Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación: Mención en Sistemas Electrónicos

Sistema electrónico para la monitorización de la fertilidad de las vacas

Autor:

Víctor Espeso Aparicio

Tutor:

Jesús M. Hernández Mangas

20 de Junio de 2019

Título:	**Sistema electrónico para la monitorización de la fertilidad de las vacas**
Autor:	**D. Víctor Espeso Aparicio**
Tutor:	**D. Jesús M. Hernández Mangas**
Departamento:	**Departamento de Electrónica**

Tribunal

Presidente:	**D. Martín Jaraíz**
Secretario:	**D. Jesús M. Hernández**
Vocal:	**D. Jesús Arias**
Suplente:	**Dña. Ruth Pinacho**
Suplente:	**D. Pedro López**

Fecha:	**20 de Junio de 2019**
Calificación:	

Resumen de TFG

El trabajo consiste en el diseño hardware de un sistema electrónico que es capaz de detectar el movimiento de vacas. Esto se realiza para detectar los ciclos de fertilidad de estas vacas. Este diseño se basa en la obtención de datos de sensores que miden el movimiento, acelerómetro y GPS. Además, este sistema es capaz de enviar esos datos a un microcontrolador, a través del protocolo LoRa. Este microcontrolador es capaz de subir la información a la nube. Adicionalmente se diseña una caja para proteger el sistema de la intemperie y se realizan ciertas pruebas de campo para medir sus capacidades.

Palabras Clave

Acelerómetro, GPS, microcontrolador, IoT y LoRa.

Abstract

The work consists on design of an electronic system that is capable to detect the movement of cows. This is done to detect the fertility cycles of these cows. This design is based on obtaining data from sensors that measure movement, accelerometer and GPS. In addition, this system can send this data to a microcontroller, through the LoRa protocol. This microcontroller can upload this information to the cloud. Additionally, a box is designed to protect the system from the weather and certain field tests are carried out to measure its capabilities.

Keywords

Accelerometer, GPS, microcontroller, IoT y LoRa.

"Por mí se va a la ciudad del llanto, por mí se va al eterno dolor, por mí se va hacia la raza condenada. La justicia movió a mi supremo Hacedor. El divino poder, la suma sabiduría y el primer amor me hicieron. Antes de mí no hubo nada creado, a excepción de lo inmortal, y yo, a mi vez, duraré eternamente. ¡Oh, vosotros, los que entráis, abandonad toda esperanza!"

Dante Alighieri

Agradecimientos

Llega acá el fin de la primera gran etapa de mi vida, sin embargo, el hecho de que use la primera persona del singular para definirlo no hace justicia a todos aquellos que estuvieron detrás apoyándome. Por tanto debiera decir, nuestra vida. Quisiera agradecer en especial a las siguientes personas:

- **A mis padres**: Montse y Luismi. Por la suerte que he tenido por tenerlos, por corregirme y educarme tal como soy, por el apoyo incondicional en todo momento, por las facilidades que me habéis dado para hacer cumplir mis sueños. Por todo aquello que habéis hecho por mí: MUCHAS GRACIAS.
- **A mi hermano**: Jorge. Por estar siempre ahí, por hacerme la cena cuando estaba estudiando esta carrera, por distraerme con los juegos de baloncesto. Mucho ánimo en unos años seguro que te veremos a ti terminar esta etapa.
- **Al Yayo**: Por su interés aparentemente indiferente, sin tú apoyo es probable que no consiguiese la suficiente motivación para terminar esto.
- **Al resto de familia**: Gracias por las muestras de interés y apoyo durante esta etapa.
- **A mi tutor**: Jesús M. Hernández Mangas. Por la pasión que muestra en su profesión y que consigue contagiar al resto de personas. Gracias por tus enseñanzas, no sólo en este proyecto, sino también en las asignaturas cursadas. Gracias por no ponérmelo fácil.
- **A los profesores**: A los profesores de la facultad, en especial a los del departamento de Electrónica de la ETSIT, dado que son los que más me han guiado en estos últimos cursos. Además, un especial agradecimiento a todos aquellos que se apasionan con la docencia y consiguen motivar a los alumnos con sus enseñanzas.
- **A los compañeros**: A los que siguen y los que ya han terminado. Sobre todo a los que les tocó sufrirme en alguna práctica grupal.
- **A Sara**: Por su infinita paciencia, desde que me aguanta no he vuelto a suspender más de dos asignaturas en un curso.
- **A los amigos**: En especial a los que conocí al principio de esta etapa, sin vosotros esta se hubiera hecho mucho más dura.

A todos los mencionados, y a los que pueda pasar por alto, muchas gracias por la paciencia que habéis tenido conmigo. Espero haber sido un buen alumno, y espero ser una buena persona.

Índice general

Índice de tablas

Índice de figuras

Siglas

A

ADC
Analog to Digital Converter. 5, 6

B

BLE
Bluetooth Low Energy. 5

BOM
Bill of Materials. 13, 49

C

CAD
Computer-Aided Design. 19

F

FIFO
First in-First out. 5

G

GLONASS
Global'naya Navigatsionnaya Sputnikovaya Sistema (traduce del ruso). 7, 8

GPIO
General Purpose Input/Output. 4, 12, 13

GPS
Global Positioning System. 3, 4, 7–10, 12–15, 21, 25, 30, 45–48, 51

I

I2C
Inter-Integrated Circuit. 4–6, 9–12, 15, 25, 26, 46, 47

IANA
Internet Assigned Numbers Authority. 31

IDE
Integrated Development Environment. 4, 25, 45

IMU

Inertial Measurement Unit. 5

IoT

Internet of Things. 1, 32, 51

L

LPWAN

Low-Power Wide-Area Network. 4

M

MEMS

Micro Electro-Mechanical System. 5

MosFet

Metal-oxide-semiconductor Field-effect transistor. 13, 30

MQTT

Message Queuing Telemetry Transport. 26, 30–32, 38, 39, 51, 52

O

OLED

Organic Light-Emitting Diode. 4, 5, 46

P

PCB

Printed Circuit Board. 1–3, 12, 13, 15, 21, 22, 41, 46, 49, 51

Q

QSPI

Quad-SPI. 5

R

RAM

Random Access Memory. 5

RSSI

Received Signal Strength Indicator. 42, 52

S

SBAS

Satellite Based Augmentation System. 7, 8

SF

Spreading Factor. 32

SPI

Serial Peripheral Interface. 4–6

SPIFFS

SPI Flash File System. 5, 51

SSL

Secure Sockets Layer. 31

U

UART

Universal Asynchronous Receiver-Transmitter. 4, 7, 12, 25

USB

Universal Serial Bus. 10, 12, 15, 21, 22

UTC

Coordinated Universal Time. 7, 8, 47

W

WAN

Wide Area Network. 4

WLAN

Wireless Local Area Network. 4, 5

WPAN

Wireless Personal Area Network. 5

Capítulo 1

Introducción

1.1 Motivación

Diversos estudios relacionan el ciclo estral de las vacas con un incremento de la actividad física [1] [2]. La necesidad de detectar estos ciclos lleva a Charles A. Kiddy, en 1977, a poner podómetros en las patas de las vacas para medir el movimiento de estas [3]. Charles concluye que las vacas dan cuatro veces más pasos en el período de estro. Por otra parte, en 1994, P.L. Senger ya propone una pequeña red de comunicaciones donde instalar varios sensores electrónicos y recoger los datos en un ordenador [4].

Con el paso de los años se ha intentado desarrollar sistemas de detección acordes a las tecnologías del momento. Cabe mencionar un último artículo donde se describe un nuevo intento de detectar la fertilidad de las vacas a través de estos sistemas [5].

1.2 Objetivos

Los objetivos de este trabajo es el estudio de sistemas Internet of Things (IoT). Hay que diseñar un sistema que detecte el movimiento de la vaca para poder detectar los ciclos de fertilidad. Esto se realizará con distintos sensores que son capaces de medir el movimiento (acelerómetro, giróscopo y GPS). Además de la necesidad de transmitir a través de radiofrecuencia los datos recogidos. Por lo tanto, hay que diseñar un transmisor y un receptor para poder captar esos datos. Se usa LoRa, dado que es un protocolo novedoso que ofrece una calidad interesante en la transmisión. Cabe mencionar que el sistema debe tener una autonomía suficiente de meses. Por último, se propone subir los datos a la nube para completar un sistema de IoT

1.3 Estructura

Este trabajo consta de siete capítulos además de la introducción, las conclusiones, y los anexos.

En el capítulo 2 se describe todo el desarrollo hardware desde la selección de componentes, hasta su captura esquemática y su posicionamiento en la placa Printed Circuit Board (PCB). En el capítulo 3 se detalla el proceso para realizar la caja, y las herramientas utilizadas en el mismo. En el capítulo 4 se muestran las bibliotecas Arduino empleadas y su uso dentro del código. En el capítulo 4.4 se hace una breve introducción al protocolo LoRa, sus especificaciones, sus usos comunes y la legislación vigente que se aplica. En el capítulo 5

[1] J.G. HALL, CECIL BRANTON, y E.J. STONE. Estrus, estrous cycles, ovulation time, time of service and fertility of dairy cattle in Louisiana. Louisiana State University, 1959. Disponible en `https://www.sciencedirect.com/science/article/pii/S0022030259906939`

[2] G.S. LEWIS y S.K. NEWMAN. Changes Throughout Estrous Cycles of Variables That Might Indicate Estrus in Dairy Cows. Journal of Dairy Science Vol. 67, No. 1, 1984. Disponible en `https://www.sciencedirect.com/science/article/pii/S0022030284812783`

[3] CHARLES A. KIDDY. Variation in physical activity as an indication of estrus in dairy cows. Journal of Dairy Science Vol. 60, No. 2, 1977. Disponible en `https://www.sciencedirect.com/science/article/pii/S0022030277838599`

[4] P.L. SENGER. The estrus detection problem: New Concepts, technologies, and possibilities. Journal of Dairy Science Vol. 77, No. 9, 1994. Disponible en `https://www.sciencedirect.com/science/article/pii/S0022030294772179`

[5] E.E.AT-TARAS Y S.L.SPAHR. Detection and characterization of estrus in dairy cattle with an electronic heatmount detector and an electronic activity tag. Journal of Dairy Science Vol. 84, No. 4, 2001. Disponible en `https://www.sciencedirect.com/science/article/pii/S0022030201745353`

se explica el por qué de hacer una aplicación móvil en formato HTML y se describe su uso. En el capítulo 6 se muestra el montaje del sistema, y cómo queda enmarcado en la caja para su uso. En el capítulo 7 se detallan los principales problemas y contratiempos surgidos en el trabajo. Por último, en el capítulo 8 se estima el coste del proyecto para distintos tipos de encargos.

1.4 Metodología

Para la elaboración de este trabajo se ha realizado un desarrollo de hardware y software a través de métodos de desarrollo ágil. Es decir, se aplica un método iterativo donde las necesidades van marcando la pauta y los objetivos inmediatos del proyecto, buscando la funcionalidad de los distintos elementos individualmente antes de la integración plena en el sistema. Para ello, entre otras facilidades, se decide usar módulos integrados con fácil accesibilidad a sus pines.

En primera instancia, se testearon uno a uno en una protoboard, buscando el modo de cablearlos correctamente, lo que podía ofrecer y su funcionamiento básico. El hecho de tener que testear los dispositivos en una protoboard viene como consecuencia de la falta de documentación o de la documentación incompleta. Por mencionar un ejemplo, existen numerosas hojas de datos del chip ESP32 y de diversos módulos, pero los vendedores del módulo utilizado en este proyecto lo único que son capaces de facilitarte es el esquemático y una imagen ilustrativa de la salidas de pines.

Una vez asegurado el funcionamiento de los sensores y su posterior comunicación con el chip maestro (ESP32), se procedió a hacer un diseño de la PCB para su fabricación y, así, poder trabajar con el sistema completo. Una vez fabricado y montado, se probó el sistema y se corrigieron los fallos con el montaje.

Por último, se procedió a generar el software completo, realizar ciertas pruebas de campo y estimar varios presupuestos.

Capítulo 2

Desarrollo hardware

En este capítulo se detalla cómo se resuelve la problemática del proyecto seleccionando los dispositivos necesarios. A su vez se resuelve su interconexión y adaptación a través de componentes pasivos y activos.

Además de la selección de componentes y el diseño de sus conexiones, se muestra el diseño punto por punto, empezando por la selección, el diseño esquemático del proyecto, y finalmente su materialización colocando todos los elementos en una PCB.

2.1 Vista general de funcionamiento

Previo a la selección de componentes se realiza un esquema básico del funcionamiento del proyecto. Están por un lado a las vacas y por el otro al ganadero. En la parte de las vacas hay que hacer una recopilación de datos de los distintos sensores y disponer de una antena de radio LoRa para emitir los datos recogidos. Por el lado del ganadero hay un microcontrolador que recoge los datos vía radiofrecuencia y almacena o envía estos a la nube. Esto se puede ver esquemáticamente en la Figura 2.1. Todo esto se puede reducir básicamente a tres puntos básicos:

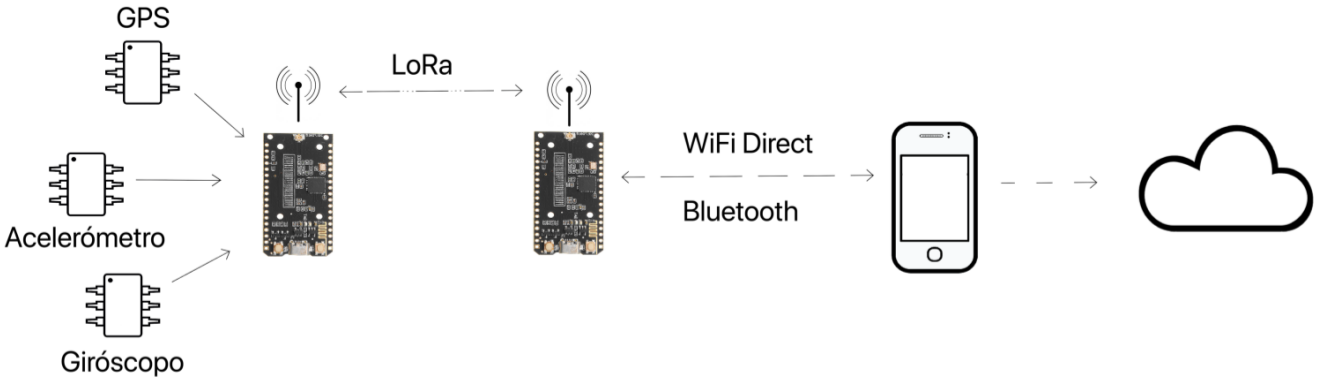


Figura 2.1: Idea esquemática del proyecto

1. Adquisición de datos. Se necesitan sensores que recojan los datos. En este caso se requiere medir de alguna forma el movimiento que realiza el animal. Esto es posible resolverlo de dos maneras distintas, se puede localizar y medir el movimiento a través de un sistema de posicionamiento inercial, y a través de un sistema de posicionamiento global. Es decir, se necesita un módulo que posea un acelerómetro y/o giróscopo, y otro módulo que posea un sistema Global Positioning System (GPS).

2. Almacenamiento y tratamiento de datos. Se necesita un chip que recoja directamente los datos de los sensores, los almacene y los trate.

3. Envío de datos y subida a la nube. Se requiere un chip que recoja todos los datos de los chips de las distintas vacas y los suba a una nube para poder visualizarlos.

Por tanto, para resolver el primer problema, se emplean dos periféricos, un módulo con acelerómetro y giróscopo, se selecciona un módulo de la familia MPU6000/6050, que se detalla en el Apartado 2.2.2. También se escoge un módulo GPS, concretamente Mini Locator GPS Navigation Module Position Tracker RoyalTek REB-5216 que se explica en el Apartado 2.2.2.

Todos los datos que proporcionen los sensores los gestiona un chip que los adquiere, almacena y trata, para poder ser enviados a un chip maestro cuando este los solicite. El chip escogido es el ESP32 el cual se detalla en el Apartado 2.2.1, es el mismo chip para ambos sistemas, esclavo y maestro. De esta forma no hay que estudiar ni diseñar varios dispositivos, ya que el mismo es capaz de solucionar los problemas básicos 1 y 2 que se acaban de describir.

2.2 Selección de componentes

2.2.1 Microcontrolador ESP32

Este chip dispone de una gran disponibilidad de interconexión con cualquier tipo de dispositivos a través de sus interfaces de comunicación. Cuenta con hasta cinco protocolos distintos de interconexión, tres cableados (Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C)) y dos inalámbricos (Bluetoothh, WiFi), los cuales se desarrollan en el Apartado 2.2.1. Además, este chip dispone de dos niveles de alimentación distintos (3,3V y 5V), así como pines de General Purpose Input/Output (GPIO) suficientes. Mencionar que estos chips tienen muchas más posibilidades y características, pero dado que no se usan directamente en el proyecto no se comentan, o se mencionan brevemente. El módulo que se usa es concretamente Heltec WiFi LoRa 32, que, además, integra LoRa, es decir, otro protocolo inalámbrico.

Dado que la problemática del proyecto yace en interconectar varios sistemas en el campo a grandes distancias, no hay que depender del WiFi o Bluetoothh, dado que son protocolos Wireless Local Area Network (WLAN). Se necesita, al menos, una red Wide Area Network (WAN). Para esto se utiliza LoRa que es una Low-Power Wide-Area Network (LPWAN), como explica en el Apartado 4.4 este protocolo es capaz de alcanzar grandes distancias y sirve para transmitir baja carga de datos con muy poca potencia.

Existen en el mercado distintos módulos ESP32 en el cual se integra la familia de chips SX127x que proporcionan una interfaz LoRa a través de SPI, y que, además, está totalmente integrado en el módulo. Estos dispositivos llevan integrado la adaptación a la antena LoRa y proporcionan la antena, de tal manera que solucionan completamente la problemática de la comunicación por radiofrecuencia.

Los chips SX127x disponen de transmisión a las frecuencias de 868 M/915MHz (SX1276) y de 433 MHz (SX1278). Conforme a las frecuencias a las que emite cada chip, se puede usar cualquiera de los dos, como se ve en el Apartado 4.4, la frecuencia de 433 MHz entra dentro de las bandas legales para radioaficionados y está liberada. La frecuencia 868 MHz es la que recomienda LoRaWan que se utilice en Europa. Lo que no se puede usar en ningún caso es el SX1276 a 915MHz.

Por último, el módulo posee un conector estándar para la conexión de baterías. Gracias a esto se convierte en un dispositivo desconectado de la red eléctrica.

Protocolos TX

El ESP32 tiene diversos protocolos de transmisión, de los que nos beneficiaremos para adquirir y transmitir los distintos datos en las etapas del proyecto.

En primera instancia cuenta con tres canales de UARTs (UART0, UART1, UART2). Este módulo tiene acceso a dos de ellas, concretamente a la UART0 y la UART2. La UART0 se utiliza para programar el ESP32 a través del Integrated Development Environment (IDE) de Arduino y muestra realimentaciones en el ordenador. Por lo que queda libre la UART2, pero estos pines coinciden con un bus de I2C que suele estar conectado a una pantalla Organic Light-Emitting Diode (OLED). Los módulos pueden ser adquiridos sin esa pantalla y poder así usar la UART2 para conectar al sensor que lo requiera. A la hora de programarlo se debe tener presente que hay que desactivar las funcionalidades que por defecto vienen configuradas para la pantalla OLED.

Tiene dos buses I2C, que se configuran como maestro o esclavo según la necesidad del proyecto. En este caso, es en modo maestro, dado que el ESP32 es el encargado de adquirir los datos. Se puede acceder a ambos

buses de I2C, pero como se acaba de comentar, uno de ellos suele estar configurado para comunicarse con una pantalla OLED, y si se desea usar la UART2, sólo se dispone de un canal de I2C. Estas interfaces de I2C pueden operar entre Standard mode (100 Kbit/s) y Fast mode (400 Kbit/s).

Por último en cuanto a protocolos cableados, hay un canal SPI de propósito general, a través del cual se conecta internamente el módulo adquirido al dispositivo SX127x, es decir, que sirve como interfaz LoRa. Además, tiene un canal Quad-SPI (QSPI) el cual se utiliza para comunicarse con la memoria Flash, y así implementar un protocolo SPI Flash File System (SPIFFS).

La familia ESP se caracteriza por la posibilidad de transmitir los protocolos inalámbricos más comunes a nivel de redes Wireless Personal Area Network (WPAN) y WLAN, como son Bluetooth y WiFi, respectivamente.

El protocolo WiFi que se implementa en el ESP32 incluye TCP/IP y las versiones 802.11 b/g/n. Este chip es capaz de actuar tanto como de punto de acceso como de estación, lo cual proporciona una gran variedad de posibilidades a la hora de transmitir datos entre distintos dispositivos.

En cuanto a Bluetooth, dispone de una interfaz a través de la cual se comunica con otros dispositivos, así como poder configurarse en modo Bluetooth Low Energy (BLE).

Por último, a través de la interfaz SPI de LoRa, se puede configurar las características del protocolo.

Flash

Este dispositivo tiene una memoria Flash de 16 MB, a través de la cual se se pueden almacenar archivos mediante del protocolo SPIFFS. Este protocolo permite usar la Flash como un directorio donde se pueden anidar otros directorios y almacenar archivos. Se implementa con instrucciones sencillas y similares a las que se usan en línea de comandos. Además, garantiza un uso eficiente de la memoria, dado que este protocolo fue creado para sistemas con memoria Flash reducida. El único detalle a tener en cuenta es la limitación del tipo de memoria ya que al ser una memoria Flash, los tiempos de escritura serán grandes comparado con el almacenamiento de los valores en Random Access Memory (RAM). Por tanto, no es recomendado escribir en la memoria Flash cada dato adquirido, si ese dato tiene una alta cadencia. Sino que es mejor acumular una serie de datos y luego almacenarlos, o hacer un procesado para reducir tanto el uso como el número de escrituras en la memoria Flash.

2.2.2 Periféricos

MPU6000/6050

Para detectar el movimiento inercial del animal de una forma eficiente, se emplea la familia de módulos MPU6000/MPU6050[1], que son placas con sensores Inertial Measurement Unit (IMU).

Estos módulos poseen principalmente un acelerómetro, un giróscopo y un termómetro. Así como la posibilidad de establecerse como maestros de otros sensores secundarios, como pudieran ser un magnetómetro y/o altímetro (no incluidos en este módulo). La principal utilidad de estos módulos es la de recoger los datos de velocidad inercial y de temperatura, además de almacenar varios datos antes de transmitirlos.

A la hora de recoger datos estos módulos tienen la capacidad de ser programados para utilizar de distintas maneras los sensores que llevan incorporados. Es decir, se pueden programar para usar uno, o varios de los sensores, y, además, cabe la posibilidad de modificar la frecuencia de muestreo de datos de los mismos. Todo esto es interesante analizarlo para concretar las necesidades fundamentales del proyecto dado que en función de qué y cómo se utilicen los sensores el sistema consumirá más o menos potencia.

El acelerómetro que posee se encuadra dentro de la tecnología Micro Electro-Mechanical System (MEMS). El funcionamiento básico del acelerómetro es el resultado del contacto de un elemento o masa libre dentro de un cubo, donde las paredes son piezoeléctricas y en el momento que la masa libre haga contacto con una o varias caras se envía un pulso eléctrico proporcional al contacto a los sistemas Analog to Digital Converter (ADC). Estos ADCs convierten en datos digitales esos pulsos y los almacenan en la memoria First in-First out (FIFO) del módulo.

El acelerómetro de este módulo puede ser programado de tal manera que el fondo de escala puede ser ±2, ±4, ±8 y ±16g. Posee a su vez integrados ADCs de 16 bits activos simultáneamente para cada eje de entrada de datos (x, y, z). Es decir que cada dato completo se compone de 6 Bytes, 2 Bytes por eje.

El giróscopo también se enmarca dentro de los elementos de tecnología MEMS. Siendo su funcionamiento algo más complejo. Teniendo en cuenta que es un sensor piezo-eléctrico, se basa en componentes que trataran

[1]Información complementaria en `https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/`

de convertir una magnitud física en una tensión o corriente eléctrica. Concretamente, este trata de convertir el momento angular analógico en datos digitales a través de los ADCs. Los giróscopos se basan en el principio de aceleración de Coriolis, donde hay una masa de prueba oscilando en un modo primario. En el instante que se modifica la oscilación en ese modo se generan nuevas fuerzas en un modo secundario. De tal manera que se puede detectar y calcular la aceleración angular.

El giróscopo este módulo se programa de tal manera que el fondo de escala puede ser ±250, ±500, ±1000, y $\pm 2000^o$/s. Posee a su vez integrados ADCs de 16 bits activos simultáneamente para cada eje de entrada de datos (x, y, z). Es decir que cada dato completo se compone de 6 Bytes, 2 Bytes por eje.

Conviene estudiar las distintas opciones que proporciona esta familia de módulos, dado que ofrece una variedad bastante interesante a la hora de ser utilizados. Me centraré en los sensores principales, acelerómetro y giróscopo, ignorando el termómetro y los posibles sensores esclavos, dado que no se prevé su utilización.

La decisión de usar uno u otro modo dependerá de dos factores clave. El primer lugar, los datos que se deseen tomar y su precisión. Y en segundo lugar la cantidad de potencia que se consuma adquiriendo esos datos. Como se observa en la Tabla 2.1 hay un modo normal de operación donde se usa el acelerómetro y el giróscopo de forma combinada, o de forma individual cada uno durmiendo el sensor no utilizado. Además, dispone de un modo de bajo consumo, exclusivamente para el acelerómetro, en el cual bajando la frecuencia de muestreo se reduce sensiblemente el consumo del dispositivo. Finalmente posee el modo de dormir el chip por completo.

CONSUMO	Giróscopo	Acelerómetro
Modo Normal	3,8 mA	
	3,6 mA	500 μA
Modo Low Power	No tiene	10 μA @ 1,25 Hz
		20 μA @ 5 Hz
		70 μA @ 20 Hz
		140 μA @ 40 Hz
Modo Sleep	5 μA	

Tabla 2.1: Modos de uso MPU6000/6050

En cuanto a la transmisión de datos al ESP32 se dispone del protocolo SPI en el dispositivo MPU6000 de hasta 1 MHz y del protocolo I2C, en ambos dispositivos de hasta 400 KHz. En la Tabla 2.2 se muestra que el protocolo I2C proporciona una velocidad suficiente para la cadencia de datos de estos módulos, por tanto, se utiliza el módulo MPU6050. Mencionar que tiene dos direcciones I2C distintas que se detallan al final del mismo apartado.

