico, lo que aumenta los costes bastante, utilizando un modelo cerrado que tiene unas restricciones de tamaño y frecuencia de mensajes mayores [1]. Como este proyecto está pensado como una prueba de concepto para comprobar si es posible el control de un robot de forma remota en entornos sin cobertura, queremos los costes sean mínimos, haciendo que LoRaWAN sea el candidato perfecto ya que permite la creación de redes privadas o utilizar redes comunitarias como *The Things Network*, convirtiéndose en la opción más flexible, accesible y abierta [2].

También se ha considerado utilizar un robot educativo como es el mBot (aunque posteriormente se hablarán de otros modelos que también podrían haber sido candidatos para el proyecto y porque se ha elegido finalmente al mBot) para realizar este experimento. Esto es porque ayuda a centrarse más en la parte de las

comunicaciones e integración con la red LoRaWAN al tener ya disponible un robot con una estructura modular y compatibilidad con Arduino para una programación sencilla, convirtiéndola en la herramienta perfecta para validar los conceptos y realizar pruebas de campo en entornos reales.

1.4. Motivación

Este proyecto nace de la idea de diseñar un sistema para la exploración de lugares remotos utilizando tecnologías emergentes como LPWAN. Se relacionará algo tan novedoso como es el Internet de las Cosas con la robótica ya que no solo se estudiará los fundamentos relacionados con ese tipo de redes, sino que también se analizará que componentes y la programación que necesita un robot para poder desempeñar dicha tarea.

1.5. Casos de uso

Se han identificado los siguientes escenarios donde el trabajo desarrollado podría ser útil:

- Exploración de zonas rurales o forestales donde no hay cobertura de redes móviles.
- Exploración de entornos que han sufrido algún tipo de catástrofes donde las infraestructuras de comunicación han quedado destruidas y es necesario evaluar el terreno antes de la intervención humana.
- Exploración de zonas de difícil acceso y potencialmente peligrosas para el ser humano como pueden áreas contaminadas, instalaciones industriales abandonadas o cuevas [3].

Sin embargo, existen más casos de uso que puede tener este proyecto, como, por ejemplo:

- Equipo de vigilancia remoto para vigilar para cubrir una gran área. Como el robot está basado en Arduino se le pueden conectar distintos tipos de sensores de sensores como puede ser de movimiento o luminosidad e incluso una cámara para transmitir la información, aunque de acuerdo con la información recogida en la tabla 1.1 no se puede enviar una gran cantidad de información

con LoRaWAN [5].

- Comunicación con vehículos autónomos utilizando LoRaWAN de forma que los vehículos autónomos envíen su posición GPS al gateway más cercano para reducir el tráfico en las ciudades inteligentes [4]. Aunque esta propuesta solo utiliza la parte de la red LoRaWAN y el envío de la posición GPS es interesante tenerla en cuenta.
- Se puede utilizar en agricultura donde se tenga campos agrícolas que necesiten una red como LoRaWAN y utilizar el robot para tomar medidas mediante sensores y realizar tareas sencillas como regar o gestión de plagas.

1.6. Comparación de robots

Para este proyecto se podrían haber elegido una gran cantidad de robots para llevarlo a cabo. Sin embargo, se acabó eligiendo el mBot así que se va a comentar las características que tienen robots similares al mBot y también el porqué de la elección final.

En primer lugar, hay que ver que necesitamos del robot para llevar a cabo un proyecto como este:

- Que la placa del robot se pueda programar para que sea capaz de interpretar las instrucciones que reciba.
- Que la conexión entre la mota y el robot se pueda realizar, y a ser posible, de forma sencilla.
- Que disponga de sensores útiles para que pueda ser manejado desde una gran distancia como puede ser un sensor de ultrasonido o infrarrojos.
- Que sea capaz de moverse por entornos con obstáculos y terrenos irregulares.
- Que no tenga un coste muy elevado para esta prueba de concepto.

Teniendo en cuenta los requisitos que debe tener el robot se van a presentar a los candidatos que se pueden ver en las figuras 1.2 (mBot), 1.3 (Thymio), 1.4 (Romi), 1.5 (Maqueen Plus) y 1.6 (Zumo).

Figura 1.2: Robot mBot de Makeblock

Figura 1.3: Robot Thymio de Mobsya

Figura 1.4: Robot Romi de Pololu

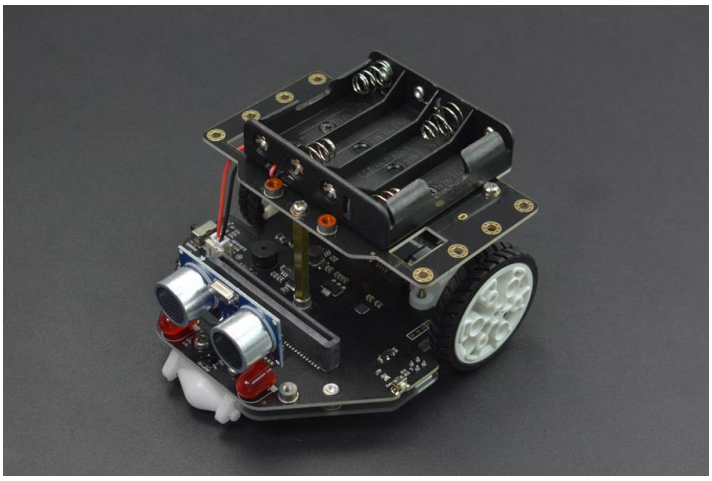


Figura 1.5: Robot Maqueen Plus de DFRobot.

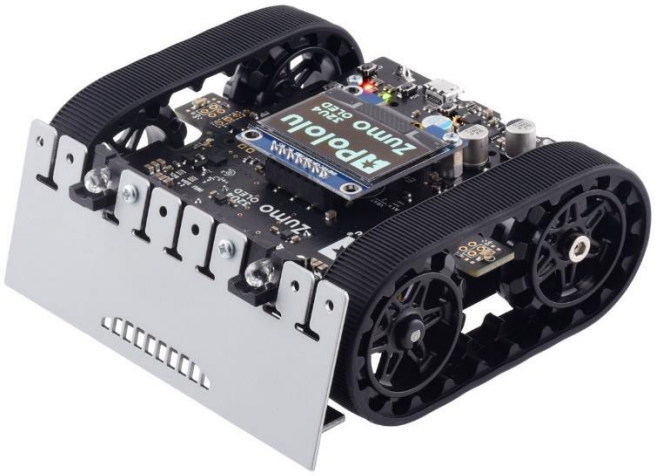


Figura 1.6: Robot Zumo de Pololu

En la tabla 1.2 se van a plasmar algunas de las características de los robots que son más relevantes para este proyecto:

Criterio	mBot (Makeblock)	Maqueen Plus (DFRobot)	Zumo (Pololu)	Thymio (Mobsya)	Romi (Pololu)
Plataforma base	Arduino UNO (ATmega328)	micro:bit/ESP32	Arduino/RPi Zero	Propietaria (VPL, ASEBA)	Arduino 32U4/Raspberry Pi
Programación	Scratch, Arduino	micro:bit, Arduino, Python	Arduino	ASEBA	Arduino, Python
Sensores integrados	IR, de línea, ultrasonidos, LED RGB	IR, LED, ultrasonidos, de luz	Depende del montaje	Temperatura, IR, ultrasonidos	Depende del montaje
Robustez (terrenos irregulares)	Baja (Chasis de plástico)	Media (base más estable)	Alta (ruedas de oruga y base metálica)	Baja	Media
Autonomía energética	Media (de 2 a 4 horas)	Alta (con packs de Li-ion)	Alta	Media	Alta
Coste aproximado	60 – 100 €	60 – 80 €	80 – 150 €	120 – 140 €	90 – 120 €

Tabla 1.2: Características de los distintos robots.

De acuerdo a las características vistas en la tabla 1.2 y los requisitos para el robot expuestos previamente se puede observar que todos estos robots son compatibles para este proyecto. Sin embargo, hay que tener en cuenta que se tiene más experiencia programando en Arduino, por lo que se descarta el robot de Thymio y que se busca que los costes sean mínimos, descartando los robots de Pololu (Zumo y Romi). Tanto el mBot como el Maqueen Plus son opciones que se adaptan perfectamente a este proyecto. Incluso el Maqueen Plus puede llegar a hacerlo mejor en la práctica ya que tiene una autonomía energética mayor y posee más robustez sobre el terreno, aunque, por otro lado, el mBot está disponible en la biblioteca de la ETSIIT, por lo que se decidió finalmente que se utilizaría el mBot.

Para esta sección se han utilizado las referencias [6], [7], [8], [9] y [10].

1.7. Estructura de la memoria

La memoria de este TFG está estructurada de la siguiente manera:

- En el **Capítulo 1: Introducción** hay un pequeño resumen sobre lo que consiste este proyecto, seguido de los objetivos que se planean conseguir, el contexto

sobre el que se va a trabajar y las motivaciones para llevarlo a cabo. Por último, se habla de algunos casos de uso a parte del enfoque principal del proyecto y de porqué se ha utilizado el mBot de entre varios candidatos.

- En el **Capítulo 2: Planificación y costes** se ha hecho en primer lugar una planificación temporal con las fases del proyecto, y después se ha estimado el coste total del proyecto en función de los recursos utilizados.
- En el **Capítulo 3: Fundamentos teóricos** se han hablado de los conceptos básicos necesarios para entender los fundamentos teóricos en los que está basado este proyecto.
- En el **Capítulo 4: Diseño e implementación** se va a hablar sobre la arquitectura del sistema, así como el diseño *hardware*, de *firmware* y del *backend*. Además, se va realizar la configuración en *The Things Network*.
- En el **Capítulo 5: Experimentos** se han analizado distintos experimentos para comprobar la funcionalidad del sistema planteado y cómo se comportaría en un entorno real.
- En el **Capítulo 6: Conclusiones y trabajo futuro** se han descrito las conclusiones que se han llegado al terminar este proyecto y se abre la vera a trabajos futuros que podrían mejorar este proyecto.

Capítulo 2

Planificación y presupuesto

En este capítulo se va a mostrar la planificación temporal que se planeó para completar este proyecto y también se analizarán todos los recursos empleados en el proyecto, ya sean de *hardware*, *software* o humanos y se realizará una estimación del coste total basada en esos datos.

2.1. Planificación temporal

En primer lugar, se va a describir las fases en las que se ha dividido este proyecto y posteriormente se mostrará un orden cronológico del desarrollo del proyecto:

- **Análisis del mBot**

  En esta primera fase se analizará al mBot: los componentes por los que está compuesto, la accesibilidad que se tiene a su placa y la capacidad para conectar componentes externos a los fabricados por *Makeblock*, las formas de conexión que permite y cómo se podría programar.

- **Estudio de la tecnología de LoRaWAN**

  Después se buscará información sobre LoRaWAN para conocer la tecnología sobre la que se va a realizar el proyecto, sobre todo conocer de donde ha surgido esta tecnología y los fundamentos teóricos que son necesarios para la elaboración del proyecto.

- **Estudio de artículos relacionados con LoRaWAN y con el proyecto**

  En tercer lugar, se hará un estudio de artículos que principalmente utilicen la tecnología de LoRaWAN para conocer el ámbito de uso de esta tecnología y también si se ha utilizado en proyectos parecido a este.

- **Análisis de la mota y gateway a utilizar**

  Después se analizará la mota, que es una Lilygo T-Beam SX1262, y el gateway,

que es un Pycom Pygate, para entender cómo funcionan y cómo utilizarlos en este proyecto.

- **Análisis de *The Things Network***

  Una vez comprendidos la mota y el gateway, se analizará la plataforma que utilizaremos para crear la red LoRaWAN, entendiendo cómo se añaden los dispositivos y comprendiendo las funcionalidades que ofrece.

- **Creación de la aplicación en Python para mandar las instrucciones**

  Posteriormente se creará la aplicación en Python que permite enviar las instrucciones que se quiere que realice el mBot a *The Things Network.*

- **Preparación del entorno para la programación de la mota y el mBot**

  Se preparará el entorno de trabajo instalando Arduino IDE con las librerías que son necesarias para la programación de la mota y el mBot.

- **Programación de la mota**

  Con las librerías instaladas, se procederá a la programación de la mota, la cual debe ser capaz establecer una comunicación bidireccional con *The Things Network* de forma que la mota envía la posición GPS y recibe las instrucciones que debe ejecutar el mBot, y además se tiene que programar la comunicación con el mBot.

- **Programación del mBot**

  Con la mota preparada para enviar las instrucciones al mBot, hay que programarlo para que este sea capaz de interpretar los mensajes que recibe de la mota y realizar la acción deseada.

- **Estudiar la conexión entre la mota y el mBot**

  Con la mota y el mBot preparados hay que estudiar qué posibilidades se tienen para realizar la conexión entre ambos y llevar a cabo la elegida finalmente.

- **Montaje del mBot con la mota**

Con la conexión entre ambos completada hay que hacer un montaje de forma que la mota y el mBot se puedan desplazar juntos sin que se puedan separar.

- **Realización de experimentos, toma de datos y discusión de los resultados obtenidos**

  Con todo el montaje finalizado se realizarán experimentos para probar si se han completado los objetivos impuestos y se discutirán los resultados en torno a los resultados obtenidos.

- **Redacción y revisión de la memoria**

  Por último, queda realizar la redacción de la memoria del proyecto donde se plasmará todo el recorrido que se ha seguido, se recogerán los datos obtenidos y se discutirán si se han alcanzado los objetivos del proyecto.

A continuación, se va a mostrar un diagrama de Gantt en la figura 2.1 donde está recogido desde cuando se empezó el proyecto hasta la finalización del mismo y también el tiempo dedicado a cada fase del mismo

Figura 2.1: Diagrama de Gantt creado con *GanttProject* [11].

En la tabla 2.1 se muestra recogido de forma más detallada el día de inicio y fin de cada una de las fases y en la tabla 2.2 la extensión temporal en horas de cada una de las fases.

Nombre	Fecha de inicio	Fecha de fin
Proyecto completo	15/11/24	13/6/25
Análisis del mBot	15/11/24	29/11/24
Estudio de la tecnología LoRaWAN	2/12/24	20/12/24
Estudio de artículos relacionados con LoRaWAN y con el proyecto	23/12/24	30/12/24
Análisis de la mota y el gateway	1/1/25	21/1/25
Análisis de The Things Network	23/1/25	4/2/25
Creación de la aplicación en Python para mandar instrucciones	5/2/25	18/2/25
Preparación del entorno para la programación de la mota y el mBot	20/2/25	27/2/25
Programación de la mota	3/3/25	28/3/25
Programación del mBot	1/4/25	14/4/25
Estudiar la conexión entre la mota y el mBot	16/4/25	29/4/25
Montaje del mBot con la mota	1/5/25	6/5/25
Realización de experimentos, toma de datos y discusión de los resultados obtenidos	8/5/25	20/5/25
Redacción y revisión de la memoria	22/5/25	13/6/25

Tabla 2.1: Fechas de inicio y fin detalladas de la planificación temporal.

Fase del proyecto	Horas
Análisis del mBot	20
Estudio de la tecnología LoRaWAN	35
Estudio de artículos relacionados con LoRaWAN y con el proyecto	15
Análisis de la mota y el gateway	20
Análisis de The Things Network	10
Creación de la aplicación en Python para mandar instrucciones	20
Preparación del entorno para la programación de la mota y el mBot	15
Programación de la mota	40
Programación del mBot	20
Estudiar la conexión entre la mota y el mBot	40
Montaje del mBot con la mota	10
Realización de experimentos, toma de datos y discusión de los resultados obtenidos	40
Redacción y revisión de la memoria	70
Tiempo total	355

Tabla 2.2: Extensión temporal en horas de cada una de las fases.

2.2. Costes del proyecto

En esta sección se hará una estimación del coste total en función de los recursos *hardware*, *software* y humanos utilizados en la realización de este proyecto.

2.2.1. Recursos *hardware*

Primero se va a hablar de los recursos *hardware* como componentes y dispositivos que se han utilizado:

- **Ordenador personal HP Pavillion Gaming Laptop 15**

  Este es el ordenador personal que se ha utilizado en el proyecto para la búsqueda de información y programación de los distintos dispositivos. Sus características técnicas se pueden ver en la tabla 2.3.

Modelo	HP Pavillion Gaming Laptop 15
Procesador	Intel Core i5-9300H (2.4 GHz)
Tarjeta gráfica	NVIDIA GeForce GTX 1050
Memoria RAM	8 GB
Almacenamiento	256 GB SSD
Sistema operativo	Windows 11 Home

Tabla 2.3: Especificaciones técnicas de ordenador personal

- **Teléfono personal Samsung Galaxy Note 10**

  Utilizado para comprobar las funcionalidades *Bluetooth* que tiene el mBot y para dar cobertura al gateway cuando se realicen experimentos al aire libre. Las especificaciones técnicas que tiene este dispositivo se pueden ver en la tabla 2.4.

Modelo	Samsung Galaxy Note 10
Procesador	Exynos 9825, 7nm
Memoria RAM	8 GB
Almacenamiento	256 GB
Sistema operativo	Android 12

Tabla 2.4: Especificaciones técnicas del teléfono personal [12].

- **Gateway Pycom Pygate**

  El gateway que utilizaremos en el proyecto es el Pycom Pygate y es el dispositivo que permite la comunicación entre *The Things Network* con la mota ya que es el encargado de dirigir los mensajes en ambos sentidos. Toda la información necesaria para ponerlo en funcionamiento se puede encontrar en [13].

Figura 2.2: Pycom Pygate utilizada en el proyecto.

- **Mota Lilygo T-Beam SX1262**

  Para este proyecto se utilizará la mota Lilygo T-Beam SX1262 ya que es una placa basada en ESP32 equipada con un chip LoRa SX1262, posee un módulo GPS para enviar su localización y permite utilizar la frecuencia de 868 MHz que emplean las redes LoRaWAN en Europa [14].

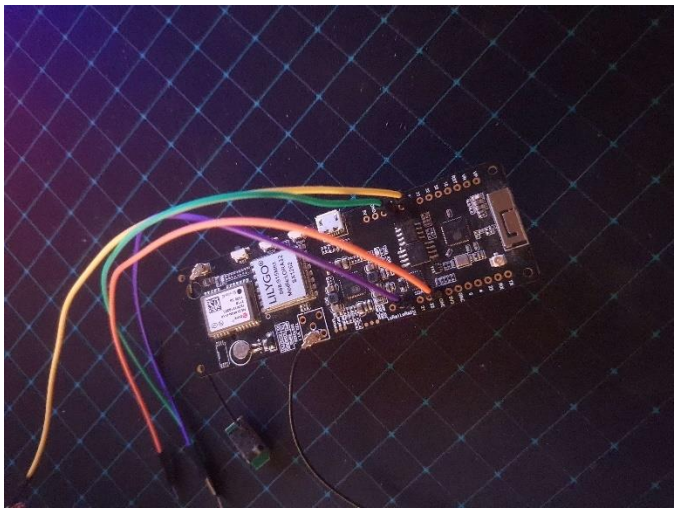


Figura 2.3: Placa Lilygo T-Beam utilizada en el proyecto.

- **mBot**

  Robot educativo que se utilizará para llevar a cabo el proyecto, siendo el dispositivo final que recibirá instrucciones y deberá realizar para cumplir con una serie de pruebas y experimentos para cumplir con los objetivos del proyecto. En la figura 1.2 se puede ver cómo es dicho robot [6].

- **Convertidor de nivel lógico bidireccional**

  Como se planteará más adelante es necesario un convertidor de nivel lógico bidireccional debido a que el mBot opera a 5 V y la mota a 3.3 V. El modelo que se utilizará es el *Sparkfun Logic Level Converter Bi-Directional* que se puede observar en la figura 2.4.

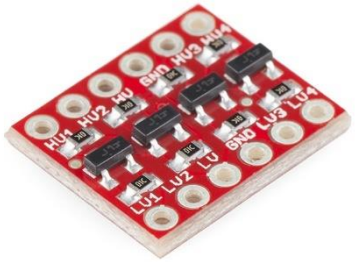


Figura 2.4: *Sparkfun Logic Level Converter Bi-Directional*.

- **Cables Dupont macho-macho**

  Como se explicará más adelante, la conexión entre el mBot y la mota se ha realizado con componentes físicos, utilizando este tipo de cables para conectar desde el mBot hasta el conversor lógico y del conversor lógico a la mota. Los cables utilizados se pueden observar en la figura 2.5.

Figura 2.5: Cables Dupont macho-macho.

Una vez comentados los recursos *hardware* que se han utilizado, en la tabla 2.5 se ha recogido el número de unidades que se han utilizado de cada uno y su precio, tanto de unidad como el total:

Componentes	Nº de unidades	Precio por unidad (€)	Precio total (€)
Ordenador personal	1	600,00	100 (8 meses)
Teléfono personal	1	191,00	191,00
Gateway	1	275,00	275,00
Mota	1	44,50	44,50
mBot	1	79,99	79,99
Convertidor lógico bidireccional	1	15,96	15,96
Cable Dupont	8	4,99 (pack de 40)	4,99
		Total	711,44

Tabla 2.5: Costes de los recursos *hardware*.

