UNIVERSIDAD DE GRANADA

PROYECTO SISTEMAS ELECTRÓNICOS INTEGRADOS

MÁSTER EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Localizador GPS para ganado

Desarrollo e implementación de un sistema de localización GPS para animales en la ganadería mediante LoRa.

Autores

David Fernández Martínez (davidfm8@correo.ugr.es)

Rafael Adán López (rafaadan6@correo.ugr.es)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Enero de 2023

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación

En la última década, la constante innovación tecnológica de las TIC ha generado nuevos paradigmas tecno-productivos en una amplia diversidad de áreas. El sector de la ganadería también se ha visto afectado, el uso de las tecnologías de la información y la comunicación ha supuesto un cambio en la forma tradicional de trabajar y tomar decisiones. La utilización de sistemas GPS permite conocer la distribución espacial de los animales e inferir conclusiones sobre su comportamiento. Esta herramienta permite mejorar el bienestar del animal, puesto que puede ser encontrado en caso de pérdida o accidente y repercute directamente en el propietario con una reducción de costes y aumento de la productividad [1].

Este producto funciona como un collar o prenda vestible (*wearable*) que porta el animal y que posee un módulo GPS que obtiene las coordenadas instantáneas de la posición. En función de como se transmite la información de la posición hacia un servidor donde el usuario pueda visualizarla o consultarla, diferencia las soluciones existentes en el mercado. La opción más asequible y sencilla requiere cargar la información mediante un puerto serie de forma manual [2], de esta forma su uso queda reducido al estudio del comportamiento del ganado, ya que no funciona en tiempo real. Las opciones más extendidas y que presentan un precio más elevado realizan la transmisión de la información hacia un servidor mediante la red GSM, posibilitando la localización del animal en tiempo real, sin embargo, requiere de una suscripción a una operadora móvil.

Una vez analizado el contexto y estado de la técnica actual, se ha buscado una diferenciación respecto a las soluciones comerciales existentes. A la hora de elegir como mandar los mensajes entre el gateway y el nodo localizador hay que tener en cuenta que necesitamos de una red de bajo consumo y que tenga un alcance considerable. La primera opción que se nos viene a la cabeza son las redes *LPWAN (Low Power Wide Area Networks)*. Este tipo

de redes tienen las siguientes características generales: gran cobertura, bajo consumo energético, baja velocidad y coste reducido. Existen numerosas tecnologías, pero las más importantes son NB-IoT, Sigfox y LoRaWAN. En la *Figura* 1.1 se observa una comparación entre ellas.

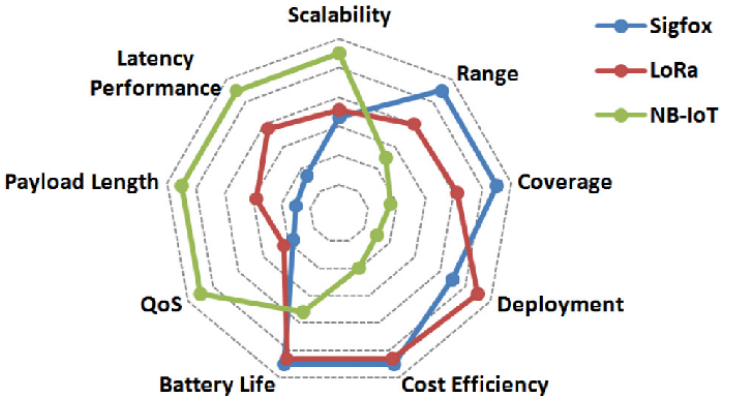


Figura 1.1: Comparación tecnologías LPWAN [3]

Se descarta la opción de NB-IoT al tratarse de una tecnología que opera en bandas con licencia (que aunque operan de manera más eficiente, suponen un coste adicional al necesitar una suscripción). Entre SigFox y LoRaWAN se acaba escogiendo LoRaWAN debido a que se planea seguir utilizándose en líneas futuras para otros proyectos, y este proyecto sirve como primera toma de contacto. Es por esto que se opta por utilizar LoRaWAN, ya que permite una red de largo alcance y baja potencia sobre una banda de frecuencias gratuita que presenta buenos resultados en entornos rurales como el caso de uso planteado en este proyecto.

1.2. Objetivos

En esta sección se enumeran los objetivos planteados:

- Comunicación LoRa en entorno rural con una distancia entre los equipos de al menos $1km$.
- Implementar mecanismos que reduzcan el consumo de batería en el equipo que contiene el módulo GPS: uso de acelerómetro.
- Funcionamiento del equipo receptor como gateway, recepción de los datos a través de LoRaWAN y retransmisión a un servidor vía WiFi.
- Mejorar la precisión del módulo GPS haciendo uso de un filtrado de los datos de posición.

- Permitir al usuario la visualización de la posición en tiempo real mediante un mapa.
- Disponer de un registro de los mensajes recibidos (logs) y almacenar los datos relevantes (posición, batería, hora) para permitir al usuario su posterior análisis.

1.3. Recursos hardware y software empleados

En esta sección se presentan los recursos, hardware y software que se han utilizado para la realización del proyecto.

1.3.1. Recursos hardware

Los recursos hardware utilizados son los siguientes:

- **FiPy.** Se trata de un microcontrolador con MicroPython integrado. Dispone de controladores de red para WiFi, BLE, LTE, Sigfox y LoRa. En el proyecto se hará uso de 2 FiPys que funcionarán como microcontrolador de los equipos que componen el sistema. (*Figura* 1.2).

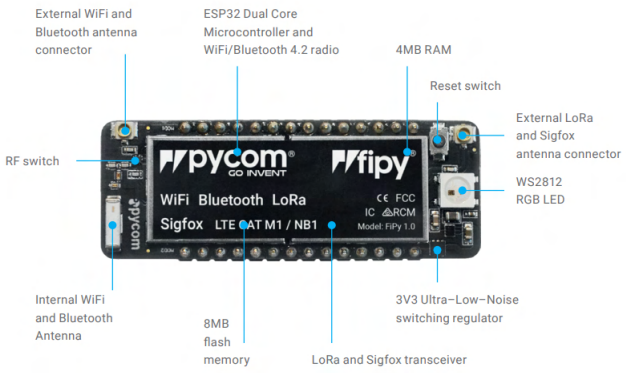
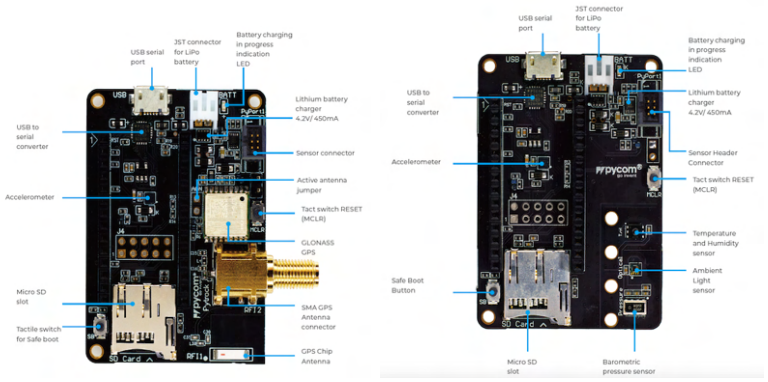