Bytes	Funciones	SPI 1MHz	I2C 400 KHz
14	Acel + Gyr +Temp	112 μs	280 μs
12	Acel + Gyr	96 μs	240 μs
6	Acel	48 μs	120 μs
2	1 Eje (de cualquier sensor)	16 μs	40 μs

Tabla 2.2: Tiempos de transmisión MPU6000/6050

Tal y como se ve en el Apartado 2.2.3, el factor crítico en el sistema es el consumo de batería y la necesidad de reducirlo. Por tanto, hay que configurar este sensor para que sea operativo con el mínimo consumo posible. Realmente la principal utilidad de este sensor es medir la cantidad de movimiento que realiza el animal, de tal manera, que la orientación en la que se mueve el animal es trivial, por tanto, se prescinde del giróscopo.

Hay dos opciones para medir la cantidad de movimiento que realiza el animal: una es con el módulo del movimiento (norma matemática del vector de movimiento) y la otra procesando los datos que nos arrojan los ejes del acelerómetro y contando pasos. En cualquier caso se usa sólo el acelerómetro, durmiendo el resto de sensores. Respecto a la frecuencia de muestreo de este, la diferencia de consumo es relativamente despreciable. Para estimar la resolución necesaria de muestreo para el MPU6050 se realizan distintos experimentos a distintas frecuencias. El experimento consiste en sensar los datos del eje Z y almacenarlos inmediatamente en la memoria Flash. Denotar que no se hace simultáneamente a la vez, pero el recorrido es similar en los tres casos. Vemos en la Figura 2.2 como a 1,25 Hz no se tiene la sensibilidad suficiente y se pierde información, cómo a 20 Hz quizá sea demasiada información y sea más complejo de tratar. Por lo expuesto, se decide muestrear a 5 Hz.

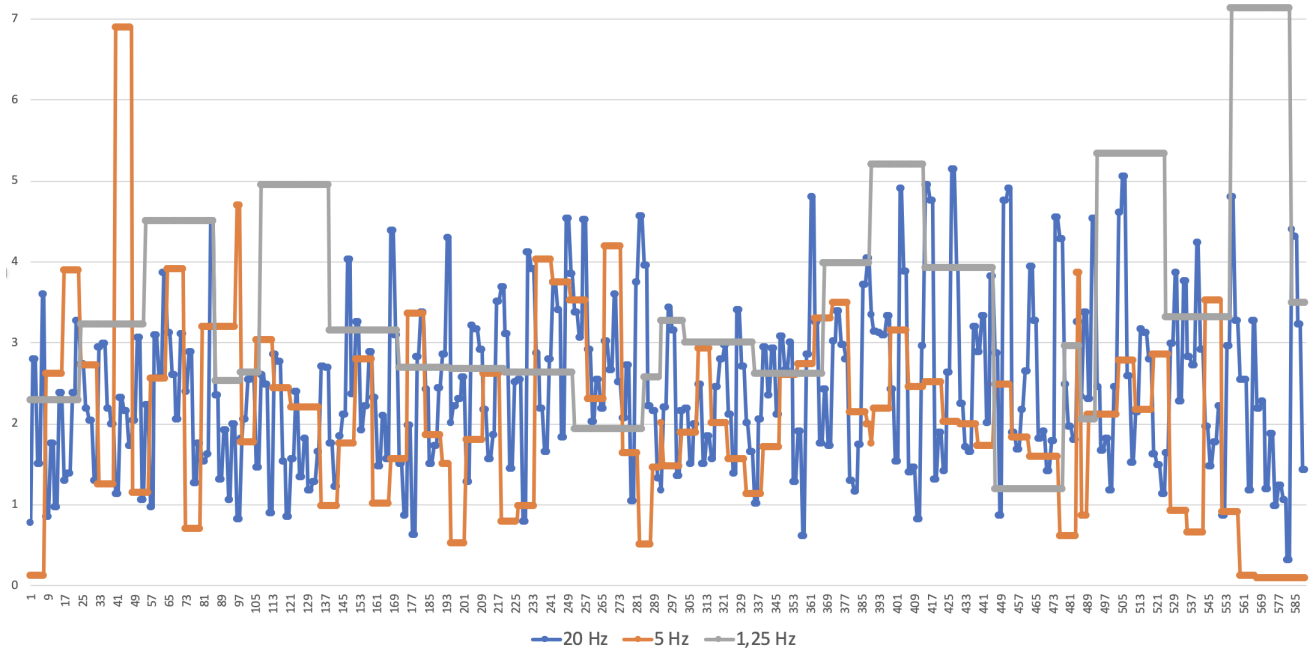


Figura 2.2: Datos del eje Z en distintos modos de ahorro

GPS

El segundo periférico que se incluye en el sistema es un dispositivo GPS. Con el fin de aprovechar la información que envíen los satélites GPS. Ya no sólo la posición a través de longitud y latitud, sino también permite adquirir datos como el día y la hora en la que se adquieren los datos.

El posicionamiento GPS se basa en la triangulación de varios satélites, concretamente con tres satélites se puede determinar la posición, pero para evitar problemas, como se ve más adelante, es recomendable incluir un cuarto satélite GPS.

Básicamente, un satélite tiene un radio de acción, que, si detecta el receptor, estima que puedes estar en cualquier punto de la circunferencia en torno a dicho satélite. Si se incluye un segundo satélite se puede determinar un plano generado por las dos circunferencias de radio en torno a ambos satélites. Incluyendo el tercer satélite, esa superficie se puede reducir a dos puntos. Muchas veces con sólo tres satélites detectados se determina con precisión la posición, puesto que de los dos puntos calculados, uno suele ser improbable debido a su altura o profundidad, no estando en el entorno de la superficie terrestre. Pero si se consigue detectar un cuarto satélite, se discierne en qué punto de los dos se encuentra el receptor.

El módulo escogido para esta funcionalidad es el Mini Locator GPS Navigation Module Position Tracker RoyalTek REB-5216. Las principales características de este dispositivo GPS son el soporte para GPS, Global'naya Navigatsionnaya Sputnikovaya Sistema (traduce del ruso) (GLONASS) y Satellite Based Augmentation System (SBAS), la posibilidad de alimentarlo a 3,3V y la conexión a través de la UART [2].

Tiene la posibilidad de un modo de bajo consumo en el cual no arrojará datos pero en la siguiente reconexión tarda menos en adquirirles. Además, dispone de un pin que proporcionará un pulso por segundo.

Este dispositivo GPS transmite los datos en cadenas de caracteres del estándar NMEA. Este estándar define una serie de frases con unos parámetros que proporcionan la información necesaria para dar la localización del dispositivo. Estas frases, o comandos en principio sólo las envía el GPS. Las frases constan de dos partes, un identificador que denota el tipo de información que va a continuación. Este identificador va precedido del símbolo $. Y a continuación siguen los parámetros y la información de la frase.

El REB-5216 principalmente envía cuatro tipos distintos de información[3]. En primer formato de frase es el que viene precedido del identificador $GPGGA. Véase el formato completo en la Tabla 2.3. Esta frase indica la hora solar (Coordinated Universal Time (UTC)) donde se sitúa el receptor, los datos de localización (latitud

[2]Hoja de datos disponible en `https://www.itead.cc/wiki/images/b/bb/REB-5216_operational_manual_v0_1_20150122.pdf`
[3]Más información sobre NMEA en `https://www.gpsinformation.org/dale/nmea.htm`

y longitud) y los satélites detectados. Mencionar que si el dispositivo no encuentra datos de un parámetro, lo deja vacío entre comas.

Parámetros	**ID**	**UTC**	**Latitud**	**N/S ID**	**Longitud**	**E/W ID**	**Posición ID**	**Satélites usados**
Ejemplo	$GPGGA	160332.421	4166.2773	N	470.6064	W	,	06
Formato		hhmmss.sss	ggmm.mmm	Norte/Sur	gggmm.mmmm	Este/Oeste		0-12

HDOP	**Altitud**	**Uds**	**Separación Geoidal**	**Uds**	**Era de las coordenadas**	**DFS ID**	**Checksum**
,	9	M	,	M	,	0000	*27
		metros		metros			

Tabla 2.3: Formato frase $GPGGA

La siguiente trama es una frase GSA. Esta frase muestra la información de interés en su identificador. Si muestra el identificador $GPGSA cuando sólo se están usando satélites de la red GPS. Si empieza por $GLGSA se están usando exclusivamente satélites GLONASS o SBAS. Y si muestra $GNGSA se usan satélites combinados de ambos sistemas.

A continuación muestra de forma detallada en la Tabla 2.4 la información de cada satélite detectado. Tienen dos cabeceras distintas pero el formato es análogo para ambas. Estas cabeceras son $GPGSV para información de satélites GPS y $GLGSV para satélites GLONASS o SBAS. En cada frase puede llevar la información de hasta tres satélites.

Parámetros	**ID**	**N^o Mensajes**	**N^o de Mensaje**	**Satélites detectados**	**Satélite ID**	**Elevación**	**Azimut**	**SNR**	**2^o Satelite ID**	**...**	**Cheksum**
Ejemplo	$GPGSV	2	1	06	08	76	040	50	06		
Formato						grados	grados	dBHz			

Tabla 2.4: Formato frase $GPGSV/$GLGSV

Por último tenemos la trama $GNRMC, la cual es bastante similar a $GPGGA. Esta resume la información obtenida en una trama como se puede ver en la Tabla 2.5. En ella se puede ver, la fecha y hora, los datos de posición (longitud y latitud), y otros datos adicionales como diversas velocidades y variaciones.

Parámetros	**ID**	**UTC**	**Estado**	**Latitud**	**N/S ID**	**Longitud**	**E/W ID**	**Velocidad sobre tierra**
Ejemplo	$GNRMC	105567.897	A (Valido)	4200.3156	N	452.9172	W	0.5
Formato		hhmmss.sss	V (Erroneo)	ggmm.mmmm	Norte/Sur	gggmm.mmmm	Este/Oeste	nudos

Grados sobre tierra	**Fecha**	**Variación Magnética**	**VS**	**Modo**	**Cheksum**
207	200619	,	,	,	*89
grados	ddmmyy	grados			

Tabla 2.5: Formato frase $GNRMC

2.2.3 Batería y estrategia de uso

Al requerirse una cierta autonomía, se hace un estudio de consumo para estimar la capacidad de la batería que se necesita para cumplirlo. Para ello, se analiza dispositivo por dispositivo en la Tabla 2.6.

Se puede apreciar cómo el GPS es el dispositivo que más consume, y que transmitir en LoRa también es un factor bastante crítico a la hora del consumo. Por tanto, hay que buscar una solución para reducir el uso del GPS y el uso de LoRa.

Para el GPS se estima un uso cada 5 minutos, es decir se toman 288 muestras al día (a priori). En condiciones óptimas tarda 35 segundos en adquirir datos (según la hoja de datos), se considera 1 min para asegurar de que logra recoger datos válidos. En este supuesto estaría encendido un 20% del día e hibernando un 80%. Es decir que el consumo del GPS prorrateado se estima en la Ecuación 2.1

$$0,8 \times 50\ \mu A + 0,2 \times 35\ mA = 7,04\ mA \tag{2.1}$$

Se puede complicar un poco esta estrategia de uso del GPS con afán de ahorrar aún más en el consumo de la batería. Pudieran plantearse dos supuestos adicionales.

En el primer supuesto, se considera que el animal tiene momentos a lo largo de la jornada en el que su movimiento va a ser reducido, por ejemplo, cuando este esté comiendo o durmiendo. En estos momentos se puede bajar la frecuencia de muestreo del GPS. Es decir, cuando se detecte que apenas se ha movido en sucesivas muestras, se reduciría el muestreo de 5 minutos a 30 minutos, y si se detecta que ha habido un movimiento no despreciable, volver al muestreo original.

Dispositivo	Modo	Consumo
GPS	Normal	35 mA
	Hibernate	50 μA
MPU6050	LP 1 Hz update	10 μA
	LP 5 Hz update	20 μA
	LP 20 HZ update	70 μA
SX127x (LoRa)	Sleep	0,2 μA
	Idle	1,5 μA
	Standby	1,6 mA
	Receive mode	12 mA
	Transmit mode 20 dBm (máx.)	120 mA
	Transmit mode 7 dBm	20 mA
ESP32	Normal mode 80 MHz	25 mA
	Low-Speed 2 MHz	4 mA
	Ligth-Speed	0,8 mA

Tabla 2.6: Consumo de los dispositivos

En el segundo supuesto, se considera que el sistema GPS puede funcionar mal o no funcionar. Es decir, esta situación se da cuando el GPS no consigue los datos de posición (véase Apartado 7.2.2), en este caso es absurdo consumir potencia cuando el sensor no nos proporciona los datos requeridos. Lo que se plantea en ese caso es que si en sucesivas lecturas no se consiguen recoger datos, se pasa a hacer una lectura más larga, de 5 minutos, para ver si se consigue recoger datos. Si no consiguiera detectar los datos de posición, pasaría a un modo de hibernación continuo hasta el día siguiente cuando se volvería a intentar detectar la posición nuevamente.

El siguiente elemento a tener en cuenta es el SX127x y la comunicación LoRa. Realmente los chips que van en las vacas dedican un tiempo bastante despreciable a transmitir, incluso se puede reducir haciendo un tratamiento de datos en el mismo chip para reducir aún más el tiempo en modo de transmisión de LoRa. Sin embargo, este sistema sí que ha de estar bastante tiempo pendiente en modo recepción a que el chip maestro requiera los datos almacenados. En el caso en el que esté el tiempo suficiente encendido esperando esta petición, parece absurdo tenerlo encendido durante periodos que es poco probable que se use, véase por la noche. Por tanto, como primera medida se tiene encendido sólo las 12 horas centrales del día (de 8:00 a 20:00 apróx.). Es decir que estaría la mitad del tiempo dormido y la otra mitad en modo receptor, se ve dicho consumo en la Ecuación 2.2.

$$0,5 \times 0,2\ \mu A + 0,5 \times 12\ mA = 600\ \mu A \tag{2.2}$$

Adicionalmente, se puede plantear que ese horario de recepción sea programable vía aplicación móvil, de tal manera que se pudiera reducir a solo un par de horas al día. Sin embargo el consumo en este modo y con la anterior estrategia de uso no es tan crítica comparado con otros componentes.

Por último, es necesario modular el consumo tanto del ESP32 como del MPU6050. Para ambos dispositivos se afronta una estrategia de uso conjunta, dado que el MPU6050 es el sensor con mayor cadencia de datos y, por tanto, el que limita principalmente el uso del ESP32.

Por tanto, con una frecuencia de 5 Hz, el chip, es capaz de tomar datos. Se muestrea cinco veces por segundo, a lo que hay que añadir el tiempo de transmisión del protocolo I2C que se ve en la Ecuación 2.3.

$$\frac{3\ datos \times 16\ Bits}{400\ KHz} = 120\ \mu s \tag{2.3}$$

Es decir, cada 200 ms tarda 120 μs en adquirir el dato. El resto del tiempo se puede dormir el ESP32. Esto significa que estaría 120 μs encendido, y 199,88 ms apagado. Por lo que se reduce drásticamente el consumo, como se puede ver en la siguiente Ecuación 2.4.

$$999,4 \times 10^{-3} \times 0,8\ mA + 0,6 \times 10^{-3} \times 25\ mA = 814,52\ \mu A \tag{2.4}$$

Se ve en la Tabla 2.7 un resumen de consumos de los dispositivos, en la cual, se denota nuevamente que el factor limitante es el dispositivo GPS. Hay que mencionar que todos estos cálculos están hechos de manera independiente para cada dispositivo, y en las mejores condiciones posibles.

Dispositivo	Consumo
GPS	7,04 mA
ESP32	0,814 mA
SX127x	0,6 mA
MPU6050	0,02 mA
Total	8,475 mA

Tabla 2.7: Consumo final de los dispositivos

Tiempo autonomía	Capacidad batería (Ah)
3 meses	18,916
2 meses	12,610
1 mes y 15 días	9,355
1 mes	6,305
15 días	3,051

Tabla 2.8: Relación tiempo autonomía y capacidad de la batería

Para estimar la batería que se requiere, se plantean una serie de tiempos de autonomía en la Tabla 2.8 y se analiza la capacidad que se necesita para cada caso.

Al final se decide implementar en el sistema una batería de 13,4 Ah, la cual va a proporcionar una autonomía de algo más de dos meses (65 días apróx.). La batería concretamente es Panasonic PICPAULLNB55. Cabe destacar, que no se tiene en cuenta en ningún momento los efectos que se puedan dar en la batería. Por ejemplo, el paulatino envejecimiento después de varios ciclos de carga, y efectos de la temperatura como la bajada de su rendimiento (con bajas o altas temperaturas).

Lo que hay que tener en cuenta es que la batería es bastante más grande de las que suelen ser utilizadas para este tipo de placas. Por tanto, como se ve en el esquemático de la placa ESP32 en el Apéndice A.3, por defecto viene con una resistencia que controla el regulador de carga de un valor de 10 K Ω. Esto implica que carga a través del puerto Universal Serial Bus (USB) con una corriente de 100 mA. Es decir, que para cargar completamente una batería de 13,4 Ah se tarda 134 h (apróx. 5 días y medio). Esto es inviable, por tanto, se deberá substituir la resistencia de 10 K Ω por una de 1 K Ω, como se observa en el Apartado 6. Con este cambio se consigue que el sistema cargue la batería con el máximo amperaje posible, es decir, 1 A. Con esta disposición se cargará completamente en 13h y 24 min.

2.3 Captura esquemática

2.3.1 Captura esquemática de los componentes

Los dos periféricos principales, tanto el GPS, como el MPU6050 son dispositivos que están disponibles en formato módulo con pines, lo que simplifica el montaje y su maquetación esquemática.

Captura esquemática del MPU6050

En la Figura 2.3 se puede ver dispositivo MPU6050 y los pines que dispone. Así en función de esta imagen y de la hoja de datos, se diseña el esquemático que se ve en la Figura 2.4.

- VCC: Alimentación del dispositivo. 2,375V-3,46V. Típicamente 3,3V.
- GND: Conexión a masa del dispositivo.
- SCL: Línea para el reloj en el protocolo I2C.
- SDA: Línea para los datos en el protocolo I2C.
- XCL: Línea para el reloj en el protocolo I2C actuando como maestro de un segundo sensor.
- XDA: Línea para los datos en el protocolo I2C actuando como maestro de un segundo sensor.

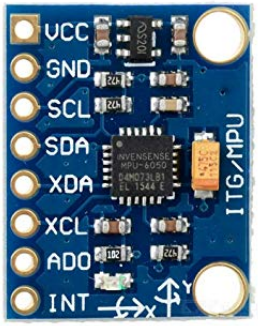


Figura 2.3: MPU6050

- AD0: El pin a través del cual se configura la dirección I2C del sensor. Este sensor tiene la capacidad de tomar dos direcciones: 0b110100x pudiendo conectar este pin a masa o a alimentación para fijar el bit menos significativo.
- INT: A través de este pin se pueden programar interrupciones.

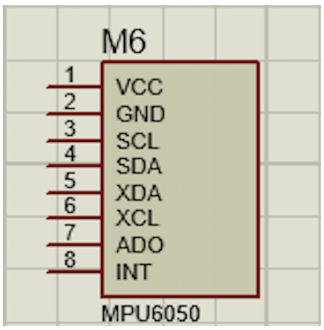


Figura 2.4: Diseño esquemático del MPU6050

Es importante destacar que el MPU6050, como se ve en el Apéndice A.1, ya dispone internamente de las resistencias de pull-up para los bus del protocolo I2C. Por tanto, para este dispositivo hay que tender dos cables de alimentación (uno para VCC y otro para GND), las dos líneas del bus I2C y la conexión de AD0. Se considera la dirección de I2C menor, es decir 0x68. Para ello hay que conectar el pin a GND, pero en la hoja de datos se recomienda poner una resistencia entre el pin y masa de 4,7 kΩ.

Figura 2.5: Mini Locator REB-5216

Captura esquemática del REB-5216

En el dispositivo Minilocator REB-5216 de la Figura 2.5, se pueden ver los pines de los que se compone. También se diseña la captura esquemática del componente que se ve en la Figura 2.6.

- VCC: Alimentación del dispositivo a 3,3V.
- GND: Conexión a masa del dispositivo. Dispone de dos pines. A pesar de estar los dos internamente conectados se recomienda conectar ambos entre sí. Esto se puede observar en el Apendice A.2.
- RXD: Línea a través de la que el dispositivo puede recibir datos del ESP32, mediante el protocolo UART.
- TXD: Línea a través de la que el dispositivo transmite datos al ESP32, mediante el protocolo UART.
- 1PPS: Arroja un pulso por segundo. (1 Pulse Per Second)
- VBAT: Es un pin de alimentación. Debe estar siempre a 3,3V y, además, tiene una funcionalidad especial. Este pin mantiene una alimentación básica cuando se desconecta la alimentación de VCC. Si se da ese caso, el dispositivo entra en un modo de hibernación. En este modo no se reciben datos pero apenas se consume potencia, como se explica en el Apartado 2.2.3. Su utilidad es proporcionar una transición rápida entre el modo dormido y el activo, asegurando una captación más rápida de satélites.

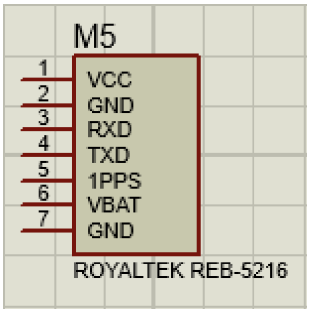


Figura 2.6: Diseño esquemático del Mini Locator REB-5216

Captura esquemática del módulo ESP32

Por último, se cuenta con la placa ESP32 compuesta por 36 pines accesibles, como se ve en la Figura 2.7.

La primera función fundamental del microcontrolador es alimentar el resto de componentes del sistema. Esto es gracias a sus reguladores y sus conexiones, tanto por el puerto USB, como a través de una batería. Es decir, los pines de alimentación (en este caso de 3,3V) proporcionarán la tensión al resto de dispositivos. Por tanto, se deben tender pistas para conectarlos. Es importante recordar que se tienen que unir todas las masas para que no haya problemas de distintas tensiones de referencia. Los circuitos reguladores están integrados en el mismo módulo y, además, se les practica una ligera modificación, tal y como se comentó al final del Apartado 2.2.3.

La siguiente función fundamental es la de recoger los datos de los sensores. Se emplea la UART2 para conectar con el GPS, usando el pin 17 como TXD y el pin 16 como RXD; y se usa el canal de I2C que está accesible, con el pin 22 para SCL y el pin 21 para SDA. El resto de funcionalidades del ESP32 se resuelven vía software. Mencionar que la filosofía para generar el diseño esquemático del ESP32 es indicar el pin de la función que se va a utilizar o la de uso más común.

2.3.2 Componentes adicionales

Además de los componentes principales que se conectan a través de unos conectores hembra a la PCB, hay otros componentes para completar el sistema. Lo primero a tener en cuenta son los condensadores de desacoplo en las alimentaciones, estos reducen el ruido en las mismas. Se dispondrá de un condensador de desacoplo para cada pin de alimentación de valor de 100 nF, a excepción del módulo GPS. El módulo REB-5216 recomienda en su hoja de datos condensadores más grandes, por tanto, se implementa un condensador de 10 μF tanto a la salida del pin VCC como a la del pin VBAT del GPS.

Para gestionar el apagado y encendido del GPS se propone un circuito de encendido. Al no poder ponerse en modo de bajo consumo a través de comandos, el modo de hibernación se consigue desconectando la alimentación de VCC. Al necesitarse un reinicio rápido, se debe dejar la alimentación conectada a VBAT. Se propone conectar un pin de GPIO para controlar la alimentación del GPS, pero el ESP32 sólo es capaz de

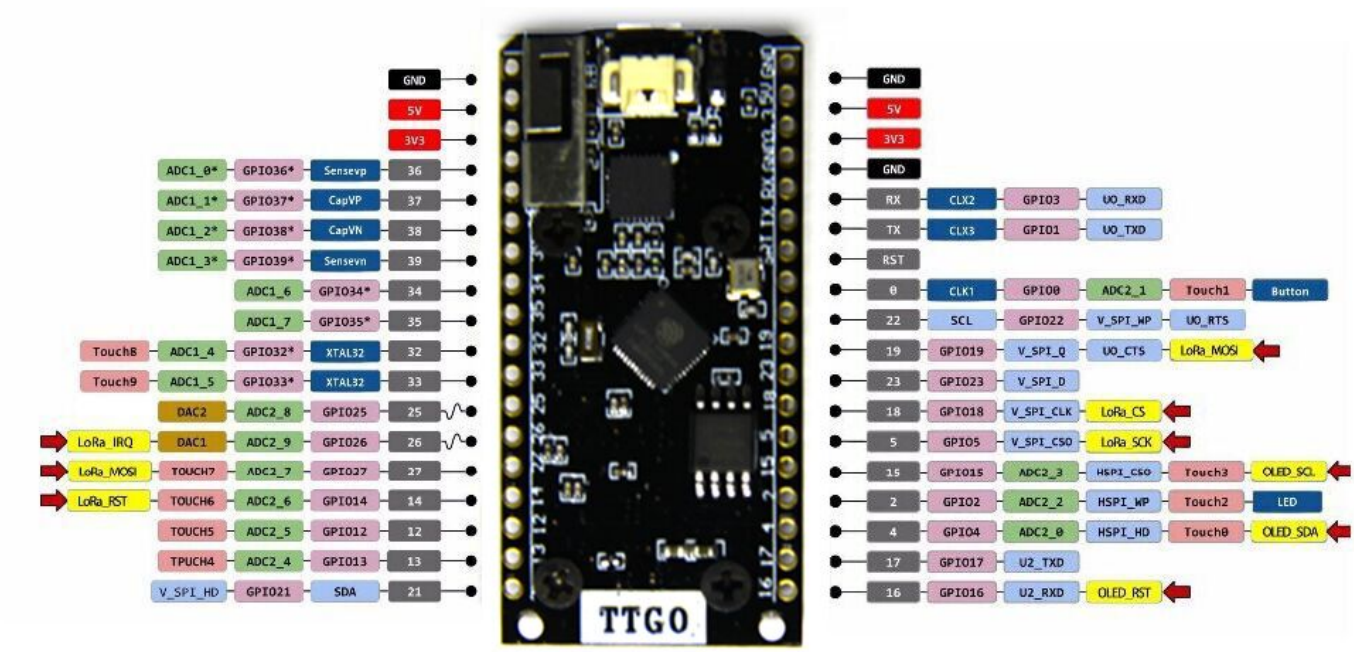


Figura 2.7: Pines ESP32 LoRa

proporcionar 30 mA a través de esos pines, y el módulo GPS necesita 35 mA. Por tanto, se usa un interruptor a través de un Metal-oxide-semiconductor Field-effect transistor (MosFet), activo en baja, que proporciona la intensidad que falta. Este MosFet se controla a través de un pin cualquiera de la GPIO. Para fijar el estado de alto se conecta una resistencia de pull-up de la alimentación al pin de GPIO de control y la fuente del MosFet.