Para considerar los costes del ordenador personal se ha considerado que va a tener una esperanza de vida de unos 4 años y que el proyecto ha durado unos 8 meses. Para los cables Dupont como se compran por packs de 40 cables y se han utilizado 8 consideramos el coste del pack completo. Los precios se han obtenido de las siguientes referencias [14], [15], [16], [17], [18] y [19].

2.2.2. Recursos *software*

En esta sección se van a comentar los recursos *software* utilizados. Aunque en este caso no aportan nada para el coste total porque son programas gratuitos se van a comentar cuáles se han utilizado y el porqué.

- **Arduino IDE**

  Plataforma de desarrollo de software gratuita y de código abierto para programar en microcontroladores compatibles con Arduino. Como la mota está basada en un ESP32 y la placa del mBot en un Arduino Uno se ha utilizado este IDE para programar ambos [20].

- **Visual Studio Code**

  Editor de código desarrollado por Microsoft que utilizaremos para programar el gateway utilizando la extensión *PyMakr* [21].

- ***The Things Network***

  Es una plataforma de código abierto pensada para IoT y basada en la tecnología LoRaWAN. La utilizaremos como medio para conectar los dispositivos que componen la red LoRaWAN, que en nuestro caso son la mota y el gateway, y monitorizar los mensajes que se envían y reciben [22].

- **GanttProject**

  Aplicación de código abierto que se ha utilizado para diseñar el diagrama de Gantt mostrado en la figura 2.1 [11].

- **Word**

  Editor de texto que se ha utilizado para la redacción de la memoria.

- ***PyCharm Community***

  Editor de código para escribir en el lenguaje de Python utilizado para crear la aplicación que envía las instrucciones. Se ha utilizado la versión *Community* de este programa para no aumentar el coste del proyecto [23].

- ***Mermaid***

  Software utilizado para crear diagramas de flujo del código de los dispositivos [33].

Como se ha mencionado previamente, las aplicaciones que se han utilizado han sido todas obtenidas de manera gratuita, por lo que no contribuyen al coste total del proyecto.

2.2.3. Recursos humanos

Por último, vamos a considerar los recursos que se han empleado en el proyecto. Los recursos humanos hacen referencia al tiempo que ha empleado cada persona en el proyecto, y cada persona tendrá un valor por hora distinto. En la tabla 2.6 se han recogido las personas que han participado, el tiempo que ha invertido cada una, el precio por hora y el coste total de estos recursos:

Nombre	Tiempo invertido (h)	Precio por hora (€)	Coste (€)
Francisco Javier Cañizares Cancho	355	25,00	8875,00
Juan José Ramos Muñoz	20	50,00	1000,00
Jorge Navarro Ortiz	20	50,00	1000,00
		Total	10875,00

Tabla 2.6: Costes de los recursos humanos

Ahora que ya tenemos los costes de todos los tipos de recursos empleados en el proyecto se va a calcular el total que conlleva realizar este proyecto. Dicho coste se encuentra recogido en la tabla 2.7.

Tipo de recurso	Coste (€)
Hardware	711,44
Software	0
Humano	10875,00
	11586,44

Tabla 2.7: Costes totales del proyecto.

Capítulo 3

Fundamentos teóricos

En este capítulo se van a explicar los conceptos teóricos sobre los que se fundamenta este proyecto. En específico, se va a explicar los conceptos de LoRa, LoRaWAN, las bandas de frecuencia ISM y el protocolo *Message Queuing Telemetry Transport* (MQTT).

3.1. LoRa

LoRa (*Long Range*) es una modulación de espectro ensanchado que fue desarrollada por Semtech y deriva de la modulación *Chirp Spread Spectrum* (CSS), que fue creada en 1940 para aplicaciones de radar. Esta modulación utiliza señales *chirp* que varían su frecuencia continuamente provocando que los desfases temporales y de frecuencia entre emisor y receptor sean equivalentes, reduciendo así la complejidad del diseño del receptor. La señal de datos deseada se obtiene troceando a una velocidad de datos superior y modulando en la señal *chirp*.

La relación entre la tasa de bits deseada, la tasa de símbolos y la tasa de *chip* para la modulación LoRa viene dada por la siguiente expresión:

$$R_b = SF * \frac{Rate\ Code}{\frac{2^{SF}}{BW}}\ bits/s$$

Donde SF es el *Spreading Factor* cuyo valor varía está comprendido entre 7 y 12, BW es el ancho de banda de la modulación y $Rate\ Code = \frac{4}{4+CR}$, siendo CR *Code Rate* con un valor entre 1 y 4 [24]. Estos parámetros son configurables para optimizar el rendimiento según el escenario.

Algunas de las propiedades que posee la modulación LoRa son las siguientes:

- **Ancho de banda ampliable:** LoRa es una modulación que permite un ancho de banda y frecuencias escalables y puede ser utilizada para aplicaciones de frecuencia de banda estrecha y de banda ancha [24].
- **Baja potencia:** LoRa es un esquema de modulación de envolvente constante

lo que significa que permite utilizar amplificadores que tienen bajo coste y consumo energético [24].

- **Gran robustez:** Debido a la naturaleza asíncrona y alta relación BT (producto entre ancho de banda y tiempo) de LoRa es resistente a las interferencias dentro y fuera de la banda [24].
- **Resistencia a la multitrayectoria y desvanecimiento:** Como el pulso chirp es de banda ancha LoRa ofrece inmunidad a la multitrayectoria y desvanecimiento haciéndola ideal para entornos urbanos y suburbanos [24].
- **Resistencia al efecto Doppler:** El efecto Doppler tiene un efecto mínimo sobre la señal LoRa [24].
- **Largo alcance:** Proporciona un alcance muy amplio. Por ejemplo, en comparación con la modulación FSK, LoRa tiene 4 veces su alcance [24].
- **Mayor capacidad de red:** Como LoRa utiliza factores de dispersión ortogonales (SF) puede tener múltiples transmisiones simultáneas en el mismo canal sin degradar la sensibilidad del receptor [24].

Algunas de las aplicaciones típicas en las que se utiliza esta modulación son aquellas que requieren conectividad remota y bajo consumo, como lo hacen las aplicaciones de IoT. Algunos ejemplos son:

- En la agricultura inteligente con sensores de riego y monitoreo de la humedad del suelo
- En las ciudades inteligentes con control de alumbrado
- En el medio ambiente monitorizando la calidad del aire y la humedad para prevenir incendios forestales.
- En seguridad y defensa con sensores remotos.

Por último, LoRa opera en la capa física del modelo OSI, lo que permite integrarla con arquitecturas de nivel superior como es LoRaWAN.

3.2. LoRaWAN

El estándar LoRaWAN es una LPWAN diseñada para conectar “cosas” de forma inalámbrica a internet en redes regionales, nacionales o globales y busca cumplir con requisitos clave de IoT como son comunicaciones bidireccionales,

seguridad *end-to-end*, movilidad y localización de servicios [25]. Para ello, LoRaWAN utiliza bandas de frecuencia sin licencia (por ejemplo, la banda de 863-869 MHz en Europa) y sus mayores ventajas son gran alcance, de hasta 15 Km en zonas rurales, poco consumo energético y su capacidad para conectar muchos dispositivos a una misma red [25].

LoRaWAN opera en la capa de red y su arquitectura utiliza una topología en estrella donde los *gateways* retransmiten los mensajes entre dispositivos finales y el servidor central. Utilizando conexiones IP, el gateway convierte paquetes RF en IP y viceversa [25].

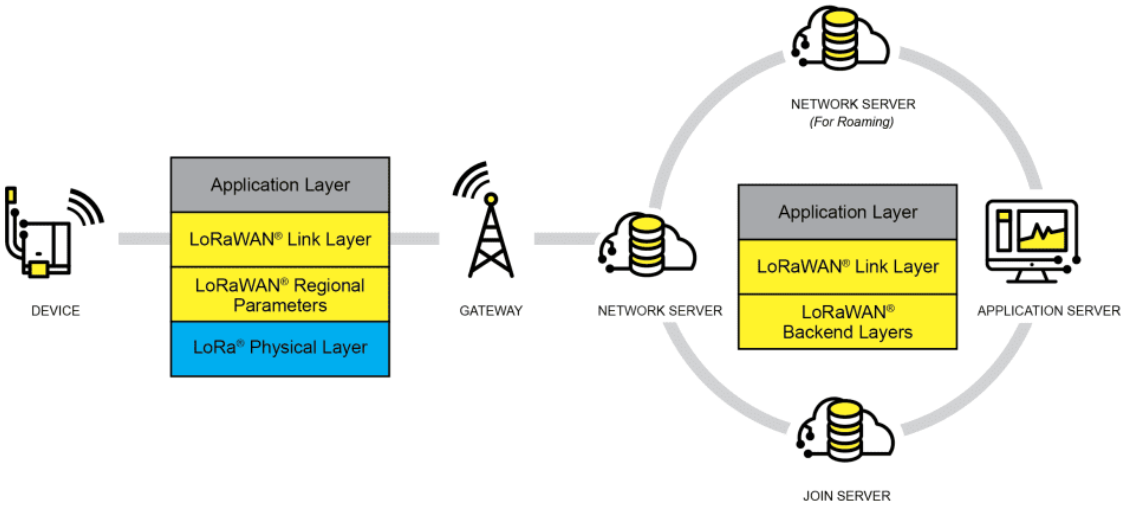


Figura 3.1: Arquitectura LoRaWAN [25].

Como se puede ver en la figura 3.1 [25] hay varios componentes principales en esta red:

- **Dispositivos finales o motas:** Transmiten datos a la red y pueden recibir comandos o configuraciones.
- ***Gateways*:** Se encargan de reenviar los mensajes como se ha comentado previamente.
- ***Network server*:** Es el corazón lógico de la red cuyas funciones son filtrar duplicados, verificar la integridad, realizar la gestión de red y reenviar datos a los servidores de aplicación.

- ***Application server*:** Es el receptor final de los datos de usuario donde se decodifican y procesan.
- ***Join server*:** Es el encargado de la activación de los dispositivos finales.

3.2.1. Modos de activación y seguridad

Para que un dispositivo final sea miembro de una red LoRaWAN tiene que ser personalizado y activado. Existen dos modos de activación para los dispositivos finales y, aunque solo es necesario utilizar uno de estos dos métodos, el dispositivo final puede implementar ambos.

En ambos modos de activación cuando el dispositivo final es permitido en la red debe almacenar las claves DevAddr que es la dirección del dispositivo, AppEUI que un identificador de aplicación, NwkSKey que es una clave de sesión de la red y AppSKey que es una clave de sesión de aplicación.

***Over-The-Air-Activation* (OTAA)**

Con el modo OTAA el dispositivo final debe seguir un procedimiento de unión antes de poder participar en el intercambio de mensajes. Además, cada vez que se pierda la información del contexto de la sesión deberá de realizar este procedimiento de unión de nuevo [26].

El dispositivo final tiene que estar personalizado con los siguientes identificadores antes de comenzar el procedimiento de unión:

- ***End-device identifier* (DevEUI):** Identificador global único del dispositivo final compuesto de 64 bits que identifica al dispositivo final en toda la red.
- ***Join-server identifier* (JoinEUI):** Es un ID de aplicación global compuesto por 64 bits que permite identificar al *Join server* que puede ayudar en el procedimiento de unión y derivación de claves de sesión.
- ***Application Key* (AppKey):** Es una clave AES-128 específica del dispositivo final que se utiliza para derivar las claves de sesión NwkSKey y AppSkey específicas de ese dispositivo final para cifrar y verificar la comunicación en la red y los datos de aplicación.

El procedimiento de unión consta de dos tramas:

- ***Join-Request frame*:** El procedimiento de unión siempre es iniciado por el dispositivo final mandando la trama *Join-Request* que sigue el formato de la figura 3.2 [26].

Size (octets)	8	8	2
Join-Request payload	JoinEUI	DevEUI	DevNonce

Figura 3.2: Trama *Join-Request* [26].

Esta trama contiene el JoinEUI y DevEUI además de DevNonce, el cual es un contador que inicia en 0 cuando el dispositivo final se enciende y se incrementa con cada *Join-Request* [26].

- ***Join-Accept*:** La red responde a la trama *Join-Request* con un *Join-Accept* si el dispositivo final ha sido permitido en la red. Es enviado como una trama *downlink* normal, pero utilizando *JOIN_ACCEPT_DELAY1* y *JOIN_ACCEPT_DELAY2* en vez de *RECEIVE_DELAY1* y *RECEIVE_DELAY2* [26]. El formato de la trama *Join-Accept* se puede ver en la figura 3.3 [26].

Size (octets)	3	3	4	1	1	(16) optional
Join-Accept payload	JoinNonce	NetID	DevAddr	DLSettings	RXDelay	CFList

Figura 3.3: Trama *Join-Accept* [26].

Esta trama está compuesta por JoinNonce que es un valor que no se repite y que se utiliza por el dispositivo final para derivar las claves de sesión NwkSKey y AppSKey, NetID que es un identificador de red, el identificador DevAddr, DLSettings que está compuesto por los valores que establecen el desfase entre la velocidad de transmisión de datos en los enlaces ascendentes y descendentes en las dos ventanas de recepción (RX1 y RX2), RXDelay que que fija el retraso que tiene el enlace descendente en la ventana RX1 y CFList que es un parámetro que puede o no estar presente [26].

***Activation by Personalization* (ABP)**

Este modo de activación empareja directamente a un dispositivo final con una red sin tener que realizar el procedimiento de unión descrito en OTAA. Eso significa que las claves que normalmente se guardan en el dispositivo final cuando es permitido en la red como DevAddr, NwkSKey y AppSKey deben estar almacenadas en él directamente, lo que significa que el dispositivo final está preparado para participar en esa red en cuanto es iniciado [26].

Cada dispositivo final debe de tener su propia pareja de NwkSKey y AppSKey para no comprometer la seguridad de las comunicaciones con otros dispositivos finales.

Seguridad

En cuanto a la seguridad, una vez entendida la explicación de ambos modos de activación, se puede ver que OTAA es bastante más segura que ABP debido a que las claves son guardadas cuando se completa el procedimiento de unión ya que las claves se derivan de ese proceso. Sin embargo, ABP ya las lleva configurada de serie al no haber un proceso de unión dinámico, lo que lo hace más vulnerables a distintos tipos de ataques como ataques por repetición, clonación de dispositivos, ataques por modificación de claves o *sniffing* y análisis del tráfico.

3.2.2. Tipos de dispositivos y estrategias de comunicación:

En LoRaWAN, la eficiencia energética, la disponibilidad de los dispositivos y la gestión del espectro están directamente relacionadas. Es por ello que se definen tres clases de operación para el dispositivo final que permiten adaptar el comportamiento del mismo dependiendo de sus necesidades de comunicación, consumo y latencia. Además, estas clases están pensadas para equilibrar el uso del canal, la sincronización y respuesta a eventos externos. El mecanismo que utiliza LoRaWAN para reducir el consumo energético reducir el tiempo que los dispositivos finales están realizando alguna acción. Para ello, LoRaWAN implementa dos ventanas de recepción (RX1 y RX2) que son los únicos momentos en los que el dispositivo final está escuchando para recibir paquetes. Si en la ventana RX1 no ha recibido ningún paquete abre la ventana RX2. En la figura 3.4 [26] se puede observar el

funcionamiento de estas ventanas de recepción.

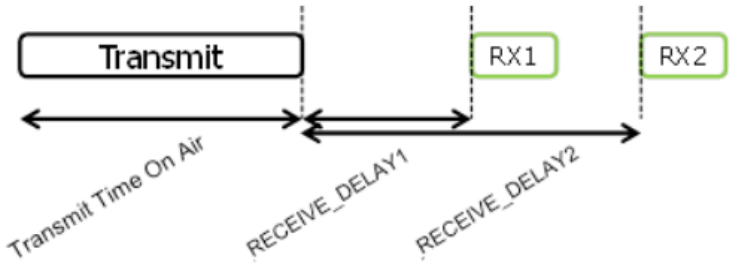


Figura 3.4: Ventanas de recepción [26].

La primera ventana de recepción (RX1) utiliza una frecuencia que es función de la frecuencia utilizada en el *uplink* y una velocidad de datos que es función de la velocidad de datos utilizada en el *uplink* (La velocidad de datos del *uplink* y de RX1 depende de cada región). Esta ventana no está abierta más de lo que indica el parámetro *RECEIVE_DELAY1* segundos después de finalizar la transmisión *uplink* [25].

La segunda ventana (RX2), si se abre, utiliza una frecuencia y velocidad de datos configurables (dependiendo de la región), y no puede estar abierta más de lo que indica el parámetro *RECEIVE_DELAY2* en segundos después de la finalización de la transmisión del mensaje *uplink* [26].

Las clases de operación son las siguientes:

Clase A

Esta clase de operación la implementan todos los dispositivos por defecto y funciona de forma que implementa una comunicación bidireccional con las ventanas de recepción que se ha comentado previamente. Esta clase se ideó para aplicaciones en las que el dispositivo final sólo necesita recibir información del servidor cuando termina de enviar sus datos.

Clase B

Esta clase funciona de forma parecida a la clase A pero, además de las ventanas RX1 y RX2, esta clase puede abrir más ventanas de recepción adicionales

sincronizadas utilizando balizas que envía el gateway de forma periódica para sincronizar todos los dispositivos en la red de forma que el dispositivo final pueda abrir una breve ventana de recepción adicional (*ping slot*) en un momento predecible [26]. En la figura 3.5 [26] se muestra de forma gráfica cómo funciona el ping slot.

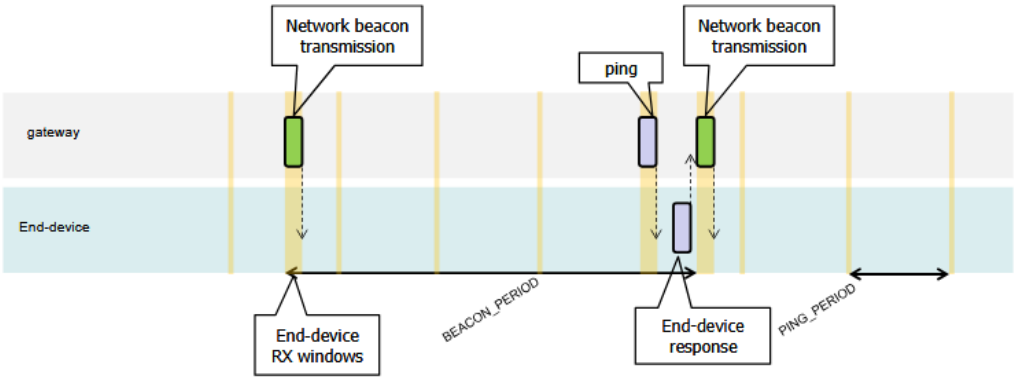


Figura 3.5: Funcionamiento del *ping slot* [26].

Clase C

En esta clase, el dispositivo final siempre mantiene activo el receptor, excepto en la transmisión del *uplink*. Esto significa que los dispositivos finales que implementan esta clase puedan recibir mensajes en todo momento, eliminando la latencia descendente. Por otro lado, al mantener activo su receptor en todo momento, el consumo energético aumenta enormemente en comparación con la clase A y B.

Como se puede observar en lo comentado en las clases de operación, los dispositivos finales transmiten LoRaWAN transmiten sin verificar si el canal está libre porque LoRaWAN utiliza un esquema de acceso basado en ALOHA puro. Aun así, utiliza técnicas para compensar esta limitación:

- ***Spreading Factor* (SF):** Los paquetes transmitidos que utilizan un valor distinto de SF (valor comprendido entre 7 y 12 como se ha mencionado en el apartado 3.1) pueden coexistir en el mismo medio sin interferirse significativamente.
- ***Adaptative Data Rate* (ADR):** El *Network server* puede ajustar dinámicamente el SF, el ancho de banda y la potencia de transmisión del dispositivo final, lo que optimiza el uso del espectro y extiende la vida útil del dispositivo.

3.3. Bandas de frecuencia ISM

Las bandas de frecuencia ISM son rangos específicos del espectro radioeléctrico reservados internacionalmente para aplicaciones industriales, científicas y médicas, que no requieren licencia previa para su uso. Están reguladas por la Unión Internacional de Telecomunicaciones (UIT) a través del artículo 5 del Reglamento de Radiocomunicaciones (ITU-R) [27]. Aunque principalmente fue concebido para equipos como microondas, láseres médicos o calentadores de RF, en la actualidad también se utilizan para las comunicaciones corto y medio alcance como son redes WiFi (2,4 GHz y 5 GHz), *Bluetooth* o RFID y tecnologías de baja potencia y largo alcance como es LoRa/LoRaWAN.