Figura 1.2: FiPy

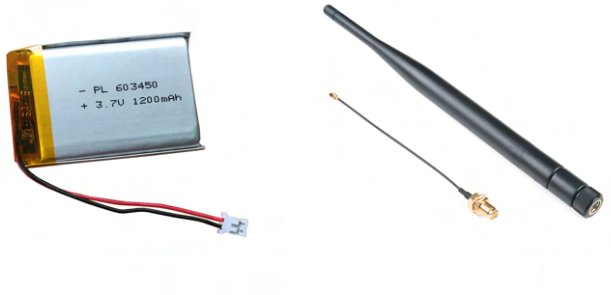
- **Placa de expansión Pytrack 2.0 X.** Placa que incorpora un módulo GPS y un acelerómetro, sobre ella puede funcionar un microcontrolador FiPy. (*Figura* 1.3a).
- **Placa de expansión PySense 2.0 X:** Placa que incorpora diferentes sensores y sobre la que puede funcionar un microcontrolador FiPy. (*Figura* 1.3b).
- **Baterías de litio de 3.7V recargarble.** Dos baterías para dotar de autonomía a las FiPy (*Figura* 1.4a).



(a) Pytrack 2.0 x (b) Pysense 2.0 x

Figura 1.3

- **Antenas LoRa.** Dos antenas que hacen posible la comunicación vía LoRa entre los equipos (*Figura* 1.4b).



(a) Bateria de litio 3.7V (b) Antena LoRa

Figura 1.4

1.3.2. Recursos software

Los recursos software utilizados son los siguientes:

- **Visual Studio Code**. Se trata de un editor de código desarrollado por Microsoft. Cuenta con una gran cantidad de plugins (entre ellos Pymakr para proyectos con Pycom), se ha utilizado para el desarrollo del código.
- **The Things Network (TTN)**. Es una red global de comunicación de dispositivos IoT basada en el protocolo LoRaWAN. TTN permite conectar dispositivos IoT a la red de forma gratuita y sin necesidad de

un contrato con un proveedor de servicios móviles. En este proyecto se ha utilizado como puente entre el gateway y el servidor local.

- **Node-RED**. Es una plataforma de programación visual de código abierto para conectar dispositivos IoT desarrollada por IBM. En este proyecto se ha utilizado para el procesado de la información, con su correspondiente almacenamiento (logs) y representación (mediante un mapa en tiempo real).
- **PyBytes**. Se trata de una plataforma para dispositivos IoT desarrollada por Pycom. Ofrece multitud de herramientas y servicios que facilitan el desarrollo de soluciones IoT. A través de esta plataforma se ha monitorizado remotamente tanto el nodo como el gateway. PyBytes permite verificar si los dispositivos están conectados y además están enviando mensajes.

1.4. Planificación y costes

La planificación temporal del proyecto se ilustra en el diagrama de *Gantt* de la *Figura* 1.5. Se ha desarrollado a lo largo de 7 semanas de trabajo, desde la semana del 12/12/2022 a la semana del 23/1/2023. La metodología

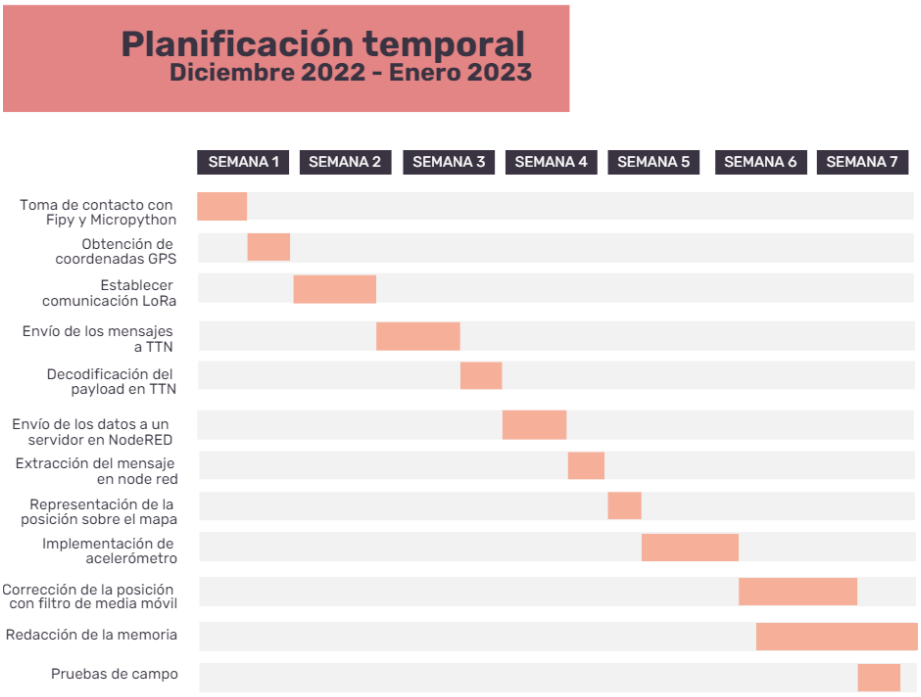


Figura 1.5: Diagrama de *Gantt* de la planificación temporal del proyecto.

de trabajo ha sido colaborativa en todas las tareas que se describen en el diagrama. La estimación del coste total del proyecto resulta de la suma de hardware, software, y el coste humano. Se ha deducido el salario por hora de un ingeniero de telecomunicaciones junior a partir del informe "El ingeniero de Telecomunicación: Perfil Socioprofesional" realizado por el COIT en el año 2013. El coste humano se recoge en la *Tabla* 1.1.

Alumno	Salario	Horas	Total
David Fernández Martínez	15€/h	70	1050€
Rafael Adán López	15€/h	70	1050€

Tabla 1.1: Costes humanos.

Los recursos software empleados, al ser todos de código abierto, no han tenido un coste asociado. Los recursos hardware, a su vez, tampoco han supuesto un coste, ya que han sido facilitados por el Departamento de Teoría de la Señal, Telemática y Comunicaciones. Aun así, se especifica el coste asociado a los recursos hardware en la *Tabla* 1.2.

Material	Unidades	Precio	Total
FiPy	2	64.28€/unidad	128.56€
Pytrack 2.0 x	1	40.65€/unidad	40.65€
Pysense 2.0 x	1	30.03€/unidad	30.03€
Antena Lora	2	11.24€/unidad	22.48€
Bateria de litio	2	6.38€/unidad	12.76€
Portatil personal	2	500€/unidad	1000€

Tabla 1.2: Costes hardware.