2.3.3 Diferentes versiones

Ya se comentó que se necesitan dos sistemas, uno para dejarlo con el animal y otro que tiene el ganadero. El diseño del sistema de la vaca es el descrito anteriormente, y el diseño del sistema del ganadero es simplemente el mismo esquemático, pero solamente con el chip ESP32. Dado que este no necesita, ni los sensores, ni los componentes para intercomunicarlos ni adaptarlos.

Además del esquemático, se muestra un documento donde se concreta el precio de los componentes y donde adquirirlos, este documento es el Bill of Materials (BOM).

Mencionar que en ambos esquemáticos hay notas explicando brevemente cada componente, y se plantea la estrategia de uso a partir de la cual se empezó a trabajar. Ambos esquemáticos se pueden encontrar en el Anexo B.

2.4 Diseño PCB

El diseño de la PCB viene marcado por varios factores, el más importante es el tamaño. Esto se debe a que la batería escogida limita el tamaño del sistema (68 mm x 80 mm x 19 mm). Entonces hay hueco de sobra para colocar los componentes. Si bien es cierto que no caben todos en una misma cara, se aprovechan ambas para su colocación.

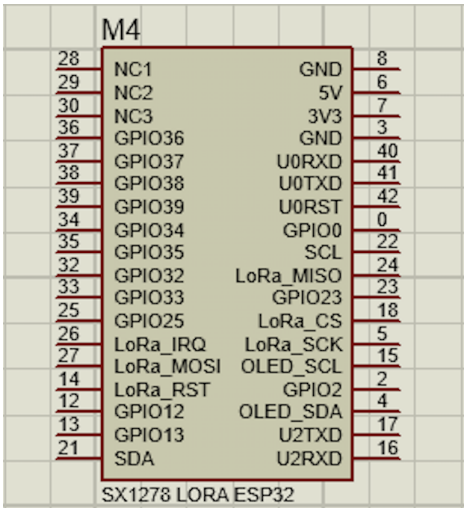


Figura 2.8: Diseño esquemático ESP32 LoRa

Previo a su colocación se diseñan las huellas que se corresponden con los componentes, añadiendo la serigrafía suficiente y necesaria para tener la máxima información a simple vista. Por ejemplo, cual son los pines de alimentación, los del protocolo de comunicación, o la orientación de los ejes en el caso del MPU6050. Se puede ver la huella del módulo ESP32 en la Figura 2.9, la del módulo GPS en la Figura 2.10 y la del MPU6050 en la Figura 2.11. Vemos que tanto en el ESP32 como en el MPU6050, faltaría serigrafía importante, esta se añade a posteriori, fuera de la huella del elemento para poder disponerla en el mejor sitio posible sin que interfiera con otros elementos. Mencionar que hay que tener un especial cuidado en el diseño de las huellas. Dado que al diseñar las capturas esquemáticas el tamaño se puede elegir arbitrariamente. En este caso, el tamaño ha de ser exactamente el del componente real, para evitar problemas posteriores en el montaje.

Figura 2.9: Huella del ESP32

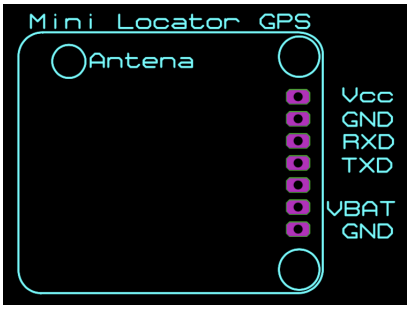


Figura 2.10: Huella del GPS

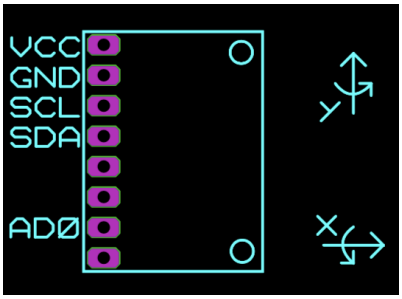


Figura 2.11: Huella del MPU6050

2.4.1 Posicionamiento de los elementos

Una vez disponibles todas las huellas de los elementos a tener en cuenta, se procede a su colocación sobre la PCB. Lo primero que se tiene en cuenta es que el puerto USB del ESP32 ha de quedar disponible al exterior, por tanto, se coloca el módulo en el borde de la placa con el puerto USB lo más cercano al exterior posible. Como se comenta al inicio de esta misma sección, no caben todos los elementos en una misma cara, por tanto, se colocan ambos módulos sensores en la otra cara. Dado que el módulo GPS dispone de una antena, y necesita ir lo más exterior posible, se dispone el GPS en el extremo opuesto al puerto USB con conexión a la antena lo más cerca del borde. Entonces el módulo MPU6050 se coloca en el hueco restante.

Para colocar los elementos pasivos se tiene en cuenta, que módulo es el que necesita dicho elemento, y se intenta colocar lo más cerca del pin posible. Así, por ejemplo, todos los condensadores de desacoplo se sitúan lo más cerca posible de las alimentaciones de cada dispositivo. Y el circuito que controla la alimentación del GPS lo más cerca del pin de GPIO12. Por último, se sitúa la resistencia de 4,7 K Ω lo más cerca posible del pin AD0 que fija la dirección I2C del MPU6050.

2.4.2 Rutado y planos de masa

Dispuesta la colocación de los elementos el rutado es bastante simple, debido a que no hay demasiados dispositivos. Se intenta rutar siempre por el camino más corto evitando hacer excesivos agujeros. Comentar que se ruta la alimentación directamente desde el pin de 3,3V del ESP32 a cada elemento.

Se coloca un plano de masa en ambas caras de la PCB para evitar problemas con el ruido y de distintas referencias. Pero no se colocan cubriendo completamente ambas caras. Esto se debe a que el mismo módulo ESP32 dispone de antenas integradas como se puede ver en la Figura 2.7. Por tanto se consulta el manual de diseño hardware del ESP32 [4] se recomienda que se deje, en la medida de lo posible, un radio de 15mm alrededor de la antena, para evitar que los elementos o el plano de masa puedan amortiguar la señal de la antena. Se ve en la Figura 2.12 una ilustración sacada del manual de diseño del ESP32 donde se explica gráficamente lo comentado. Por tanto, en base a lo consultado en el manual, se libera de plano de masa y de elementos la zona colindante a la antena.

2.4.3 Serigrafía y plano mecánico

El plano mecánico de la placa, se disponen cuatro agujeros en las esquinas de la PCB para dar la posibilidad de sujetarla en el recinto donde se vaya a colocar. Además, se replican los agujeros de los componentes, por si necesitasen una sujeción complementaria. Mencionar que el tamaño final de la placa es de 64x64 (mm).

Se completa la serigrafía que no se pudo colocar en las huellas de los elementos. Se evita que la serigrafía esté sobre huellas o agujeros para su completa comprensión. La serigrafía colocada es relativa a los protocolos de comunicación, por si se necesitan conectar al osciloscopio para realizar estudios, testeos o correcciones; y además, se añade serigrafía relativa a la alimentación. Huelga decir que cada elemento esta identificado con su sigla que corresponde con la del esquemático, esto simplifica el montaje y previene de errores en el mismo.

Por último, como se observa en el diseño final de la PCB en las Figura 2.13 y 2.14, se añade tres identificadores a la placa. Uno indica el nombre del autor, otro la versión hardware y finalmente se deja un cuadrado para poder escribir la versión software que se implemente. Además se incluye un logo de la facultad en la que se ha realizado este proyecto (ETSIT UVa).

[4] Disponible en `https://www.espressif.com/sites/default/files/documentation/esp32_hardware_design_guidelines_en.pdf`

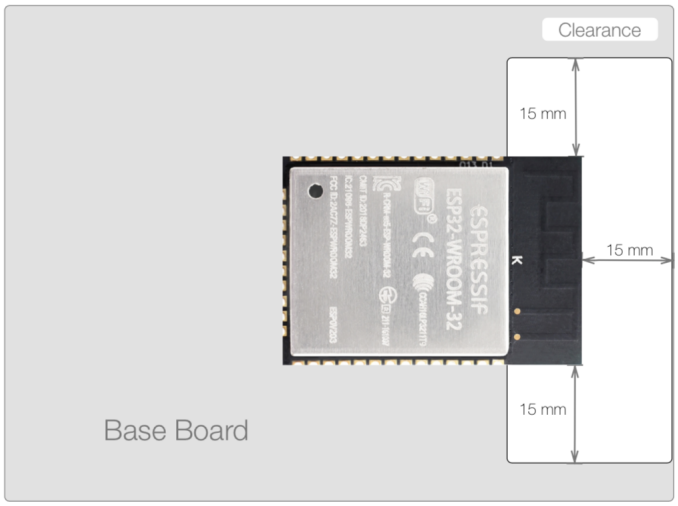


Figura 2.12: Zona libre de elementos para la antena del ESP32

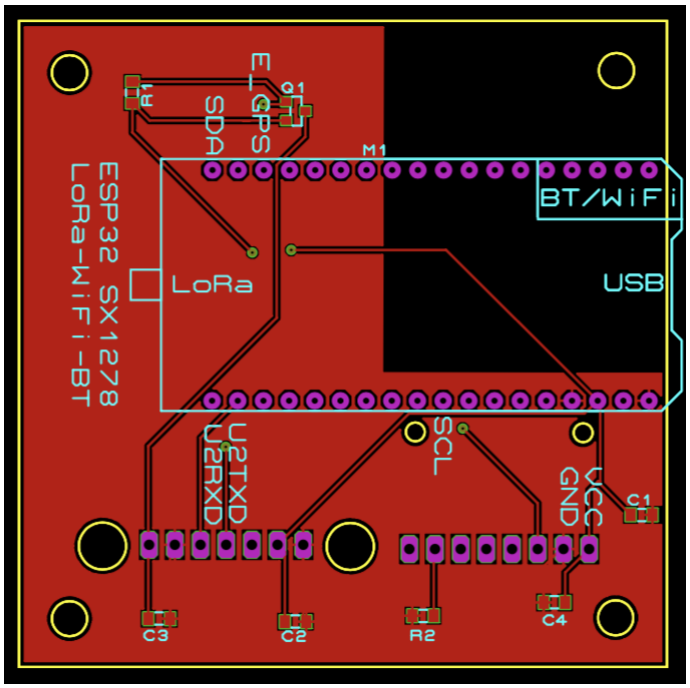


Figura 2.13: Diseño de la parte superior de la PCB

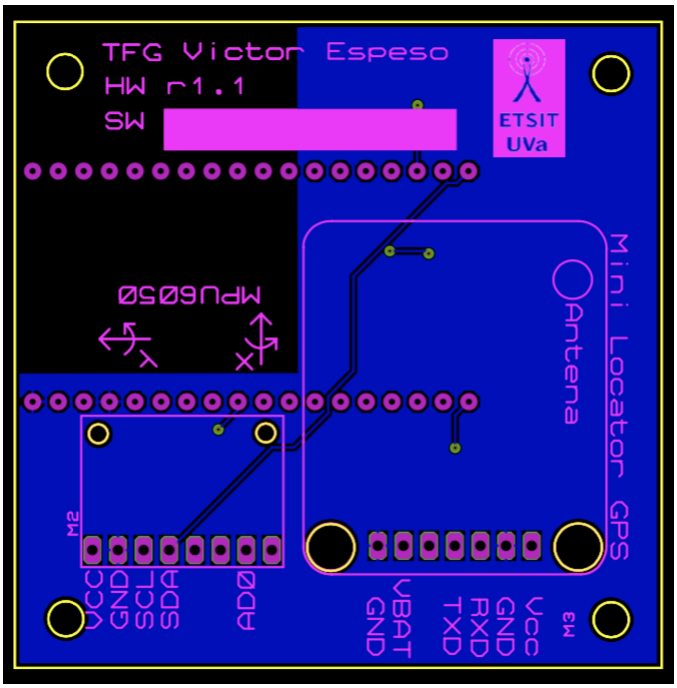


Figura 2.14: Diseño de la parte inferior de la PCB

Capítulo 3

Caja

Se fabrica una caja para resguardar el sistema electrónico de las condiciones atmosféricas y también para facilitar la sujeción al animal.

Para diseñar la caja existen múltiples herramientas Computer-Aided Design (CAD) que facilitan la tarea. Se puede optar por herramientas on-line e incluso herramientas que son instalables en el ordenador. Además, se puede elegir, por un lado, generar un diseño mediante bloques gráficos, es decir, a través de una interfaz en la cual se arrastran los elementos y se fijan las características en dichos objetos. Un ejemplo de esta herramienta es Tinkercad [1]. Por otro lado, existen herramientas que generan el objeto a través instrucciones escritas o código. Este último tipo de herramienta es el usado para el diseño de la caja en el proyecto.

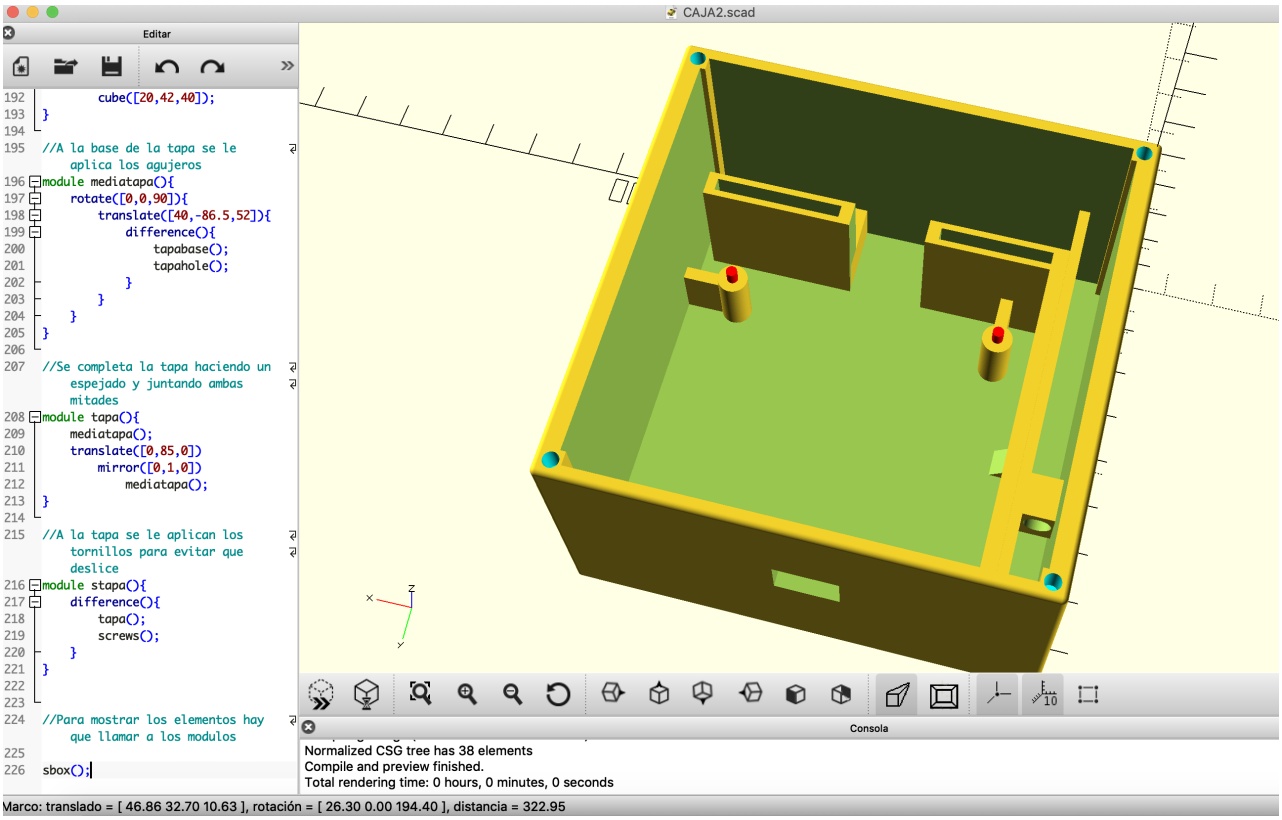


Figura 3.1: Interfaz OpenSCAD

[1] https://www.tinkercad.com

3.1 OpenSCAD

La herramienta usada es OpenSCAD, es una herramienta que se puede instalar en los sistemas operativos más usados (Linux/UNIX, Windows y MacOs) y que genera el objeto a través de simples instrucciones. Cabe destacar que es un software libre de código abierto y gratuito. Se puede descargar en su página `https://www.OpenSCAD.org/index.HTML` y en la que, además, se puede visualizar la documentación [2] de las instrucciones para poder construir los distintos objetos 3D.

Como ya se ha comentado, hay que generar un archivo que después de compilarse, se genera un objeto que se puede previsualizar de manera rápida. Una vez se tenga el objeto se puede renderizar para crear un modelo más exacto de lo que se quiere construir. Por último, si esa renderización no arroja ningún error, el programa es capaz de generar un archivo .stl, un tipo de archivo que geometriza un objeto 3D.

Se ve la interfaz gráfica del OpenSCAD mostrando la versión dos de la caja en la Figura 3.1, a la derecha tiene para escribir las instrucciones, debajo un cuadro que emite retroalimentaciones a la hora de compilar el código y en el centro se puede ver la previsualización del objeto diseñado. Huelga decir que los colores mostrados del objeto en OpenSCAD son una ayuda para el diseño, ya que al final el objeto se imprime en el color del plástico que se disponga en la impresora 3D.

3.1.1 Comandos básicos OpenSCAD

Se comentan las instrucciones básicas con las que se construyen objetos, destacar que la unidad de los parámetros en todas las instrucciones es milímetros:

- cube([ancho,largo,alto]); Se introducen los parámetros, y genera un ortoedro de las dimensiones detalladas.
- cylinder(altura,radio inferior, radio superior); Se genera un cilindro con la altura y el radio introducido. Variando adecuadamente los parámetros podemos generar con esta instrucción pirámides y conos.
- sphere(radio | d=diámetro); Genera una esfera con el radio o el diámetro estipulado.

Para los elementos que incluyen cierta curvatura, véase el cilindro o la esfera, se le puede añadir un parámetro adicional. Este parámetro es $fn y modula el número de segmentos que se utilizan para dibujar la curva. Por ejemplo, si se generase un cilindro con el parámetro $fn=4 se obtendría un polígono de cuatro caras laterales, donde no se apreciaría curvatura alguna. Se recomienda trabajar con el parámetro reducido, cuanto mayor sea este valor más tarda en generar la previsualización. En cambio para generar el producto final es recomendable subir el parámetro ostensiblemente para un mejor acabado.

Se dispone también de modificadores para estos elementos, estos modificadores se colocan previamente a los elementos, y finalizan en el código con punto y coma (;), dado que la instrucción termina con la generación del elemento modificado.

- translate([x,y,z]) Traslada el elemento los milímetros indicados en cada eje.
- rotate([x,y,z]) Rota el objeto en los grados indicados en cada eje.
- scale([x,y,z]) Multiplica las dimensiones del objeto por el factor indicado en cada parámetro.
- mirror([x,y,z]) Hace la función espejo del objeto en los ejes en los cuales haya un uno.
- difference() Se emplea para eliminar partes de un objeto. Es decir, lo que hace esta función es eliminar el espacio que ocupa el segundo objeto indicado en la función. Gracias a esta función se generan elementos más complejos. Por ejemplo, es posible hacer una esfera hueca, siendo el primer elemento una esfera mayor y el segundo una esfera menor. O por ejemplo, se puede hacer huecos para poder introducir tornillos, eliminado cilindros en otros objetos.
- minkowski() Esta función es la más compleja que se utiliza. Se necesitan, al menos, dos objetos. Consiste en fijar el segundo elemento en cada punto del borde del primer objeto, es decir, que se hace una especie de integral a lo largo del borde. Esto es bastante útil si se requiere redondear los bordes de un cubo (o cualquier otro polígono), en este caso, el segundo elemento será una esfera, así en cada arista y vértice habrá una curvatura. [3]

[2] `https://www.OpenSCAD.org/cheatsheet/index.HTML`
[3] más información sobre el método minkowski `https://doc.cgal.org/latest/Minkowski_sum_3/index.HTML`

Por último, dado lo complejo que se puede volver el elemento, se pueden crear objetos propios para poder integrarlos y modificarlos uno a uno sin tener que modificar todo el código del objeto final. Esto se realiza con la instrucción module. Se puede asignar un nombre al módulo y entre llaves generar el objeto deseado. Posteriormente cada vez que se requiera de ese objeto, simplemente hay que llamarle para que sea utilizado. Prácticamente es el equivalente a una función o una subrutina en cualquier lenguaje de programación.

3.2 Objetivos de diseño y versiones

El diseño debe permitir que, en el mínimo espacio posible, pueda caber el sistema completo junto con las dos antenas (GPS y LoRa) y la batería.

En primer lugar, se dispone de un espacio central donde se aloja la placa. Hay que tener en cuenta que la PCB tiene elementos en ambas caras, y los elementos no es recomendable que estén apoyados directamente en el fondo. Como se explica en el Apartado 2.4, la PCB cuenta con cuatro agujeros cerca de los vértices, en previsión de poder colocar unos pivotes o unos tornillos, para elevar y sujetar la PCB. Es preciso hacer a su vez un agujero en la placa para el puerto USB, a través del cual se programa el sistema y se cargar la batería.

Además del espacio central, se aprovechan los bordes exteriores para colocar las antenas, en uno de ellos se debe colocar la antena GPS. Esta antena hay que procurar que este a cielo abierto, por tanto, se dedica todo una cara para ella, completamente separada de la PCB y se deberá colocar con la cara receptora hacia el exterior. La antena LoRa se sitúa en el borde más cercano al vacío de plano de masa que se dejó en la PCB de la cual va separada físicamente. Con esto se evita que se amortigüe la señal de algún modo.

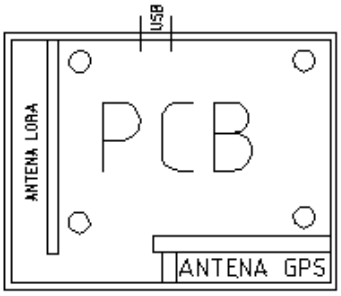


Figura 3.2: Idea original de la caja

Por último, en el plano superior se coloca la batería, que es el componente que delimita el tamaño de la caja. Esta apoyada sobre los propios muros que separan los huecos de la PCB y las antenas. Para evitar que se desplace, es necesario añadir un trozo de muro a mayores. Se puede ver la idea original en un esbozo en la Figura 3.2.

3.2.1 Versión 1

La primera versión de la caja se diseña con OpenSCAD, siendo el resultado es el de la Figura 3.3. Se añade rieles en dos caras paralelas de la caja para poder deslizar una tapa. Dicha tapa, Figura 3.4, tiene dos asas a través de las cuales se introduce una cinta elástica para poder sujetarla alrededor del cuello del animal. Y, además, se incluyen cuatro agujeros en las esquinas para colocar unos tornillos y que no deslice la tapa. Por último, para robustecer a los pivotes que sujetan la PCB se les añade un segmento a cada uno.

Esta versión tiene ciertos fallos. El principal fallo es que los pivotes no encajan con agujeros realizados en PCB. Además, la caja tarda bastante en fabricarse, por tanto, se recomienda estrechar muros o eliminarlos.

3.2.2 Versión 2

En la segunda versión de la caja (véase Apéndice C) se intenta corregir la posición de los pivotes (sin éxito) y se hacen huecos los muros más anchos. Además, se introduce una muesca para intentar fijar la PCB a los pivotes. El resultado con el sistema electrónico montado se puede ver en la Figura 3.5. Se ve como van colocados los componentes, y como adicionalmente se deja un pequeño hueco para la batería que se usa en la fase de diseño. La tapa es la misma para ambas versiones dado que no se la detecta fallo alguno.

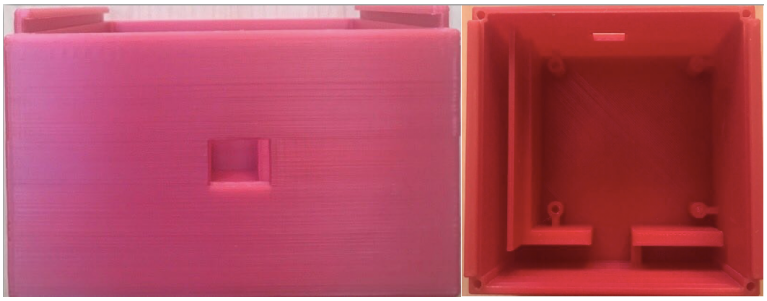


Figura 3.3: Versión 1 de la caja

Figura 3.4: Tapa de la caja

3.2.3 Versión 3

La versión tres no se diseña ni se fabrica. Pero si detallan ciertos aspectos a corregir.

- Hay que reducir el ancho de los bordes, y de los muros que separan zonas. Pudiendo, así, reducir ligeramente el tamaño de la caja.
- Hay que lograr colocar los pivotes correctamente. Estos pivotes estaban originalmente para que se sujetase a ellos la PCB y no se moviese en el eje horizontal, pero en el vertical se puede salir. Por tanto, se propone ahuecar los pivotes para poder atornillar la PCB a los mismos.
- Hay que buscar algún sistema para poder cerrar el agujero para el puerto USB y minimizar, así, los huecos por los cuales pueda entrar agua u otros elementos no deseados, dado que la caja estará a la intemperie.