Dependiendo de la región en la que se esté la banda ISM utilizada por LoRa y LoRaWAN cambia. En la tabla 3.1 se muestra la banda ISM autorizada, la frecuencia central típica y ancho de banda permitido en cada región para LoRaWAN.

Región	Banda ISM	Frecuencia central típica (MHz)	Ancho de banda permitido (KHz)
Europa	EU868	868 - 868,6	125 / 250 / 500
Estados Unidos	US915	902 - 928	125 / 500
Asia	AS923	915 - 928 / 923	Variable según el país
India	IN865	865 – 867	125

Tabla 3.1: Reglamento para las bandas ISM según la región [28].

En Europa, las normativas que debe seguir la banda ISM están reguladas por la *European Telecommunications Standards Institute* (ETSI) y las principales características que hay que seguir son las siguientes [29]:

- *Duty cycle*: Es el porcentaje máximo de tiempo que un dispositivo puede transmitir en una frecuencia específica en el periodo de 1 hora. Dependiendo de la subbanda el *duty cycle* varía: en la banda de 868 – 868,6 MHz es de 1 %, en la banda de 868,7 – 869,2 MHz es de 0,1 % - 0,01 % y en la banda de 869,4 – 869,65 MHz es de un 10 %.

- Potencia máxima de transmisión: Por defecto es de 14 dBm (25 mW) aunque en algunas subbandas se permiten hasta 27 dBm (500 mW) con restricciones adicionales.
- Separación de canales: La normativa especifica que los canales estén suficientemente espaciados. Como los canales suelen ser de 125 KHz o 250 KHz se recomienda una separación mínima de 200 KHz con canales adyacentes.
- Técnicas de mitigación de interferencias: La ETSI exige que los dispositivos finales incorporen técnicas para reducir el impacto sobre otros usuarios. Algunas de estas técnicas son:
    - ***Listen Before Talk* (LBT):** El dispositivo escucha antes de transmitir, aunque solo es obligatoria si supera el *duty cycle* o en ciertas subbandas.
    - ***Adaptative Frequency Agility* (AFA):** Cambio diámico de canal para evitar colisiones o congestión.
    - ***Time-on-Air Control*:** El firmware del dispositivo debe evitar que la transmisión exceda los límites de la duración establecidos.

3.4. *Message Queuing Telemetry Transport* (MQTT)

El protocolo MQTT es un protocolo de mensajería ligero que fue diseñado específicamente para dispositivos con recursos limitados y redes con poca fiabilidad o alto retardo. Fue creado en 1999 y hoy en día está estandarizado por *Organization for the Advancement of Structured Information Standards* (OASIS).

MQTT emplea una arquitectura cliente-servidor con un modelo de comunicación de publicar/suscribir para que sea ligero, abierto, simple y diseñado para una fácil implementación, lo que lo hace ideal para las comunicaciones máquina a máquina e IoT [30]. Los componentes principales de este protocolo son:

- *Broker* (Intermediario): Es un servidor central que se encarga de recibir publicaciones, filtrarlas por tema (*topic*) y enviarlas a los clientes suscritos.
- Clientes MQTT: Los clientes MQTT pueden realizar diversas acciones

como realizar publicaciones a un *topic* o suscribirse a un *topic* para recibir mensajes de él. Un mismo cliente puede actuar tanto de publicador como de suscriptor.

No se va a entrar mucho en detalle del funcionamiento interno del protocolo como es la información de los mensajes que utiliza, pero si se va a hablar de los niveles de calidad de servicio (QoS) que tiene definidos dependiendo de los requisitos necesitados. Son tres los niveles de calidad de servicio:

- **"Como mucho una vez" (QoS = 0):** La entrega no está garantizada ya que el mensaje solo se entrega una vez. Por tanto, si el mensaje se pierde el suscriptor no lo recibirá. Este nivel es utilizado por aquellos que periódicamente reciban actualizaciones y no sea un problema grave perder una de ellas.
- **"Por lo menos una vez" (QoS = 1):** Se garantiza que el suscriptor reciba el mensaje, pero puede llegar a entregarse más de una vez. Cuando se reciba el mensaje se tiene que responder con un mensaje *PUBACK*.
- **"Exactamente una vez" (QoS = 2):** La entrega está garantizada sin duplicados, pero tiene una mayor complejidad y coste. Este nivel de calidad de servicio se utiliza en aplicaciones que no permiten redundancia ni pérdida de paquetes.

Capítulo 4

Diseño e implementación

En este capítulo se va a explicar la arquitectura del sistema propuesto, se va a mostrar el diseño *hardware* y la implementación *firmware* que se ha llevado a cabo, desde el diseño del *backend* y junto con la configuración de *The Things Network* hasta la implementación en los dispositivos.

4.1. Arquitectura del sistema

La arquitectura del sistema propuesta está compuesta por varios bloques, basada en una arquitectura distribuida:

1. **Aplicación de usuario:** Cliente de control remoto desde donde se envían las instrucciones al robot. Utiliza el protocolo MQTT para comunicarse con TTN.
2. ***The Things Network* (TTN):** Plataforma LoRaWAN pública encargada de gestionar los mensajes entre el dispositivo final (Lilygo T-Beam) y la aplicación de usuario.
3. ***Gateway* (Pycom Pygate):** Dispositivo encargado de retransmitir los mensajes tanto en el enlace ascendente como descendente, habilitando la comunicación entre el dispositivo final y el servidor LoRaWAN.
4. **Dispositivo final (Lilygo T-Beam):** Dispositivo final de la red LoRaWAN que recibe los mensajes de TTN, los decodifica y los transmite vía UART hacia el mBot.
5. **Conversor de nivel lógico *Sparkfun*:** Permite la comunicación UART de forma segura entre el dispositivo final y el mBot ya que trabajan a distinto voltaje.
6. **Robot mBot:** Robot móvil encargado de interpretar y realizar las instrucciones.

A continuación, se muestra en la figura 4.1 un diagrama de la arquitectura completa junto con el flujo de datos que tiene.

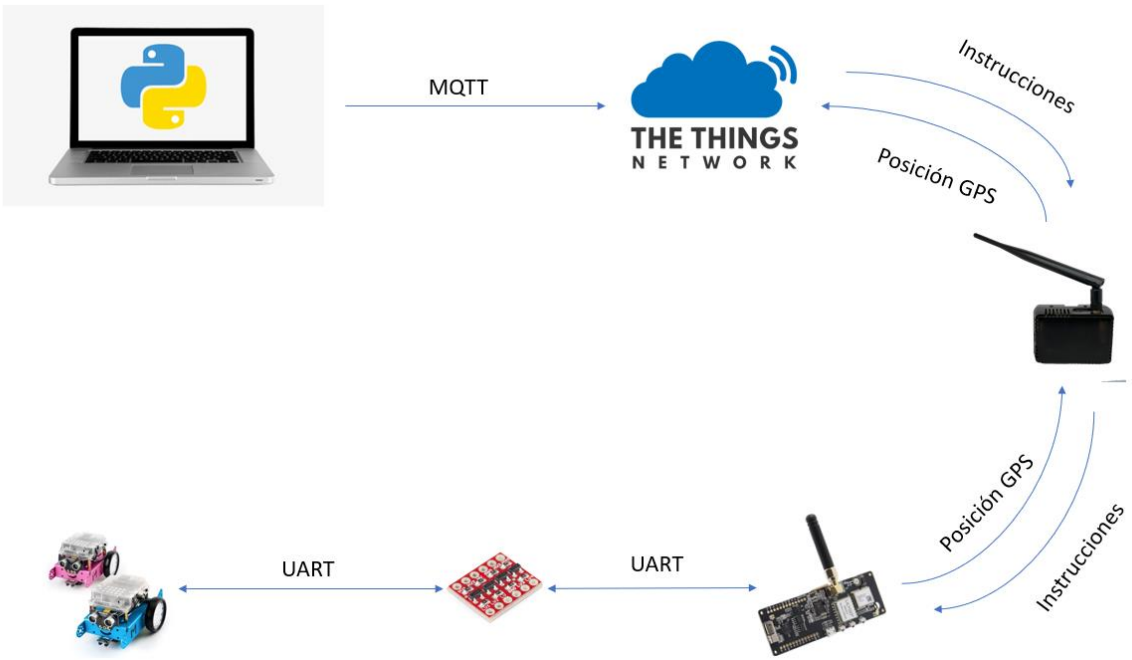


Figura 4.1: Diagrama de flujo.

Posteriormente se hablará más en detalle cómo se han utilizado estos protocolos en la programación de los dispositivos, pero el diagrama de la figura 4.1 nos permite ver la comunicación que se da entre los distintos bloques. Para comenzar, vemos que la aplicación de usuario tiene una comunicación unilateral con TTN debido a que en este sistema la aplicación de usuario actuará como cliente MQTT que publicará los tópicos (que serán las instrucciones que queramos enviar) en el *bróker*, el cual es TTN. Después se observa la comunicación bidireccional que se da entre TTN y la mota, pasando por el *gateway*, donde los mensajes que envía TTN serán las instrucciones y los que envía la mota su posición GPS. Por último, se observa que la comunicación entre la mota y el mBot se realiza utilizando el protocolo UART que permite crear una conexión física entre ambos dispositivos. Los mensajes que envía la mota son las instrucciones del mBot.

Hay que tener en cuenta que esta arquitectura permite cumplir con el objetivo general del proyecto pero que presenta limitaciones técnicas que lo condicionan, las cuales son:

- **Latencia en los mensajes en *downlink*:** La mota está configurada para funcionar como clase A y, como se ha visto anteriormente, esta clase solo abre ventanas de recepción cuando hace una transmisión, lo que impide la

recepción en tiempo real y puede causar retrasos en la ejecución de órdenes críticas.

- ***Duty cycle* limitado:** Como se ha comentado previamente, el *duty cycle* está limitado por la ETSI.
- **Capacidad de datos reducida:** LoRaWAN está diseñado para que el *payload* de los mensajes sea pequeño (en SF 12 es menor a 51 bytes), lo que condiciona la estructura de los mensajes. Además, no permite transmitir ni imágenes ni video.
- **Fiabilidad del canal LoRa:** Aunque LoRa es robusto ante el ruido se pueden dar colisiones por ALOHA puro (sobre todo en redes densas), interferencias en entornos urbanos debido a tecnologías que utilicen esas bandas ISM y pérdidas de paquetes si el *gateway* no tiene buena sensibilidad o se encuentra entre obstáculos físicos.
- **Dependencia del GPS:** El módulo GPS de la mota impone limitaciones ya que cuando se enciende tiene un tiempo de fijación hasta obtener la señal GPS, poca fiabilidad en zonas que no tienen acceso directo al cielo y si no se hace una buena gestión puede aumentar considerablemente el consumo energético de la mota. Además, si la mota no es capaz de obtener la posición GPS no podrá enviar su mensaje, por lo que no se abrirían ventanas de recepción.
- **Limitaciones del mBot:** Como el mBot es un robot pensado para fines educativos presenta limitaciones en entornos exteriores reales que tengan terrenos irregulares y obstáculos.
- **Limitaciones con TTN:** Como el sistema depende de la disponibilidad de TTN que es una red pública tiene limitaciones como falta de cobertura (porque puede que no haya *gateways* por esa zona), añade retardo debido al enrutamiento y verificación de paquetes y limitaciones por utilizar la versión gratuita.

4.2. Diseño *hardware*

Se va a hacer un análisis sobre que formas se pueden conectar la mota y el mBot y como se ha llevado a cabo la que se ha elegido, teniendo en cuenta algunas consideraciones, el esquema de la conexión y las limitaciones y posibles mejoras que se podrían tener en cuenta.

4.2.1. Análisis de conexión entre mBot y mota

Para conocer las opciones que tenemos para conectar el mBot y la placa se va a estudiar primero la placa del mBot (denominada mCore). En la figura 4.2 se observa la placa del mBot.

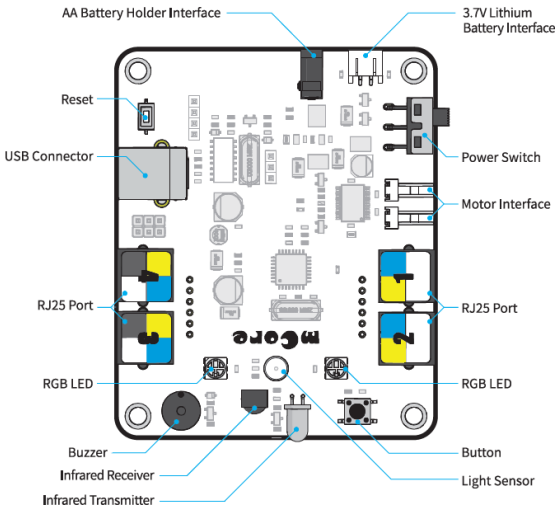


Figura 4.2: Estructura del mCore [31].

En la figura 4.2 [31] se observa una estructura simple del mCore en donde se encuentra la posición de los sensores que tiene, aunque para realizar la conexión entre la mota y el mBot lo único que podemos tener en cuenta son los puertos RJ25, que posibilitan la opción de realizar una conexión por I2C con la mota. Sin embargo, tenemos más opciones para realizar la conexión como se muestran en la figura 4.3, en la cual se muestra una foto tomada del mCore donde se mostrarán componentes más detallados, como son el módulo *Bluetooth* (situado en la parte superior de la placa y recubierto por un rectángulo rojo) que habilita la posibilidad de realizar una conexión *Bluetooth*, y donde se pueden observar que hay pines digitales (situados al lado de los puertos RJ25 y recubiertos de rectángulos rojos) que permiten la creación de un conexionado físico utilizando cables *Dupont*.

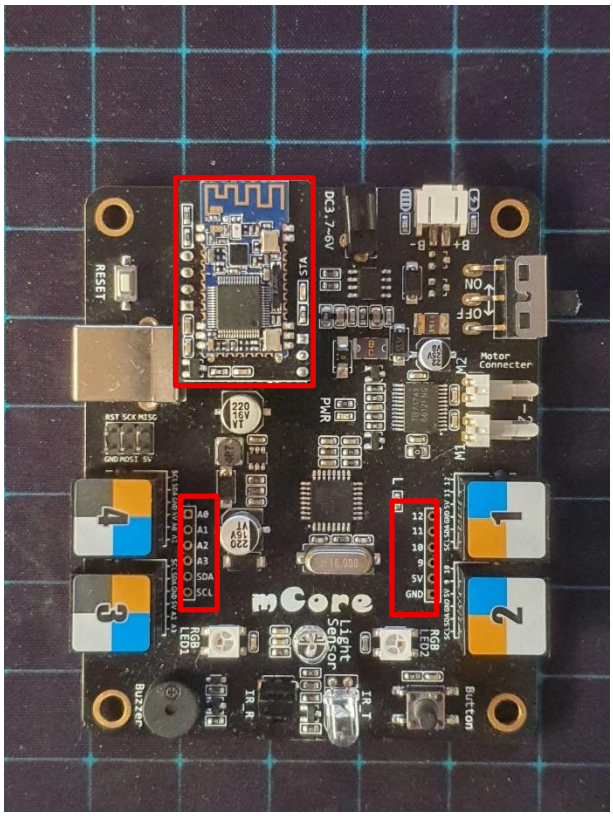


Figura 4.3: Foto tomada de la placa mCore.

En la figura 4.4 se muestra una foto tomada de la mota para considerar las opciones que tenemos para realizar la conexión.

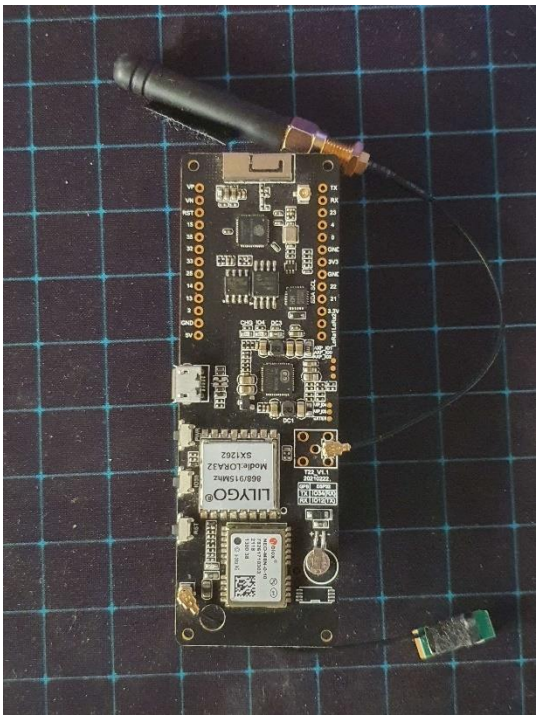


Figura 4.4: Foto tomada de la mota Lilygo T-Beam

Resumiendo, estas son las opciones que tenemos para realizar la conexión mBot-mota:

- **Conexión I2C:** La placa mCore tiene los pines SCL (línea de reloj) y SDA (línea de datos) propios del protocolo I2C integrados en los puertos RJ25 del mCore. No obstante, como se observa en la figura 4.4, la mota no se posee estos pines, por lo que la conexión I2C queda descartada.
- **Conexión por *Bluetooth*:** La placa mCore tiene módulo *Bluetooth* como se ha comentado previamente y la mota, al estar basada en un microcontrolador ESP32, incluye soporte para *Bluetooth* y *Bluetooth Low Energy* (BLE). Para comprobar que tipo de *Bluetooth* implementa el mCore se ha utilizado la aplicación *nRF Connect* que detecta las comunicaciones BLE disponibles. Al utilizar esta aplicación se observa que se puede conectar con el mCore como se muestra en la figura 4.5.

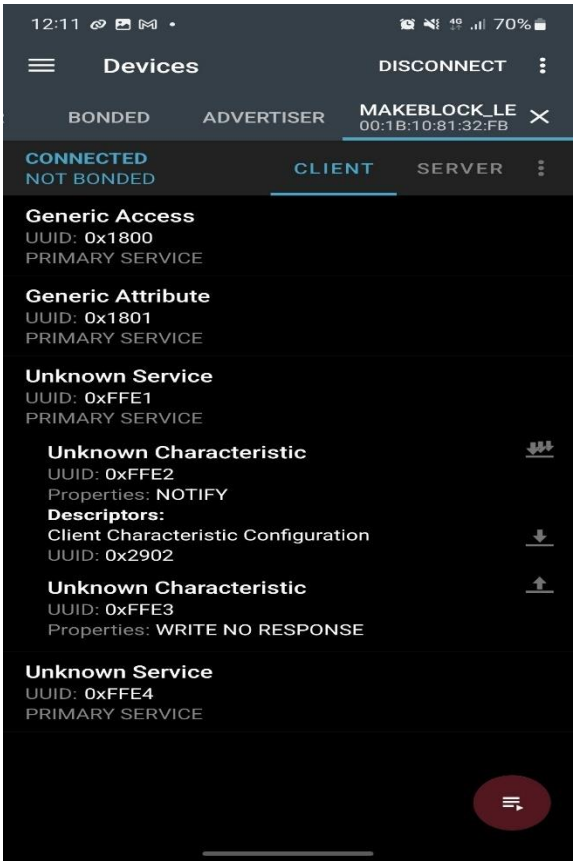


Figura 4.5: Conexión BLE con *nRF Connect*.

En la figura 4.5 se muestran los servicios y características que permite esta conexión *Bluetooth*. Utilizando las librerías de Arduino para BLE como son *BLEDevice.h, BLEUtils.h, BLEScan.h, BLEClient.h* y *BLEAdvertisedDevice.h*

se podría programar el *firmware* de la mota para la conexión BLE. No obstante, observando la figura 4.5, se ve que hay servicios y características desconocidas que son los utilizados para mandar las instrucciones al mBot ya que tienen propiedades *WRITE NO RESPONSE*. Si se intenta enviar algún mensaje a esas características la conexión BLE se pierde automáticamente y el mBot no los recibe. Para intentar solucionar este problema habría que contactar con el fabricante *Makeblock* y pedirle cómo se podría cambiar el firmware del módulo *Bluetooth*.

- **Conexión por UART:** Como se ha observado en las figuras 4.3 y 4.4, ambos dispositivos poseen de pines digitales, por lo que utilizando cables *Dupont* es posible realizar una conexión física. No obstante, hay que tener en cuenta que la mota, para la conexión con TTN, utiliza una gran cantidad de pines, por lo que hay que tenerlo en cuenta para escoger los que no estén siendo utilizados. Además, también hay que considerar que el mBot y la mota tienen distintos niveles de operación (siendo 5 V el mBot y 3,3 V la mota). Por ello, habría que utilizar un dispositivo que sea capaz de convertir la tensión de 5 V a 3,3 V y viceversa.