El coste total del proyecto será de 2100€(costes humanos totales) más 1234.48€(costes hardware totales) que resulta en 3334.48€.

Capítulo 2

Descripción del sistema

El sistema implementado se ilustra en la *Figura* 2.1. Está compuesto por dos dispositivos: el equipo transmisor (nodo) incorpora un módulo GPS y se encuentra en el animal, el equipo receptor (gateway) recibe la posición por LoRa y la retransmite via WiFi al servidor alojado por *TTN* (*The Things Network*). Es por esto que el gateway debe estar situado en una zona con conexión WiFi para poder conectarse a los servidores de TTN. Los componentes del sistema son los siguientes:

- **Nodo (localizador GPS).** Se ha utilizado una FiPy sobre una placa de desarrollo Pytrack 2.0 X. Esta placa incorpora un módulo GPS que permite obtener las coordenadas de posición. Para enviar los datos a través de LoRa se hace uso del módulo LoRa de la Pytrack 2.0 X y la antena LoRa externa. Esta placa también tiene integrado un

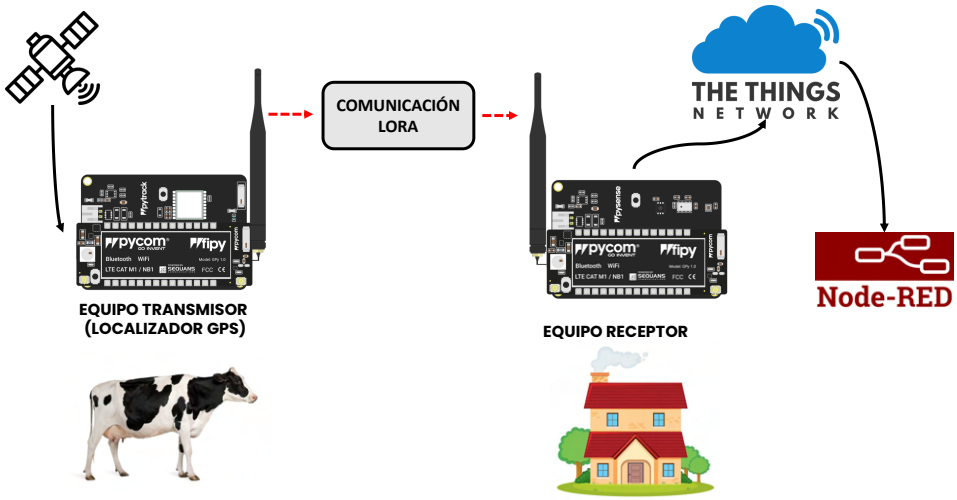


Figura 2.1: Esquema del sistema implementado.

acelerómetro del que se hace uso para detectar movimiento.

- **Gateway.** Se ha utilizado una FiPy sobre una placa de desarrollo Pysense 2.0 X. El gateway recibe la posición a través de LoRa por lo que se hace uso del módulo LoRa de la Pysense 2.0 X y de la antena LoRa externa. El gateway no hace uso de los sensores que incorpora la Pysense 2.0 x por lo que la Fipy podría funcionar sobre cualquier placa de desarrollo de Pycom indistintamente, siempre y cuando tenga disponible un módulo LoRa y WiFi.

El funcionamiento del sistema es el siguiente: el nodo estará recibiendo la posición GPS vía satélite y extraerá las coordenadas (latitud y longitud). Cada mensaje enviado por el nodo contiene los siguientes datos: nombre, latitud, longitud y nivel de batería. Esta información será enviada mediante LoRa al gateway que se encontrará en una zona con acceso a WiFi. El gateway retransmite los datos a un servidor de la red *The Things Network* donde se decodifica el mensaje y se extrae el payload. Los mensajes decodificados se envían desde TTN hasta un servidor local en Node-RED, donde se representan los datos para el usuario. En el servidor local el usuario puede visualizar sobre un mapa la posición del animal en tiempo real, además puede extraer un archivo csv que contiene todos los mensajes recibidos durante el tiempo que el servidor local ha estado activo. Este archivo contiene los datos nombre, latitud, longitud y nivel de batería (los mismos que ha enviado el nodo) y además la hora a la que se ha recibido cada mensaje. Se proporciona un script de *Matlab* en el que se muestra el estado de la bateria , la distancia entre nodo y gateway (para ello hay que indicar la posición del gateway que será conocida) y representa sobre un mapa el recorrido que ha realizado el animal durante el tiempo de funcionamiento. Con estas funcionalidades se permite al usuario conocer la posición en tiempo real y tener un registro de la posición para analizar el comportamiento a posteriori. A continuación se describe de forma detallada el funcionamiento de cada uno de los actores que forman parte del sistema.

Capítulo 3

Diseño e implementación del sistema

3.1. Funcionamiento del nodo

El nodo es el encargado de recibir las coordenadas GPS y transmitirlas hacia el gateway. La frecuencia con la que se envían estos mensajes diferencia las dos versiones implementadas en este proyecto. En la primera aproximación los mensajes se envían cada segundo, en la versión más avanzada se hace uso de un acelerómetro, de modo que el nodo únicamente enviará la posición cuando detecte que se está moviendo. Para mejorar la precisión de la posición extraída por el módulo GPS de la Pytrack 2.0 X se ha implementado un filtro de media móvil. A continuación se explica el funcionamiento de cada versión en particular.

3.1.1. Versión inicial

Para obtener las coordenadas GPS en la Pytrack 2.0 X se hace uso de la clase de MicroPython L76GNSS [4]. Un filtro de media móvil realiza la media de la muestra actual con N muestras anteriores. En las pruebas realizadas sin el filtro de media móvil podíamos observar errores puntuales entre mensajes consecutivos en el rango de $[1, 10]m$ aproximadamente. Esto se traduce en desviaciones respecto de la posición real que pueden percibirse sobre la visualización de la posición en el mapa de Node-RED. Por ejemplo, cuando el nodo está completamente quieto y se recibe una posición con un determinado error, en el mapa veremos un desplazamiento puntual indeseado. Para solventar este problema se ha implementado un filtro de media móvil de 10 muestras. Bajo el supuesto de que un animal de ganado se mueva a una velocidad de $1m/s$ y que el tiempo en enviar 1 mensaje es de 1 segundo, deriva en que en 10 mensajes consecutivos tendremos una variación de como máximo 10 metros. Este filtro permite suavizar esas desviaciones

puntuales y mejorar así la precisión del módulo GPS de la Pytrack 2.0 X. Se puede encontrar el código en el archivo *main_simple.py* en la carpeta del nodo localizador (Pytrack). El funcionamiento del código se resume en los siguientes pasos:

1. Una vez iniciado, el nodo trata de conseguir cobertura de los satélites y obtener las coordenadas GPS.

2. El filtro de media móvil tiene dos buffers, cada uno de 10 muestras: uno para la latitud y otro para la longitud. Ambos se inician a cero.