3.3 Impresora y procesado previo

Por último, antes de poder fabricar el producto final, se necesita convertir el archivo .stl en un tipo de archivo que sepa interpretar la impresora 3D. El tipo de archivo que se necesita es un archivo .gcode o RS-274. Por ello, se utiliza el programa Ultimaker Cura, descargable en `https://ultimaker.com/en/products/ultimaker-cura-software`.

Este programa es bastante versátil, dado que tiene una pequeña base de datos con bastantes modelos de impresora 3D, donde tiene almacenado el espacio imprimible y las disponibilidades generales que tiene de cada modelo. Además, permite configurar parámetros de impresión tales como el espesor del plástico usado y la utilización de soportes para elementos que no estén directamente colocados sobre el suelo o una pared (los soportes son trozos de plástico de baja densidad debajo del elemento flotante, que posteriormente, se puede retirar con cierta facilidad). Se ve en la Figura 3.6 como se pueden colocar los elementos en la bandeja de la impresora y como calcula los metros de plástico y el tiempo que tarda en fabricar el objeto.

Mencionar que la impresora utilizada es la Anycubic i3 Mega (véase Figura 3.7), puesto que es la que se disponía en la Universidad para su utilización.

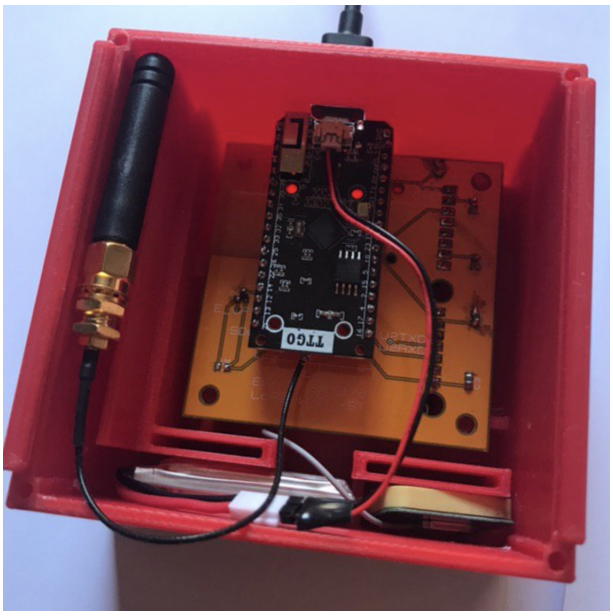


Figura 3.5: Versión 2 de la caja con el sistema montado

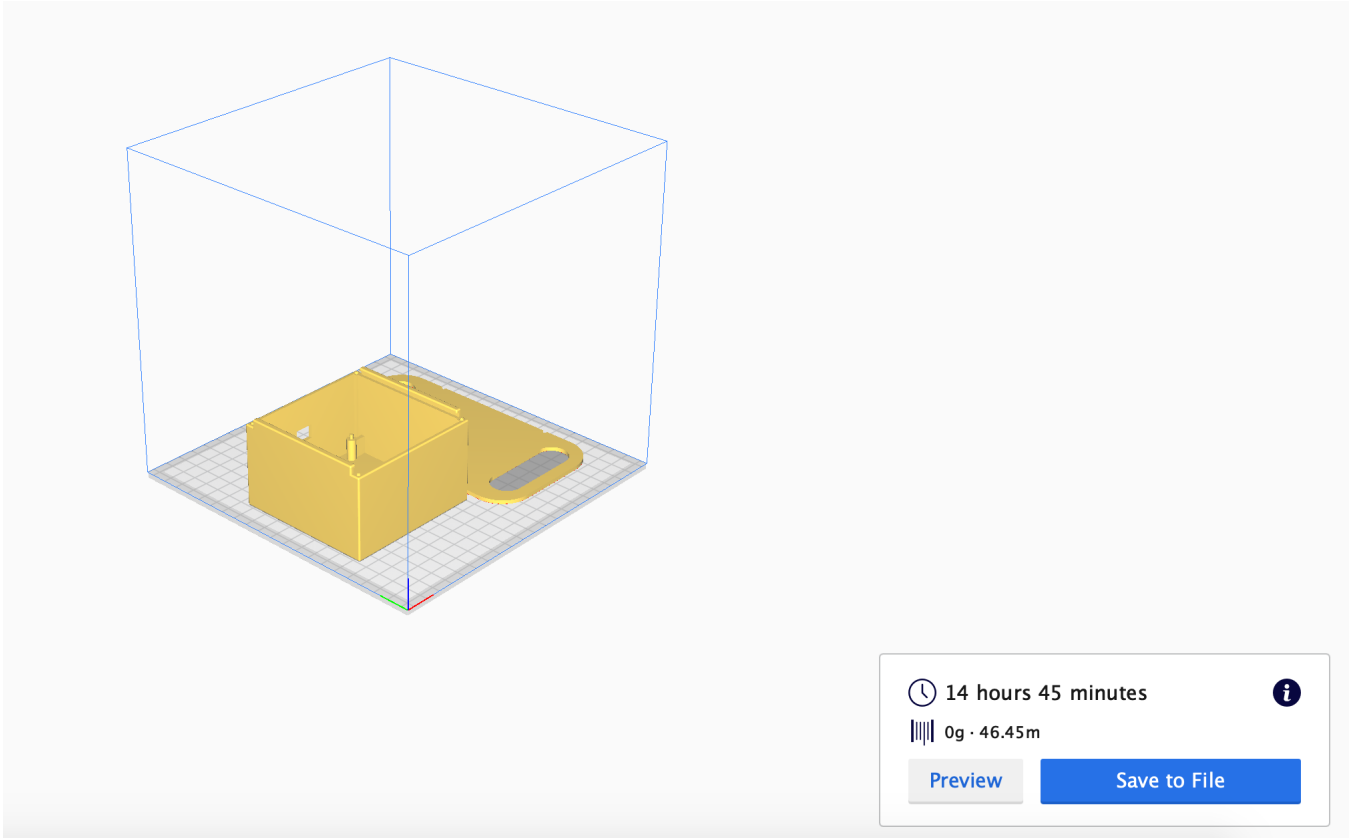


Figura 3.6: Previsualización en Ultimaker Cura del objeto

Figura 3.7: Impresora 3D Anycubic i3 Mega

Capítulo 4

Desarrollo Firmware

Se usa el IDE de Arduino por que dispone de asistencia a programación a placas no propias. Es decir se puede crear código a partir de este lenguaje de programación (que es prácticamente C++) y programarlo en las placas que lo permitan (como los ESP). La metodología de Arduino es bastante simple, se basa en el principio de que la mayoría de las cosas ya se han probado. Por tanto, se buscan bibliotecas que contengan ejemplos que se puedan adecuar a la problemática, y este código se adapta para la utilidad del proyecto. Al final el objetivo es no tener que crear una biblioteca propia teniendo que bajar a bajo nivel, sino aprovechar lo que ya funciona de otras bibliotecas y conseguir adaptarlo para solucionar el problema del proyecto.

4.1 Bibliotecas

Lo primero que se ha de hacer es instalar la tarjeta que se utiliza. Para ello se debe ir a las preferencias de Arduino e introducir en el apartado de gestión de URLs de tarjetas adicionales el siguiente url: `https://dl.espressif.com/dl/package_esp32_index.json`. Una vez aplicado los cambios, se habrán fijado los parámetros para poder trabajar con esta placa y, además, se habrán instalado una serie de bibliotecas asociadas a este hardware que simplifican la búsqueda de soluciones en internet.

Concretamente se utilizan las siguientes bibliotecas:

- stdlib.h: Es la biblioteca estándar de C. Contiene funciones básicas.
- time.h: Biblioteca estándar de C para el uso del tiempo.
- stdio.h: Biblioteca estándar en C para funciones de entrada y salida de datos.
- heltec.h[1]: Sirve para controlar los aspectos básicos de un módulo Heltec como el que se dispone. Tiene funciones para controlar la pantalla OLED, el protocolo LoRa y el WiFi.
- MPU6050.h[2]: Dispone de las funciones necesarias para controlar el sensor MPU6050 y adquirir sus datos. Esta biblioteca requiere de Wire.h, biblioteca que viene por defecto en el IDE de Arduino ya que sirve para manejar I2C, un protocolo básico en las comunicaciones.
- Adafruit_GPS.h[3]: Sirve para controlar el dispositivo GPS de una manera más intuitiva, proporciona, además, decodificación para el protocolo NMEA lo cual simplifica mucho el problema. Necesita de HardwareSerial.h, que sirve para usar los distintos canales de UART que se disponen, viene ya instalada en las configuraciones de la placa.
- SPIFFS.h: Es la biblioteca que gestiona las funciones para usar el sistema de archivos en la Flash. Requiere de FS.h para funcionar. Ambas vienen en el pack de la placa a la hora de instalarse.
- WiFi.h: Biblioteca para usar el WiFi del dispositivo. Se complementa con WiFiAP.h para poder poner el sistema como punto de acceso. Se encuentran dentro de las bibliotecas hardware al instalar la placa.

[1] Disponible en `https://platformio.org/lib/show/6051/Heltec%20ESP32%20Dev-Boards/examples`
[2] Disponible en `https://github.com/jarzebski/Arduino-MPU6050`
[3] `Disponibleenhttps://github.com/adafruit/Adafruit_GPS`

- Adafruit_MQTT.h y Adafruit_MQTT_client.h[4]: Bibliotecas que proporcionan las funciones necesarias para usar Message Queuing Telemetry Transport (MQTT) con la página de io.Adafruit.

4.2 Software del sistema que lleva la vaca

El programa que se ejecuta en el sistema electrónico que lleva la vaca tiene dos objetivos principales: recoger datos de los sensores y enviarlos por LoRa cuando se le requieran. Para ello se necesita, como punto importante, generar un temporizador para poder escuchar LoRa mientras se espera al muestreo de los sensores. Es decir, no depender de la instrucción delay(), ya que esta detiene completamente el programa y no realiza ninguna instrucción más. Y, además, se requiere de almacenar los datos en la memoria Flash para no depender de búferes demasiado grandes, que encima se puedan perder si se resetea el microcontrolador o se acaba la batería.

4.2.1 Temporizador

En los ESP32 cuando se inicia el sistema se activa un contador de milisegundos que esta disponible para consultarse en la función millis(). Se usar esa función para temporizar. La manera de uso es simple, se crea una sentencia condicional if, cuya condición se cumple si el resto entre millis() y el período es cero. De esta manera se puede variar el parámetro del período para hacer que se entre con más o menos frecuencia a esa función.

En general se crean todos los temporizadores de ambos sistemas de esta manera. Pero hay subrutinas que interesa que estén supeditadas a una rutina más prioritaria, y que si no se realiza esta, no se realicen. Además, tener muchas condiciones de esta manera con millis(), tampoco es la mejor idea. Esto se debe a que las subrutinas tardan cierto tiempo en realizarse y puede hacer sobrepasar el período de otra subrutina, haciendo que esta tenga que esperar otro período entero a realizarse. Por tanto, en este código se supedita todo al número de lecturas del MPU6050, dado que es el sensor que más se utiliza y que menos tarda en realizar su subrutina asociada. Se tiene un contador de lecturas del MPU6050, y, se sabe, dado el diseño realizado, que va a operar a 5 Hz, por tanto, cada cinco lecturas es un segundo. Las tareas que se supeditan a esto es el guardado en memoria Flash y la lectura del GPS. El tiempo que no se realiza nada, el ESP32 esta escuchando LoRa.

4.2.2 MPU6050

Como ya se ha comentado, para este sensor se cuenta con la biblioteca MPU6050.h asistida por Wire.h. En esta biblioteca se encuentran las funciones básicas para usar el sensor. Desde su inicialización, configuración, lectura de datos, etc. Mencionar que en esta biblioteca se hacen dos pequeñas modificaciones. Primero se añade un cuarto valor a la estructura Vector, añadiendo el módulo del valor de los tres ejes. Este módulo no se calcula automáticamente, hay que programarlo en el código, pero proporciona una estructura más completa para manejar todos los valores de una manera más conjunta. Se añade una variable tipo float con el nombre XYZmod. La segunda modificación es añadir una función en MPU6050.cpp para activar los modos de bajo consumo del acelerómetro. Esta tiene dos variables de entrada, una para fijar la frecuencia del acelerómetro y otra por si se quiere tener encendida la temperatura. Básicamente se acude a bajo nivel a escribir directamente en los registros los valores deseados, creándose una variable de dos bytes y enviándose por I2C. Vemos como se apagan por defecto los ejes del giróscopo y que, además, esta función se pudiera completar dando la opción a dormir el chip completamente.

```
1  void MPU6050::setCycleMode(uint8_t frec, bool tempEn){
       uint16_t value = 0b0010000000000111;
       // 6B PWR_MGMT_1
       // 0   Device Reset
       // 0   Sleep
6      // 1   Cycle (Para encender muestrear apagar)
       // 0   Sin determinar
       // ?   Temp Disabl
```

[4]Disponibles en `https://github.com/adafruit/Adafruit_MQTT_Library`

```
    // 000 ClockSel[2:0]
    // 6C PWR_MGMT_2
    // ?? Frecuencia 40(11)-20(10)-5(01)-1,25(00)
    // 0  STBY_XA
    // 0  STBY_YA
    // 0  STBY_ZA
    // 1  STBY_XG
    // 1  STBY_YG
    // 1  STBY_ZG

    if (tempEn){
        value |= (1 << 11);
    } else {
        value &= ~(1 << 11);
    }
    value |= ((frec & 0b11)<< 6);
        writeRegister16(MPU6050_REG_PWR_MGMT_1, value);
}
```

Además, para el correcto funcionamiento de la función hay que guardar las siguientes definiciones de registros en MPU6050.h. Los tres primeros son los registros de control de potencia del MPU6050. Las cuatro siguientes declaraciones marcan lo que se ha de escribir en los registros para poder configurar cada uno de los modos de bajo consumo. Se aprovecha, a su vez para definir tres funciones que normalizan el valor de los registros a unidades de aceleración (g).

```
#define MPU6050_REG_USER_CTRL   (0x6A) // User Control
#define MPU6050_REG_PWR_MGMT_1  (0x6B) // Power Management 1
#define MPU6050_REG_PWR_MGMT_2  (0x6C) // Power Management 2

#define MPU6050_Cycle_40        (0b11)
#define MPU6050_Cycle_20        (0b10)
#define MPU6050_Cycle_5         (0b01)
#define MPU6050_Cycle_1_25      (0b00)

#define NormX                   (Acel.XAxis * (9.81/16384.0))
#define NormY                   (Acel.YAxis * (9.81/16384.0))
#define NormZ                   (Acel.ZAxis * (9.81/16384.0))
```

Finalmente se define la función dentro de la clase MPU6050, configurado para que si se la llama sin aportar variables, se configure con la temperatura apagada y una frecuencia de muestreo de 40 Hz.

```
void setCycleMode(uint8_t frec = MPU6050_Cycle_40, bool tempEn = 0);
```

Para el uso del MPU6050 se recomienda calibrarlo de alguna manera antes de su uso. Para ello se acude a internet, y una función[5] que calibra el sensor, se modifica, resultando la función void calibrar(). En ella se intenta fijar los valores de offset del sensor lo más cerca del 0 posible en el caso de x e y, y lo más cerca de 9.81 m/ s_2 (1 g en reposo el sensor). Independientemente de si se logra en mayor medida, se hace inmediatamente después una lectura de los valores en reposo, y se fijan en el vector de zeros. Este vector será la referencia. Es decir, a los valores muestreados se les comparara con este vector para conseguir un valor real de la medida. Esto se puede ver en la función void readMPUData(), que sirve adquirir datos y a su vez genera la norma matemática del vector. Por último, se crea una función de inicialización del MPU6050 (void initMPU6050()) donde se comprueba si esta conectado el sensor, se calibra y se configura el modo de bajo consumo.

Tratamiento de datos y uso de la Flash

Para el uso de la memoria Flash se usan las siguientes funciones. Todas ellas sacadas de los ejemplos que venían al instalar la biblioteca SPIFFS.h.

[5]Función original en `https://www.diarioelectronicohoy.com/blog/configurar-el-mpu6050`

- void listDir(fs::FS & fs, const char * dirname, uint8_t levels): Muestra el directorio pedido, con el nivel de profundidad indicado.
- void readFile(fs::FS & fs, const char * path): Lee el archivo indicado. Además, se le añade a la función primitiva, que no sólo muestre la información por el serial, sino que también lo almacene en un búfer para su posterior utilización.
- void writeFile(fs::FS & fs, const char * path, const char * message): Crea y escribe el archivo indicado. Mencionar que pisa el anterior archivo si ya existía uno con ese nombre.
- void appendFile(fs::FS & fs, const char * path,/* const*/ char* message): La cadena de texto es añadida al archivo deseado. Mencionar que se elimina la característica const para evitar problema de incompatibilidades entre funciones.
- void renameFile(fs::FS & fs, const char * path1, const char * path2): Renombra un archivo.
- void deleteFile(fs::FS & fs, const charr * path): Elimina el archivo.

Hay que denotar que los accesos a memoria Flash, tanto de escritura cómo de lectura, suelen ser lentos. Por tanto no es buena idea guardar el dato cada vez que se obtiene. Para esto se plantean varias soluciones, crear un vector para almacenar previamente varios valores y luego dedicar unos instantes a guardar todos esos datos en un solo acceso a memoria. Y tratar los datos para que el volumen de datos que se han de transmitir sea menor. Se opta por ambas soluciones, primero se tratan los datos, después se van guardando en un búfer previo a su memorización en Flash.

Para hacer el tratamiento de datos se plantea convertir los datos arrojados por el MPU6050 en pasos, es decir, convertirlo en un podómetro. Mencionar que la rutina que se describe y se aplica es propia y no tiene ninguna base científica más que la simple inspección de lo que arrojan las curvas de movimiento. De todas formas el hecho de que no cuantifique exactamente los pasos de la vaca no es importante. Esto se debe a que lo que pretende el sistema es ver si en algún momento hay un período de mayor movimiento. Por tanto, la detección de eso será en relación a datos propios de la misma vaca, es decir, que tanto si cuenta la mitad como el doble de pasos, al final el momento de nerviosismo de la vaca se nota cuadruplicando (según Charles A. Kiddy) ese movimiento.

Lo primero que se hace para crear esta rutina de procesado es generar unas curvas de movimiento. Para ello se programa el sensor para que recoja los datos del eje Z, que se estima que será el de mayor variación en el andar, y que también se calcule en todo momento el módulo del movimiento. La gráfica se puede observar en la Figura 4.1. Esta gráfica se genera recogiendo los datos que se almacenan en la memoria Flash mientras se da un ligero paseo. Posteriormente se introducen en una hoja de cálculo y se genera las dos gráficas.

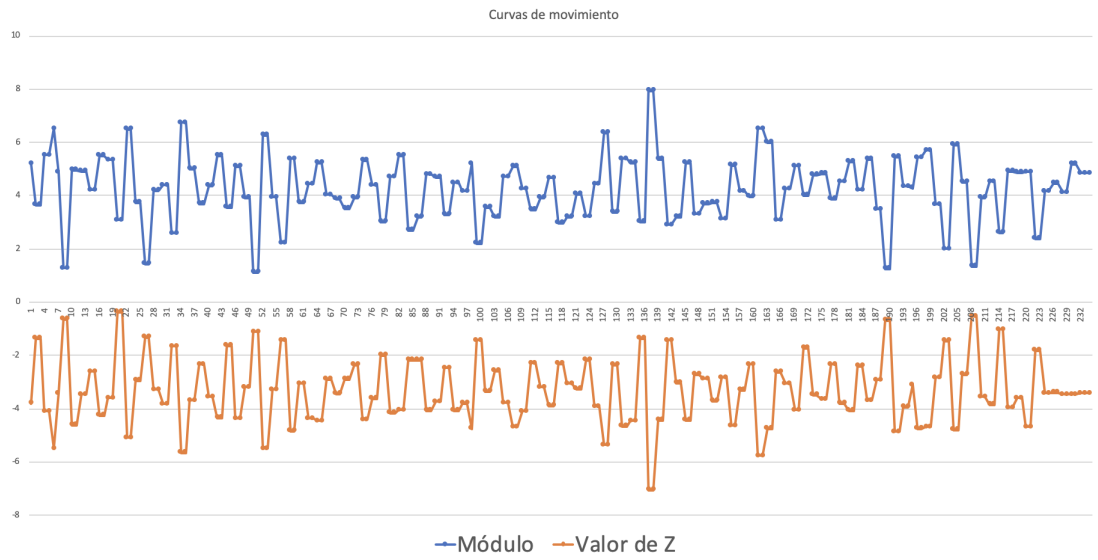


Figura 4.1: Gráfica de movimiento

Se ve que ambas gráficas son similares, finalmente se decide usar la del módulo, dado que en un sistema ideal el eje Z esta completamente orientado verticalmente, pero es posible que no se coloque correctamente en el animal, y sean los otros ejes los que denoten el movimiento. Este movimiento se detecta con el módulo del vector, sea cual sea el eje que se coloque en la vertical. Viendo la Figura 4.1 se llega a la conclusión que los pasos se dan cuando hay un máximo, y ese máximo supera en cierto umbral al anterior mínimo. Por tanto, generamos a través de la herramienta de pseudocódigo Pseint[6] el algoritmo. Se ve el diagrama de bloques del algoritmo en la Figura 4.2. Este lo que realiza es, básicamente, un primer guardado de los valores del mínimo y del máximo, si la función sube, se actualiza el máximo y si la función deja de subir se comprueba si entre el mínimo y el máximo guardado hay un umbral necesario para contar pasos.

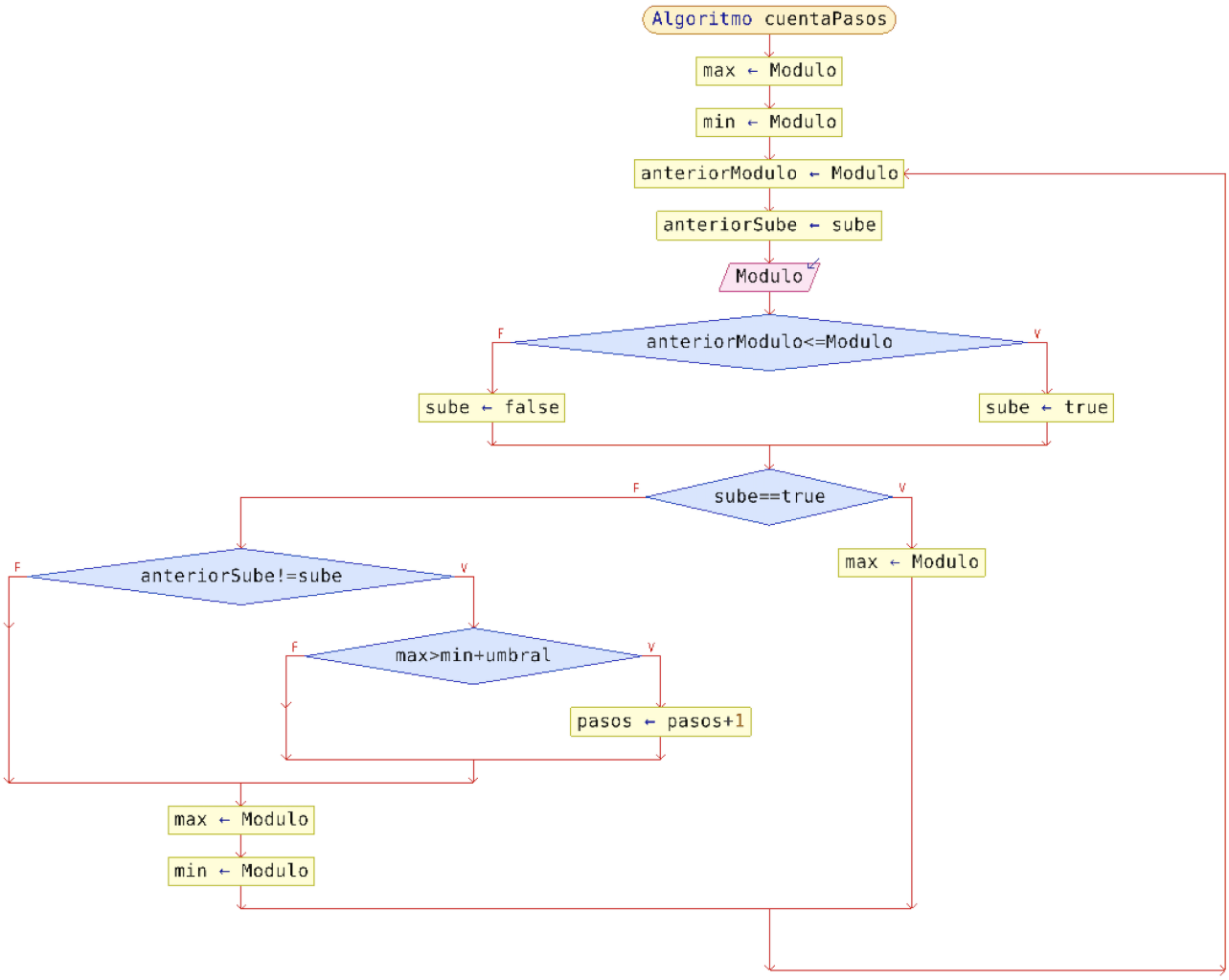


Figura 4.2: Diagrama de bloques del algoritmo contar pasos

Este algoritmo se aplica a la hoja de cálculo donde estaban almacenados los datos y se ve en la Figura 4.3 como efectivamente, el incremento de pasos coincide con el máximo del módulo.