De acuerdo con las tres opciones presentadas, finalmente se ha escogido la opción de conexión por UART para realizar la conexión mBot-mota debido a que por cómo están diseñados las placas la conexión por I2C no es posible y por *Bluetooth* hay que modificar el *firmware* del módulo *Bluetooth*, añadiendo complejidad extra a esta prueba de concepto.

4.2.2. Cableado final entre mBot y mota

Como se ha elegido la conexión por UART hay que conocer el número de pines que se van a utilizar y cuáles de ellos, así como el número de cables necesarios y cómo se conectan.

Primero se va a comentar el número de pines. Cómo el objetivo es que la comunicación entre el mBot y la mota sea bidireccional, se necesitarán por lo menos 2 pines, uno de transmisión y otro de recepción, en cada uno de los dispositivos. Por otro lado, como se ha comentado previamente trabajan a distintas tensiones, y para utilizar el conversor de nivel lógico se requiere además que cada dispositivo utilice dos

pines extras, uno que será una tierra común entre dispositivos y el otro que será el pin de 5 V del mBot y el pin de 3,3 V de la mota. Por tanto, se van a utilizar en total 4 pines en cada dispositivo, con un total de 8 cables ya que se necesitan 4 cables para conectar de los pines del mBot al conversor y otros 4 para conectar los pines de la mota al conversor.

En cuanto a cuáles de ellos por la parte del mBot tenemos todos los pines libres, por lo que se van a utilizar el pin 10 para recepción (Rx) y el pin 11 para transmisión (Tx). También, como se ha comentado antes, se utilizará el pin de 5 V y un pin de GND. Por parte de la mota hay que revisar la configuración del *firmware* de la mota, donde se encuentra que los pines 13 y 25 están libres. Por ello, se utilizará el pin 25 para recepción y el pin 13 para transmisión. En la figura 4.6 se muestra gráficamente la conexión entre estos dispositivos.

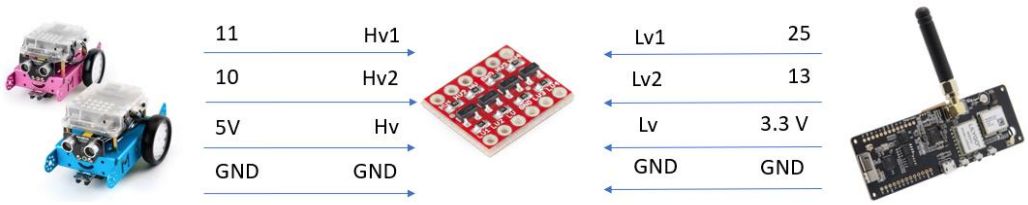


Figura 4.6: Esquema de conexiones de forma gráfica.

Cabe mencionar que los cables *Dupont* que se utilizarán tienen ambos extremos macho.

4.2.3. Limitaciones y posibles mejores

Se van a comentar algunas limitaciones que tiene este diseño *hardware,* tanto en la integración física como en el rendimiento de los componentes:

- **Montaje provisional y poco robusto:** Para conseguir que ambos dispositivos se puedan desplazar a la vez se ha utilizado cinta de doble cara para que queden fijados, lo que puede causar desconexiones del cableado físico. En la figura 4.7 puede verse como ha quedado finalmente este montaje.
- **Exposición de la mota:** Como la mota no cuenta con una carcasa protectora, está expuesta a polvo, humedad y posibles golpes durante el desplazamiento del mBot, lo que puede provocar daños a esta placa.
- **Autonomía limitada por baterías:** Ambas placas dependen de baterías independientes. El mBot está alimentado por 4 baterías AA mientras que para

alimentar la mota se ha utilizado una batería portátil de 10000 mA. El mayor problema de utilizar baterías independientes entre las placas es que posiblemente una de ellas se agota antes.

- **Limitaciones físicas del mBot:** El mBot no está diseñado para transportar componentes adicionales a los creados por el fabricante *Makeblock*, lo que significa que el tamaño para transportar la mota es muy reducido.
- **Precisión de las antenas:** La orientación de las antenas en la mota es crítica para el funcionamiento del sistema, y en el montaje realizado (figura 4.7) se observa que las antenas no están en la posición óptima para su correcto funcionamiento.

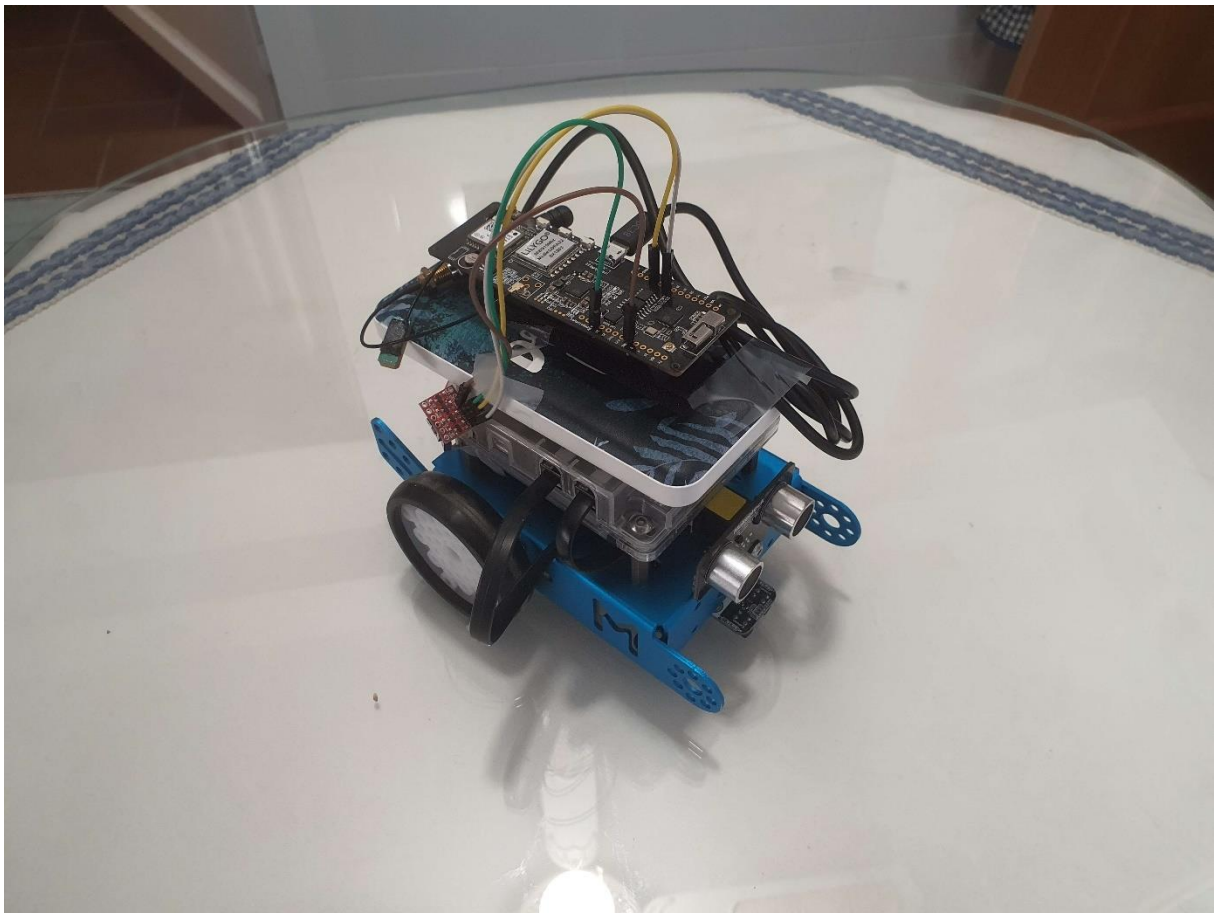


Figura 4.7: Montaje completo.

Algunas mejores que podrían mejorar o eliminar estas limitaciones son:

- **Diseño de una carcasa para la mota:** Se podría diseñar una carcasa para lo mota e imprimirla con una impresora 3D para mejorar la fijación con el mBot y proteger la mota de los peligros comentados.
- **Mejoras para la alimentación:** Se podría mejorar la alimentación de los

dispositivos utilizando una fuente de energía común, evitando que uno de los dispositivos muera primero y también simplificaría la carga. Además, se podrían añadir sensores que permitan monitorizar la batería para que la mota pudiese transmitir, junto con la señal GPS, la batería de los dispositivos.

- **Diseñar soporte para antenas:** Se puede diseñar un soporte para las antenas GPS y LoRa para que estas mantengan una posición óptima independientemente de los movimientos realizados por el mBot.
- **Fijación del cableado:** Se podría soldar el cableado para evitar que con el movimiento del mBot se desacoplen los cables.

4.3. Implementación *firmware*

En este apartado se va a mostrar la implementación del *backend* junto con el envío de mensajes de forma detallada que se da en este bloque, la configuración que se ha utilizado en *The Things Network*, la implementación que se ha realizado en la mota para realizar la conexión con la red LoRaWAN y la transmisión de datos al mBot y la implementación de la lógica del mBot para recibir e interpretar los comandos recibidos. Para ello se van a mostrar parte del código utilizado, aunque el código completo utilizado para cada caso se puede encontrar aquí [34].

4.3.1. Implementación del *backend*

Como se ha mencionado previamente, la aplicación de usuario ha sido desarrollada en *Python*. Las librerías que se han utilizado son las siguientes:

- **Tkinter:** Permite crear la interfaz gráfica que utilizará el usuario para pulsar la instrucción que se desee enviar.
- **Paho MQTT:** Permite mandar las instrucciones utilizando el protocolo MQTT. Para ello, la aplicación de usuario crea un cliente MQTT que publica tópicos en el *bróker* (TTN).
- **Base64:** Permite codificar los mensajes en base 64. Es necesario para obtener los mensajes compatibles con TTN.
- **JSON:** Permite estructurar los mensajes enviados a TTN.

En primer lugar, se va a describir la configuración inicial de la aplicación. En la figura 4.8 se observa dicha configuración:

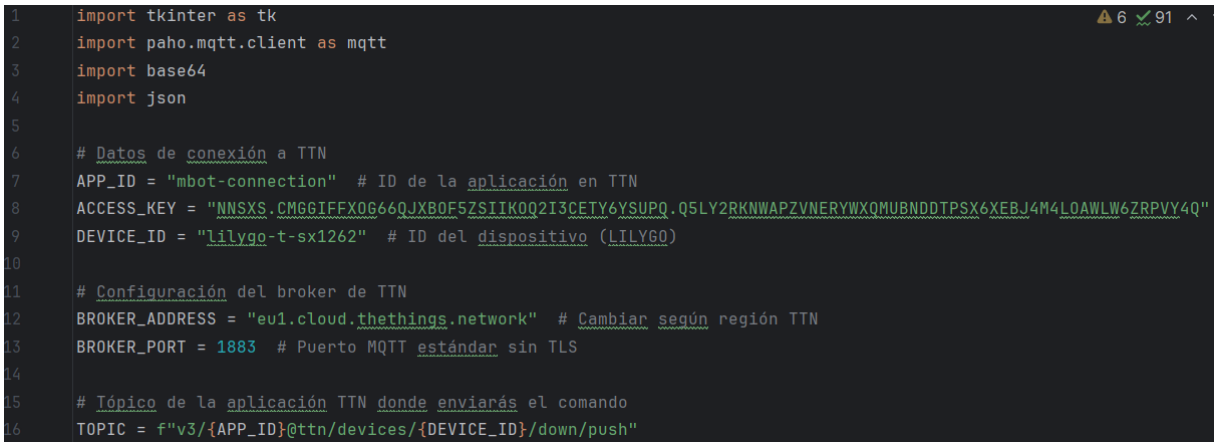
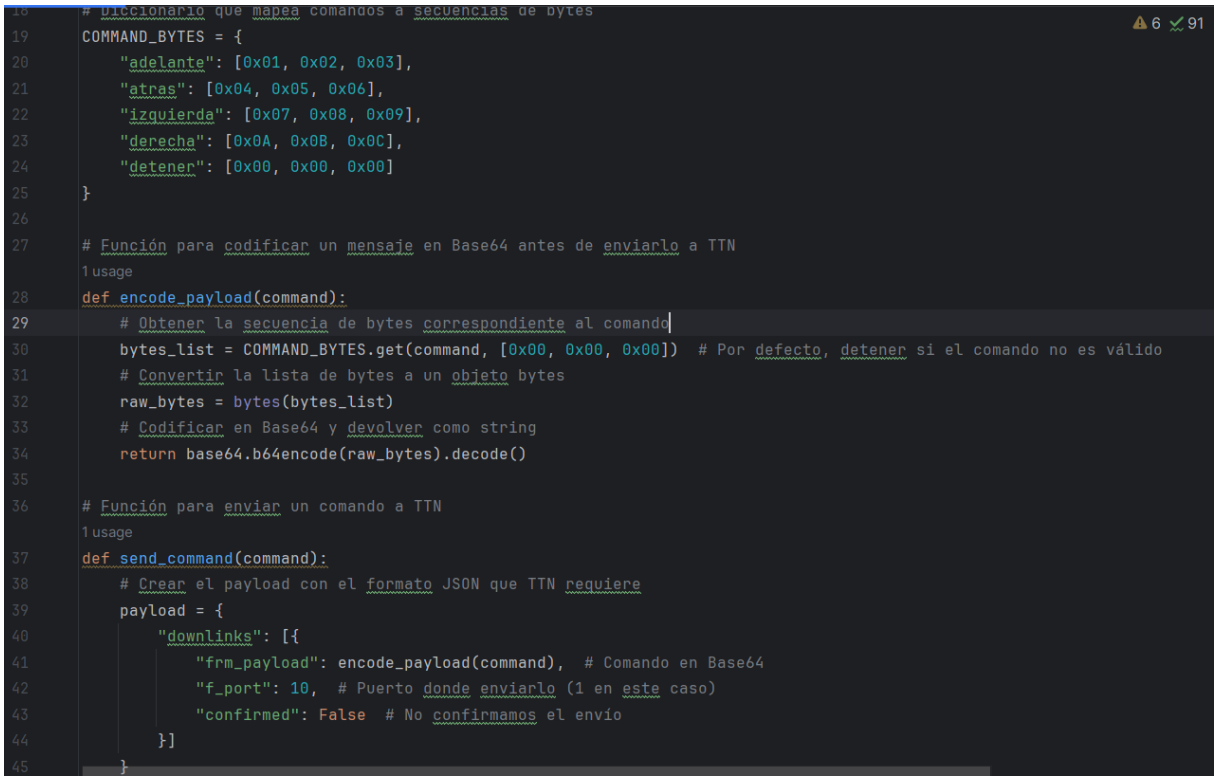


Figura 4.8: Configuración inicial de la aplicación de usuario.

En la figura 4.8 se muestra cómo se han importado las librerías mencionadas previamente y se han inicializado las siguientes variables:

- **APP_ID:** Es el identificador de la aplicación registrada en TTN.
- **ACCES_KEY:** Es la clave de acceso de la aplicación en TTN, que funciona junto con una contraseña para autenticar al cliente MQTT. Se consigue de forma automática en TTN añadiendo una integración MQTT (En la explicación de la configuración de TTN se muestra como se ha conseguido).
- **DEVICE_ID:** Es el identificador único del dispositivo registrado en la aplicación de TTN.
- **BROKER_ADDRESS:** Es la dirección del *bróker* MQTT. En este caso es TTN y esta dirección varía según en la región en la que nos encontremos.
- **BROKER_PORT:** Es el puerto que se utilizará para la conexión MQTT CON el *bróker*. El puerto estándar es 1883.
- **TOPIC:** Es el tópico en el que se publican los mensajes para enviar comandos al dispositivo.

Con estos parámetros inicializados, se procede con el mapeado de los comandos en secuencias de bytes. Los comandos están compuestos por secuencias de 3 bytes. Por ejemplo, para enviar la instrucción “ADELANTE” correspondería con la secuencia 0x01 0x02 0x03. En la figura 4.9 se puede observar el mapeado de todos los comandos, así como la función para codificar estos comandos en base 64 y la función que se encarga del envío de estos comandos.



```
# Diccionario que mapea comandos a secuencias de bytes
COMMAND_BYTES = {
    "adelante": [0x01, 0x02, 0x03],
    "atras": [0x04, 0x05, 0x06],
    "izquierda": [0x07, 0x08, 0x09],
    "derecha": [0x0A, 0x0B, 0x0C],
    "detener": [0x00, 0x00, 0x00]
}

# Función para codificar un mensaje en Base64 antes de enviarlo a TTN
1 usage
def encode_payload(command):
    # Obtener la secuencia de bytes correspondiente al comando
    bytes_list = COMMAND_BYTES.get(command, [0x00, 0x00, 0x00])  # Por defecto, detener si el comando no es válido
    # Convertir la lista de bytes a un objeto bytes
    raw_bytes = bytes(bytes_list)
    # Codificar en Base64 y devolver como string
    return base64.b64encode(raw_bytes).decode()

# Función para enviar un comando a TTN
1 usage
def send_command(command):
    # Crear el payload con el formato JSON que TTN requiere
    payload = {
        "downlinks": [{
            "frm_payload": encode_payload(command),  # Comando en Base64
            "f_port": 10,  # Puerto donde enviarlo (1 en este caso)
            "confirmed": False  # No confirmamos el envío
        }]
    }
```

Figura 4.9: Mapeo de comandos y funciones para codificar y enviar comandos.

Cabe mencionar que la codificación en base 64 es necesaria ya que cuando TTN recibe un mensaje lo decodifica en base 64 por defecto. También hay que destacar que en la función “send_command” existe la variable “f_port” ya que es un campo de LoRaWAN para identificar el puerto de aplicación al que se envía el mensaje en *downlink* desde TTN. En la mota también se encuentra está misma variable con el mismo valor para que pueda recibir los mensajes.

Para crear el cliente MQTT se utilizan las funciones de la librería Paho MQTT, las cuales pueden verse en la figura 4.10.

```
    # Conexión al broker MQTT de TTN
    client = mqtt.Client()
    client.username_pw_set(APP_ID, ACCESS_KEY)  # Autenticación con App ID y Access Key

    try:
        # Conectar al broker
        client.connect(BROKER_ADDRESS, BROKER_PORT, keepalive: 60)

        # Publicar el mensaje en el tópico
        client.publish(TOPIC, json.dumps(payload))

        # Actualizar estado en la interfaz
        status_label.config(text=f"Comando '{command}' enviado a TTN.", fg="green")

        # Cerrar la conexión
        client.disconnect()
```

Figura 4.10: Creación del cliente MQTT y publicación de tópicos.

Las funciones utilizadas para este propósito son:

- **mqtt.Client():** Esta función crea una instancia del cliente MQTT.
- **username_pw_set(APP_ID, ACCESS_KEY):** Configura las credenciales para autenticarse con TTN.
- **client.connect(BROKER_ADDRESS, BROKER_PORT):** Conecta al cliente en la dirección "BROKER_ADDRESS" y en el puerto "BROKER_PORT" con un *keepalive* de 60 segundos.
- **client.publish(TOPIC, json.dumps(payload)):** Publica el mensaje (*payload* JSON con el comando codificado) en el tópico definido previamente.
- **client.disconnect():** Después del mensaje finaliza la conexión.

Por último, queda definir como se ha creado la interfaz gráfica. Como se ha comentado previamente, se ha utilizado la librería Tkinter y en la figura 4.11 se muestra los comandos utilizados. Las funciones utilizadas son las siguientes:

- **window = tk.TK:** Crea la ventana principal de la aplicación.
- **title(""):** Establece el título de la ventana.
- **tk.Frame(window):** Crea un *widget frame* (es un contenedor invisible para agrupar y organizar otros widgets, como son los botones) dentro de la ventana principal (porque se le ha pasado el parámetro *window*).
- **frame.pack():** Crea coloca el *frame* en la ventana utilizando el gestor de geometría *pack*.
- **commands = ["adelante", "atras", "izquierda", "derecha", "detener"]:** No es de la librería Tkinter, pero se necesita una lista de los comandos para la creación de los botones.
- **tk.Button(frame, …):** Crea un *widget button* dentro del *frame*. Se ha utilizado un bucle *for* para la creación de los botones.
- **tk.Label:** Crea un *widget label* para mostrar mensajes de estado. Dará al usuario información sobre el mensaje enviado como por ejemplo si ha llegado correctamente.
- **window.mainloop():** Inicia el bucle principal de eventos de Tkinter, manteniendo la interfaz gráfica abierta y respondiendo a las interacciones del usuario.

```python
# Crear la interfaz gráfica usando Tkinter
1 usage
def create_gui():
    # Crear la ventana principal
    window = tk.Tk()
    window.title("Control del mBot")

    # Crear un frame para organizar los botones
    frame = tk.Frame(window)
    frame.pack(padx=20, pady=20)

    # Lista de comandos
    commands = ["adelante", "atras", "izquierda", "derecha", "detener"]

    # Crear un botón para cada comando
    for command in commands:
        button = tk.Button(frame, text=command.capitalize(), width=15, height=2,
                           command=lambda cmd=command: send_command(cmd))
        button.pack(pady=5)

    # Etiqueta de estado para mostrar resultados
    global status_label
    status_label = tk.Label(window, text="", fg="black")
    status_label.pack(pady=10)

    # Ejecutar el loop principal de la interfaz gráfica
    window.mainloop()
```

Figura 4.11: Creación de la interfaz gráfica utilizando Tkinter.