3. Cuando el nodo ya ha establecido conexión GPS de forma satisfactoria, se entra a un bucle donde se extraen las 10 primeras coordenadas y se van almacenando en los buffers hasta llenarlos.

4. A partir de aquí el nodo obtiene una nueva coordenada GPS cada segundo. A esta latitud y longitud se le aplica el filtro de media móvil y las coordenadas resultantes se envían al gateway. La latitud y longitud obtenidas por el módulo GPS (antes de pasar por el filtro de media móvil) entran a los buffers y sale la muestra más antigua. De este modo, los buffers se van actualizando cada vez que se obtienen unas nuevas coordenadas.

Las primeras 10 muestras que inician los buffers se envían tal cual han sido obtenidas por el módulo GPS de la Pytrack 2.0 X por lo que pueden presentar alguna desviación. A partir de este momento se aplicará el filtro de media móvil y la posición se irá estabilizando. Cabe destacar que este funcionamiento presenta un tiempo de respuesta cuando se parte del reposo. Supongamos que el animal que porta el nodo está completamente quieto. Durante este tiempo las coordenadas GPS que obtiene son iguales o muy parecidas, por lo tanto, a la salida del filtro de media móvil se tiene un valor muy estable. En el momento que el animal comienza a desplazarse, las coordenadas que entran en los buffers comienzan a ser sustancialmente distintas de las que había dentro, sin embargo, para que la salida del filtro comience también a cambiar sustancialmente, deberá esperar a que entren en el buffer más coordenadas correspondientes al desplazamiento. Siendo los buffers de 10 muestras y enviándose 1 muestra cada segundo el tiempo de respuesta para visualizar el cambio de posición sobre el mapa, partiendo del reposo, será aproximadamente de 5 segundos que será el tiempo que tardan los buffers en recibir 5 muestras significativamente distintas a las 10 anteriores.

3.1.2. Versión avanzada (acelerómetro)

El objetivo de esta versión es hacer uso del acelerómetro para reducir el consumo del dispositivo. En el caso anterior se está continuamente enviando

la posición cada segundo, la idea es enviar mensajes únicamente cuando el animal se esté moviendo, de esta forma se estará evitando consumir recursos para enviar información redundante (cuando el animal está quieto y pasa mucho tiempo sin moverse). Para implementar el acelerómetro se hace uso de la clase de MicroPython LIS2HH12 [5]. En esta versión el número de muestras del filtro de media móvil se ha reducido a la mitad. Esto se debe a que realizando pruebas de campo con tamaño 5 y 10 muestras se ha observado que el mejor funcionamiento en términos del tiempo de respuesta ha sido con 5 muestras. Si se utiliza 10 muestras se obtiene un tiempo de respuesta demasiado lento (debido a que se envían menos muestras que en la versión inicial al ser solo con movimiento). Se puede encontrar el código en el script *main_avanzado.py* en la carpeta del nodo localizador (Pytrack). El funcionamiento del código se resume en los siguientes pasos:

1. Una vez iniciado, el nodo trata de tener conexión con los satélites y obtener las coordenadas GPS. De igual forma que en la versión inicial, los buffers de latitud y longitud del filtro de media móvil se inicializan a cero.

2. Cuando se ha establecido conexión GPS comienza el bucle donde se extraen las 5 primeras coordenadas y se almacenan en los buffers.

3. En este punto se define una interrupción para el acelerómetro. Esta interrupción se activa cuando detecta movimiento y llama a una función *handler*.

4. La función *handler* obtiene las coordenadas GPS y les aplica el filtro de media móvil, la latitud y longitud resultante será la que se envía al gateway. Como en el caso anterior, las coordenadas obtenidas entran a los buffers y salen las más antiguas.

De este modo, los mensajes se envían únicamente cuando se activa la interrupción del acelerómetro.

3.2. Comunicación entre nodos: LoRaWAN

En este apartado se explica como se ha implementado la comunicación LoRaWAN entre el nodo localizador y el gateway, así como la retransmisión y procesamiento de la información a los servidores de The Things Network. Para hacer uso de los módulos LoRa de las FiPy se hace uso de la clase LoRa de MicroPython [6].

3.2.1. Gateway

El gateway es el encargado de retransmitir la información que envía el nodo localizador. Para ello, se conecta vía WiFi a la red TTN. Por tanto,

el gateway necesitará de conexión constante a la red. La configuración de este se realiza mediante los script *nanogateway.py*, *config.py* y *main.py*. De forma resumida, el funcionamiento del código es el siguiente:

- Mediante el script *config.py* se especifica el SSID y la contraseña del punto de acceso al que nos vamos a conectar vía WiFi, así como la frecuencia a la que va a operar. En nuestro caso elegimos la banda de 868 MHz, que es la usada en Europa.
- En el script *nanogateway.py* se encuentra todo el funcionamiento del gateway especificado en una clase. Este código ha sido tomado del repositorio de Pycom [7].
- En el script *main.py* se crea un objeto de la clase NanoGateway y se inicia para que empiece a funcionar.

3.2.2. Nodo localizador

El funcionamiento del nodo localizador se ha explicado en la Sección 3.1, por lo que en este apartado nos centraremos en como se implementa la comunicación vía LoRaWAN. Como ya se dijo anteriormente, se va a utilizar la banda de frecuencia de 868 MHz que es la correspondiente para Europa. Por otro lado, es necesario escoger como se van a conectar el nodo y el gateway. Existen dos tipos de conexión:

- **OTAA (Over-The-Air-Activation):** se trata de la manera más segura de conectarse a la red. Su principal ventaja es que la conexión se renueva cada vez que el dispositivo es reiniciado o apagado, lo que dificulta que alguien pueda robar la sesión y clonar el dispositivo.
- **ABP (Activation-By-Personalisation)**: no tan seguro como OTAA, pero mucho más sencillo. La principal ventaja es que no se necesita hacer un join a la red y el servidor no tiene que confirmar los mensajes. La desventaja que tiene es que la llave de encriptación es fija y puede ser extraída y clonada por un atacante.

Debido a la simplicidad y que los datos enviados no son de tipo sensible, decidimos escoger ABP. Por tanto, en el script del nodo localizador será necesario establecer unos parámetros fijos para esta conexión. Estos son:

- **DevAddress**: dirección lógica (equivalente a una dirección IP).
- **NetworkSessionKey**: clave de cifrado para la transmisiones y para validar la integridad de los mensajes.
- **ApplicationSessionKey**: clave de cifrado (a través de la aplicación) para las transmisiones y para validar la integridad de los mensajes.

3.2.3. The Things Network

Una vez los datos llegan al gateway, estos son retransmitidos a los servidores de TTN. Como ya se ha mencionado, se necesita de una conexión a Internet para que el gateway se pueda comunicar con los servidores. Para poder conectarse a los servidores de The Things Network es necesario crear una cuenta (gratuita). Dentro de la interfaz inicial se presentan dos apartados diferentes:

- **Aplicaciones:** en esta sección se configurará la aplicación que recibirá los datos. Es decir, se configura el nodo localizador (end device) así como el formateo de los datos.
- **Gateways**: en esta sección se configura el gateway y se visualizan los mensajes entrantes.