Con esto se consigue manejar un menor volumen de datos. Concretamente en el ejemplo que se acaba de ver, se usan 107 valores del módulo para generar un único valor que contiene 17 pasos. Aún así parece innecesario que cada vez que se genere un paso se almacene en la memoria Flash, por tanto, se crea una unidad de movimiento en un determinado tiempo. Es decir se cuentan los pasos cada cierto tiempo y se almacenan en la memoria Flash. Se decide usar la unidad pasos cada cinco minutos, dado que es una buena unidad para ir muestreando y comparando con valores adyacentes. Recordando que se supedita la escritura en Flash a el número de lecturas del MPU6050, teniendo en cuenta que se hacen cinco lecturas por segundo (5 Hz),

[6]Disponible en `http://pseint.sourceforge.net/slide/pseint.HTML`

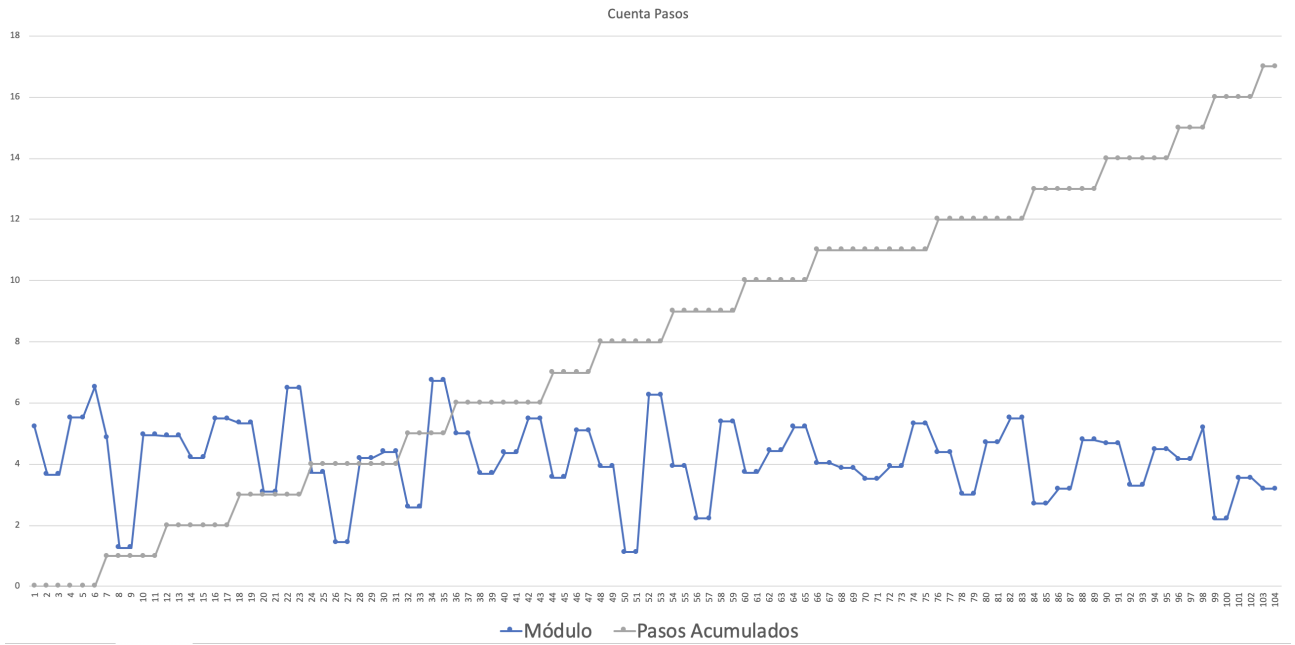


Figura 4.3: Algoritmo de contar aplicado en hoja de cálculo

se guarda en flash cada 1500 lecturas. La conclusión es evidente, se ha pasado de escribir 1500 veces en la memoria Flash a sólo una, ahorrando el tiempo que conlleva y utilizándole para el uso común del sistema. Además de usar el espacio de dicha memoria de manera más eficiente.

4.2.3 GPS

Para el sistema GPS se utilizan las librerías Adafruit_GPS.h y HardwareSerial.h. Realmente sólo se necesitaría la segunda, pero para captar los datos se necesita demodular los datos que ofrece el sistema en codificación NMEA (explicada en el Apartado 2.2.2). La primera biblioteca simplifica bastante esa labor, decodificando e identificando las variables interesantes. Realmente en el código hay poco desarrollado sobre este sensor, esto es debido a la problemática explicada en el Apartado 7.2.2. Básicamente, no se logra conseguir los datos de posición en ningún momento con este dispositivo.

Lo que sí se programa es el control de alimentación a través del pin GPIO25 (Después de la corrección que se describe en el Apartado 7.1.1). Este pin estando en alto (3,3V) mantiene abierto el interruptor MosFet, y cuando se cumple el período de muestreo se activa el GPS bajando el pin GPIO25 a 0 V para cerrar el interruptor y que alimente el sistema. Se mantiene encendido durante un minuto, y cuando se cumple ese minuto se leen los datos arrojados por el REB-5216. Estos datos se guardan en Flash. Si no se encuentran los datos de posición también se notifica en el archivo el fallo, y si tres veces consecutivas falla el GPS ya no se vuelve a encender para no consumir potencia sin rendimiento alguno.

4.3 Software del sistema que lleva el ganadero

El principal objetivo del sistema del ganadero es servir de enlace entre los sistemas de las vacas y las interfaces y aplicaciones móviles. Para ello se aprovechan los protocolos de conexión inalámbrica que con los que cuenta el microcontrolador, tanto como para pedir los datos sensados a los chips de las vacas, como para subirlos a la nube para su posterior visualización. Y todo esto ha de hacerse lo más transparente posible al usuario. La idea general se plasma en la Figura 4.4 donde se puede ver que primero se pone el sistema ESP32 como punto de acceso. A través de este punto de acceso se lanzará una sencilla aplicación móvil para pedir los datos de la red WiFi propia y así poder conectarse a internet. Una vez conectado a internet se usa el protocolo MQTT para subir y actualizar la información obtenida.

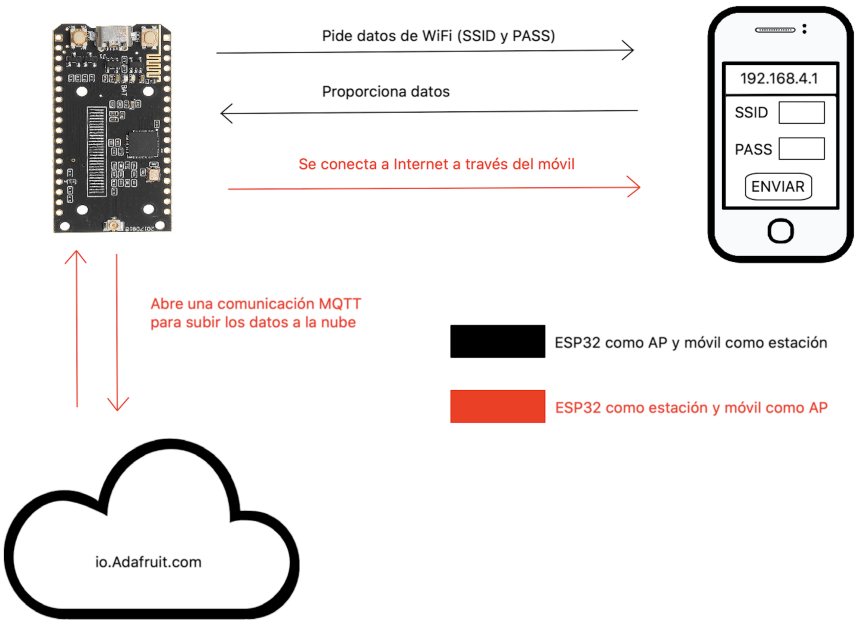


Figura 4.4: Idea comunicaciones con el ESP32 del ganadero

4.3.1 WiFi

Los microcontroladores ESP32 tienen dos protocolos básicos de conexión inalámbrica, que son WiFi y Bluetooth. Se decide usar WiFi, ya no sólo por las ventajas que nos ofrece, sino por que está mucho más desarrollado y explotado por los usuarios, y es por tanto más probable encontrar en la red ejemplos que podamos adaptar a nuestro proyecto. Para ello se necesita la biblioteca WiFi.h.

Como se ve en la idea general, el primer objetivo es poner el ESP32 como punto de acceso de WiFi. Este código se adapta del ejemplo encontrado en `https://bit.ly/2UQw1LE`. Se realiza es la aplicación HTML que se puede ver en el Apéndice E, esta es un simple cuestionario. El código Arduino funciona de tal manera que una vez que se da al botón se devuelve una cadena de texto con toda la información de la página web. Simplemente hay que detectar los parámetros posteriores a inSSID e inPASS y confirmar que ambos se han introducido, esto se realiza con las funciones indexOf() y substring() del objeto String.

Posteriormente hay que conectarse a una red WiFi, para ello en los ejemplos de la misma biblioteca WiFi.h se encuentra la manera de conseguirlo. Se modifica de tal manera que, en vez de usar una SSID y una contraseña fijada en el código, ambos datos son los adquiridos por la página web. Mencionar para que el ESP32 pase de punto de acceso a estación hay que cambiar el modo de funcionamiento e incluso resetear el WiFi del dispositivo.

Se aprovecha que se conecta a internet para actualizar los datos de fecha y hora que se utilizan posteriormente.

4.3.2 MQTT

Para subir los datos a la nube se utiliza el protocolo MQTT. MQTT es un protocolo que se inventa en 1999 por IBM y Arcom (Andy Stanford-Clark y Arlen Nipper). Se basa en la suscripción y publicación. Pretende ser para redes de bajo ancho de banda pero pudiéndose aprovechar de una alta cadencia de datos. El puerto que usa MQTT asignado por la Internet Assigned Numbers Authority (IANA) es el 1833 (es el que se usa en el proyecto). Opcionalmente se puede usar el puerto 8883 para usar MQTT sobre Secure Sockets Layer (SSL). El problema de esto, es que SSL añade gran cantidad de datos de cabecera y agranda el ancho de banda necesario para usar el protocolo.

Existen varios elementos claves para el uso de MQTT. Se necesita un servidor y uno o más clientes. El servidor acepta conexiones y publicaciones de los clientes, y controla las suscripciones. Este básicamente actúa como intermediario entre clientes que publican y clientes que se suscriben a esas publicaciones. Los clientes son los dispositivos usuarios de MQTT, estos han de conectarse al servidor para realizar publicaciones, suscribirse

o desuscribirse a publicaciones y consultar esas suscripciones. Existen los tópicos, estos son los que reciben esa información cuando se publica y son consultados cuando se solicita una suscripción. [7]

Por tanto, podemos usar MQTT para comunicar varios clientes, generando varios tópicos. Esta información es bidireccional, es decir, que se puede controlar parámetros dentro del sistema a través de este protocolo. Se decide usar el servidor de Adafruit y sus librerías (Adafruit_MQTT.h y Adafruit_MQTT_client.h) debido a la sencillez de su uso. La estrategia de uso de este método se describe en el Apartado 5.2. En resumen, tendremos tres tópicos, dos publicaciones, pasos y feedbacks; y una subscripción, pedirLoRa. A través de pasos se publican los pasos recogidos, a través de feedbacks se publican realimentaciones varias y a través de pedirLoRa se puede solicitar desde otro cliente la comunicación LoRa con los dispositivos de las vacas.

4.4 Comunicación LoRa

4.4.1 Introducción a LoRa

LoRa es un protocolo de radio que se basa en el espectro extendido, concretamente en la modulación chirp. Es decir que utiliza más ancho de banda del que se requiere en un origen la señal. Esto se hace así para obtener cierta robustez a interferencias, operar a baja densidad espectral, etc. LoRa en cuestión es el protocolo de radio a nivel de capa física a nivel de de red es LoRaWAN, este protocolo lo definen en la página de LoRa-Alliance[8]. Esta red es usada para adaptar las IoT a dispositivos que no cuenten con una conexión fija a internet. Esto proporciona una gran capacidad de reducción de costes, al sólo tener que pagar una linea de datos en vez de muchas. Esta se dispone en arquitectura estrella, de tal manera, que el chip conectado a la red recoge los datos de los no conectados, creando así múltiples chips que se pueden considerar pseudo- IoT.

Las principales características de LoRa a nivel físico es que dispone de una alta sensibilidad, gran radio de comunicación, entorno a 10-15km, baja densidad de datos (255bytes) y disponibilidad de usar tres bandas de frecuencia: 915, 868 y 433 MHz. Además de la robustez a interferencias.

Prácticamente la modulación LoRa se basa en ir modulando la señal en una frecuencia que varía constantemente. En cuestión LoRa lo que hace es hacer un barrido de frecuencias lineal, desde el más bajo hasta el más alto, y una vez en el más alto, vuelve al más bajo. Además LoRa dispone de dos parámetros regulables a mayores, el ancho de banda de la señal (de 7.8KHz a 500 KHz) y el factor de de ensanchamiento (Spreading Factor (SF)). Este último indica la cantidad de bits que tienen los símbolos. Mencionar que al hacer el barrido de frecuencias, cuando LoRa termina de enviar un símbolo no empieza de nuevo con la frecuencia mas baja, sino que empieza en una frecuencia fijada para cada símbolo. Es decir que si tiene un SF de 2, son símbolos de 2 bits y existen 4 posibles saltos de frecuencia.

Como se ve en el Apartado 4.4.2, el aumento del ancho de banda sube la tasa de datos pero también el SF influye, siendo cuanto mayor, más baja la velocidad de transmisión.

Además, hay que estudiar es la disponibilidad de la bandas de frecuencias en España. Para ello se acude a la página del gobierno[9] donde se explican y se asignan todas las bandas de frecuencias. Y se concluye los siguiente:

La banda de 433 MHz se puede utilizar dado que esta enmarcada dentro de las bandas de radioaficionados. Ver Figura 4.5.

La banda de 868 MHz tiene cierta limitaciones pero se enmarca dentro de la radiodifusión, por tanto, también puede usarse. Ver Figura 4.6.

Por último la banda de 915 MHz no se puede usar en Europa. Ver Figura 4.7.

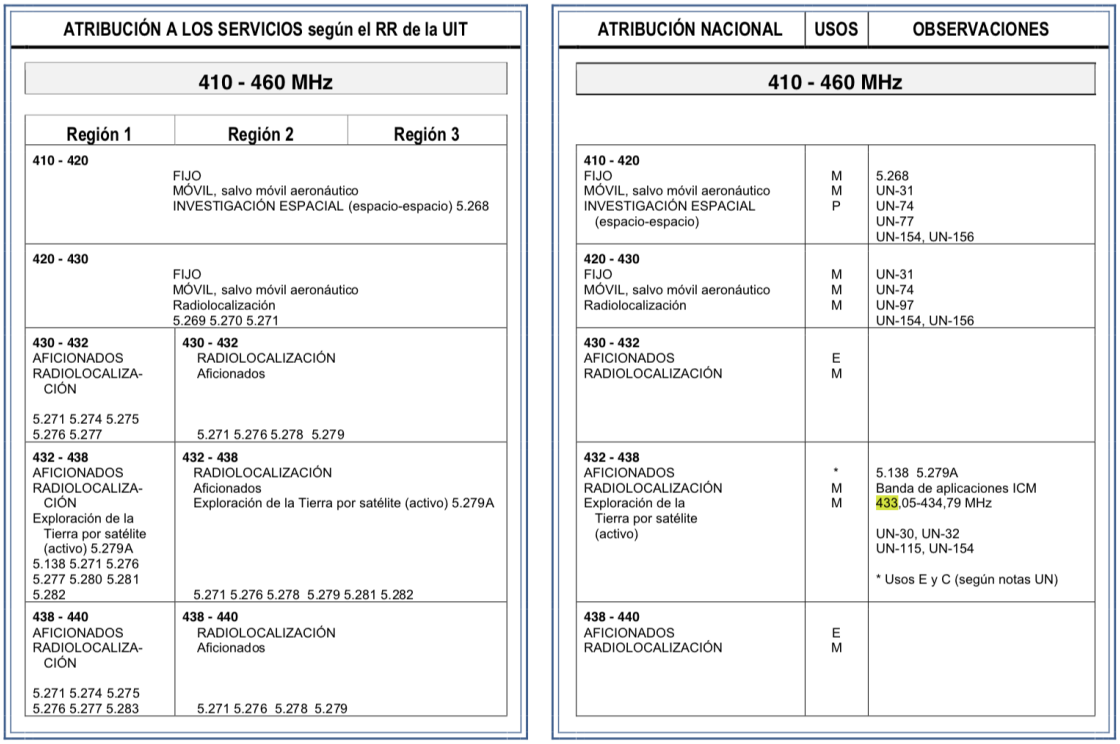
Acudiendo a la documentación de LoRaWan se recomienda usar 433 MHz en China, 868 MHz en Europa y 915 MHZ en América.

4.4.2 Familia SX127x

Se decide acudir a la hoja de datos[10] de la familia SX127x para analizar las características básicas que podemos implementar. Factor de ensanchamiento de espectro de 6 a 12. Hasta 14 dBm en transmisión. Cabecera de 12 bytes. Carga útil de 64 bytes. Velocidades de transmisión que se detallan en la Figura 4.8.

Realmente no se especifica el radio de acción de este dispositivo, pero conforme a lo que dice la LoRa-Alliance debiera cumplirlo minimamente. Se ve en el Apartado 6.2 como esto no se cumple en ningún caso.

[7] Más información en `http://mqtt.org`
[8] `https://lora-alliance.org`
[9] `https://avancedigital.gob.es/espectro/Paginas/cnaf.aspx`
[10] Disponible en `https://bit.ly/2WiELKM`



ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT

410 - 460 MHz

Región 1	Región 2	Región 3
410 - 420	FIJO MÓVIL, salvo móvil aeronáutico INVESTIGACIÓN ESPACIAL (espacio-espacio) 5.268	
420 - 430	FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización 5.269 5.270 5.271	
430 - 432 AFICIONADOS RADIOLOCALIZA-CIÓN 5.271 5.274 5.275 5.276 5.277	**430 - 432** RADIOLOCALIZACIÓN Aficionados 5.271 5.276 5.278 5.279	
432 - 438 AFICIONADOS RADIOLOCALIZA-CIÓN Exploración de la Tierra por satélite (activo) 5.279A 5.138 5.271 5.276 5.277 5.280 5.281 5.282	**432 - 438** RADIOLOCALIZACIÓN Aficionados Exploración de la Tierra por satélite (activo) 5.279A 5.271 5.276 5.278 5.279 5.281 5.282	
438 - 440 AFICIONADOS RADIOLOCALIZA-CIÓN 5.271 5.274 5.275 5.276 5.277 5.283	**438 - 440** RADIOLOCALIZACIÓN Aficionados 5.271 5.276 5.278 5.279	

ATRIBUCIÓN NACIONAL | **USOS** | **OBSERVACIONES**

410 - 460 MHz

410 - 420 FIJO MÓVIL, salvo móvil aeronáutico INVESTIGACIÓN ESPACIAL (espacio-espacio)	M M P	5.268 UN-31 UN-74 UN-77 UN-154, UN-156
420 - 430 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización	M M M	UN-31 UN-74 UN-97 UN-154, UN-156
430 - 432 AFICIONADOS RADIOLOCALIZACIÓN	E M	
432 - 438 AFICIONADOS RADIOLOCALIZACIÓN Exploración de la Tierra por satélite (activo)	* M M	5.138 5.279A Banda de aplicaciones ICM 433,05-434,79 MHz UN-30, UN-32 UN-115, UN-154 * Usos E y C (según notas UN)
438 - 440 AFICIONADOS RADIOLOCALIZACIÓN	E M	

Figura 4.5: Legislación para la banda de 433 MHz

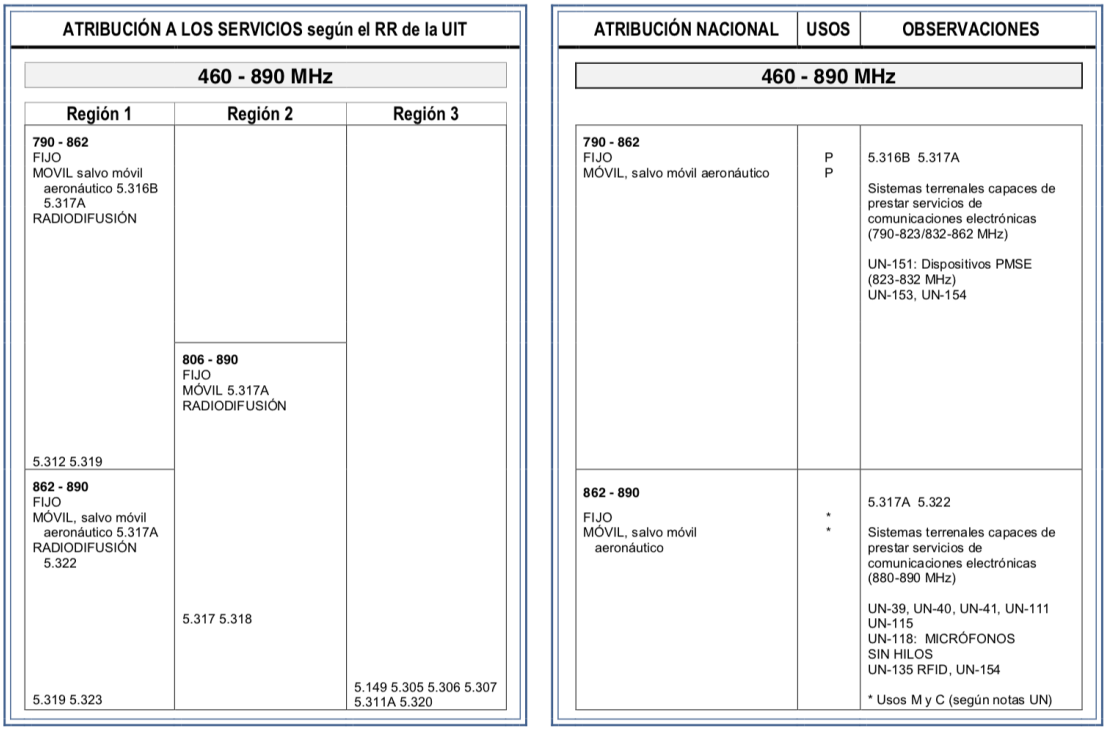
4.4.3 Aplicación y Modificación de LoRa

Lo primero que hay que mencionar, es que se modifica el protocolo LoRa en su capa física para adaptarlo al proyecto. Esto se debe a que cada ESP32 tiene una dirección única de seis bytes y no se encuentra la manera de conseguir encapsularlo en un byte sin que vayan a coincidir direcciones. Es decir que las direcciones LoRa que se usan, son de seis bytes. Esto incrementa bastante la cabecera. Se pasa de tener dos bytes de direcciones a doce. Esto implica severas modificaciones de las funciones de emisión y recepción proporcionadas en las bibliotecas de heltec.h.

Además, se decide usar el identificador del mensaje como un indicador de la fragmentación del paquete. Es decir que, si el paquete se ha fragmentado, el identificador indica cuantos mensajes quedan para concluir el paquete y poder encapsularlo completamente.

Una vez estipulada estas variaciones en la cabecera y adaptadas las funciones básicas que envían (void sendMessage(String Outgoing, String destination)()) y reciben mensajes LoRa (void onReceive(int packetSize)()). Creándose incluso una para fragmentar el paquete (void sendLora(String paquete, String destination)()). Se decide implementar una pequeña versión de la red. Lo que se define es un protocolo incompleto y que sólo recoge los datos de los pasos del chip del animal.

En primer lugar el ESP32 maestro envía un mensaje a todos los receptores pidiendo datos y este se queda esperando durante un cierto instante de tiempo (regulable a través de variable). Si durante ese instante no recibe nada, repite la petición periodicamente hasta que reciba algo. El ESP32 recibe esa petición, la identifica y si cumple los parámetros correctos de pedida de datos, prepara los datos y procede a enviarlos. Finalmente, el ESP32 del ganadero recibe los datos y los prepara para subirlos a la nube para su posterior visualización.