En la figura 4.12 se muestra cómo queda la interfaz gráfica de la aplicación de usuario.

Figura 4.12: Interfaz gráfica de la aplicación de usuario.

Cómo se observa en la figura 4.12, esta aplicación es bastante sencilla, contando únicamente con los botones de las instrucciones y un mensaje de estado sobre el último comando enviado. En la figura 4.13 se muestra un diagrama de flujo del comportamiento de la aplicación de usuario.

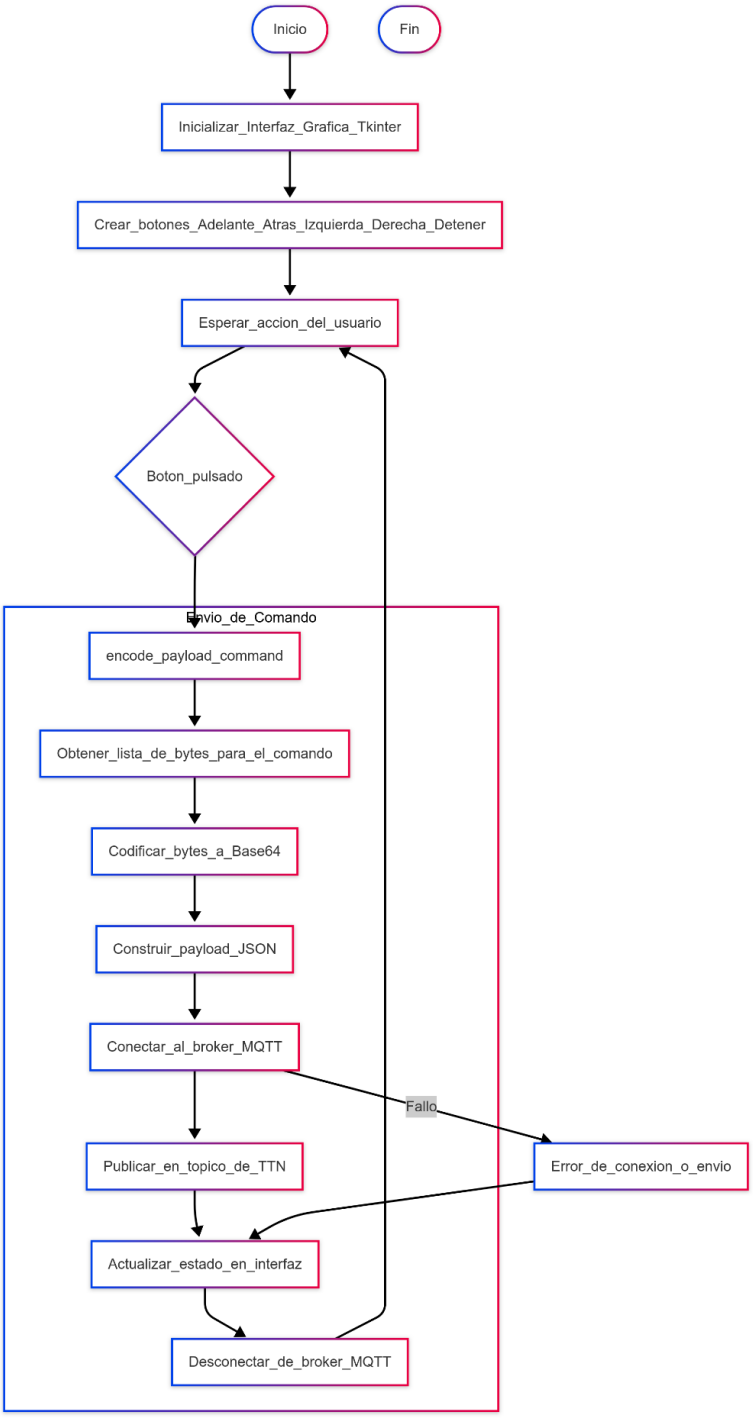


Figura 4.13: Diagrama de flujo de la aplicación de usuario.

Como se observa en el diagrama de flujo (figura 4.13), la aplicación se inicia y crea la interfaz gráfica. Cuando se pulsa alguno de los botones se crea la instrucción

en el formato correcto y después se conecta con el *bróker*. Cuando se ha enviado el mensaje, muestra el estado en la interfaz gráfica, se desconecta del *bróker* MQTT y se repite el proceso.

El código completo de la aplicación de usuario se puede encontrar en el repositorio de mi *github* personal [34].

4.3.2. Implementación del *firmware* de la mota

Para la implementación del *firmware* de la mota se ha partido del proyecto *t-beam-lorawan* de Jorge Navarro Ortiz [32]. En este repositorio se busca poder conectar una placa Lilygo T-Beam con un módulo LoRa SX1262 específicamente a TTN, y además conseguir que esta placa envíe la posición. En este proyecto se ha adaptado este repositorio para que la placa tenga almacenada sus credenciales correctamente y modificado la configuración de pines para la conexión UART, reciba las instrucciones y las transmita y se haga la creación del canal UART para transmitir y recibir mensajes. Para utilizar este repositorio hay que tener en cuenta que versión de placa se está utilizando (en este caso es la rev1) y hay que instalar previamente las librerías que se van a utilizar, que son:

- **Basicmac de LacunaSpace:** Esta librería permite utilizar el módulo SX1262 de la placa con LoRaWAN. Se utiliza por la librería LMIC no soporta este módulo.
- **MCCI Catena/Arduino-LMIC:** Con esta librería se implementa la capa MAC de LoRaWAN, se hace gestión del envío y recepción, admite los modos de activación OTAA y ABP y podemos controlar la frecuencia que se utilizará.
- **TinyGPSPlus:** Decodifica los datos del módulo GPS y permite extraer la latitud, longitud y altitud.
- **ThingPulse SSD1306 OLED:** Para el control del *display* OLED SSD1306 aunque la placa que se ha utilizado en este proyecto no posee este *display*.
- **AXP202X Library:** Esta librería es necesaria para la Rev1 de la placa y nos permitirá reducir el consumo de energía (modo *deep sleep*).

Ahora se va a comentar las partes más importantes del código de la mota, sobre todo cómo se hace la conexión con TTN, la gestión del envío de mensajes y recepción de los mismos y la transmisión de estos por UART.

A parte de las librerías necesarias para que el código funcione hay que añadir la librería ***HardwareSerial*** que se utilizará para la creación del UART. Con las librerías definidas se definen instancias de estas.

También se ha definido un *buffer* de datos (*txBuffer*) donde se almacenan los datos GPS para enviar a TTN, un control del modo *deep sleep* con el contador *bootCount* (contador de reinicios) y *wakeCause* (motivo del despertar, que puede ser por pulsación de botón o temporizador). Por último, se han definido unos *flags* para controlar el envío de paquetes LoRaWAN (*packetSent* y *packetQueued*) y para gestionar las interrupciones (*pmu_irq*).

Las funciones principales de este código son:

- **trySend():** Verifica si hay una posición GPS válida, lo muestra por la pantalla OLED y construye el paquete de datos utilizando la función *buildPacket()* definida en otro archivo de configuración. Por último, se encarga de enviar el paquete a TTN utilizando la función *ttn_send()* y soporta las confirmaciones LoRaWAN si está definido.
- **doDeepSleep(uint64_t msecToWake):** Esta función se encarga de gestionar el modo de bajo consumo apagando la pantalla, la radio LoRa y el GPS. Permite que el dispositivo despierte por temporizador o pulsación del botón. Se llama a *esp_deep_sleep_start()* para entrar en *deep sleep.*
- **sleep():** Se encarga de ejecutar el *deep sleep* después de enviar un paquete o si no hay un bloqueo GPS tras un tiempo definido en *GPS_WAIT_FOR_LOCK.*
- **callback(uint8_t message):** Con esta función se manejan los eventos LoRaWAN como son la unión a TTN, el envío de paquetes, las respuestas y errores. En esta función es donde se ha definido el procesamiento de los mensajes recibidos de TTN (en *EV_RESPONSE*), de forma que convierte *payloads* de tres bytes en comandos para el mBot. Por ejemplo, el código 01 02 03 lo transforma a “ADELANTE”, y lo envía al mBot utilizando el UART con el comando *SerialMbot.println(command)*. Para depuración, el código de 3 bytes recibido y la instrucción enviada al mBot se muestran por el serial y pantalla.

- **axp192Init():** Inicializa el AXP192 que configura salidas de energía para LoRa, GPS y pantalla OLED. También habilita interrupciones para eventos de batería como carga o conexión/desconexión y monitoriza el estado de carga.
- **setup():** Función que se utilizará para inicializar los puertos serie (UART2 para transmitir las instrucciones y Serial para depuración), configurar el GPS y el módulo LoRaWAN e intentar unirse a TTN (*ttn_join*).
- **loop():** Función que ejecuta bucles para el GPS, LoRaWAN y la pantalla OLED, gestiona el envío de paquetes cada tantos milisegundos (definidos en *SEND_INTERVAL*) mediante *try_send()*, maneja las pulsaciones prolongadas del botón para reiniciar las preferencias de red y hace que la placa entre en modo *deep sleep* tras enviar un paquete o si no hay bloqueo GPS.

Para que la conexión con TTN se consiga hay que configurar un par de archivos de este repositorio, que son *credentials.h* (en la figura 4.14 se observan las claves ABP), archivo donde se introducirán las credenciales necesarias para la conexión con TTN que dependen del modo de activación que se vaya a utilizar, y el archivo *configuration.h* (en la figura 4.15 se muestran un fragmento de este archivo), que es un archivo de configuración donde hay que introducir parámetros como los pines que se utilizan (que depende de la placa y de la versión de la misma), parámetros relacionados con la red LoRaWAN (como el *spreading factor*, el puerto LoRaWAN que se va a utilizar o cada cuanto se envía mensaje, que está definido para cada 10 ms), pines que se van a utilizar para comunicación I2C con la pantalla OLED (previamente se ha mencionado que esta placa no posee los pines SDA y SCL de I2C por lo que se han definido pines digitales como si fuesen el SDA y SCL para conectar la pantalla) y parámetros relacionados con el GPS.

```
#ifdef USE_ABP

    // LoRaWAN NwkSKey, network session key
    static const u1_t PROGMEM NWKSKEY[16] = { 0x30, 0x25, 0x53, 0x83, 0x87, 0x9B, 0x0B, 0x45, 0xF6, 0x52, 0xB8, 0x07, 0xB4, 0xC3, 0x7C, 0xE9 };
    // LoRaWAN AppSKey, application session key
    static const u1_t PROGMEM APPSKEY[16] = { 0x0E, 0xA0, 0x00, 0xF0, 0x1E, 0x1F, 0xFF, 0x51, 0xA0, 0xA3, 0x48, 0xF3, 0x80, 0x3E, 0x0C, 0xE7 };
    // LoRaWAN end-device address (DevAddr)
    // This has to be unique for every node
    static const u4_t DEVADDR = 0x260B4513;

#endif
```

Figura 4.14: Claves ABP en *credentials.h*.

```
63  const uint8_t EU_DR_SF12 = 0, EU_DR_SF9 = 3, EU_DR_SF7 = 5, EU_DR_SF7_BW250 = 6;
64
65  #define DEBUG_PORT              Serial          // Serial debug port
66  #define SERIAL_BAUD             115200          // Serial debug baud rate
67  #define SLEEP_BETWEEN_MESSAGES  false           // Do sleep between messages
68  #define SEND_INTERVAL           (10 * 1000)     // Sleep for these many millis
69  #define MESSAGE_TO_SLEEP_DELAY  5000            // Time after message before going to sleep
70  #define LOGO_DELAY              5000            // Time to show logo on first boot
71  #define LORAWAN_PORT            10              // Port the messages will be sent to
72  #define LORAWAN_CONFIRMED_EVERY 1               // Send confirmed message every these many messages (0 means never)
73  #define LORAWAN_SF              EU_DR_SF9      // Spreading factor (recommended DR_SF7 for ttn network map purposes, DR_SF10 works
74  #define LORAWAN_ADR             0               // Enable ADR
75  #define REQUIRE_RADIO           true            // If true, we will fail to start if the radio is not found
76
77  // If not defined, we will wait for lock forever
78  #define GPS_WAIT_FOR_LOCK       (120 * 1000)     // Wait after every boot for GPS lock (may need longer than 5s because we turned
```

Figura 4.15: Fragmento del archivo *configuration.h*.

Las funciones relacionadas con TTN (como *ttn_join()* o *ttn_send()* que se han mencionado previamente) están descritas en archivo denominado *ttn.ino*. En la figura 4.16 se muestra la función *ttn_send()*.

```
433  void ttn_send(uint8_t * data, uint8_t data_size, uint8_t port, bool confirmed){
434      // ttn_set_cnt(); // we are about to send using the current packet count
435
436      // Check if there is not a current TX/RX job running
437      if (LMIC.opmode & OP_TXRXPEND) {
438          _ttn_callback(EV_PENDING);
439          return;
440      }
441
442      // Prepare upstream data transmission at the next possible time.
443      // Parameters are port, data, length, confirmed
444      LMIC_setTxData2(port, data, data_size, confirmed ? 1 : 0);
445
446      _ttn_callback(EV_QUEUED);
447      count++;
448  }
```

Figura 4.16: Función *ttn_send()* del archivo *ttn.ino*.

Para comprender mejor el comportamiento de este programa se ha creado un diagrama de flujo (figura 4.17). En ella, se observa que lo primero que se hace es inicializar con el *setup()* todo lo comentado anteriormente. Una vez inicializado, se intenta conectar con TTN si está disponible (si no está disponible entra en un *deep sleep* infinito). Una vez registrado en TTN comienza el bucle principal del GPS, TTN y de la pantalla. Si se presiona el botón se borran los ajustes con TTN y se reinicia el dispositivo. Si no, se intenta enviar la posición GPS (en caso de que no sean válidos los datos GPS se entra en modo *deep sleep*, se despierta después de un tiempo y vuelve al comienzo del bucle) construyendo el paquete LoRaWAN, lo envía y entra en modo *deep sleep*. Cuando recibe un mensaje desde LoRaWAN, interpreta el mensaje y lo transmite al mBot, finalizando con la vuelta al bucle principal.

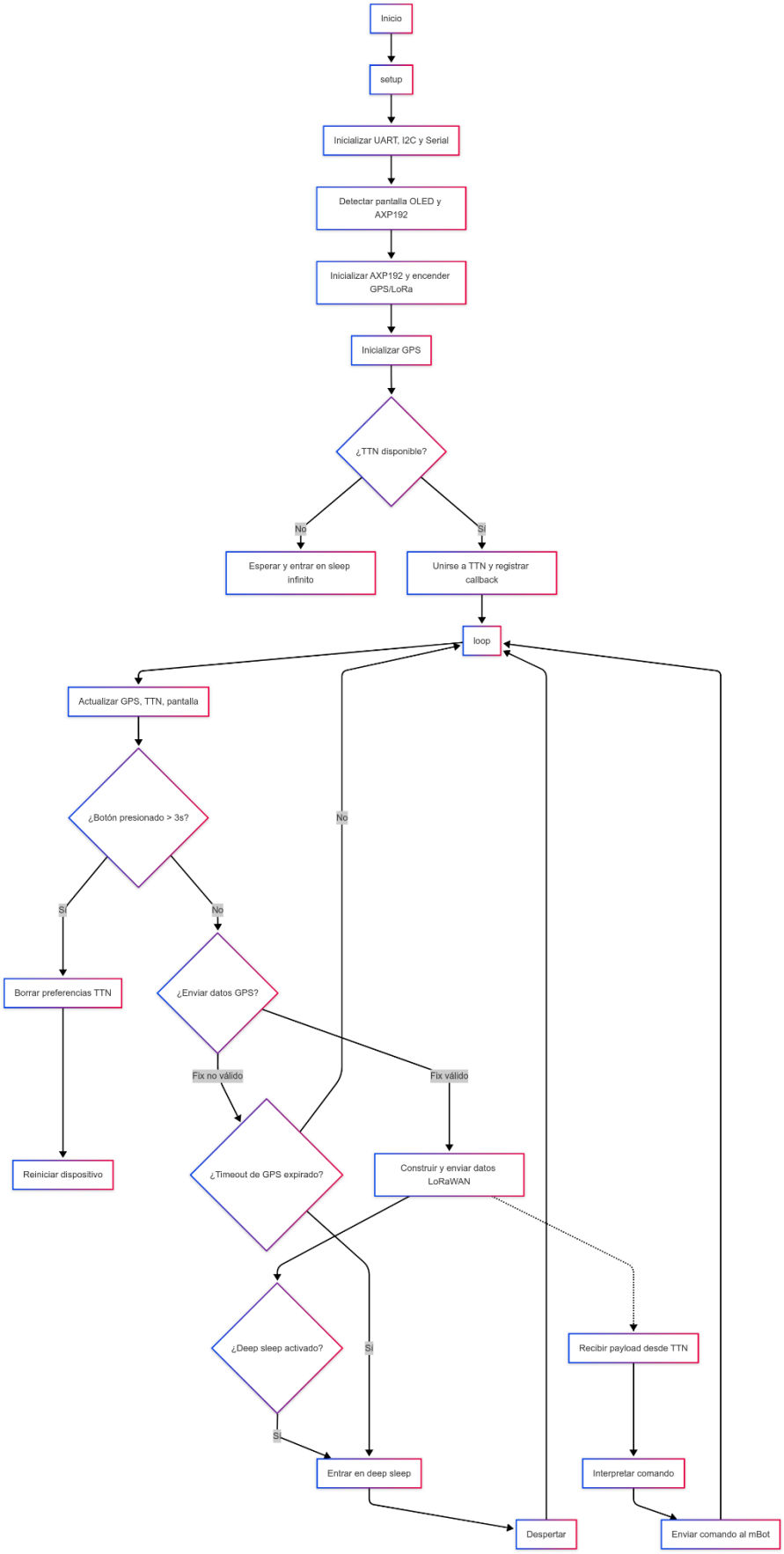


Figura 4.17: Diagrama de flujo del comportamiento del programa de la mota.

Por último, comentar que con este código sólo se ha implementado la clase A, dejando la implementación de la clase B y C para trabajo futuro, y que el avance que se hizo para conectar la mota y el mBot utilizando BLE está comentado, dejando la implementación de las otras clases para trabajo futuro. El código completo de la mota se puede encontrar en el repositorio de mi *github* personal [34].

4.3.3. Implementación del firmware del *gateway*

Ahora se va a comentar el *firmware* del gateway para conseguir que se conecte a nuestra red LoRaWAN en TTN a través de una red WiFi. El *gateway* está programado en MicroPython y las librerías que se han utilizado (figura 4.18) son:

- **network:** Para conectar el *gateway* a redes WiFi.
- **time:** Para manejar retrasos y esperar conexiones.
- **machine:** Para interactuar con el *hardware* del dipositivo.
- **machine.RTC:** Para sincronizar la hora mediante NTP.
- **pycom:** Para funciones específicas del Pycom Pygate.
- **binascii:** Para convertir la dirección MAC en un identificador hexadecimal.

```
1  from network import WLAN
2  import time
3  import machine
4  from machine import RTC
5  import pycom
6
7  import binascii
```

Figura 4.18: Librerías utilizadas en el *gateway*.

Las variables que se han definido son "*devices_list*", que es un diccionario de Python en el que se han definido los dispositivos (este Pycom Pygate se ha utilizado en otras asignaturas con parte de este código), y "*FIXEDLORAWANSERVER*" que indica la dirección del servidor LoRaWAN.

También se han utilizado funciones propias de este dispositivo para deshabilitar *pybytes* y el modo *LTE* y se ha definido un *callback* para algunos eventos como se puede ver en la figura 4.19.

```
# Define callback function for Pygate events
def machine_cb (arg):
    evt = machine.events()
    if (evt & machine.PYGATE_START_EVT):
        # Green
        pycom.rgbled(0x103300)
    elif (evt & machine.PYGATE_ERROR_EVT):
        # Red
        pycom.rgbled(0x331000)
    elif (evt & machine.PYGATE_STOP_EVT):
        # RGB off
        pycom.rgbled(0x000000)

# register callback function
machine.callback(trigger = (machine.PYGATE_START_EVT | machine.PYGATE_STOP_EVT | machine.PYGATE_ERROR_EVT), handler=machine_cb)
```

Figura 4.19: *Callback* para eventos del *gateway*.