Una vez configurado tanto el gateway como el nodo localizador, es necesario especificar como se van a procesar los datos y que se va a hacer con ellos. Nuestro objetivo es mandar los datos a un servidor local como se explicará más adelante en la Sección 3.3. Es por eso que necesitamos pasarle un formato de datos apropiado. En nuestro caso decidimos enviar un mensaje en formato JSON. Para ello, en la configuración del nodo localizador es necesario crear una función que decodifique los datos procedentes de este nodo (en bytes) al formato que queremos (JSON). El lenguaje que se utiliza en esta interfaz es JavaScript, el código que decodifica los datos se muestra en la *Figura* 3.1.

3.3. Servidor local y procesamiento de datos

Una vez procesado los datos en TTN, estos se enviarán a Node-RED para su procesamiento y representación.

Formatter code *

```
function Decoder(bytes, port) {

  GPSCoordinates = String.fromCharCode.apply(null, bytes)
  mensaje = GPSCoordinates.split(',');

  return {

    'lattitude': mensaje[0],
    'longitude': mensaje[1],
    'name': mensaje[2],
    'battery': mensaje[3]
  };

}
```

Figura 3.1: Payload formatter.

3.3.1. Procesamiento de datos

Mediante Node-RED desplegamos un servidor que se ejecutará localmente en nuestro ordenador (localhost en el puerto 1880). El funcionamiento de este servidor se ilustra en el diagrama de bloques de la Figura 3.2.

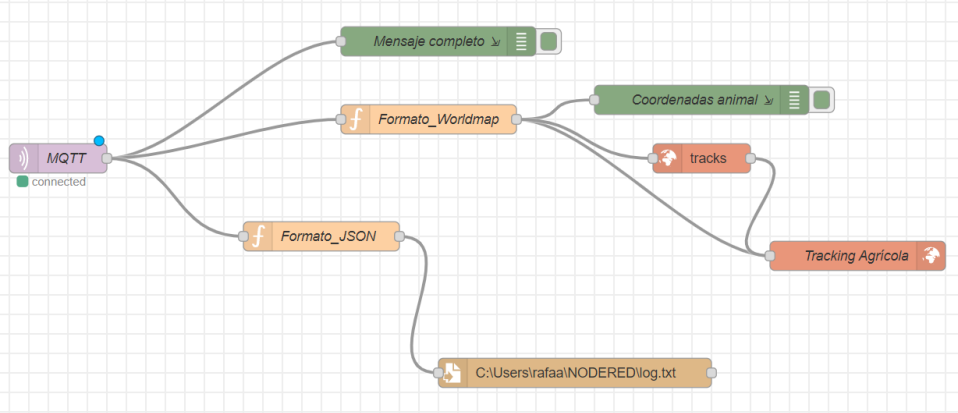


Figura 3.2: Esquema en Node-RED

El sistema está compuesto de los siguientes nodos:

- **MQTT**. Este nodo utiliza el protocolo MQTT para comunicarse con el servidor de TTN y recibir los datos (en formato JSON). Para ello, ha sido necesario crear una API key en TTN y así configurar este nodo.
- **Formato_Worldmap**. Este nodo es de tipo *function* y permite procesar el mensaje antes de pasarlo al siguiente nodo. En este caso se convierten los datos al formato que acepta como entrada el nodo Worldmap (Tracking Agrícola). En la *Figura* 3.3 se observa como los parámetros necesarios son la latitud y longitud (coordenadas), el nombre que tendrá el nodo y el icono con el que queremos representarlo.

```
var latitud
var longitud
var name
var battery


latitud = parseFloat(msg.payload.uplink_message.decoded_payload.lattitude)
longitud = parseFloat(msg.payload.uplink_message.decoded_payload.longitude)
name = msg.payload.uplink_message.decoded_payload.name
battery = parseFloat(msg.payload.uplink_message.decoded_payload.longitude)
msg.latitud = latitud
msg.longitud = longitud

return {payload:{name: name, lat:latitud, lon:longitud, icon: "paw"}};
```

Figura 3.3: Formato_Worldmap

- **Formato_JSON**. De forma analóga al caso anterior, se convierten los datos al formato que queremos que tenga el log que contendrá un registro de los mensajes recibidos durante el tiempo de funcionamiento del servidor local. El código se muestra en la *Figura* 3.4, se añade la hora a la que se ha recibido el mensaje en Node-RED.

```
var latitud
var longitud
var name
var battery

var d = new Date()

var hour = d.getHours()
var minute = d.getMinutes()
var second = d.getSeconds()


latitud = parseFloat(msg.payload.uplink_message.decoded_payload.lattitude)
longitud = parseFloat(msg.payload.uplink_message.decoded_payload.longitude)
name = msg.payload.uplink_message.decoded_payload.name
battery = parseFloat(msg.payload.uplink_message.decoded_payload.battery)
msg.latitud = latitud
msg.longitud = longitud

return {payload:{Nombre: name, Latitud:latitud, Longitud:longitud, Batería: battery, Hora: hour, Minuto: minute, Segundo: second}};
```

Figura 3.4: Formato_JSON

- **Mensaje completo y Coordenadas animal**. Estos dos nodos sirven para poder visualizar la salida del nodo MQTT y Formato_Worldmap, son de tipo debug.
- **Nodo de escritura de fichero**. Recibe la salida del nodo Formato_JSON y la escribe en un fichero .txt.
- **Nodo tracks y Tracking Agrícola**. Estos nodos se utilizan para la representación de las coordenadas en un mapa en tiempo real. El nodo tracks se encarga de dejar una traza de todas las ubicaciones y el nodo Worldmap de mostrarlo en una interfaz en localhost.

En la Figura 3.5 se muestra un ejemplo de como se veía el mapa durante una de las pruebas de campo realizadas (Sección 4).