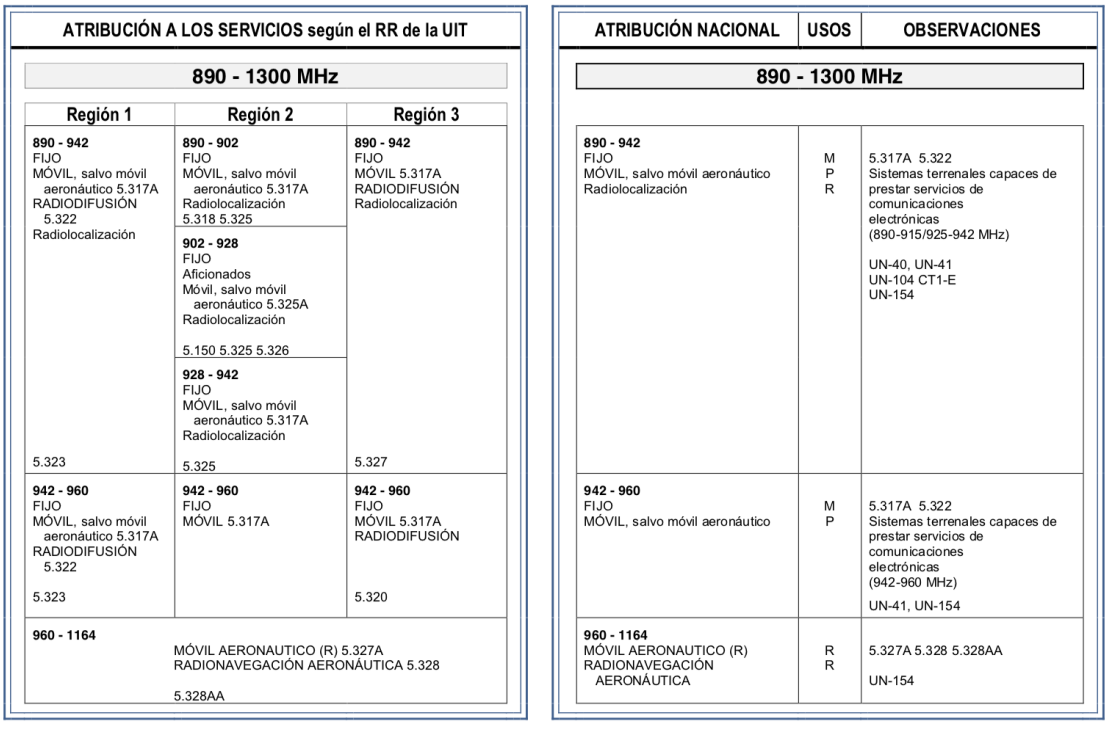
ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT

460 - 890 MHz

Región 1	Región 2	Región 3
790 - 862 FIJO MOVIL salvo móvil aeronáutico 5.316B 5.317A RADIODIFUSIÓN 5.312 5.319		
	806 - 890 FIJO MÓVIL 5.317A RADIODIFUSIÓN	
862 - 890 FIJO MÓVIL, salvo móvil aeronáutico 5.317A RADIODIFUSIÓN 5.322 5.319 5.323	5.317 5.318	5.149 5.305 5.306 5.307 5.311A 5.320

ATRIBUCIÓN NACIONAL	USOS	OBSERVACIONES
460 - 890 MHz		
790 - 862 FIJO MÓVIL, salvo móvil aeronáutico	P P	5.316B 5.317A Sistemas terrenales capaces de prestar servicios de comunicaciones electrónicas (790-823/832-862 MHz) UN-151: Dispositivos PMSE (823-832 MHz) UN-153, UN-154
862 - 890 FIJO MÓVIL, salvo móvil aeronáutico	* *	5.317A 5.322 Sistemas terrenales capaces de prestar servicios de comunicaciones electrónicas (880-890 MHz) UN-39, UN-40, UN-41, UN-111 UN-115 UN-118: MICRÓFONOS SIN HILOS UN-135 RFID, UN-154 * Usos M y C (según notas UN)

Figura 4.6: Legislación para la banda de 868 MHz



ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT

890 - 1300 MHz

Región 1	Región 2	Región 3
890 - 942 FIJO MÓVIL, salvo móvil aeronáutico 5.317A RADIODIFUSIÓN 5.322 Radiolocalización 5.323	**890 - 902** FIJO MÓVIL, salvo móvil aeronáutico 5.317A Radiolocalización 5.318 5.325	**890 - 942** FIJO MÓVIL 5.317A RADIODIFUSIÓN Radiolocalización 5.327
	902 - 928 FIJO Aficionados Móvil, salvo móvil aeronáutico 5.325A Radiolocalización 5.150 5.325 5.326	
	928 - 942 FIJO MÓVIL, salvo móvil aeronáutico 5.317A Radiolocalización 5.325	
942 - 960 FIJO MÓVIL, salvo móvil aeronáutico 5.317A RADIODIFUSIÓN 5.322 5.323	**942 - 960** FIJO MÓVIL 5.317A	**942 - 960** FIJO MÓVIL 5.317A RADIODIFUSIÓN 5.320
960 - 1164 MÓVIL AERONAUTICO (R) 5.327A RADIONAVEGACIÓN AERONÁUTICA 5.328 5.328AA		

ATRIBUCIÓN NACIONAL	USOS	OBSERVACIONES
890 - 1300 MHz		
890 - 942 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización	M P R	5.317A 5.322 Sistemas terrenales capaces de prestar servicios de comunicaciones electrónicas (890-915/925-942 MHz) UN-40, UN-41 UN-104 CT1-E UN-154
942 - 960 FIJO MÓVIL, salvo móvil aeronáutico	M P	5.317A 5.322 Sistemas terrenales capaces de prestar servicios de comunicaciones electrónicas (942-960 MHz) UN-41, UN-154
960 - 1164 MÓVIL AERONAUTICO (R) RADIONAVEGACIÓN AERONÁUTICA	R R	5.327A 5.328 5.328AA UN-154

Figura 4.7: Legislación para la banda de 915 MHz

Bandwidth (kHz)	Spreading Factor	Coding rate	Nominal Rb (bps)
7.8	12	4/5	18
10.4	12	4/5	24
15.6	12	4/5	37
20.8	12	4/5	49
31.2	12	4/5	73
41.7	12	4/5	98
62.5	12	4/5	146
125	12	4/5	293
250	12	4/5	586
500	12	4/5	1172

Figura 4.8: Velocidad de transmisión LoRa en la familia SX127x

Capítulo 5

Aplicación Móvil

5.1 HTML

Hay que hacer una aplicación móvil para dar una interfaz a la comunicación entre el ESP32 y el dispositivo desde el que se quiera controlar el sistema del ganadero. Existen distintas opciones, la primera que se contempla es crear una aplicación Android. Esto se debe a que recientes informes sitúan muy por encima a usuarios de Android frente a la competencia (iOS y Windows Phone). Sin embargo la complejidad que conlleva esta, hace plantear otras alternativas.

Se decide usar una aplicación HTML. Las ventajas de usar este lenguaje de programación es que se puede usar en cualquier dispositivo móvil que cuente con un navegador web, es decir la inmensa mayoría. E incluso poder usar dicha aplicación desde otros dispositivos, como pueden ser ordenadores.

El método de vinculación del ESP32 al teléfono móvil (o dispositivo usuario) consta de dos partes. La primera parte el ESP32 estará en modo punto de acceso de WiFi. Una vez conectados a ese punto de acceso, se debe abrir el navegador web e introducir la dirección IP en la cual esta alojada la aplicación. Esta es 192.168.4.1, una vez en ella, el ESP32 nos lanza la interfaz HTML que se ve en la Figura 5.1. Básicamente muestra unas sencillas instrucciones para rellenar el formulario. Este formulario, capta la SSID y la contraseña introducida y el ESP32 pasa de estar en modo punto de acceso a modo estación para intentar conectarse al punto de acceso introducido. Se da cierto margen, y si no logra conectarse se resetea el microcontrolador. Es importante destacar que el código HTML de la aplicación va programado en la memoria del ESP32. Por tanto, diseño de una página web más amigable, con más características, imágenes, etc, implicaría un uso mayor de la memoria y un tiempo de programación ostensiblemente mayor.

Figura 5.1: Aplicación móvil en distintos dispositivos

5.2 io.adafruit

Con lo descrito anteriormente se dispone el chip maestro del ganadero conectado a internet. El siguiente paso es ir a la página de io.adafruit[1]. Esta página web dispone una interfaz para el protocolo MQTT, donde se pueden crear varios feeds, dashboards, triggers y servicios con aplicaciones de terceros. Como ya se comenta anteriormente, los feeds son los tópicos o las variables que se quieren publicar y mostrar. Los dashboards permiten mostrar esos tópicos de distintas maneras, de forma que sea más fácil visualizarlos e interpretarlos. Por ejemplo, en un mismo dashboard puedes combinar varias gráficas con distintos feeds, a su vez un cuadro de texto que vaya mostrando lo que se sube, crear una serie de botones para controlar esos feeds desde la misma, etc. Y los triggers son pequeños programas con los cuales se puede generar ciertas salidas en función de los datos de los feeds.

Esta página ofrece dos planes, uno gratis, que es el que se utiliza en el proyecto. Y otro de pago (10$ / mes). Se menciona esto, porque se limitan ciertas características en función del plan. En el plan gratuito sólo se puede tener 10 feeds, construir 5 dashboardas y una cadencia de subida de datos de 30 datos por minuto; estas características se tienen en cuenta a la hora del diseño software del sistema. Y, además, sólo conserva los datos durante un mes. En el plan de pago no hay límite de feeds ni dashboards, y se dobla el número de interacciones que se pueden hacer así como se duplica el tiempo de guardado de datos.

En este proyecto se aprovecha esa disponibilidad gráfica de exponer los datos y controlar ciertas comunicaciones, a pesar de la baja cadencia de datos. Además, se cuenta con los servicios asociados que podemos vincular. En primer lugar se configuraran los feeds y el dashboard para mostrarlos. Además, se programará un trigger. Por último se vinculará un servicio.

5.2.1 Feeds, Dashboard y Trigger

Se crean tres tópicos:

- Feedback: Este feed sirve para visualizar la retroalimentación que se ofrece desde el ESP32, esa retroalimentación es principalmente la conexión a un punto de acceso WiFi, la hora de la que se envían los datos y el estado de la comunicación LoRa (si esta esperando algún mensaje o ha finalizado la comunicación).
- Pasos: En esta variable se almacenan los pasos recibidos.
- PedirLoRa: Este feed no es modificable desde el ESP32, sino que está subscrito a él para leer su estado. Sirve para pedir los datos vía LoRa directamente desde el dashboard.

El dashboard que se configura consta de tres elementos. El primero de ellos es un botón que maneja el feed PedirLoRa. Pulsando este botón, varia su estado entre ON/OFF. El segundo elemento es un gráfico de los pasos. Este gráfico puede ser configurable mostrando los datos almacenandos en el feed Pasos, desde 30 días atrás hasta en la última hora (con distintos intervalos intermedios). El tercer elemento que se configura es un cuadro para la salida de texto, este cuadro muestra todos los nuevos datos de los feeds. Se puede ver el resultado final consultando el dashboard en un ordenador en la Figura 5.2, en un dispositivo móvil los elementos van uno sobre otro, pero se visualiza perfectamente deslizando la pantalla hacia abajo.

Por último se configura un trigger. Este trigger actúa cuando el ESP32 envía un feedback diciendo que se ha finalizado la transmisión. Dado que es absurdo volver a pedir datos cuando se acaba de hacer un vaciado de datos, este trigger lo que hará es fijar el feed PedirLoRa a OFF. Esto implica que no se vuelve a pedir datos a continuación, hasta que el usuario lo vuelva a solicitar pulsando al botón nuevamente.

5.2.2 Servicio

Io.adafruit permite, además, vincular la actividad a aplicaciones de terceros a través de servicios. Realmente un servicio es un trigger pero aprovechándose de otras aplicaciones. Es decir, se puede programar unas condiciones que han de suceder con los feeds y como consecuencia se actúa en otra aplicación vinculada. Se decide usar servicios a través de IFTTT [2], este portal aúna una gran cantidad de aplicaciones de terceros de distintos tipos como pueden ser Dropbox, Google Drive, Twitter, Wikipedia, ios Calendar entre muchos otros. Lo que realiza esta página es recoger datos de una de estas aplicaciones y en función de lo que se programe,

[1] `https://io.adafruit.com`
[2] `https://ifttt.com/discover`

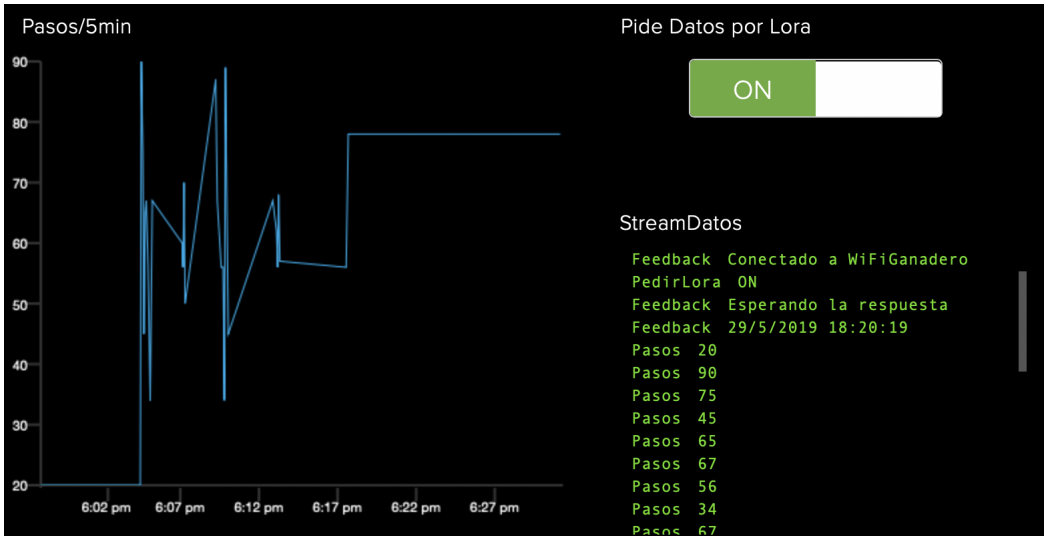


Figura 5.2: Dashboard

realiza respuestas en otra aplicación distinta. Estos programas que se realizan y automatizan tareas se llaman Applets.

El objetivo del servicio que se va a configurar es solucionar la limitación de tener solamente 30 días los datos en io.adafruit, y, además, tener una fácil exportación de datos. Para ello lo que se pretende es que el tópico Pasos que se transmita desde el ESP32 a través de MQTT, acaben en una hoja de cálculo. Para ello se necesita, a parte de lo ya configurado en io.adafruit, una cuenta de Google Drive donde se crea esa hoja de datos y se almacena.

En la página de IFTTT, para configurar una nueva Applet[3] se selecciona la opción "New Applet". Una vez en esa página, se ofrece una condición básica de programación, "if this+ then that+ ". Es decir, si en una aplicación sucede algo, en la otra se genera una reacción. Clicando en la palabra "this" se abre un menú con todas las aplicaciones disponibles, se selecciona Adafruit. Adafruit ofrece dos opciones, una sencilla, que cada vez que se actualiza un feed haya una consecuencia, o una segunda más compleja que conlleve una condición. Es decir, que en función del valor que tome el feed seleccionado se tendrá una consecuencia u otra. Se opta por la primera opción (Any New Data) y se selecciona el tópico que se va a monitorizar. En este caso Pasos. Una vez seleccionado el feed, se vuelve a la primera página de configuración de la Applet. Entonces se debe seleccionar la aplicación donde se lleve a cabo el desarrollo de la Applet clicando en la palabra "that". Ahora hay que escoger Google Sheets. Esta aplicación nos ofrece dos posibilidades. La primera es actualizar una celda concreta en una hoja de datos, y la segunda es añadir una fila con los datos del feed. Se usa la segunda opción (Add row to spreadsheet). Cuando se selecciona esa opción se muestra un menú de configuración donde se puede elegir el nombre de la hoja de datos, el formato de los datos guardados y la carpeta donde se guarda dentro de Google Drive. Una vez configurado la hoja de datos, se muestra como queda configurado el Applet con una pequeña descripción y se pide una confirmación del mismo dando la opcioón de darle un nombre. Mencionar que la hoja de datos y la carpeta se crean si no están ya creadas, y si están creadas, los datos del feed se añaden a continuación de lo ya almacenado.

Es una gran ventaja conseguir disponer de los datos en una hoja de datos automáticamente, dado que esto ya abre un gran abanico de posibilidades al usuario que puede usar, tratar e interpretar esos datos como desee y de forma autopersonalizada.

[3]Se dispone de documentación para crear este tipo de programas en `https://platform.ifttt.com/docs`

Capítulo 6

Montaje y Pruebas de campo

6.1 Montaje

Una vez recibida la placa (Figura 6.1) se procede al montaje de la misma. Se suelda sin problemas los componentes escogidos y se procede al testeo básico de los componentes. Conforme a lo expuesto en el Apartado 7.1.1 se ha de practicar una modificación de la PCB para poder programar el ESP32, esta se ve aplicada en la Figura 7.2. Por último, se ha de practicar la substitución de la resistencia R8 de la placa Heltec WiFi LoRa ESP32. Consultando el esquemático (Figura A.3) se llega a la conclusión que se ha de substituir la resistencia marcada en la Figura 6.2.

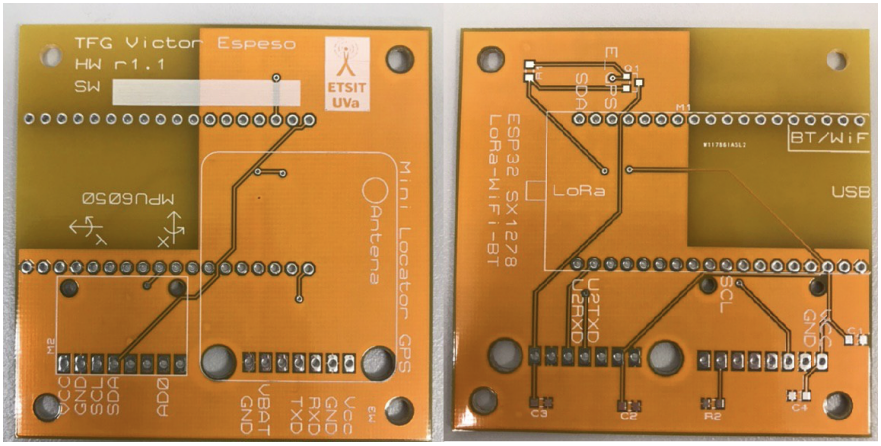


Figura 6.1: Placa original recibida

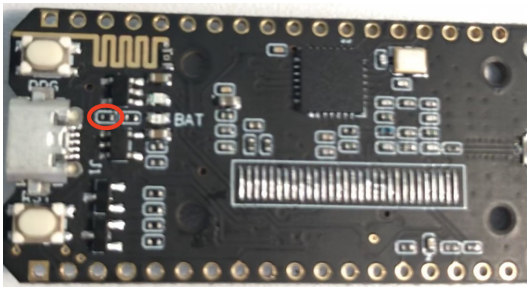


Figura 6.2: Modificación de R8

Se aplican también ligeras modificaciones artesanales a la caja. Los pivotes se taladran para poder atornillar la placa y que no se desplace por la caja.

6.2 Prueba de alcance de LoRa

Se prueba la comunicación LoRa para determinar el alcance del dispositivo y ver el rendimiento que es capaz de proporcionar. Mencionar que los experimentos se realizan fijando una potencia de transmisión de 14 dBm, un factor de ensanchamiento de 9 y un ancho de banda de 250 KHz. Simplemente se programa el emisor para enviar un mensaje cada cinco segundos, y el receptor para intente captar dicho mensaje, mida la potencia a la que se recibe y lo suba a la nube para poder visualizar una gráfica. Se realizan dos pruebas, una en ámbito urbano y otro en ámbito campestre.

El primer experimento localizamos el módulo receptor en un habitáculo interior, mientras ambos dispositivos comparten habitáculo el Received Signal Strength Indicator (RSSI) se mantiene en torno a -17 dBm. Sin embargo, en el momento que mueve el emisor a otra habitación y se pone una pared de por medio, como se ve en la Figura 6.3 el RSSI cae ostensiblemente llegando a -113 dBm. A continuación se sigue alejando el dispositivo emisor en línea recta y visión directa a la ventana que comunica con el receptor, y parece que la señal se mantiene. Pero en el primer edificio que hay que sortear la señal cae y se pierde completamente.

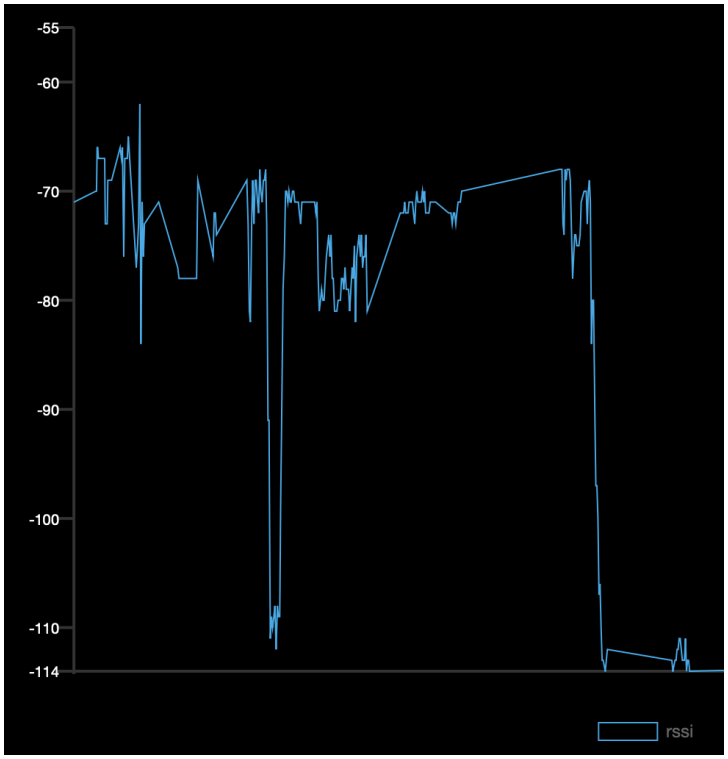


Figura 6.3: RSSI en zonas urbanas (dBm)

La segunda prueba de alcanza se realiza en un sitio más óptimo y parecido a donde pueda estar posicionado si el proyecto llega a su finalización. Se realiza el experimento en un ámbito rural, donde apenas hay edificación ni elementos que puedan interferir. En este caso, al irse alejando el emisor del receptor se va bajando paulatinamente la potencia recibida, pero no hay saltos bruscos como el anterior experimento. El experimento concluye cuando se pierde completamente la señal, se intenta tomar varios puntos desde distintas direcciones que muestren el máximo alcance de recepción. Se toman tres puntos, estos junto a la localización del emisor se introducen en la página de `http://sigpac.mapama.gob.es/fega/visor/` la cuál nos facilita la medida de distancias entre dos puntos geográficos de España. Los resultados se pueden ver en las Figuras 6.4, 6.5 y 6.6, estos denotan que la comunicación apenas llega a 600 m. Teniendo en cuenta que en los puntos marcados se recibían paquetes sueltos, es decir, no se recibían todos los paquetes emitidos. Estos resultados son algo decepcionantes, debido a que lo visto en el Apartado 4.4 se consideraba un máximo alcance de 15 Km, es decir no llega ni a un 3% de lo prometido.

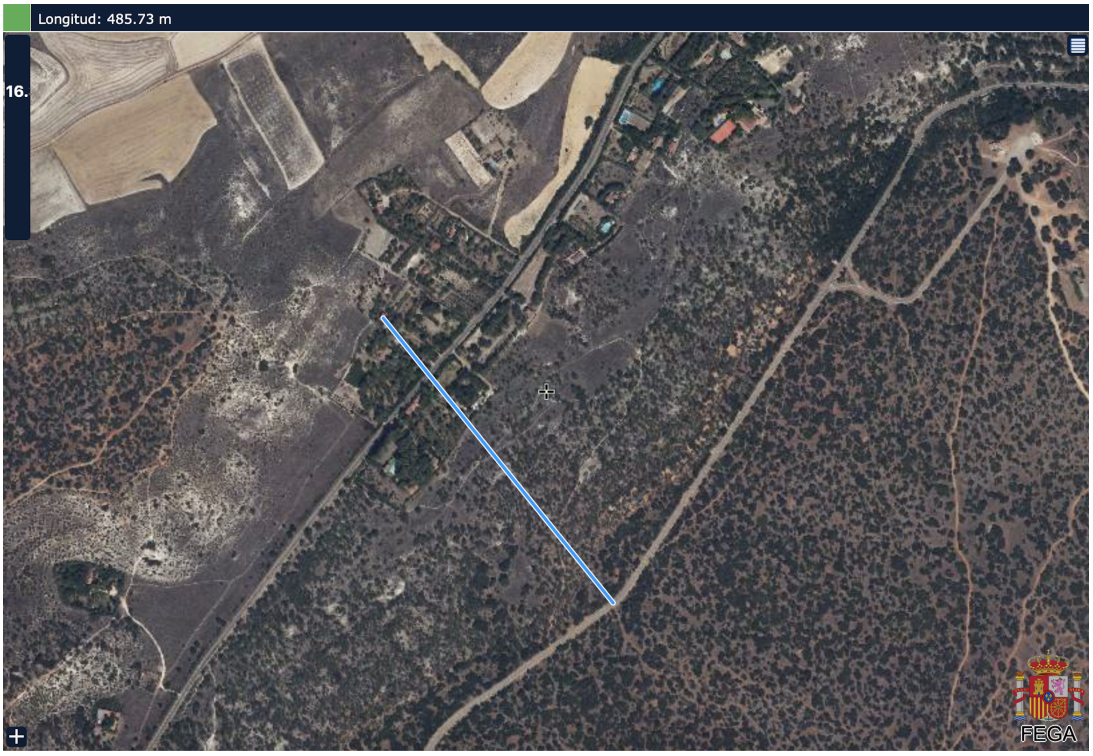


Figura 6.4: Recepción a 485 m

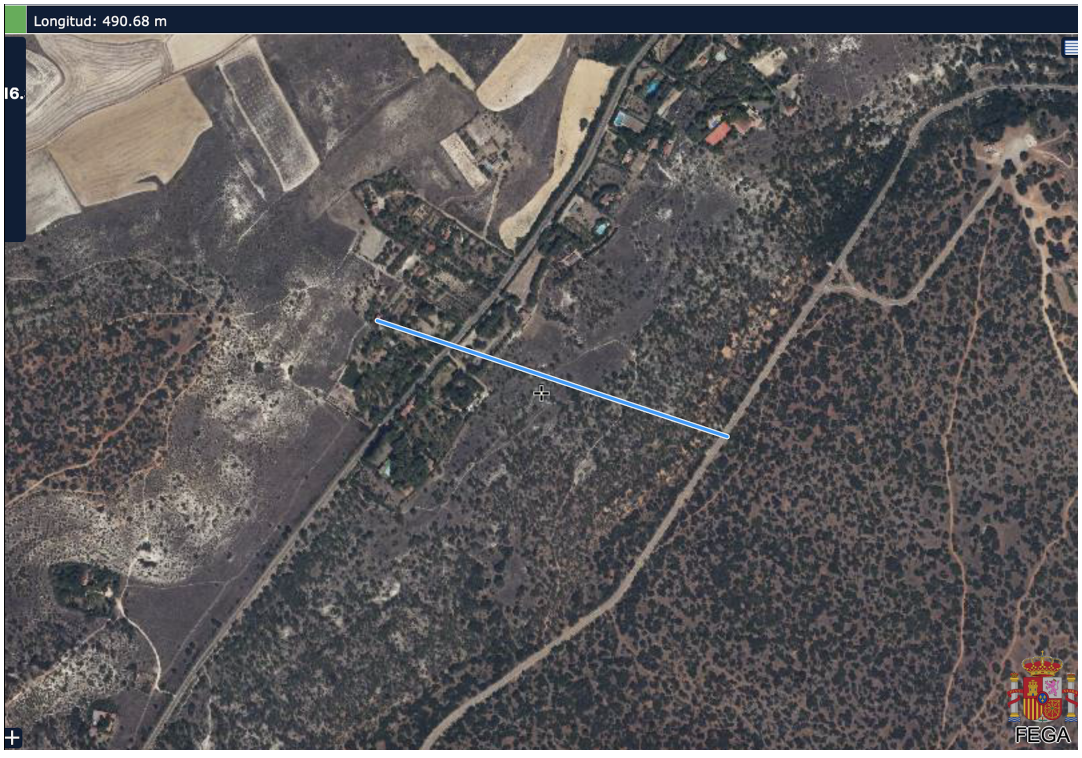


Figura 6.5: Recepción a 490 m

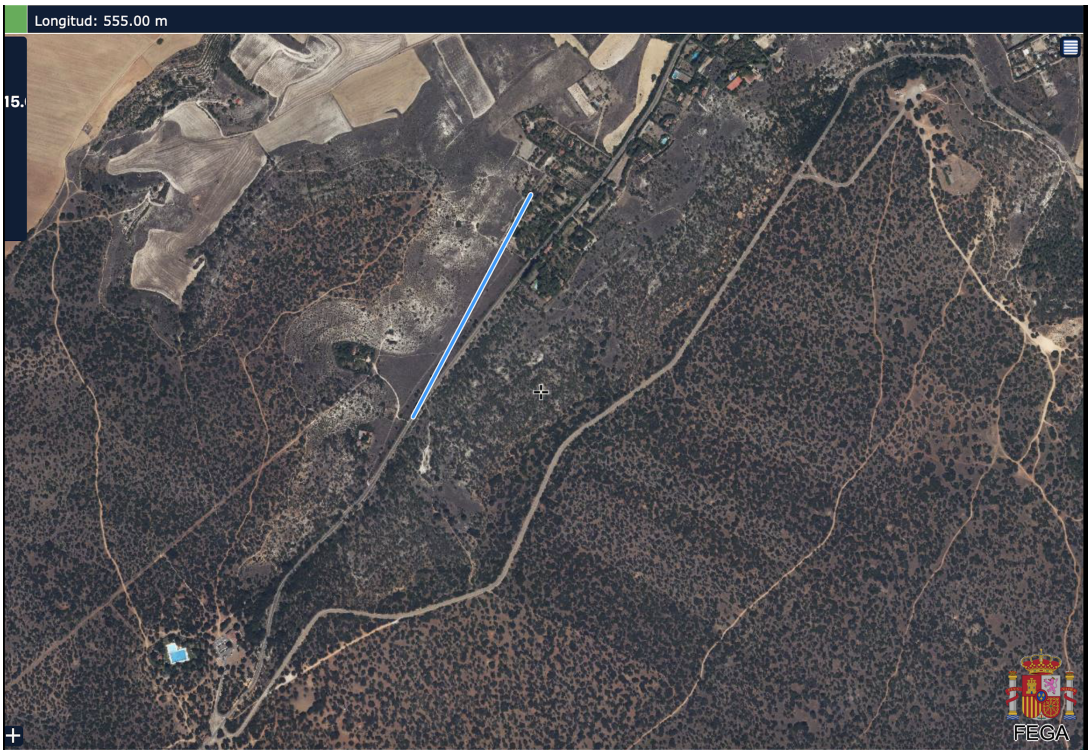


Figura 6.6: Recepción a 555 m

Capítulo 7

Errores y Contratiempos

7.1 Errores de diseño Hardware

7.1.1 GPIO12

Al intentar programar el dispositivo, una vez montado todo el sistema, el IDE de Arduino arrojaba un error de que no detectaba la memoria Flash y que era incapaz de programar el ESP32.