Se utilizan algunas funciones de la librería *network* para conectar a la red WiFi y sólo se conectará a la red que hayamos definido en el *firmware* con el siguiente bloque de código (figura 4.20). Las redes WiFi que están definidas son “WIMUNET-LORAWAN” (utilizada para otro proyecto) y “MOVISTAR_0B20” (red WiFi personal).

```
# Check for known networks
bFoundKnownNetwork=False
while not bFoundKnownNetwork:
    nets = wlan.scan()
    ssid_list = [net.ssid for net in nets]

    # 1st priority network
    #if 'RIM-AP' in ssid_list:
    selectedSSID=""
    if deviceAP in ssid_list:
        bFoundKnownNetwork=True
        selectedSSID=deviceAP
        print('Found ' + selectedSSID + ' network!')
#       wlan.ifconfig(config=('192.168.100.' + str(100 + deviceNo), '255.255.255.0', '192.168.100.100', '8.8.8.8')) # (ip, subnet_mask, gateway, DNS_server)
        wlan.ifconfig(config=('192.168.100.' + str(100 + deviceNo), '255.255.255.0', '192.168.100.100', '192.168.100.100')) # (ip, subnet_mask, gateway, DNS_server)
        wlan.connect(ssid=deviceAP, auth=(WLAN.WPA2, "wimunet!"))
    # 2nd priority network
    elif 'WIMUNET-LORAWAN' in ssid_list:
        bFoundKnownNetwork=True
        selectedSSID='WIMUNET-LORAWAN'
        print('Found ' + selectedSSID + ' network!')
        wlan.connect(ssid='WIMUNET-LORAWAN', auth=(WLAN.WPA2, "wimunet!"))
    # 3rd priority network
    elif 'MOVISTAR_0B20' in ssid_list:
        selectedSSID='MOVISTAR_0B20'
        bFoundKnownNetwork=True
        print('Found ' + selectedSSID + ' network!')
        wlan.connect(ssid='MOVISTAR_0B20', auth=(WLAN.WPA2, "[redacted]"))
    # No known network found
    else:
        print('No known network found! Waiting 5 seconds to retry...')
        time.sleep(5)
```

Figura 4.20: Conexión del *gateway* a red WiFi.

Cuando se conecta a una de estas redes WiFi se le asigna una dirección IP estática. La totalidad del código se encuentra en mi *github* personal [34].

4.3.4. Implementación del *firmware* del mBot

Por último, queda la implementación del *firmware* del mBot. El objetivo del mBot es recibir las instrucciones, interpretarlas y ejecutarlas. Se ha programado en Arduino, escogiendo la placa Arduino Uno ya que el mCore, que tiene un microcontrolador ATmega328, está basado en Arduino Uno. Para conseguirlo se han utilizado las librerías:

- **Arduino:** Para utilizar funciones básicas del entorno de Arduino.
- **MeMCore:** Librería específica para utilizar funciones del mBot como es mover

los motores.

- **SoftwareSerial:** Para implementar el canal UART y tener la comunicación mBot-mota.

En primer lugar, se declara la comunicación UART en los pines 10 (Rx) y 11 (Tx) y se instancian los dos motores y el *buzzer* del mBot (figura 4.21).

```
5   SoftwareSerial SerialMbot(10, 11);  // UART virtual con pines Rx = 10 y Tx = 11 (Original Rx=10 y Tx=11)
6   MeDCMotor motor1(M1); // Motor izquierdo
7   MeDCMotor motor2(M2); // Motor derecho
8   MeBuzzer buzzer; // Zumbador del mBot
```

Figura 4.21: Iniciación comunicación UART e instanciación de componentes del mBot.

En el *setup* del mBot se inicializa el canal del serial (para depuración) y del UART para la comunicación, así como los motores y el *buzzer* del mBot (figura 4.22).

```
18  void setup() {
19    Serial.begin(115200); // Monitor serie por USB
20    SerialMbot.begin(9600); // UART virtual en pines 10 (RX) y 11 (TX)
21    Serial.println("UART virtual iniciado");
22
23    motor1.stop();
24    motor2.stop();
25    buzzer.setpin(8); // Configura el pin del zumbador
26  }
```

Figura 4.22: *Setup* del mBot.

La función para interpretar las instrucciones hace una comparación de *strings* para saber si es correcta la instrucción. Si no es correcta para el movimiento de los motores. Además, siempre que se recibe una instrucción, ya sea correcta o no, el mBot realiza un zumbido (en principio se añadió para depuración). En la figura 4.23 se observa un fragmento de esta función.

```
28  void executeCommand(String command) {
29    Serial.print("Comando recibido por UART: ");
30    Serial.println(command);
31    buzzerActive = true; // Activar zumbido
32    buzzerStartTime = millis(); // Registrar tiempo de inicio
33    buzzer.tone(BUZZER_FREQ, BUZZER_DURATION); // Iniciar zumbido (no bloqueante)
34
35    if (command == "ADELANTE") {
36      speedValue = 100;
37      commandDuration = 3000; // Ejecutar por 3 segundos
38      motor1.run(-speedValue); // Motor izquierdo hacia adelante
39      motor2.run(speedValue);  // Motor derecho hacia adelante
40      commandStartTime = millis();
41      Serial.println("Moviendo hacia adelante a velocidad " + String(speedValue) + "%");
```

Figura 4.23: Fragmento de la función *executeCommand* del mBot.

En el bucle principal, se comprueba si se ha recibido algún mensaje por el UART y ejecuta la función *executeCommand* si lo ha hecho, se maneja el zumbido de forma que no se quede bloqueado una vez se activa y se detiene el mBot al cabo de un tiempo tras recibir el comando. En la figura 4.24 se observa dicho *loop*.

```
72  void loop() {
73    if (SerialMbot.available()) {
74      String command = SerialMbot.readStringUntil('\n');
75      command.trim(); // Elimina caracteres de control como \r o \n
76      executeCommand(command);
77    }
78
79    // Manejar el zumbido de forma no bloqueante
80    if (buzzerActive && millis() - buzzerStartTime >= BUZZER_DURATION) {
81      buzzer.noTone(); // Detener el zumbido cuando pase el tiempo
82      buzzerActive = false;
83      Serial.println("Zumbido terminado.");
84    }
85
86    // Verificar si debe detenerse después de un tiempo
87    if (commandDuration > 0 && millis() - commandStartTime >= commandDuration) {
88      motor1.stop();
89      motor2.stop();
90      Serial.println("Comando finalizado por tiempo.");
91      commandDuration = 0; // Reiniciar duración
92    }
93
94    delay(10); // Pequeño retraso para evitar lecturas rápidas excesivas
95  }
```

Figura 4.24: *loop* del código del mBot.

Para entender mejor el funcionamiento se muestra el siguiente diagrama de flujo (figura 4.25). En él, se observa que, cuando se inicia el mBot se inicializa el canal UART, los motores y el *buzzer* y se entra en el bucle principal. A partir de ahí, lo primero que se hace siempre es comprobar el estado del canal UART. Si ha recibido una instrucción lee el comando, activa el *buzzer* y lo compara con las instrucciones programadas. Cuando realiza alguna de estas acciones continua el bucle principal para gestionar el zumbido (si ha pasado más de 200 ms lo apaga, en caso contrario no). Por último, gestiona el movimiento del mBot de forma que si se supera el tiempo de acción del comando para los motores. Después de un *delay* de 10 segundos vuelve al comienzo del bucle principal. El código utilizado del mBot se encuentra en el repositorio de mi *github* personal [34].

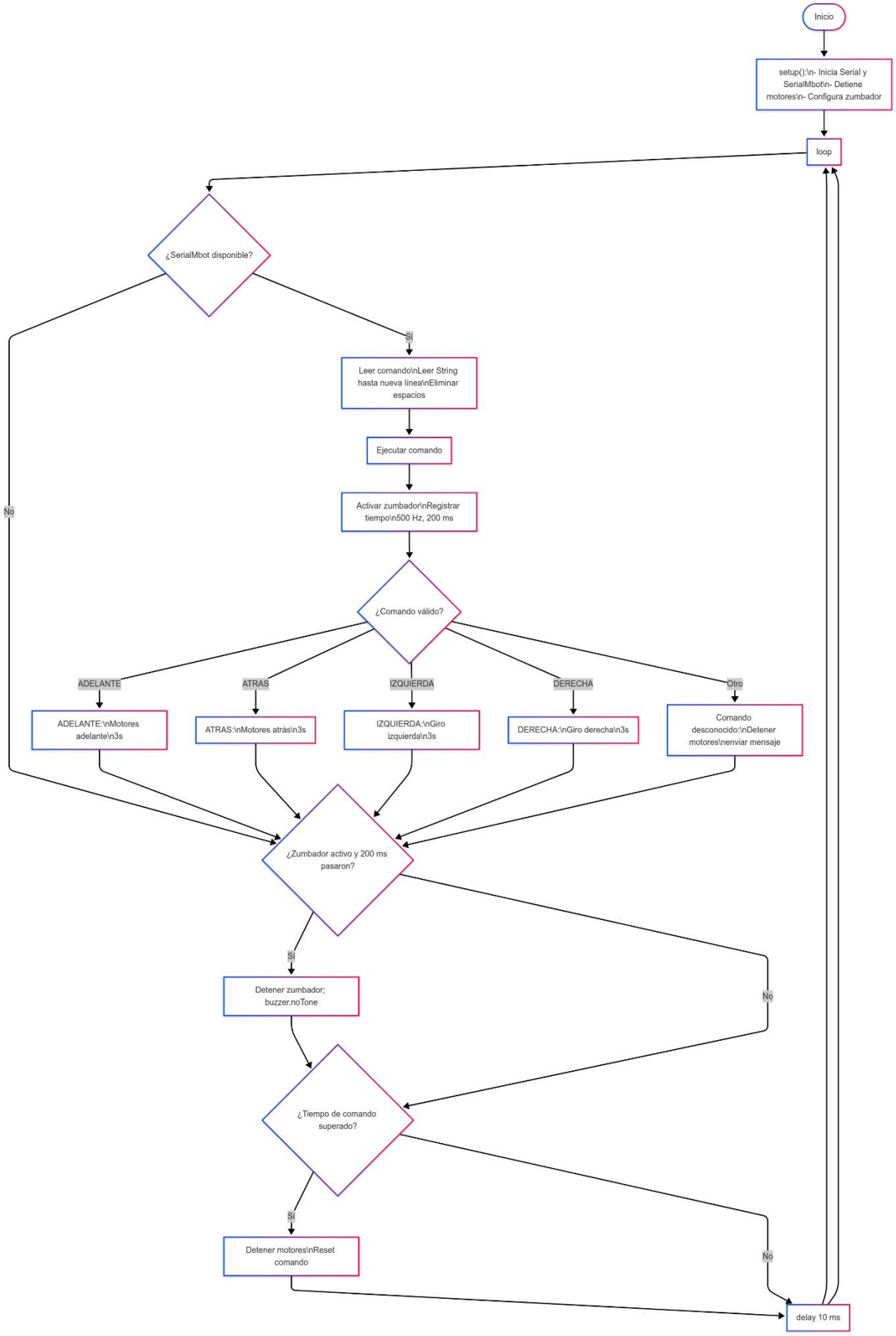


Figura 4.25: Diagrama de flujo del programa del mBot.

4.4. Configuración de *The Things Network*

En este último apartado se ha a mostrar la configuración que hay que tener en *The Things Network* para crear la red LoRaWAN y registrar los dispositivos correctamente.

Para comenzar, hay que crear una cuenta de TTN que nos permitirá acceder a la consola de la plataforma. Una vez creada, la primera vez que iniciemos a la plataforma tendremos que elegir el servidor LoRaWAN en el que registraremos nuestros dispositivos. Este servidor se elegirá conforme a la región que se esté (Europa en este caso). Dentro aparecerá un menú como el de la figura 4.26 donde se podrá acceder al apartado de *gateways* para añadir los *gateways* que queramos y a las aplicaciones (los dispositivos finales se añaden a las aplicaciones ya que es posible utilizar un mismo dispositivo en varias aplicaciones).

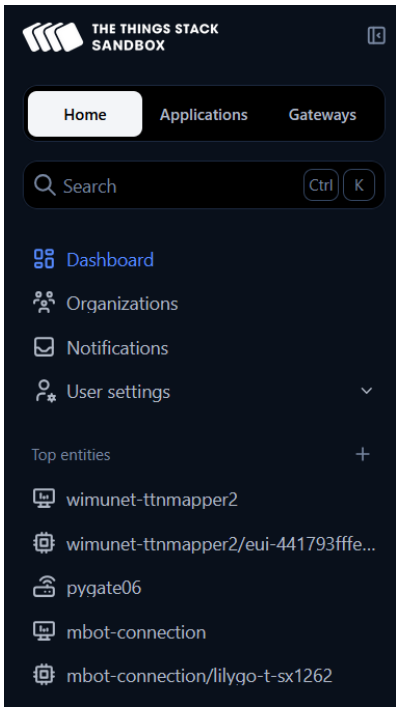


Figura 4.26: Menú de TTN.

Ahora se va a describir como añadir los dispositivos a la red LoRaWAN.

4.3.1. Añadir *gateways*

Para añadir *gateways* accedemos a la pestaña de *gateways* mostrada en la figura 4.26 y se pulsa en *register gateway* (figura 4.27).

Figura 4.27: Registrar *gateway*.

Lo primero que se pedirá introducir es un identificador único del *gateway* proporcionado por el fabricante (figura 4.28). Una vez introducido, hay que introducir algunos parámetros del *gateway* como es el *gateway ID* (identificador que le dará el usuario para reconocer al usuario en la red y que no se puede cambiar una vez se le ha dado), el *gateway name* (nombre que se le dará para que el usuario pueda reconocer de forma más sencilla al *gateway*) y el *frequency plan* (se seleccionará uno u otro dependiendo de la región) como se observa en la figura 4.28. Las demás opciones son opcionales.

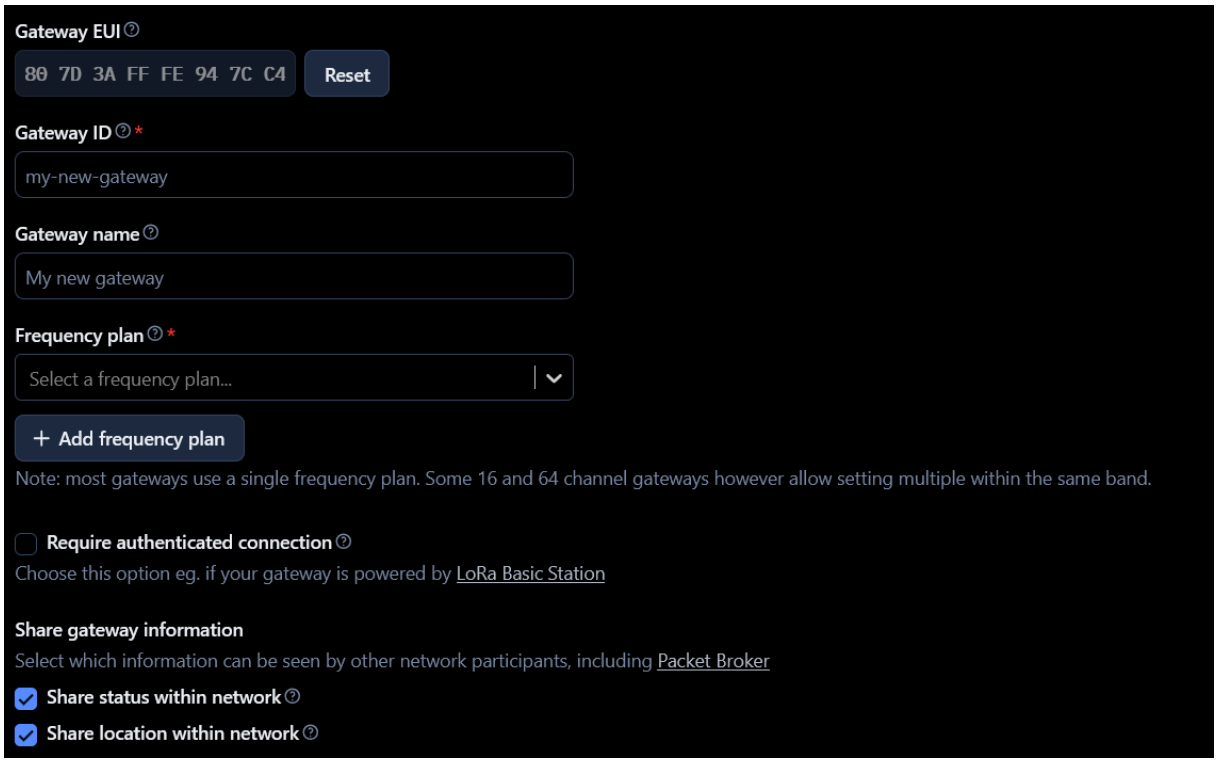


Figura 4.28: Configuración del *gateway*.

Cuando se hayan rellenado estos campos tendremos una vista general del *gateway* registrado como la de la figura 4.29. Ahí se observan los parámetros que se han introducido al registrar el *gateway* junto con algunas opciones de red. En la parte izquierda hay un menú con distintas funciones, siendo la más interesante la de *live data* para ver los mensajes que pasan en tiempo real a través del *gateway*. En la parte derecha se muestra el

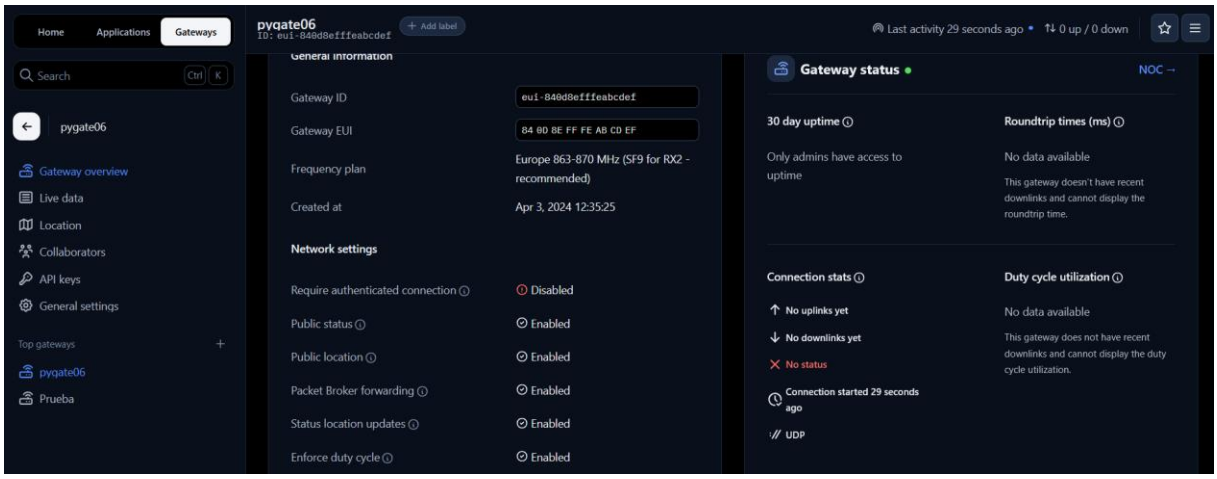


Figura 4.29: Pantalla inicial del *gateway* registrado.

Por último, hay que modificar unas opciones que no aparecen al registrar el *gateway*. Para ello, se accede a *general settings* y allí hay que buscar y marcar unas opciones relacionadas con LoRaWAN que son para habilitar los mensajes en sentido descendente (para enviar los mensajes a la mota) y obligar a cumplir con el *duty cycle* que se comentó en el capítulo 3. En la figura 4.30 se muestran estas opciones.

Figura 4.30: Opciones de LoRaWAN del *gateway*.

4.3.2. Crear aplicación y añadir dispositivo final

Para añadir la mota a la red LoRaWAN primero hay que crear una aplicación en la consola. Para ello, en el apartado de *Applications* se pulsa en *Add application* (figura 4.31).

Figura 4.31: Crear aplicación en la consola de TTN.

Para crearla se pedirá que se introduzca el *application ID* (identificador para identificar la aplicación en la red) y el *application name* (nombre que se le dará para que el usuario acceda a ella). También se puede añadir una pequeña descripción de la aplicación, pero es opcional. En la figura 4.32 se muestran estos parámetros.

Figura 4.32: Parámetros para crear aplicación en TTN.

Cuando se haya creado, hay que añadir el dispositivo final a esta aplicación. Para ello, se accede a la aplicación y se accede al menú de *end devices*, donde se pulsará en *Register end device* (figura 4.33).