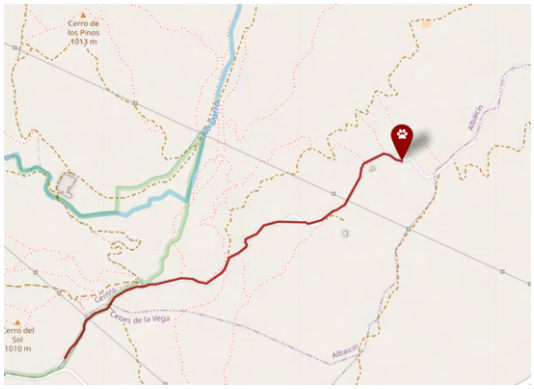


Figura 3.5: Mapa en servidor local

Capítulo 4

Pruebas y resultados obtenidos

Las pruebas de campo se han realizado en la zona alta de Cenes de la Vega, en concreto en el cerro del oro. El gateway se situó en las coordenadas [37.168965, −3.539765]. En la *Figura* 4.1 se muestra la posición del gateway y la zona en la que se han realizado las pruebas sobre *Google Maps.* La metodología seguida fue ejecutar el servidor local de Node-RED y poner a funcionar el gateway y el nodo, uno de nosotros se desplazaba con el nodo para determinar la máxima distancia a la que se recibían mensajes para las dos versiones descritas en la Sección 3.1. Dado que el servidor local se está ejecutando y se está visualizando en tiempo real la posición sobre el mapa, se grabó la pantalla durante las pruebas. Para analizar de forma posterior a la prueba la distancia a la que se han encontrado nodo y gateway se hace uso del archivo que proporciona Node-RED con el contenido de todos los

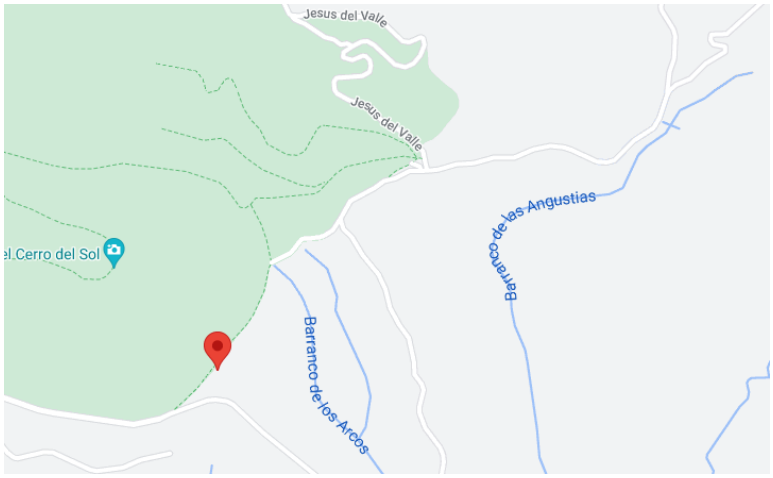


Figura 4.1: Ubicación del gateway durante las pruebas

mensajes recibidos durante el tiempo de funcionamiento. Se ha preparado un script de *Matlab* que recibe este fichero y representa la distancia, estado de la batería, diferencia temporal entre mensajes recibidos y representa sobre un mapa el recorrido seguido por el nodo (lo mismo que se ha visualizado en tiempo real sobre el mapa del servidor local). Este script junto a los ficheros de las 2 pruebas realizadas se pueden encontrar en la carpeta *Resultados* de la entrega. Debido al mal estado del conector en la placa Pytrack 2.0 X no se ha podido realizar las pruebas de campo con la batería de litio, por lo que los resultados de la batería no son relevantes en estas pruebas. El nodo funcionó con una batería *"Power Bank"* que se conecta mediante USB.

4.1. Versión simple

En primer lugar, se hizo una prueba con la versión del nodo descrita en la Sección 3.1.1 en la que se envían mensajes cada segundo una vez que el nodo se ha encendido. Durante la prueba se grabó la pantalla del mapa en el servidor local de Node-RED, este vídeo se encuentra en la carpeta *Videos_pruebas* de la entrega. A partir del fichero *"version_simple.xlsx"* se analizan los resultados obtenidos. En la *Figura* 4.2 se muestra la distancia entre nodo y gateway (posición conocida) a lo largo del tiempo de funcionamiento. La duración de esta prueba ha sido de 35 minutos y se ha alcanzado una distancia máxima de recepción de mensajes entre nodo y gateway de casi 2000 metros. El recorrido realizado por el nodo, que se ha visualizado en tiempo real sobre el mapa del servidor de Node-RED, se muestra en la

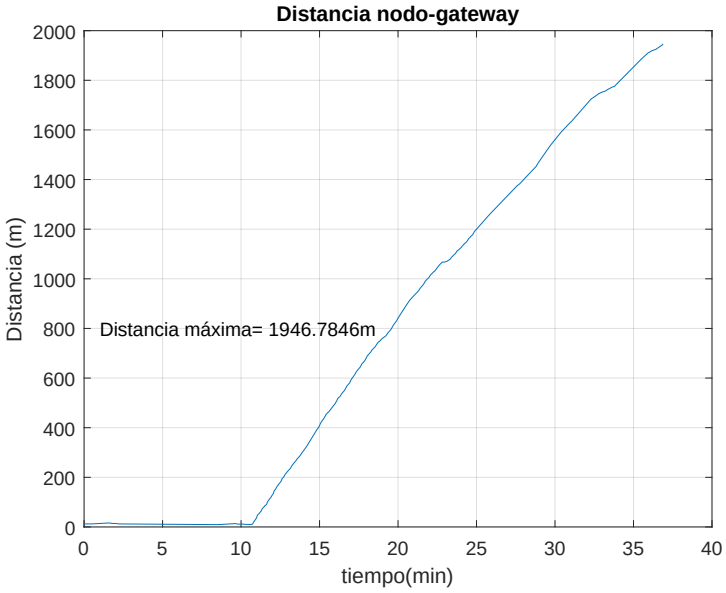


Figura 4.2: Distancia entre nodo y gateway en la prueba de la versión simple.

Figura 4.3. Se ha hecho uso de la función *geoplot* de *Matlab* que a partir de los valores de latitud y longitud permite representar la posición sobre un mapa.

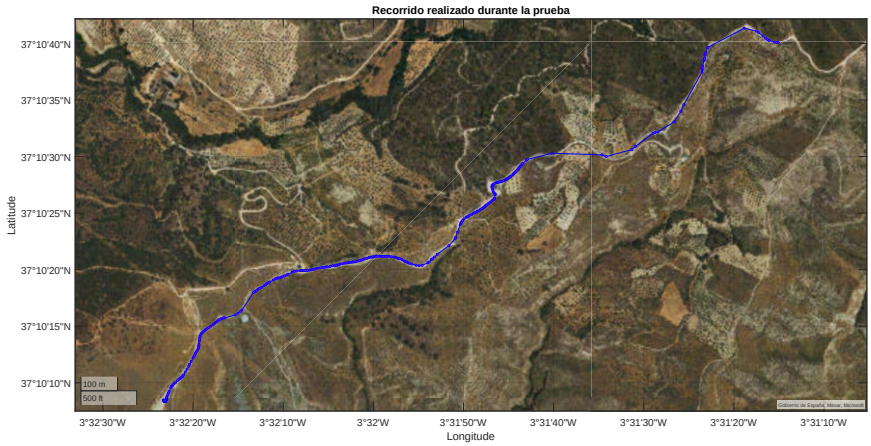


Figura 4.3: Recorrido realizado por el nodo en la prueba de la versión simple.