La primera manera de intentar detectar el fallo fue comprobar si había cortocircuitos en la placa. Una vez comprobado y descartados los fallos provenientes de cortocircuitos, se planteo otra estrategia.

Se decide desmontar uno a uno los componentes e intentar programar el chip para detectar qué elemento es el que daba problemas. Se llega a la conclusión que la R1 era la que daba el problema, dado que con ella desoldada se podía programar el chip sin problemas. Hay que recordar que esta resistencia ejercía de pull-up para el pin de la GPIO12 y el transistor a través del los cuales se controlaba la alimentación del GPS.

Antes de intentar resolver el problema se decide buscar la fuente original del mismo y se procede a una investigación exhaustiva de las distintas funciones del pin en cuestión.

Consultando el esquemático se observa que la GPIO12 comparte función en el mismo pin con: ADC2:5, TOUCH5, RTC15, SPIQ Y MTDI. Siendo esta última su función principal. Se empieza buscando en la hoja de datos por la función principal de este pin. La función MTDI se enmarca dentro de los pines robustos del ESP32. Estos pines son GPIO0, GPIO2, GPIO5, MTDI (GPIO12) y MTDO (GPIO15). Estos pines se utilizan para configurar el modo boot del dispositivo, los voltajes de operación de VDD_SDIO y otras configuraciones iniciales del sistema. Poseen una resistencia de pull-up y pull-down para controlar internamente su nivel lógico. Se pueden programar vía software accediendo al registro GPIO_STRAPPING, o se pueden programar vía hardware colocando de forma exterior una resistencia de pull-up o pull-down.

Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3 V		1.8 V	
MTDI	Pull-down	0		1	
Booting Mode					
Pin	Default	SPI Boot		Download Boot	
GPIO0	Pull-up	1		0	
GPIO2	Pull-down	Don't-care		0	
Enabling/Disabling Debugging Log Print over U0TXD During Booting					
Pin	Default	U0TXD Toggling		U0TXD Silent	
MTDO	Pull-up	1		0	
Timing of SDIO Slave					
Pin	Default	Falling-edge Sampling Falling-edge Output	Falling-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Figura 7.1: ESP32 Strapping pins

Para programarse el chip necesita confirmar el nivel lógico de MTDI. MTDI fija el nivel de alimentación de las memorias Flash. Concretamente el módulo ESP32 pose una memoria Flash incrustada (embedded flash) que opera a 3,3 V. Por tanto como se puede ver en la Figura 7.1 (sacada de la hoja de datos del dispositivo) se ve que que MTDI ha de estar fija a 0V. El diseño del proyecto fijaba este pin a 3,3V por tanto, un valor lógico de 1. El chip entiende entonces que la flash que hay que comprobar es de 1,8V y por tanto, no la detecta y no puede escribir el programa en ella. La solución artesanal que se da, es cortar las pistas que conectaban al pin GPIO12 y se usa otro pin de GPIO que no nos de problemas. Como se ve en la Figura 7.2, se usa el pin de GPIO25 como nuevo controlador de la alimentación del GPS. Se aprovechan los agujeros de las pistas en la placa para soldar el nuevo cableado, y se añade la resistencia de 4,7 K Ω lo más cerca del pin posible.

Una vez hecha esta modificación, ya se puede programar el microcontrolador y se controla la alimentación del sistema GPS sin problema.

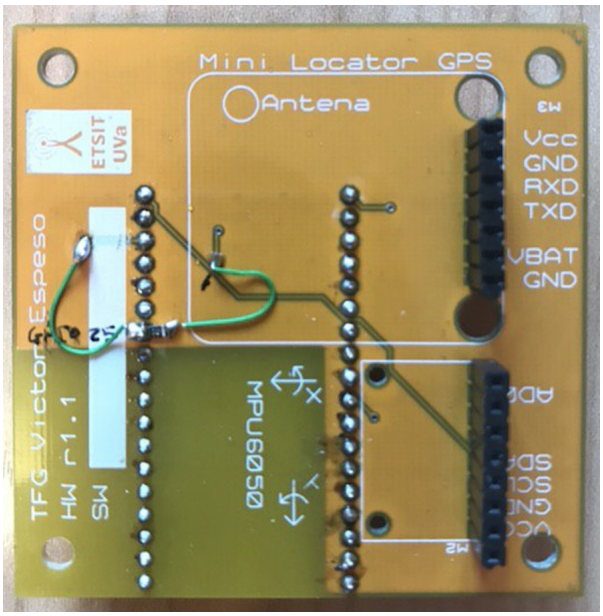


Figura 7.2: Cableado de corrección de la PCB

7.1.2 Huella del ESP32

Si se observa detenidamente las huellas de los dispositivos (Figuras 2.11, 2.10) y el diseño final de la PCB (Figuras 2.13 y 2.14) se puede observar en la parte correspondiente al cobre para soldar los componentes, que todas las huellas tienen una fina linea verde que rodea ese elemento de la huella, a excepción de la huella del ESP32 (Figura 2.9). Esto significa no se retira el plastico de la PCB para acceder al cobre de esas huellas durante la fabricación, por tanto, es imposible soldar ese elemento a la PCB. Este error lo denoto la empresa intermediaria que manda fabricar las PCB al observar los ficheros gerber enviados. Afortunadamente, ellos mismos pudieron abrir las vías al exterior y no hubo que rehacer el diseño.

7.2 Componentes

7.2.1 ESP32 LoRa

El principal problema de este dispositivo ya se comenta en el Apartado 1.4, es que no se proporciona una hoja de datos al uso del modulo en cuestión. Hay que intuir con el esquemático, la imagen de referencia y la hoja de datos general del ESP32, lo que se puede hacer y cómo.

El primer problema que se encuentra es que la mayoría de bibliotecas, ejemplos y fuentes, están codificadas para usar la pantalla OLED. Esto supone que se usa el canal de I2C para la pantalla, y que al coincidir en las salidas nos impide usar la UART2. Por tanto, se debe fijar al iniciar en nuestro programa, que no se usa el display, y que se usa el canal de I2C a través de los pines 22 (SDA) y 21 (SCL). Y fijar que se usan los

pines 16 y 17 para la UART2. Si no se explicita en el código en actualizaciones de las bibliotecas se puede encontrar la sorpresa de que cambian el canal de salida de I2C y no se detecta el MPU6050.

7.2.2 GPS

Una vez conseguido conectar el REB-5216 al ESP32 y comunicarlo se decide hacer pruebas de tiempo de adquisición. Para ello se programa en el microcontrolador un simple código que todo lo que reciba por la UART2 lo muestre por el serial. Así conseguiremos ver los datos NMEA sin procesar tal como se explicaron en el final del Apartado 2.2.2.

Se parte de la premisa de que no es capaz de funcionar en un interior. Por tanto se decide a ir a un exterior urbano. Después de 10 min recibiendo datos con el GPS encendido de manera ininterrumpida, no se recibe ningún dato. Se concluye que los edificios colindantes pueden interferir en la comunicación satélite. Se decide hacer la prueba en un ámbito rural, el cual es más acorde a la situación que se enmarca en el proyecto.

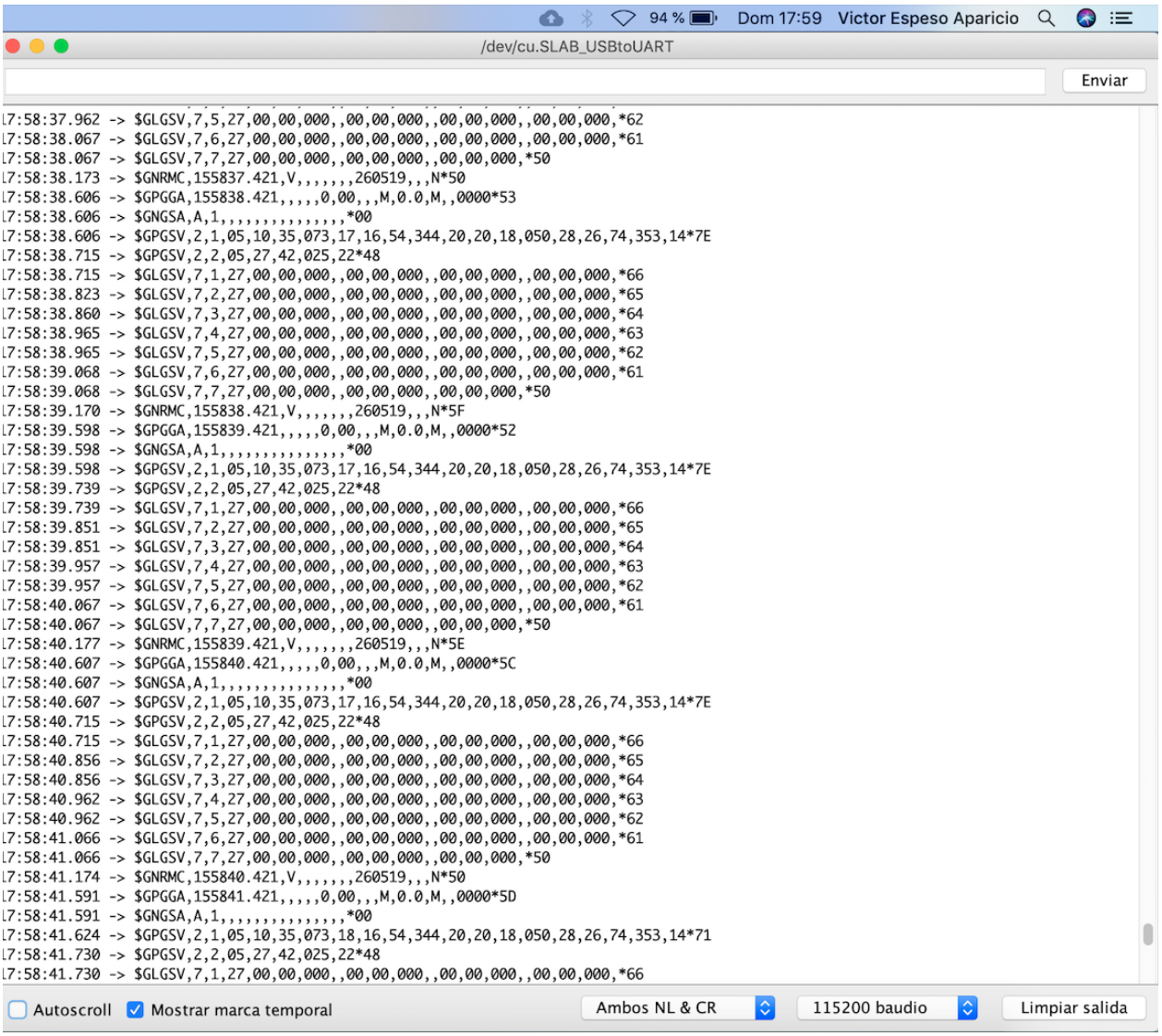


Figura 7.3: Frases NMEA después de 15 min

Esta segunda prueba se realiza en mitad del campo en lo alto de un pequeño cerro. En la Figura 7.3 se ven las frases que emite el REB-5216 después de 15 minutos conectado. En una lectura rápida interpretando los datos vemos que la frase $GPGGA y $GNRMC son capaces de dar la UTC [1]. Además, $GNRMC detecta correctamente la fecha, pero también con el carácter V después de la hora indica que los datos captados

[1] España en horario de verano tiene dos horas de diferencia respecto a UTC

pueden no ser correctos. Esto se debe a que el CRC de esa frase es erróneo. En la Tabla 7.2.2 se ve que se llegan a detectar hasta cinco satélites, pero al indicar en dos de ellos "null" en la SNR, significa que en ese instante no se tiene seguimiento del mismo. Por tanto, simultáneamente sólo se tiene información de tres de ellos. Como se comento, teóricamente, se pudiera estimar una posición con tres satélites, pero este dispositivo no es capaz de hacerlo.

ID	**Elevación** (^{o})	**Azimut** (^{o})	**SNR (dB-Hz)**
10	35	073	17
16	54	344	20
20	18	050	28
26	74	353	*null*
27	42	025	*null*

Tabla 7.1: Decodificación de las frases $GPGSV recibidas

Una vez hechas estas pruebas, se denota que no se cumple lo que se muestra en la hoja de datos del REB-5216, que asegura que en 35 segundos se obtienen datos validos. Por lo expuesto los tiempos de adquisición son ostensiblemente más grandes. Se plantea el supuesto que en 30 minutos de encendido ininterrumpido el dispositivo GPS logra captar los datos de posición. Y que su estrategia de uso es captar un dato cada hora. Se obtiene el consumo que se muestra en la Ecuación 7.1. Este consumo duplicaría el consumo del sistema, limitando mucho más la autonomía del sistema.

$$0,5 \times 50\ \mu A + 0,5 \times 35\ mA = 17,525\ mA \tag{7.1}$$

Se plantean dos posibles soluciones a este problema:

1. Localizar vía GPS sólo una vez al día. Conlleva un nuevo problema, dado que esto no sería capaz de cuantificar el movimiento del animal de manera efectiva.

2. Localizar el animal vía GPS bajo demanda. Es decir que el ganadero a través de su sistema solicite la posición de un animal concreto. El problema de esto es que el tiempo que el ganadero va a estar esperando a localizar el animal es grande.

Como ya se ha comentado en el Apartado 4 no se programa apenas código relativo al GPS dado a estos problemas.

Capítulo 8

Presupuesto

Se realiza un presupuesto del proyecto, para ofertarlo en el caso que este se considere ya como un producto final.

Consideraremos dos tipos de gastos. Unos fijos, y otros variables. Los segundos dependerán del número de dispositivos que se adquieran.

Gastos Fijos

En los gastos fijos se definen los gastos en inversión, diseño, prototipado e inversión en horas. Se pueden desglosar de la siguiente manera:

- Diseño y Prototipos: En ella se incluye el precio estipulado en el BOM de los sistemas diseñados para tener un prototipo. Así como el gasto adiccional en las baterías de pruebas, las PCB y los componentes estropeados durante el proceso.
- Inversión en material: La impresora 3D para poder imprimir las cajas de plástico.
- Sueldo: La ganancia estipulada para este proyecto se calcula en función de los siguientes supuestos. Se estima un sueldo de 2700€ al año, trabajando 1800 horas. Por tanto, se calcula en torno a 15 €/ hora. El proyecto esta planteado para 300 horas, pero se calcula que han sido en torno a 350 horas trabajadas. Entonces ascendería a 5.250€ la ganancia por este proyecto.
- Se vende fijo, al menos un sistema maestro para el ganadero, independientemente de los sistemas para los animales.

Gastos Variables

Los gastos variables vienen definidos por el número de placas que se realicen. Es más, hay muchos componente que cuanto más se pidan, su precio por unidad se reduce ostensiblemente. Por ejemplo, la batería, una cuesta más de 80€ y si se piden más de 25 cuestan 53€ . Hay que añadir, además, la mano de obra y otros gastos como gastos imprevistos.

Presupuesto final

En las siguientes Figuras (8.1, 8.2 y 8.3) vemos el presupuesto para sucesivas cantidades de sistemas comprados. Al final se obtiene 6000€ de gastos fijos, se pidan las placas que se pidan. Este gasto influye menos en el número de sistemas pedidos, si se piden más, es decir, que se divide entre más sistemas. Finalmente se le añade unas ganancias industriales de 25%. Mencionar que no se aplican impuestos, en ningún caso. Se hace el presupuesto para tres ganaderías. Una pequeña de 50 hembras reproductoras, una mediana de 250 y finalmente, una grande de 500. Se estima, además, que se compra un 25% más de sistemas, para poder tener rápido recambio al agotarse las baterías y para tener de sobra para cuando se rompa algún sistema. Al final se ve como va bajando el precio por unidad.

Gasto Fijo	6.000,61 €
Diseño y Prototipado	
BOM Sistema del Animal	121,91 €
BOM Sistema del Ganadero	16,71 €
PCB	20,00 €
ESP32 SX1276	24,18 €
ESP32 SX1278	15,66 €
Baterias pruebas	18,49 €
Total	216,95 €
Impresora 3D	
Anycubic i3 Mega	300,00 €
Sueldo	
Sueldo estimado anual	27.000,00 €
Horas estimadas anuales	1800
Horas estimadas invertidas	350
Total	5.250,00 €
Un chip maestro	
BOM Sistema del Ganadero	16,71 €

Gastos variables		5.946,72 €
Unidades	63	
Gasto por unidad	94,392 €	
ESP32 SX1276	12,090 €	761,67 €
REB-5216	15,490 €	975,87 €
MPU6050	6,000 €	378,00 €
Batería	50,780 €	3.199,14 €
Resistencias	0,067 €	4,20 €
Condensadores	0,324 €	20,41 €
Transistor	0,127 €	8,00 €
PCB	1,437 €	90,51 €
Pines	0,350 €	22,05 €
Plástico para la caja	4,000 €	252,00 €
Cinta elástica	0,600 €	37,80 €
Tornillos	0,128 €	8,06 €
Montaje	2,000 €	126,00 €
Otros gastos	1,000 €	63,00 €

PRESUPUESTO	11.947,33 €
PRESUPUESTO TOTAL	14.934,16 €

Ganancia Industrial (25%)	2.986,83 €
PRESUPUESTO €/UD	237,05 €

Figura 8.1: Presupuesto para la venta de 63 sistemas

Gasto Fijo	6.000,61 €
Diseño y Prototipado	
BOM Sistema del Animal	121,91 €
BOM Sistema del Ganadero	16,71 €
PCB	20,00 €
ESP32 SX1276	24,18 €
ESP32 SX1278	15,66 €
Baterias pruebas	18,49 €
Total	216,95 €
Impresora 3D	
Anycubic i3 Mega	300,00 €
Sueldo	
Sueldo estimado anual	27.000,00 €
Horas estimadas anuales	1800
Horas estimadas invertidas	350
Total	5.250,00 €
Un chip maestro	
BOM Sistema del Ganadero	16,71 €

Gastos variables		29.241,24 €
Unidades	313	
Gasto por unidad	93,423 €	
ESP32 SX1276	12,090 €	3.784,17 €
REB-5216	15,490 €	4.848,37 €
MPU6050	6,000 €	1.878,00 €
Batería	50,780 €	15.894,14 €
Resistencias	0,042 €	13,27 €
Condensadores	0,204 €	63,85 €
Transistor	0,097 €	30,20 €
PCB	0,642 €	200,82 €
Pines	0,350 €	109,55 €
Plástico para la caja	4,000 €	1.252,00 €
Cinta elástica	0,600 €	187,80 €
Tornillos	0,128 €	40,06 €
Montaje	2,000 €	626,00 €
Otros gastos	1,000 €	313,00 €

PRESUPUESTO	35.241,85 €
PRESUPUESTO TOTAL	44.052,32 €

Ganancia Industrial (25%)	8.810,46 €
PRESUPUESTO €/UD	140,74 €

Figura 8.2: Presupuesto para la venta de 313 sistemas

Gasto Fijo	6.000,61 €
Diseño y Prototipado	
BOM Sistema del Animal	121,91 €
BOM Sistema del Ganadero	16,71 €
PCB	20,00 €
ESP32 SX1276	24,18 €
ESP32 SX1278	15,66 €
Baterias pruebas	18,49 €
Total	216,95 €
Impresora 3D	
Anycubic i3 Mega	300,00 €
Sueldo	
Sueldo estimado anual	27.000,00 €
Horas estimadas anuales	1800
Horas estimadas invertidas	350
Total	5.250,00 €
Un chip maestro	
BOM Sistema del Ganadero	16,71 €

Gastos variables		58.299,04 €
Unidades	625	
Gasto por unidad	93,278 €	
ESP32 SX1276	12,090 €	7.556,25 €
REB-5216	15,490 €	9.681,25 €
MPU6050	6,000 €	3.750,00 €
Batería	50,780 €	31.737,50 €
Resistencias	0,037 €	22,88 €
Condensadores	0,170 €	106,38 €
Transistor	0,071 €	44,38 €
PCB	0,563 €	351,67 €
Pines	0,350 €	218,75 €
Plástico para la caja	4,000 €	2.500,00 €
Cinta elástica	0,600 €	375,00 €
Tornillos	0,128 €	80,00 €
Montaje	2,000 €	1.250,00 €
Otros gastos	1,000 €	625,00 €

PRESUPUESTO	64.299,65 €
PRESUPUESTO TOTAL	80.374,57 €

Ganancia Industrial (25%)	16.074,91 €
PRESUPUESTO €/UD	128,60 €

Figura 8.3: Presupuesto para la venta de 625 sistemas

Capítulo 9

Conclusiones

9.1 Resultado del proyecto

Al final del proyecto se consigue un prototipo funcional. Es decir un prototipo que cumple con las especificaciones dadas, pero que aún le quedan fases de desarrollo para llegar a un producto final comercializable. A lo largo de el proyecto se han estudiado una serie de protocolos no vistos durante el Grado. Estos protocolos están estrechamente relacionados con el desarrollo de las IoT, siendo estos WiFi, Bluetooth, LoRa, MQTT. Además de otros protocolos para tareas más específicas como NMEA para los sistemas GPS y SPIFFS para el uso de memorias Flash en pequeños dispositivos.

El resultado denota el especial hincapié en el desarrollo Hardware, siendo este el que más pulido queda. Dejando documentadas todos los errores a corregir y rediseñando el esquemático y la PCB (véase Apéndice B). Además, el estudio de consumo y una estrategia de uso eficiente, cercioran que se puede conseguir un sistema con autonomía relativamente alta. Es por tanto, un producto funcional.

A este producto se le añade un firmware que muestra sencillamente como es capaz de recoger datos de los sensores y llevarlos hasta la nube. El hecho de que no se considere un producto final, es dado que el software debe sufrir severas mejoras para cumplir perfectamente con el objetivo de monitorizar los animales.

Por último, las diversas pruebas de campo complementan bastante bien la documentación. Viendo las características básicas del sistema y viendo sus principales limitaciones.

9.2 Líneas de futuro

Se concluye este proyecto afirmando que queda mucho trabajo para llevar este prototipo a un producto final. Para ello se proponen una serie de aspectos que se denotan que hay que mejorar o finalizar.

- GPS: Buscar una solución para el sistema GPS. Convendría hacer un estudio más exhaustivo del mercado de estos módulos y conseguir un dispositivo que consiga la localización en el mínimo tiempo posible, sin comprometer ni el consumo ni los costes.

- La caja: La finalización de la caja correctamente. Colocación correcta de los tornillos, espaciar bien la antena LoRa. Además, se recomienda realizar un diseño más ergonómico para la vaca y a su vez más amigable para que sea agradable a la vista.

- Memoria Flash: No se tiene en cuenta la limitación de la memoria Flash en ningún momento. Realizar un estudio para asegurar que no se pierde ningún dato. Y en el caso de que tenga espacio de sobra, establecer un log de archivos en función de las últimas peticiones de datos por LoRa. De esta manera no se eliminaría los datos según se emiten, sino que se dejan como copia de seguridad, hasta que se llene la memoria y se pisen con los datos más nuevos.

- Modos de bajo consumo: En el planteamiento y el diseño hardware se plantea utilizar el modo ligth speed para que el microprocesador consuma menos. Esto no se llega finalmente a concretar en el código del proyecto. Este punto es crítico ya que sin poder activar el modo de bajo consumo, el sistema apenas aguantaría unos días y no cumple las especificaciones del producto final.