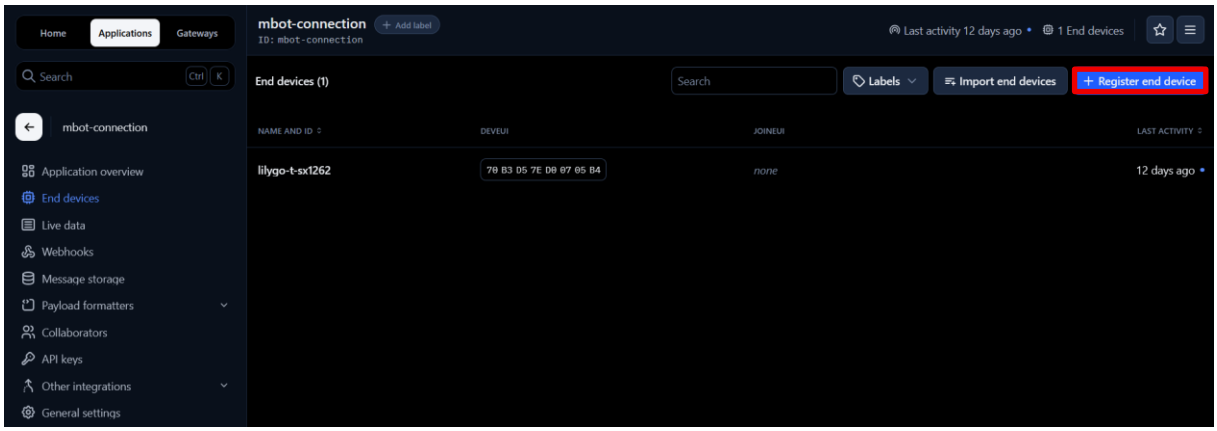


Figura 4.33: Registrar dispositivo final en la aplicación.

Lo primero que se pregunta es el tipo de dispositivo final que se va a registrar. Se puede buscar nuestro dispositivo en el repositorio de dispositivos de LoRaWAN, aunque la opción que se recomienda es introducir manualmente las especificaciones de nuestro dispositivo. Los parámetros que se introducirán son *Frequency plan* (que

depende de la región y se escogerá el mismo que en el *gateway*), *LoRaWAN version* (que es la versión LoRaWAN que se quiere utilizar y que en este caso se ha utilizado la 1.0.2 debido a una de las librerías que se utilizan) y *Regional Parameters versión* (que son la versión de la especificación de *LoRa Alliance* que soporta nuestro dispositivo y que en este caso será *RP002 Regionals Parameters 1.0.2*). En la figura 4.34 se muestran estos parámetros.

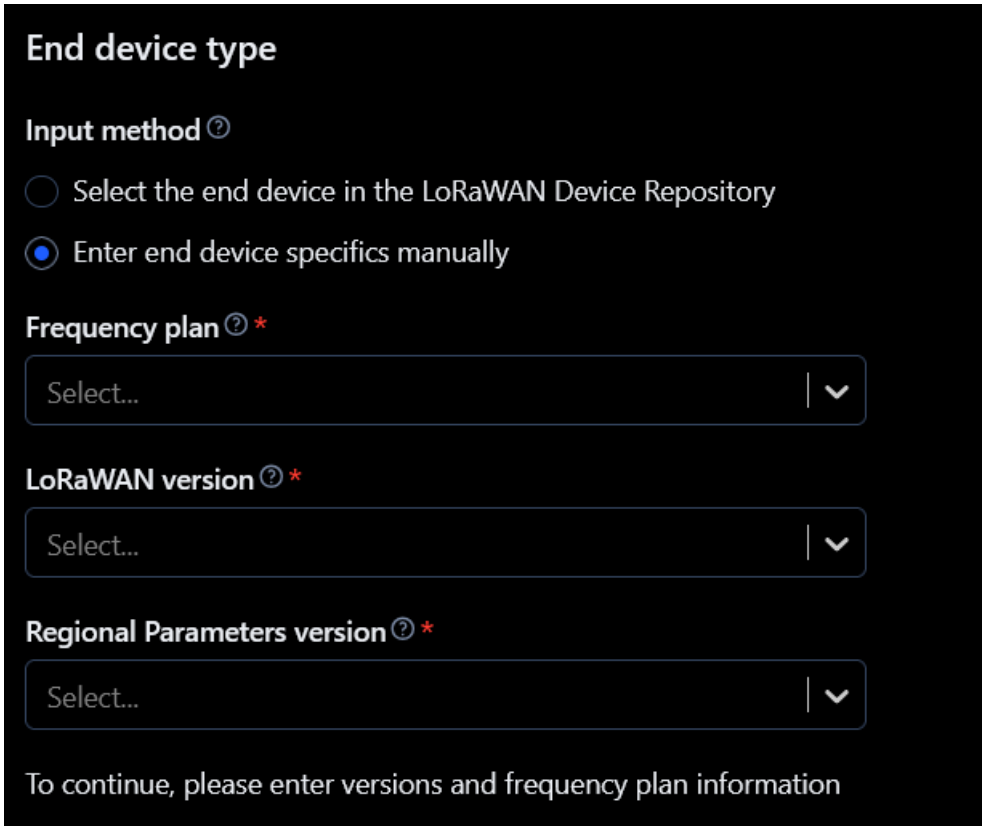


Figura 4.34: Parámetros para registrar dispositivo final.

Después, en esa misma pestaña, hay que pulsar en *show advanced settings* para poder añadir configuración extra como el modo de activación (en este caso se ha utilizado ABP), la clase del dispositivo (clase A por defecto y no se cambiará) e introducir las claves que se utilizarán con ABP (se pueden generar aleatoriamente y son las que se utilizarán en el archivo *credentials.h* mencionado previamente). También hay que introducir un *end device ID* (para identificar el dispositivo en la red). En la figura 4.35 se muestran estos parámetros.

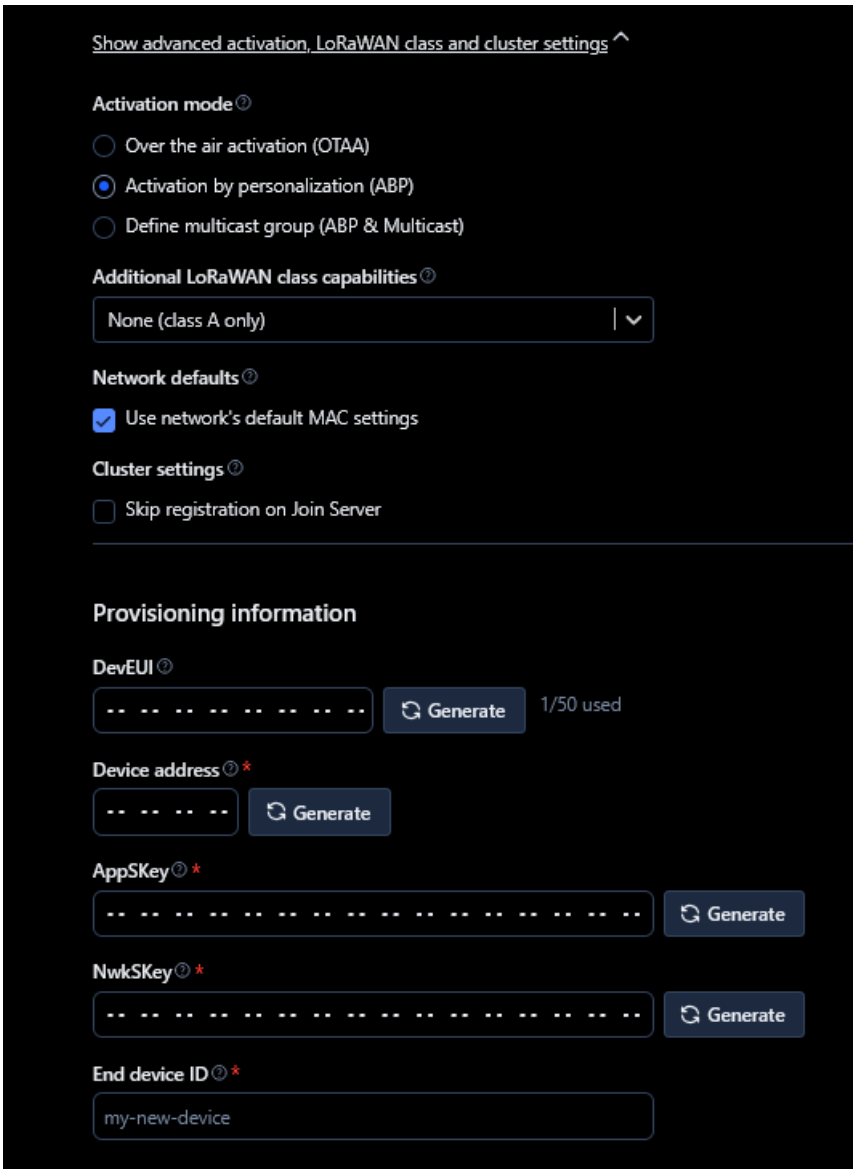


Figura 4.35: Parámetros para registrar dispositivo final.

Con el dispositivo final registrado queda activar una última opción que no aparece al registrar el dispositivo y que se encuentra en los ajustes de la capa de red, ajustes MAC avanzados y la opción que hay que marcar es *Resets frame counters* que se utiliza para reiniciar el contador de tramas cuando el dispositivo sufre alguna desconexión. Es importante activarlo ya que podría perjudicar la conexión con TTN, impidiendo que pudiese recibir los mensajes el dispositivo. Por último, para que TTN haga de *broker* MQTT hay que utilizar la integración MQTT. Para ello, tomando como referencia la figura 4.33, se pulsa en *other integrations* y después en MQTT. Se abrirá un menú como el de la figura 4.36, donde tendremos que pulsar únicamente en *Generate new API key*. La clave generada se utiliza en el *firmware* de la aplicación de

usuario (es la variable *ACCESS_KEY*).

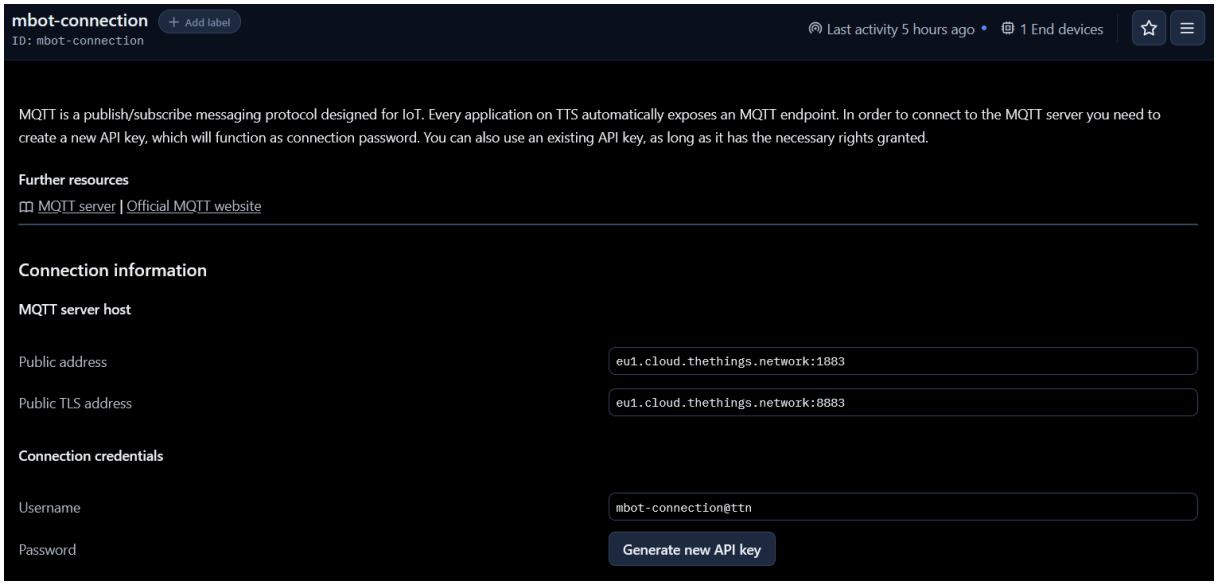


Figura 4.36: Integración MQTT en TTN.

Capítulo 5

Experimentos

En este capítulo se van a realizar una serie de experimentos para comprobar el correcto funcionamiento de este proyecto. El primer experimento consistirá en la verificación de transmisión de comandos (desde la aplicación de usuario hasta el mBot), el segundo experimento se realizará sobre el alcance de la red LoRaWAN, el tercer experimento tratará sobre la latencia del sistema (se medirá cuanto se tarda en desde que se envía la instrucción con la aplicación hasta que el mBot la ejecuta) y el último experimento para ver cómo responde el sistema al perder la conexión con la red LoRaWAN y al recuperarla.

5.1. Experimento 1: Verificación de transmisión de comandos

En este primer experimento se va a validar que el sistema funciona comprobando que el envío del mensaje en la aplicación de usuario, la recepción en TTN y retransmisión a la mota, recepción del mensaje en la mota y transmisión de este al mBot y, finalmente, recepción del mensaje en el mBot y realización de la acción funcionan correctamente. Para ello, se va a monitorizar la consola de TTN donde se mostrará la instrucción recibida por el cliente MQTT y la retransmisión de la misma a la placa. También se monitorizarán los seriales de depuración para validar la transmisión del mensaje de la placa al mBot.

En la figura 5.1 se observa la consola de TTN del dispositivo final *lilygo-t-sx1262*. Los mensajes que se ven son *uplinks* donde se envía la posición GPS de la placa y los *downlinks* que son las instrucciones del mBot. En cada mensaje se observa el *DevAddr* configurado previamente y los bytes (*payload*) que se han mandado en cada mensaje. En los *downlinks* los bytes son códigos de 3 bytes (en este experimento se mandó la instrucción “ADELANTE” con código 01 02 03 y la instrucción “DETENER” con código 00 00 00). También se muestran los valores de parámetros

correspondientes a cada mensaje como es el *FPort* que es el puerto de aplicación, el *Data rate* que es la tasa de datos indica el *spreading factor* que se ha utilizado junto con el ancho de banda, el valor de *SNR* (indica el nivel de calidad de la señal en dB y cuánto más alto mejor calidad) y de *RSSI* (indica el nivel de intensidad de la señal recibida en dBm y cuanto más mayor mejor).

Figura 5.1: Consola de la aplicación *mbot-connection*.

En la consola del *gateway* se observan esos mensajes junto con mensajes sobre el estado del *gateway* (figura 5.2).

Figura 5.2: Consola del *gateway*.

Por otro lado, en el serial de la mota se puede ver la posición GPS que tiene, como la transmite y abre la ventana de recepción y el mensaje recibido de TTN (figura 5.3).

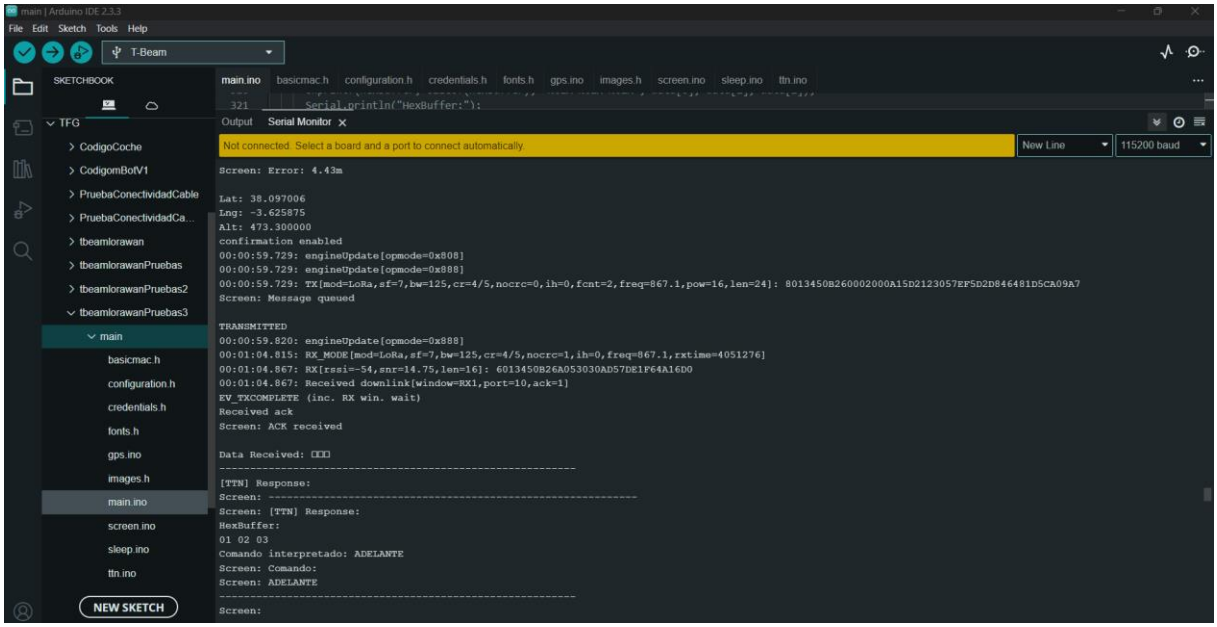


Figura 5.3: Serial de depuración de la mota.

Por último, en el serial del mBot se observa como ha recibido la instrucción y la ha interpretado (figura 5.4), confirmando que el sistema propuesto funciona.

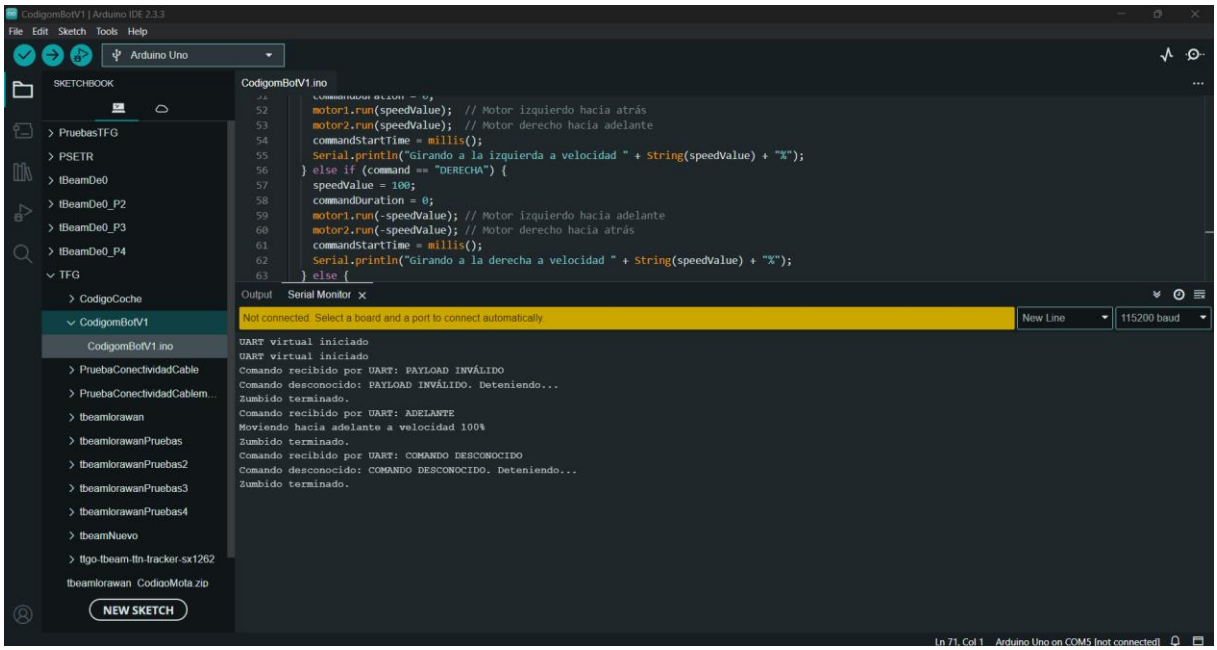


Figura 5.4: Serial de depuración del mBot.

En la figura 5.4 también se observan mensajes de comando desconocido, pero el

firmware del mBot está pensado para que cuando reciba una codificación de bytes que desconozca frene los motores. Se puede visualizar este experimento en vídeo en el repositorio de mi *github* personal [34].

5.2. Experimento 2: Alcance de la red LoRaWAN

En este experimento se posicionará el *gateway* en el domicilio personal y se tomarán las medidas del SNR y RSSI variando la distancia a la que se toman. Los resultados que deberíamos obtener serán, conforme la mota se aleja del *gateway*, valores de SNR y RSSI que irán disminuyendo. Cuando se deje de recibir mensajes significará que se ha superado el alcance máximo o que la mota está transmitiendo a un *gateway* más cercano a su posición. Las medidas se tomarán desde la consola de TTN como aparecen en la figura 5.1 y 5.2. En la tabla 5.1 se encuentran recogidos los valores obtenidos en este experimento donde se tendrán medidas de SNR y RSSI en 3 intentos distintos.