Finalmente, se realiza un análisis del tiempo transcurrido entre la recepción de mensajes consecutivos. Se hace uso de un histograma (*Figura* 4.4) en el que se visualiza la distribución de tiempos. El valor más repetido es de 3 segundos, el nodo envía la posición cada segundo, por lo que basándonos en este resultado el tiempo desde que se transmite hasta que se recibe en

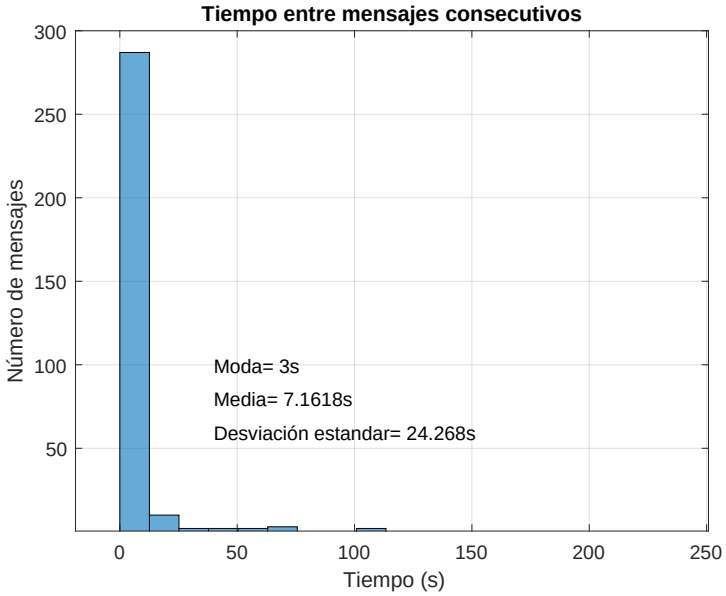
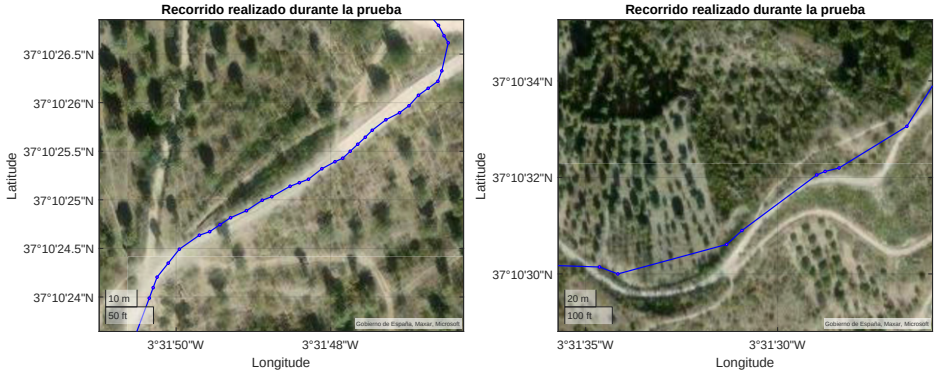


Figura 4.4: Histograma del tiempo entre mensajes consecutivos en la prueba de la versión simple.

Node-RED es de aproximadamente 3 segundos, un retardo asumible para el caso de uso que plantea este proyecto. Cuando el tiempo entre 2 mensajes consecutivos es muy superior a estos 3 segundos quiere decir que hay mensajes que se han perdido y no han sido recibidos en el gateway. Este fenómeno es consecuencia de la distancia entre nodo-gateway y de la visibilidad entre las antenas. En el recorrido realizado en esta prueba hay tramos en los que existe una visibilidad buena o parcial (únicamente árboles o arbustos que pueden reducirla) y otros tramos en los que la visibilidad es nula, puesto que hay una montaña entre ambas antenas. En la *Figura* 4.3 se ha representado la posición de cada mensaje y se han unido los puntos mediante una línea que ilustra el recorrido. En la *Figura* 4.5 se ha hecho zoom sobre dos tramos. En la *Figura* 4.5a se puede apreciar como las posiciones están cerca unas de otras, en este caso no se están perdiendo mensajes. En la *Figura* 4.5b se muestra un tramo en el que se están perdiendo mensajes y como se puede observar la distancia entre posiciones consecutivas es mayor.



(a) Buena visibilidad (b) Mala visibilidad

Figura 4.5

4.2. Versión avanzada (acelerómetro)

En esta segunda prueba se realiza el mismo procedimiento que en el apartado anterior, pero haciendo uso de la versión del nodo que hace uso del acelerómetro, descrita en la Sección 3.1.2. Ahora, únicamente se enviarán mensajes cuando el nodo se esté moviendo. Para la prueba se caminó de forma constante haciendo interrupciones cortas durante el recorrido para verificar que en estado de reposo no se enviaban mensajes. Puesto que la distancia máxima ya fue analizada en la prueba anterior, el trayecto realizado en esta prueba fue más corto, la distancia alcanzada entre nodo-gateway se muestra en la *Figura* 4.6. La duración de esta prueba ha sido de 18 minutos. Las posiciones recibidas en cada mensaje se representan mediante geoplot

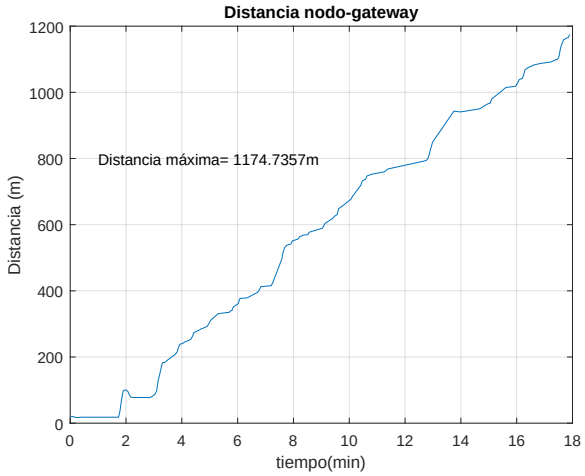


Figura 4.6: Distancia entre nodo y gateway en la prueba de la versión con acelerómetro.

en la *Figura* 4.7. En este caso, la distribución de tiempos entre mensajes consecutivos es más dispersa, puesto que no se generan en el nodo a un ritmo constante de 1 segundo. Como se puede observar sobre el histograma de la *Figura* 4.8 el tiempo más repetido vuelve a ser de 3 segundos, sin embargo, en este caso la desviación estándar es más baja. Esto se debe a que en el tramo recorrido en esta prueba la visibilidad es buena o parcialmente buena en todo el recorrido, los tiempos elevados entre mensajes se deben a los momentos de reposo del nodo y en menor medida a mensajes perdidos. Se puede concluir que la distancia máxima a la que nodo y gateway pueden comunicarse ha sido de hasta 2000 metros en las pruebas realizadas. Para

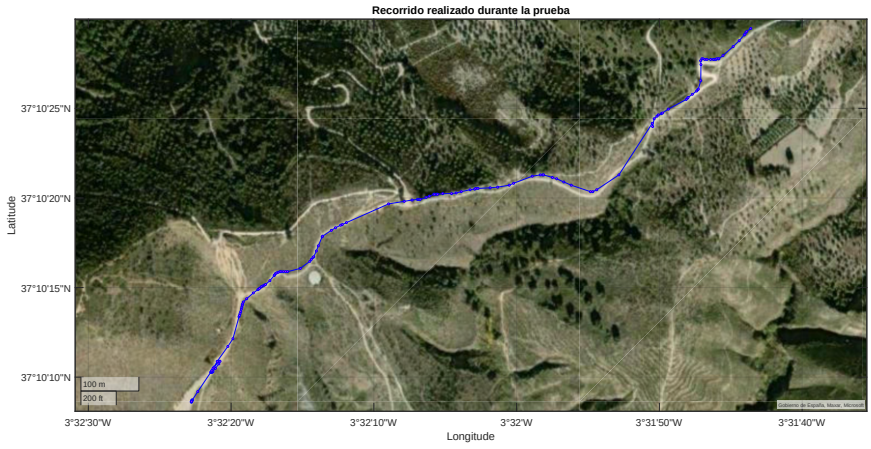


Figura 4.7: Recorrido realizado por el nodo en la prueba de la versión con acelerómetro.

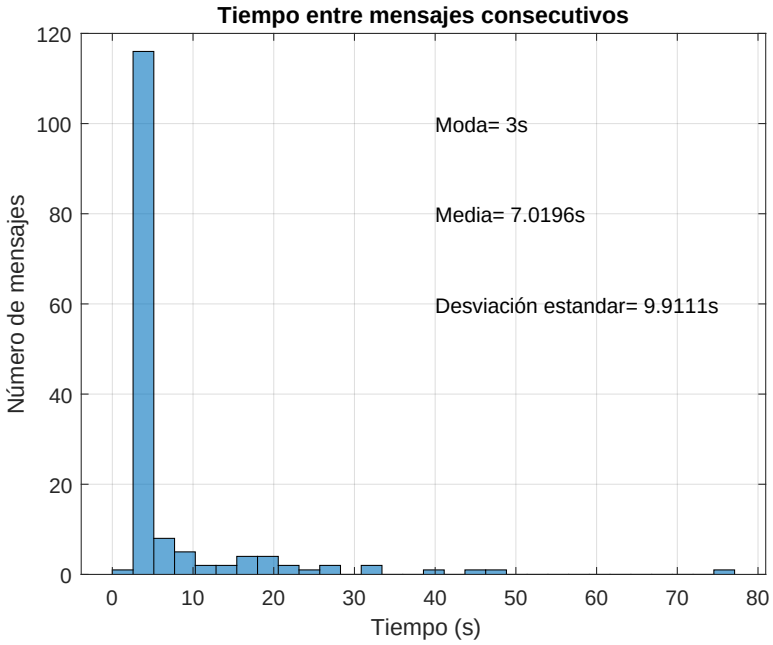


Figura 4.8: Histograma del tiempo entre mensajes consecutivos en la prueba de la versión con acelerometro.

el escenario planteado en este proyecto, del que se espera un entorno rural con condiciones de visibilidad favorables entre nodo y gateway, la distancia lograda está dentro de las expectativas planteadas.

Capítulo 5

Conclusiones y líneas futuras

5.1. Líneas futuras

5.1.1. Reducción del consumo

Como se ha comentado en la Sección 3.1 se han implementado dos versiones del nodo. El objetivo de la versión con acelerómetro es reducir el consumo de la batería. Con el código desarrollado únicamente se accede a las coordenadas GPS y se envían datos a través de LoRa cuando se detecta movimiento, en caso contrario únicamente se queda esperando a una interrupción del acelerómetro. Sin embargo, la FiPy no se pone en modo de bajo consumo realmente, para ello es necesario hacer uso del método py.go_to_sleep() de la librería pycoproc. En nuestro caso, la versión de pycoproc compatible con el firmware instalado en la FiPy incluye el método py.go_to_sleep() pero este manda a dormir a todo el microcontrolador por completo. La idea es que el acelerómetro se mantenga despierto para que en el momento que se detecte movimiento se active la interrupción y se despierte a la FiPy para extraer la posición y enviarla a través de LoRa. Tras varios intentos utilizando diferentes versiones de pycroproc e intentando adaptar el código de la versión compatible con el firmware para mantener el acelerómetro despierto, no se ha conseguido realizar, por lo que esta es una de las líneas futuras que quedan abiertas y que permitirían reducir aún más el consumo de batería.

5.1.2. Servidor remoto

Otra mejora sustancial respecto al proyecto presentado en este informe es trasladar el servidor local a un servidor remoto, de modo que el usuario pueda acceder en cualquier momento sin necesidad de tener que estar ejecutando el servidor de forma local.

5.1.3. Estudio del consumo de batería

Ha quedado pendiente el estudio del consumo de la batería en las pruebas de campo con la versión que usa el acelerómetro, como se ha comentado, esto se ha debido al mal estado del conector de la Pytrack 2.0 X que imposibilitaba la realización de las pruebas de campo con la batería de litio conectada.

5.1.4. Integración en un collar comercial

La línea de futuro más evidente en este proyecto es la integración del equipo que funciona como nodo en un collar que pueda portar un animal de forma segura.

5.2. Conclusiones

Tras la finalización del proyecto, se ha llegado a las siguientes conclusiones:

- La barrera económica de entrada al mercado es una de las principales desventajas que presentan estas soluciones. GSM puede presentar mejores prestaciones que LoRaWAN, pero requiere de un coste adicional debido a la suscripción, ya sea mensual o anual. Utilizando LoRaWAN, se han alcanzado unas prestaciones aceptables para el caso de uso que se marcaba como objetivo.
- LoRaWAN es una tecnología que, debido a tratarse de una banda sin licencia, permite la experimentación y adquisición de conocimiento tanto de redes inalámbricas como LPWAN, lo que para estudiantes de ingeniería como nosotros nos supone un campo que debemos conocer.
- La elaboración de un proyecto no es como hemos estudiado teóricamente. En un proyecto surgen problemas constantes y cambios de rumbo, en los cuáles se ponen a prueba nuestros conocimiento de ingeniero de telecomunicación.

Bibliografía

[1] Geolocalización en ganadería: una solución para el bienestar animal. https://blog.orange.es/innovacion/geolocalizacion-ganaderia/.

[2] Ezequiel Gorandi, Nicolás Clemares, and Andrés Moltoni. Collar con tecnología gps para monitoreo animal. *Investigación y Desarrollo en Electrónica*, 2015.

[3] Kais Mekki, Eddy Bajic, Frédéric Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. 5:1–7, 03 2019.

[4] https://docs.pycom.io/firmwareapi/pycom/expansionboards/l76gnss/.

[5] https://docs.pycom.io/firmwareapi/pycom/expansionboards/lis2hh12/.

[6] https://docs.pycom.io/firmwareapi/pycom/network/lora/. (.

[7] pycom/pycom-libraries: Micropython libraries and examples that work out of the box on pycom's iot modules. https://github.com/pycom/pycom-libraries.