	SNR (dB)			**RSSI (dBm)**			**Media**	
Nº de vez / Alcance (m)	**1**	**2**	**3**	**1**	**2**	**3**	**SNR (dB)**	**RSSI (dBm)**
0	9.5	9.2	8.8	-9	-7	-7	9,16	-7,6
50	9.2	9.5	7.8	-77	-84	-75	8,83	-78,6
100	7.5	7.8	10.2	-100	-85	-87	8,5	-90,6
150	8.8	7	9	-96	-90	-89	8,26	-91,6
200	-5.2	-5.2	-5.5	-112	-111	-112	-5,3	-111,6

Tabla 5.1: Medidas recogidas en experimento 2.

Los resultados obtenidos para el alcance no son los esperados ya que, como se comentó en el capítulo 1, LoRaWAN tiene un alcance teórico en zonas urbanas de hasta 5 Km, y en este experimento se dejó de recibir mensajes del *gateway* a un poco más de los 200 m. Esto es debido a que en ciudad hay muchos edificios y obstáculos que provocan interferencias. No obstante, hay que tener en cuenta que no se ha utilizado el *adaptative rate* que permite el cambio de SF dependiendo del SNR y que, por defecto, se utiliza SF 7, que es el que menos alcance da a cambio de mayor velocidad. También habría que realizar este mismo experimento en zonas rurales,

libres de obstáculos e interferencias, pero por falta de tiempo no ha sido posible.

5.3. Experimento 3: Latencia del sistema

En este experimento se va a medir la latencia del sistema, es decir, lo que tarda el mBot en realizar la acción desde que se envía por la aplicación de usuario hasta que el mBot la ejecuta. Para ello se va a utilizar un cronómetro y se tomarán las medidas unas 10 veces para obtener una media significativa. En la tabla 5.2 se pueden observar los tiempos medidos en cada intento:

Nº de intento	Latencia (s)
1	14,63
2	14,44
3	13,50
4	10,43
5	7,47
6	10,44
7	5,58
8	11,74
9	10,37
10	13,39

Tabla 5.2: Medidas de latencia en experimento 3.

De acuerdo con las medidas tomadas, se observa una latencia mínima de 5,58 s y una latencia máxima de 14,64 s. La diferencia entre estos dos valores es de casi unos 10 segundos y posiblemente es debido al momento en el que se envía el mensaje. Como se ha comentado, como la mota trabaja con clase A, solo abre ventanas de recepción cuando transmite. Por ello, es posible que el mínimo se haya obtenido debido a que se envió el mensaje justo cuando había transmitido o cerca de este momento, mientras que el tiempo máximo puede que se haya obtenido justo cuando se acabara la ventana de recepción, teniendo que esperar a que la mota transmita y abra la ventana de recepción. Como se comentó en el capítulo 4, la mota transmite cada 10 segundos. La media de latencia que se obtiene con estas medidas es de 11,199 s. La desviación típica es de 2,981 s, lo que implica que el conjunto de valores que se tomaron no se separa mucho de la media. Aun así, como se ha comentado ya, este tipo de redes no están pensadas para una comunicación continua en directo.

5.4. Experimento 4: Robustez del sistema

El último consiste en comprobar la robustez del sistema cuando se pierde y recupera la cobertura. Para simular esta pérdida de cobertura lo que se hará es desconectar y conectar el *gateway*. Utilizando la consola de TTN se comprobará el comportamiento del sistema y se medirá el tiempo que tarda en volver a funcionar el sistema. Para medir este tiempo, en la consola de TTN se indican mensajes de estado como *Disconnect gateway* o *Connect gateway*. En la figura 5.5 se muestran estos mensajes de estado.

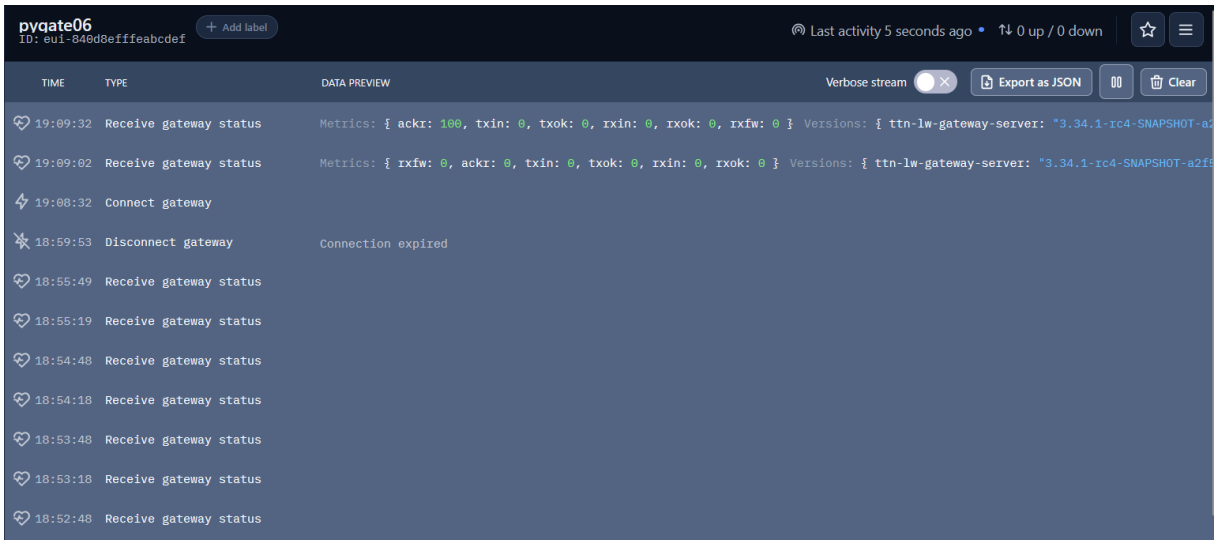


Figura 5.5: Mensajes de estado en la consola de TTN.

La idea entonces es, desconectar el *gateway*, mandar mensajes con la aplicación de usuario y medir el tiempo que tarda en enviar el mensaje desde que recupera la conexión. Se repetirá este proceso para sacar una media que nos dé un resultado más significativo. En la tabla 5.3 están recogidos dichos valores junto con el número del intento.

Nº de intento	Tiempo de reacción (s)
1	9,63
2	5,65
3	13,46
4	5,25
5	13,03
6	12,52
7	8,21
8	7,14
9	5,32
10	10,94

Tabla 5.3: Medidas recogidas en el experimento 4.

Cuando el *gateway* pierde la conexión la plataforma TTN puede guardar los mensajes enviados por el usuario en un *buffer* y cuando el *gateway* recupera la conexión, estos se mandan automáticamente en las próximas ventanas de recepción que abra. No obstante, TTN no detecta que el *gateway* ha perdido la conexión hasta pasados unos 4 minutos (tiempo que tarda el *gateway* en enviar dos mensajes de estado del *gateway*). Aun así, como el *gateway* se supone que debe estar a una red más controlada y se supone que no va a sufrir desconexiones aleatorias, este problema no es tan grave como suena. El tiempo medio que tarda en mandar el mensaje desde que recupera la conexión el *gateway* y lo ejecuta el mBot es de 9,115 s, valor parecido al que hemos obtenido en el experimento 3 y con sentido, ya que al igual que pasaba antes, se puede dar el caso que justo cuando el *gateway* recupere la conexión reciba un mensaje de la mota y abra la ventana de recepción o que pasen bastantes segundos hasta enviar un mensaje la mota y tarde bastante más en abrir la ventana de recepción. La desviación estándar es de 3,257 s, lo que verifica el comportamiento descrito antes ya que es similar al obtenido en el experimento 3.

Capítulo 6

Conclusiones y trabajo futuro

El objetivo principal de este TFG era desarrollar un sistema de control remoto para un robot en zonas sin cobertura utilizando la tecnología LoRaWAN, permitiendo tanto el envío de su posición GPS como la recepción de instrucciones para controlar el movimiento del robot. Para ello, se diseñó un sistema en el que se integró una placa (Lilygo T-Beam) a un robot (mBot), se configuró una red LoRaWAN utilizando la plataforma pública *The Things Network* y se diseñó una aplicación para el usuario en Python que emplea el protocolo MQTT para enviar los comandos.

En conclusión, el proyecto ha sido un éxito ya que se ha logrado:

- Establecer una comunicación bidireccional efectiva en el mBot y TTN.
- Transmitir periódicamente la posición GPS del robot utilizando LoRa.
- Recibir y ejecutar los comandos del control del robot en remoto, confirmando el funcionamiento de la arquitectura propuesta.
- Realizar experimentos sobre el comportamiento del sistema que validan la efectividad del mismo.

Aun así, en los experimentos realizados se han visto las limitaciones de este proyecto, como por ejemplo en el experimento del alcance de red donde de forma teórica se esperaba un valor mucho mayor al obtenido en el experimento. De todas formas, en mi humilde opinión, el proyecto ha sido un éxito ya que se conseguido diseñar un sistema de control remoto para robot en entornos sin cobertura de redes convencionales, aunque hay que tener en cuenta las limitaciones físicas del robot y limitaciones de la red LoRaWAN.

Para finalizar, se proponen algunas líneas de trabajo futuro para mejorar y expandir el proyecto:

- **Mejora del alcance y estabilidad de la red LoRaWAN:** Se pueden utilizar

antenas mejores para ampliar el alcance, probar a cambiar la localización del *gateway* a zonas más elevadas y optimizar los parámetros de transmisión (como el SF o la potencia de transmisión) siempre que se esté dentro de lo permitido.

- **Implementación de confirmaciones de recepción en el mBot:** Incorporar una lógica para confirmar que se han recibido y ejecutado los comandos y permitir retransmisiones automáticas en caso de pérdida de paquetes.
- **Ampliación de las capacidades del robot:** Se pueden integrar sensores y enviar por la red LoRaWAN los datos recogidos por estos. También se podría diseñar el uso de rutas autónomas condicionadas por eventos externos o sensores.
- **Implementación de *TTN Mapper*:** Con *TTN Mapper* se podría visualizar la posición GPS del robot en un mapa siempre que esté conectado a TTN. Además, se podrían añadir indicadores del estado de conexión y cobertura.
- **Exploración de otros protocolos de largo alcance:** Se podría repetir el proyecto utilizando *Sigfox* o *NB-IoT*.
- **Implementación de las otras clases:** Para este proyecto se ha utilizado la clase A, pero se podrían implementar las clases B y C que pueden ser más adecuadas.

Bibliografía

[1] Kais Mekki, Eddy Bajic, Frederic Chaxel, Fernand Meyer «A comparative study of LPWAN technologies for large-scale IoT deployment,» ScienceDirect, vol. 5, nº 1, p. 7, March 2019.

[2] Deutsche Telekom IoT, «NB-IoT, LoRaWAN, Sigfox: An up-to-date comparison,» Deutsche Telekom AG, Germany, March 2021.

[3] TEKTELIC, «LoRaWAN Vs. NB-IoT, Sigfox, And LTE: Which IoT Technology is Right For You?,» TEKTELIC COMMUNICATIONS, June 2025.

[4] Avinash Gudeli, Abhishek Gupta, Somdatta Sable, Kaushik Tandel, Jyoti Gurav, «Lora Based Controlled Surveillance Spy Robot,» IOSR Journal of Electronics and Communication Engineering, June 2022.

[5] Saharuna Saharuna, Tjahjo Adiprabowo, Muhammad Yassir, Dian Nurdiana, Puput Dani Prasetyo Adi, Akio Kitagawa, Arief Suryadi Satyawan, «LoRaWAN-Based Communication for Autonomous Vehicles: Performance and Development,» ILKOM Jurnal Ilmiah, vol. 16, nº 3, p. 19, December 2024.

[6] «Makeblock,» [En línea]. Available: https://www.makeblock.com/pages/mbot-robot-kit. [Último acceso: June 2025]

[7] «DFRobot,» [En línea]. Available: https://wiki.dfrobot.com/SKU_MBT0021-EN_Maqueen_Plus_STEAM_Programming_Educational_Robot. [Último acceso: June 2025]

[8] «Zumo,» [En línea]. Available: https://www.pololu.com/category/129/zumo-robots-and-accessories. [Último acceso: June 2025]

[9] «Thymio,» [En línea]. Available: https://www.thymio.org/. [Último acceso: June 2025]

[10] «Romi,» [En línea]. Available: https://www.pololu.com/category/202/romi-chassis-and-accessories. [Último acceso: June 2025]

[11] «GanttProject,» [En línea]. Available: https://www.ganttproject.biz/. [Último acceso: June 2025]

[12] «Especificaciones técnicas Samsung Galaxy Note 10,» [En línea]. Available: https://www.xataka.com/analisis/samsung-galaxy-note-10-analisis-caracteristicas-precio-especificaciones. [Último acceso: June 2025]

[13] «Pycom Pygate,» [En línea]. Available: https://docs.pycom.io/datasheets/expansionboards/pygate/. [Último acceso: June

2025]
[14] «Lilygo T-Beam,» [En línea]. Available: https://lilygo.cc/products/t-beam?srsltid=AfmBOoqgQF3QekcSsHatl4ilSVUIH_Qu8GLtNLMvCzBX_oL4bEbD41St. [Último acceso: June 2025]
[15] «Samsung Galaxy Note 10,» [En línea]. Available: https://www.samsung.com/es/business/smartphones/galaxy-note10/. [Último acceso: June 2025]
[16] «LPWAN.es,» [En línea]. Available: https://lpwan.es/tag/pygate/. [Último acceso: June 2025]
[17] «Lilygo T-Beam SX1262,» [En línea]. Available: https://lilygo.cc/?srsltid=AfmBOopK-PO-lNsNIIwRNk3IyNwEcTxjo03ucFiPUV-G-a6IalMlkxH-. [Último acceso: June 2025]
[18] «Makeblock mBot,» [En línea]. Available: https://www.makeblock.com/pages/mbot-robot-kit. [Último acceso: June 2025]
[19] «Amazon,» [En línea]. Available: https://www.amazon.es/dp/B00M7U5DV2?ref=ppx_yo2ov_dt_b_fed_asin_title. [Último acceso: June 2025]
[20] «Arduino IDE,» [En línea]. Available: https://www.arduino.cc/en/software/. [Último acceso: June 2025]
[21] «PyMakr,» [En línea]. Available: https://docs.pycom.io/gettingstarted/software/vscode/. [Último acceso: June 2025]
[22] «The Things Network,» [En línea]. Available: https://www.thethingsnetwork.org/. [Último acceso: June 2025]
[23] «PyCharm Community Edition,» [En línea]. Available: https://www.jetbrains.com/pycharm/download/other.html. [Último acceso: June 2025]
[24] Semtech, «LoRa Modulation Basics,» Semtech Corporation, May 2015.
[25] «LoRa Alliance,» [En línea]. Available: https://lora-alliance.org/about-lorawan/. [Último acceso: June 2025]
[26] L. Alliance, «LoRaWAN L2 1.0.4 Specification,» LoRa Alliance, 2020.
[27] UIT, «Terms and definitions». Available: https://life.itu.int/radioclub/rr/art1.pdf. [Último acceso: June 2025]
[28] ITU, «ITU Radio Regulations,» 2020. [En línea]. Available: https://www.itu.int/pub/R-REG-RR-2020. [Último acceso: June 2025].

[29] ETSI, «Short Range Devices (SRD) operating in the frequency range 25 MHz to 1 000 MHz,» [En línea]. Available: https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.01.01_60/en_30022002v030101p.pdf. [Último acceso: June 2025].

[30] OASIS, «MQTT Version 5.0,» OASIS, March 2019.

[31] Makeblock, «A Beginner's Guide to mBot,» Makeblock, 3 March 2023. [En línea]. Available: https://support.makeblock.com/hc/en-us/articles/12822859943959-A-Beginner-s-Guide-to-mBot. [Último acceso: 12 Junio 2025].

[32] J. N. Ortiz, «Repositorio Github,» 12 Dec 2021. [En línea]. Available: https://github.com/jorgenavarroortiz/t-beam-lorawan. [Último acceso: 13 June 2025].

[33] «Mermaid» [En línea]. Available: https://mermaid.js.org/. [Último acceso: 16 June 2025].

[34] F.J. C. Cancho, «Repositorio Github, » 16 June 2025. [En línea]. Available: https://github.com/AKAHoen/Robot-control-system-with-LoRaWAN. [Último acceso: 16 June 2025]

I. Anexo: Manual de usuario

En este anexo se van a presentar las instrucciones que hay que llevar a cabo para poder replicar este proyecto:

1. **Descargar los *firmwares*:** Accede al repositorio de mi *github* personal [34] para descargar todo el contenido del proyecto.
2. **Configuración de TTN:** Seguir los pasos que se han realizado en el apartado 4.3 para configurar TTN.
3. **Preparar los entornos de programación:** Descargar las librerías necesarias para compilar el *firmware* en todos los dispositivos.
4. **Configuración de la aplicación de usuario:** En el firmware de la aplicación de usuario, cambiar el valor de las variables de acuerdo a los dispositivos que se vayan a utilizar y a la configuración de TTN. No hay que olvidar generar la clave para utilizar la integración MQTT de TTN.
5. **Configuración del *gateway*:** En el *firmware* del *gateway* modificar las variables necesarias como la lista de dispositivo añadir la red WiFi a la que se conecte el *gateway* para recibir una IP fija.
6. **Configuración de la mota:** En el *firmware* de la mota, escribir en el archivo *credentials.h* las credenciales que se necesiten según se utilice OTAA o ABP y cambiar los parámetros del archivo *configuration.h* según la placa que se utilice y necesidades de red.
7. **Compilar *firmwares*:** Compilar los *firmwares* en el dispositivo correspondiente para que empiecen a ponerse en marcha.
8. **Monitorizar del entorno:** Para comprobar que todo funciona, monitorizar la consola de TTN para verificar la recepción y envío de mensajes por la red LoRaWAN y utilizar las herramientas de depuración para comprobar que la comunicación UART entre la mota y el mBot funciona.

II. Anexo 2: Posibles errores

En este anexo se van a cubrir algunos de los posibles errores que pueden surgir durante la replicación del proyecto. Los errores que se pueden encontrar con TTN son:

- **Dispositivos mal registrados:** Para saber si un dispositivo no se ha registrado bien, simplemente se comprobará la consola del dispositivo. Para el *gateway*, si está conectado (con los LEDs que indican está conectado a una red WiFi encendidos) pero no aparecen mensajes por su consola significa que no está bien registrado. Con los dispositivos finales será igual, aunque se puede dar el caso que por la consola del *gateway* se reciban mensajes de la mota pero no aparezcan en la consola de TTN. Para solucionar estos problemas simplemente se eliminarán de TTN y se volverán a registrar los dispositivos.
- **La mota funciona una vez pero cuando se desconecta no vuelve a funcionar:** Si se ha utilizado ABP, en los ajustes del dispositivo final hay que marcar la opción *Resets frame counter* ya que es posible que cuando se reconecte el dispositivo su contador de tramas continúe por donde se desconectó. Con esta opción, el contador de tramas siempre se resetea cuando se conecta a TTN.

En la aplicación de usuario no hay mucho margen de error ya que lo único que hay que hacer es cambiar las variables de acuerdo a nuestra configuración con TTN. Si se ha hecho correctamente no debería de surgir ningún error.

En la configuración del *gateway* y sucede lo mismo. Lo único que hay que hacer es cambiar las variables y añadir la red a la que se va a conectar el *gateway*. Si estos pasos se realizan correctamente no debería de surgir ningún error.

Con el mBot pueden surgir errores como:

- **No funciona comunicación UART:** Hay que comprobar que en el *firmware* se han definido bien los pines Rx y Tx y que están bien conectados todos los pines, tanto a la placa como al conversor. También hay que tener en cuenta que, si por ejemplo se utilizan los pines digitales del lado derecho (Tienen los puertos RJ25 al lado), no se pueden utilizar los puertos RJ25.
- **Mal funcionamiento del mBot:** Si el mBot no funciona correctamente, hay que

comprobar que el montaje se ha realizado correctamente.

- **Arduino no detecta el puerto al que está conectado:** Para poder subir código el mBot tiene que estar encendido para que se detecta el puerto al que está conectado. Tener en cuenta también que al compilar en Arduino hay que utilizar la placa Arduino Uno.

Con la mota pueden surgir los siguientes errores:

- **Errores de compilación:** Si surgen errores de compilación puede ser porque alguna librería no está bien instalada, porque se modificado algún parámetro con un valor que no puede contener o se ha utilizado la placa que no es (hay que utilizar la placa T-Beam).
- **Compila, pero no funciona:** Si se ha compilado en la placa pueden pasar dos cosas para que no funciones: la placa no ha pillado la posición GPS y por eso no envía mensajes o las credenciales del modo de activación no son correctas y por eso no se reciben mensajes en TTN.
- **La comunicación UART no funciona:** Hay que comprobar que en la configuración se han elegido pines libres para Rx y Tx y que todos los pines utilizados están bien conectados a la placa y al conversor.