- Servidor: El sistema de subida de datos a través de MQTT a io.adafruit es bastante sencillo e intuitivo, y proporciona una cantidad de servicios ingentes. Las limitaciones del ancho de banda que supone (tanto en la versión gratis como la de pago), la complicación de acceso y el poco tiempo de almacenamiento de los datos, hacen que no sea la mejor opción. Se necesitaría un sistema donde se pueda volcar una cantidad ingente de datos, aunque luego su previsualización sea más compleja de proporcionar. Además, hay que diferenciar los datos que vienen de los distintos chips de las vacas y almacenarlos en sitios distintos para guardar cierto orden.
- Comunicación LoRa: La comunicación LoRa es algo deficiente, tanto en el alcance, como en el protocolo implementado. Hay que conseguir que el sistema pueda comunicarse al menos a unos pocos kilometros de distancia. Y el protocolo que se implementa en el código es bastante simple, pidiéndose los datos a todos, no se tiene un control de flujo y apenas se controlan los errores. Esto puede que provoque interferencias entre sistemas y no se logre la comunicación. Podría implementarse, por ejemplo, que en primer lugar el maestro haga un broadcast y los esclavos disponibles envíen una respuesta corta y simple. Y en función de la RSSI y del tiempo que se lleve sin recoger los datos de un animal, el maestro abra una comunicación única con ese chip.

Apéndice A

Esquemático Componentes

A.1 Esquemático MPU6050

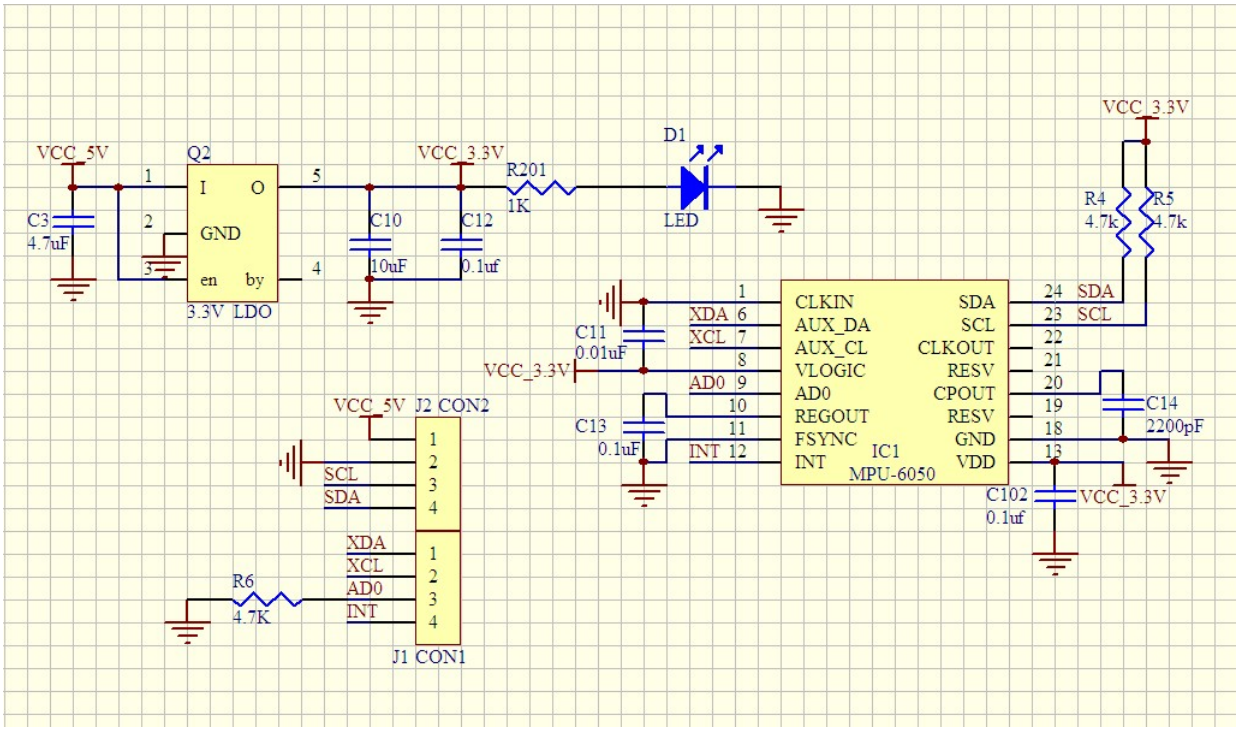


Figura A.1: Esquemático MPU6050

A.2 Esquemático REB-5216

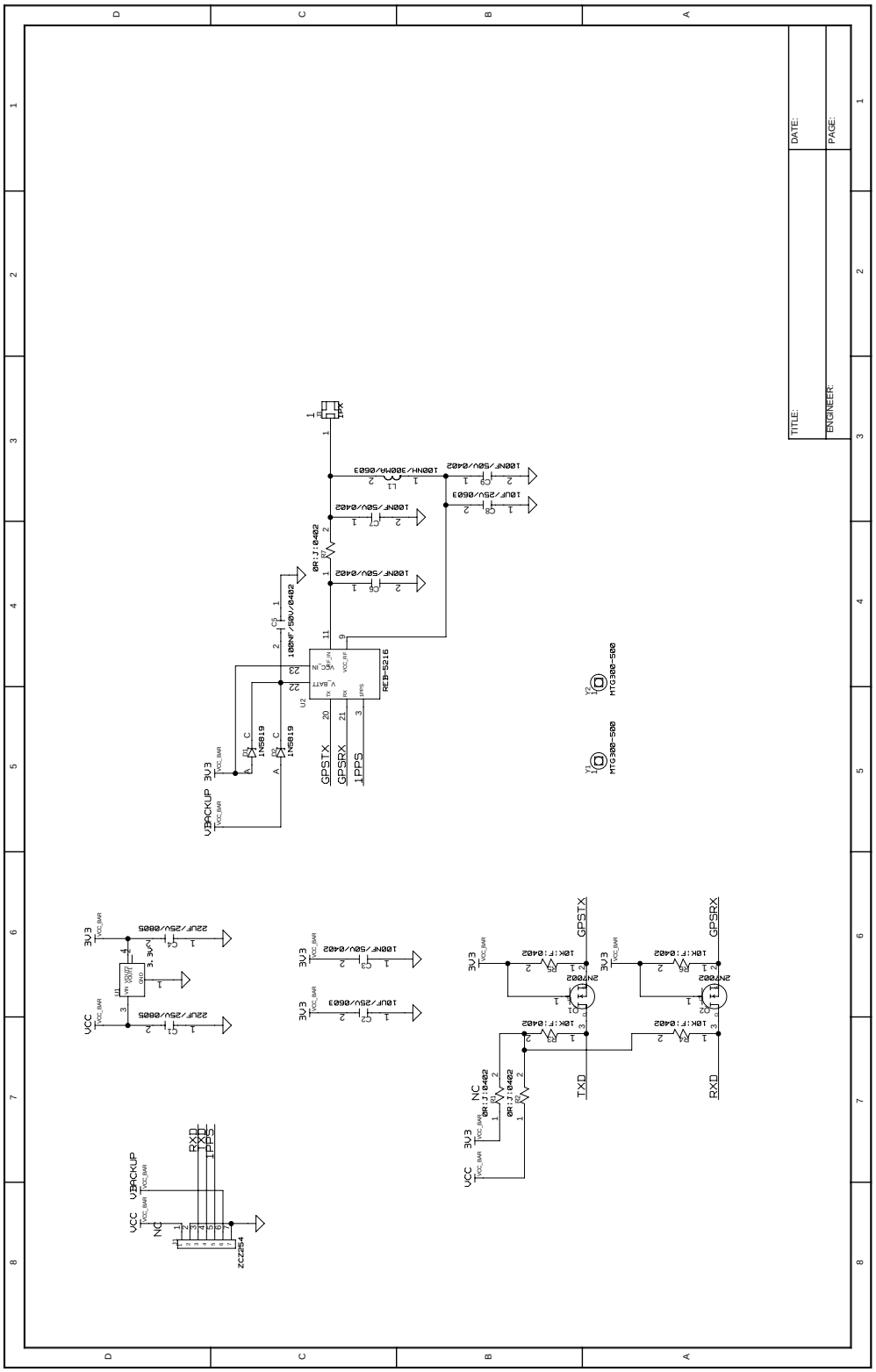


Figura A.2: Esquemático MPU6050

A.3 Esquemático ESP32

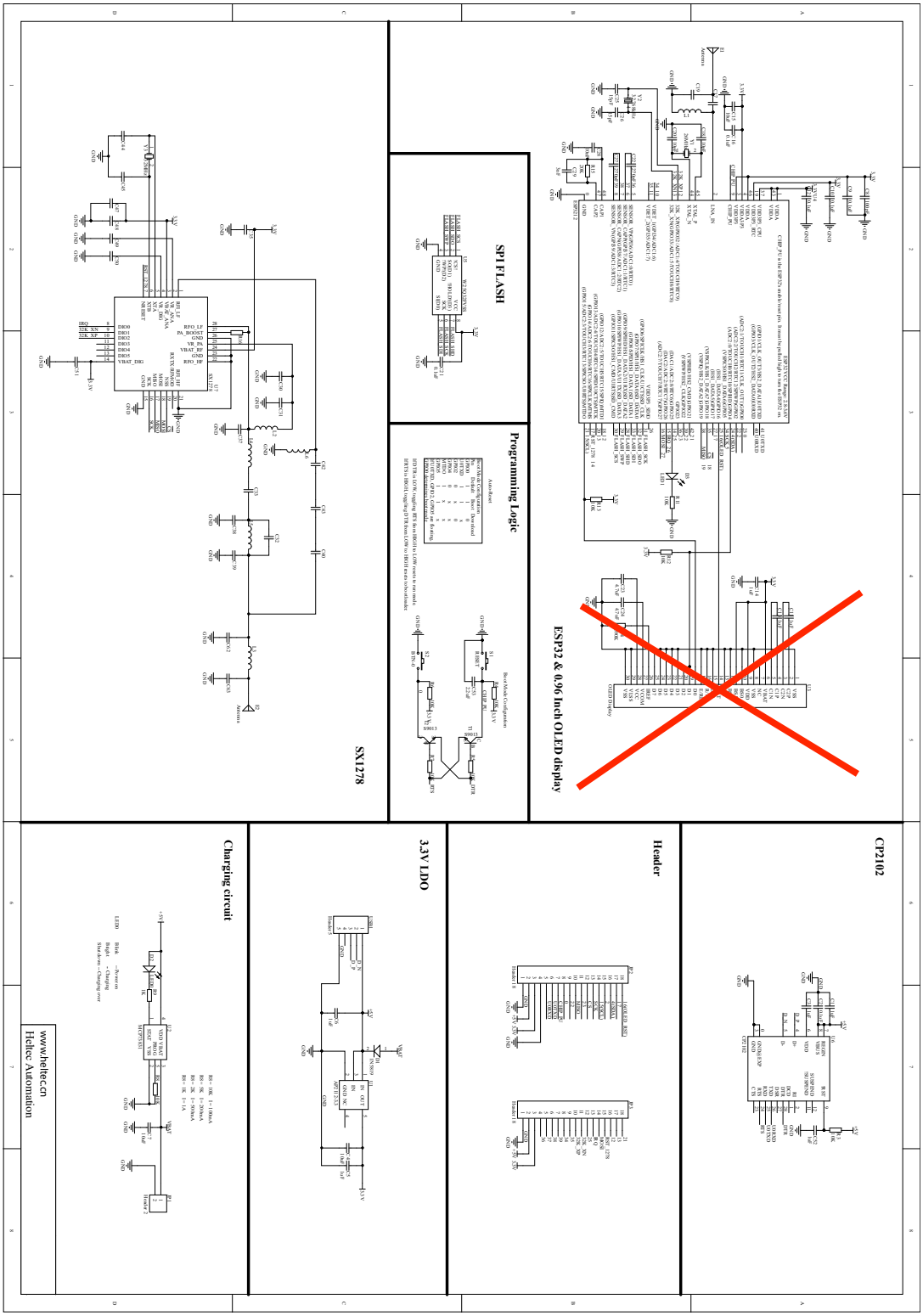
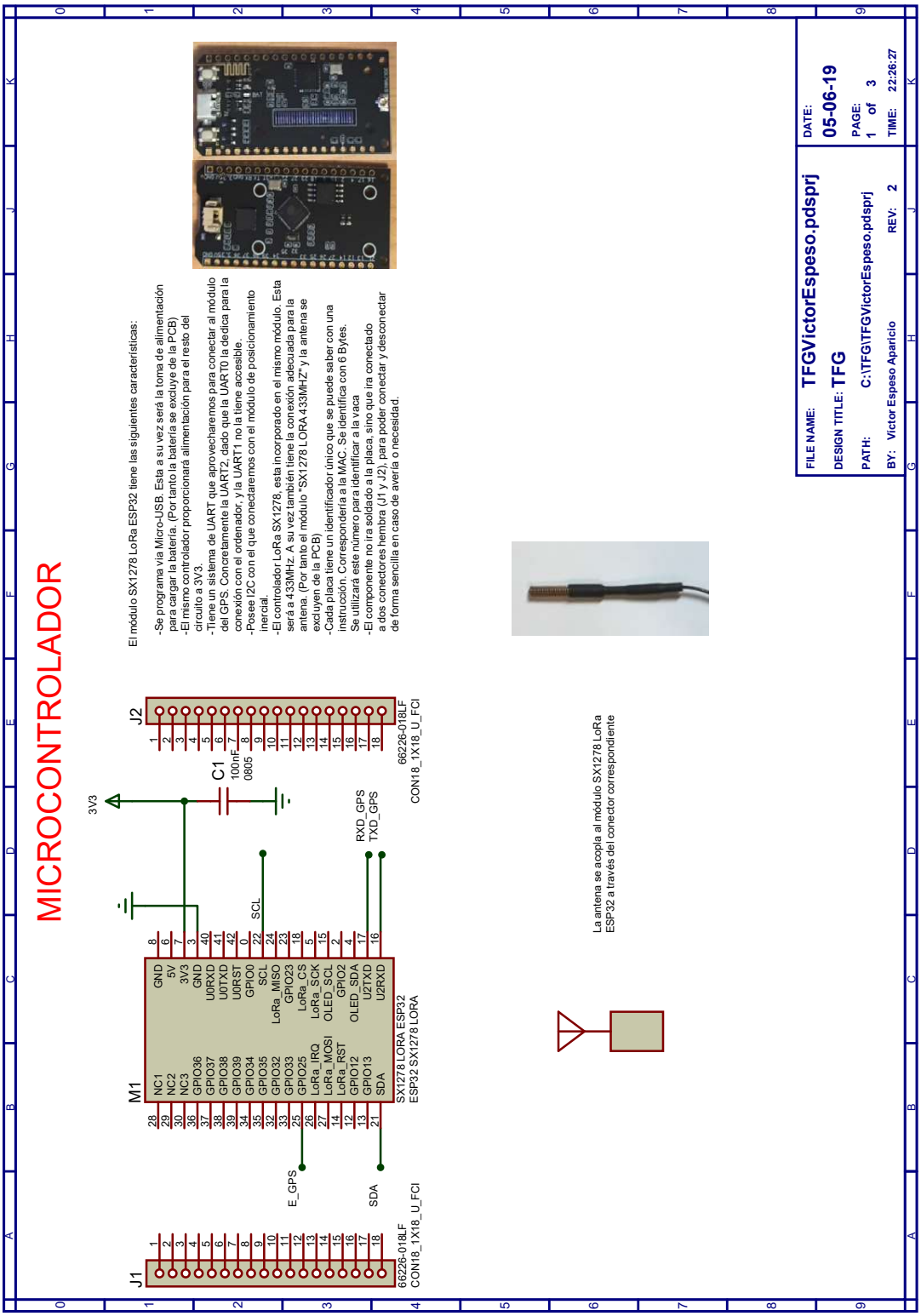


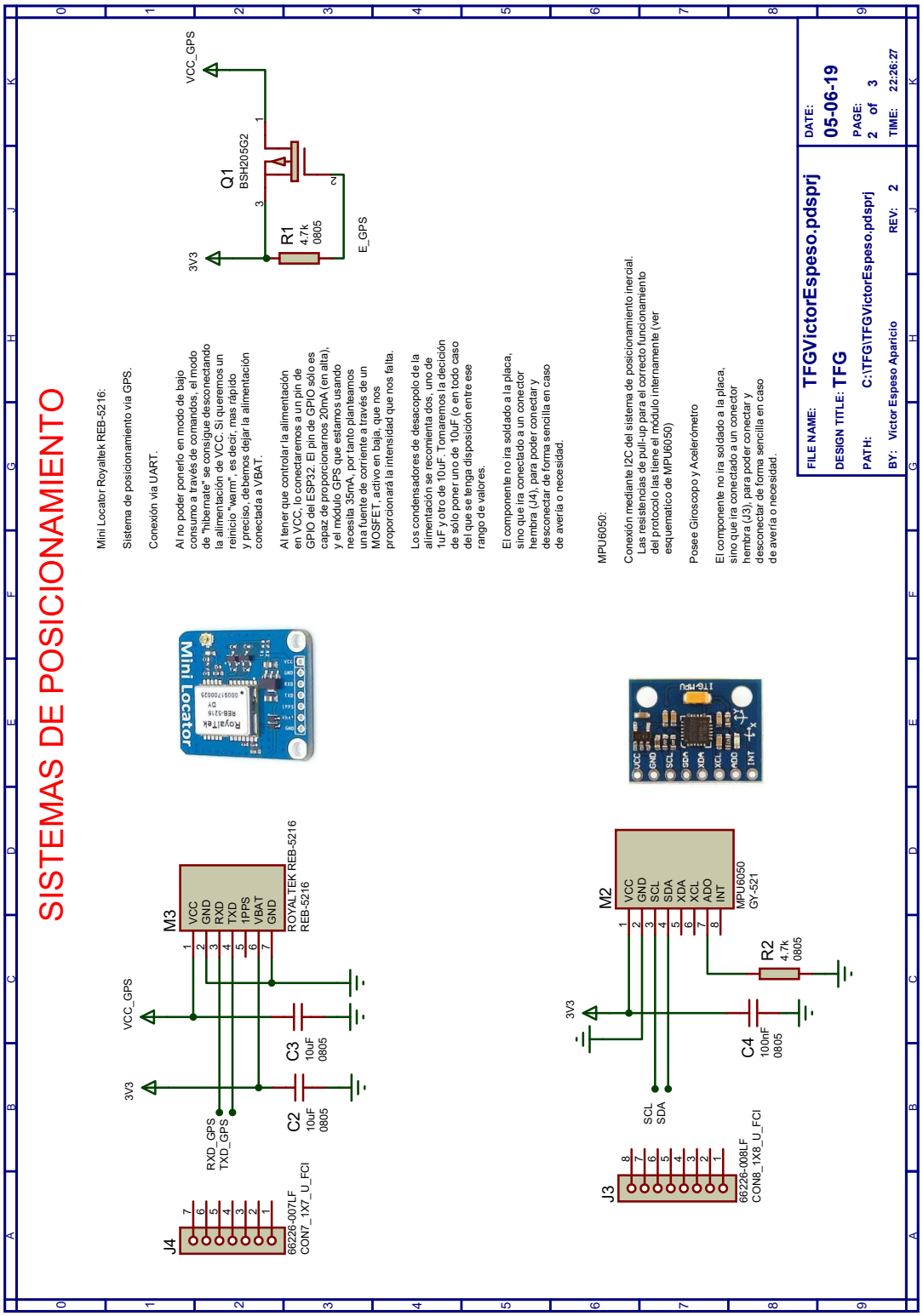
Figura A.3: Esquemático del módulo ESP32 LoRa

Apéndice B

Diseño Hardware

B.1 Esquemático del sistema de la vaca


MICROCONTROLADOR
El módulo SX1278 LoRa ESP32 tiene las siguientes características:
-Se programa via Micro-USB. Esta a su vez será la toma de alimentación para cargar la batería. (Por tanto la batería se excluye de la PCB)
-El mismo controlador proporcionará alimentación para el resto del circuito a 3V3.
-Tiene un sistema de UART que aprovecharemos para conectar al módulo del GPS. Concretamente la UART2, dado que la UART0 la dedica para la conexión con el ordenador, y la UART1 no la tiene accesible.
-Posee I2C con el que conectaremos con el módulo de posicionamiento inercial.
-El controlador LoRa SX1278, esta incorporado en el mismo módulo. Esta será a 433MHz. A su vez también tiene la conexión adecuada para la antena. (Por tanto el módulo "SX1278 LORA 433MHZ" y la antena se excluyen de la PCB)
-Cada placa tiene un identificador único que se puede saber con una instrucción. Correspondería a la MAC. Se identifica con 6 Bytes. Se utilizará este número para identificar a la vaca
-El componente no irá soldado a la placa, sino que irá conectado a dos conectores hembra (J1 y J2), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
La antena se acopla al módulo SX1278 LoRa ESP32 a través del conector correspondiente
M1
SX1278 LORA ESP32
ESP32 SX1278 LORA
J1
66226-018LF
CON18_1X18_U_FCI
J2
C1
100nF
0805
3V3
SCL
SDA
E_GPS
RXD_GPS
TXD_GPS
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio
REV: 2
DATE: 05-06-19
PAGE: 1 of 3
TIME: 22:26:27


SISTEMAS DE POSICIONAMIENTO
Mini Locator Royaltek REB-5216:
Sistema de posicionamiento via GPS.
Conexión via UART.
Al no poder ponerlo en modo de bajo consumo a través de comandos, el modo de "hibernate" se consigue desconectando la alimentación de VCC. Si queremos un reinicio "warm", es decir, mas rápido y preciso, debemos dejar la alimentación conectada a VBAT.
Al tener que controlar la alimentación en VCC, lo conectaremos a un pin de GPIO del ESP32. El pin de GPIO sólo es capaz de proporcionarnos 20mA (en alta), y el módulo GPS que estamos usando necesita 35mA, por tanto planteamos una fuente de corriente a través de un MOSFET, activo en baja, que nos proporcionará la intensidad que nos falta.
Los condensadores de desacopolo de la alimentación se recomienta dos, uno de 1uF y otro de 10uF. Tomaremos la decición de sólo poner uno de 10uF (o en todo caso del que se tenga disposición entre ese rango de valores.
El componente no ira soldado a la placa, sino que ira conectado a un conector hembra (J4), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
MPU6050:
Conexión mediante I2C del sistema de posicionamiento inercial. Las resistencias de pull-up para el correcto funcionamiento del protocolo las tiene el módulo internamente (ver esquematico de MPU6050)
Posee Giroscopo y Acelerómetro
El componente no ira soldado a la placa, sino que ira conectado a un conector hembra (J3), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
VCC_GPS
Q1
BSH205G2
R1
4.7k
0805
E_GPS
3V3
M3
VCC
GND
RXD
TXD
1PPS
VBAT
GND
ROYALTEK REB-5216
REB-5216
C3
10uF
0805
C2
10uF
0805
RXD_GPS
TXD_GPS
J4
66226-007LF
CON7_1X7_U_FCI
M2
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
MPU6050
GY-521
R2
4.7k
0805
C4
100nF
0805
SCL
SDA
J3
66226-008LF
CON8_1X8_U_FCI
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio
REV: 2
DATE: 05-06-19
PAGE: 2 of 3
TIME: 22:26:27

ALIMENTACIÓN Y CONSUMO
Consumo (3V3):
-REB5216
-Normal 35 mA
-Hibernate 50 uA
-MPU 6050 3,9 mA
-Gyr+Acr+DMP 10 uA (1 Hz update)
-LowPowAc 20 uA (5 Hz update)
70 uA (20 Hz update)
-SX1278 LoRa
-Sleep 0,2 uA
-Idle 1,5 uA
-Standby 1,6 mA
-Receive mode 12 mA
-Transmit mode 120 mA (+20 dBm)
87 mA (+17 dBm)
29 mA (+13 dBm)
20 mA (+7 dBm)
-ESP32
-240 MHz 50 mA
-160 MHz 34 mA
- 80 MHz 25 mA (Normal)
- 2 MHz 4 mA (Low-Speed)
-Ligth-Sleed 0,8 mA
Estrategia uso:
LoRa: Modo recepción 12 h/día. 6,00075 mA = (12 mA x 0,5)+(1,5 uA x 0,5)
GPS: Cada 5 minutos funcionando. 4,244 mA = (50uA x 0,88)+(35mA x 0,12)
Acelerómetro: A 40 Hz.
ESP32: 80 MHz sólo cuando hay que hacer algo, y Ligth-Speed el resto del tiempo.
Estara conectado mientras este funcionando el GPS y el Acelerómetro. Alimentará al módulo LoRa para cuando haya que hacer una transmisión que le despierte por interrupciones.
GPS = 10368s/día encendido
Acelerómetro = (40(datos/s)/4(datos/Tx))*0,48(ms/Tx)*86400(s/día)
= 414,72 s/día
Total ESP = 10782,72 s/día 12,48% del día
(25mA * 01248) + (0,8mA * 0,8752) = 3,82016 mA
Total 14,20491 mA
5000mAh - 14 días 16 h
10000mAh - 29 días 8 h
13400mAh - 39 días 7 h
*Había una tolerancia de un par de días dado que en el cálculo no se prevee una Tx y el tiempo que esta el procesador activo para dicha Tx.
BATERY
68 mm x 80 mm x 19 mm (+-1mm)
13400 mAh
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio REV: 2
DATE: 05-06-19
PAGE: 3 of 3
TIME: 22:26:27

Bill Of Materials for TFG

Design Title	TFG
Author	Víctor Espeso Aparicio
Document Number	1
Revision	2
Assembly Variant	Esclavo (Vaca)
Design Created	jueves, 6 de septiembre de 2018
Design Last Modified	viernes, 31 de mayo de 2019
Total Parts In Design	15

3 Modules

Quantity	References	Value	Unit Cost	PCB Package	LINK
1	M1	SX1278 LORA ESP32	€7,83	ESP32 SX1278 LORA	goo.gl/Ecz42g
1	M2	MPU6050	€6,00	GY-521	goo.gl/yYTFMf
1	M3	ROYALTEK REB-5216	€15,49	REB-5216	goo.gl/2W7824
Sub-totals:			€29,32		

4 Capacitors

Quantity	References	Value	Unit Cost	PCB Package	LINK
2	C1,C4	100nF	€0,11	0805	https://goo.gl/rQquqm
2	C2-C3	10uF	€0,11	0805	https://goo.gl/rQquqm
Sub-totals:			€0,43		

2 Resistors

Quantity	References	Value	Unit Cost	PCB Package	LINK
2	R1-R2	4.7k	€0,03	0805	goo.gl/97vypG
Sub-totals:			€0,07		

1 Transistors

Quantity	References	Value	Unit Cost	PCB Package	LINK
1	Q1	BSH205G2	€0,32	SOT23	https://bit.ly/2JV3xzh
Sub-totals:			€0,32		

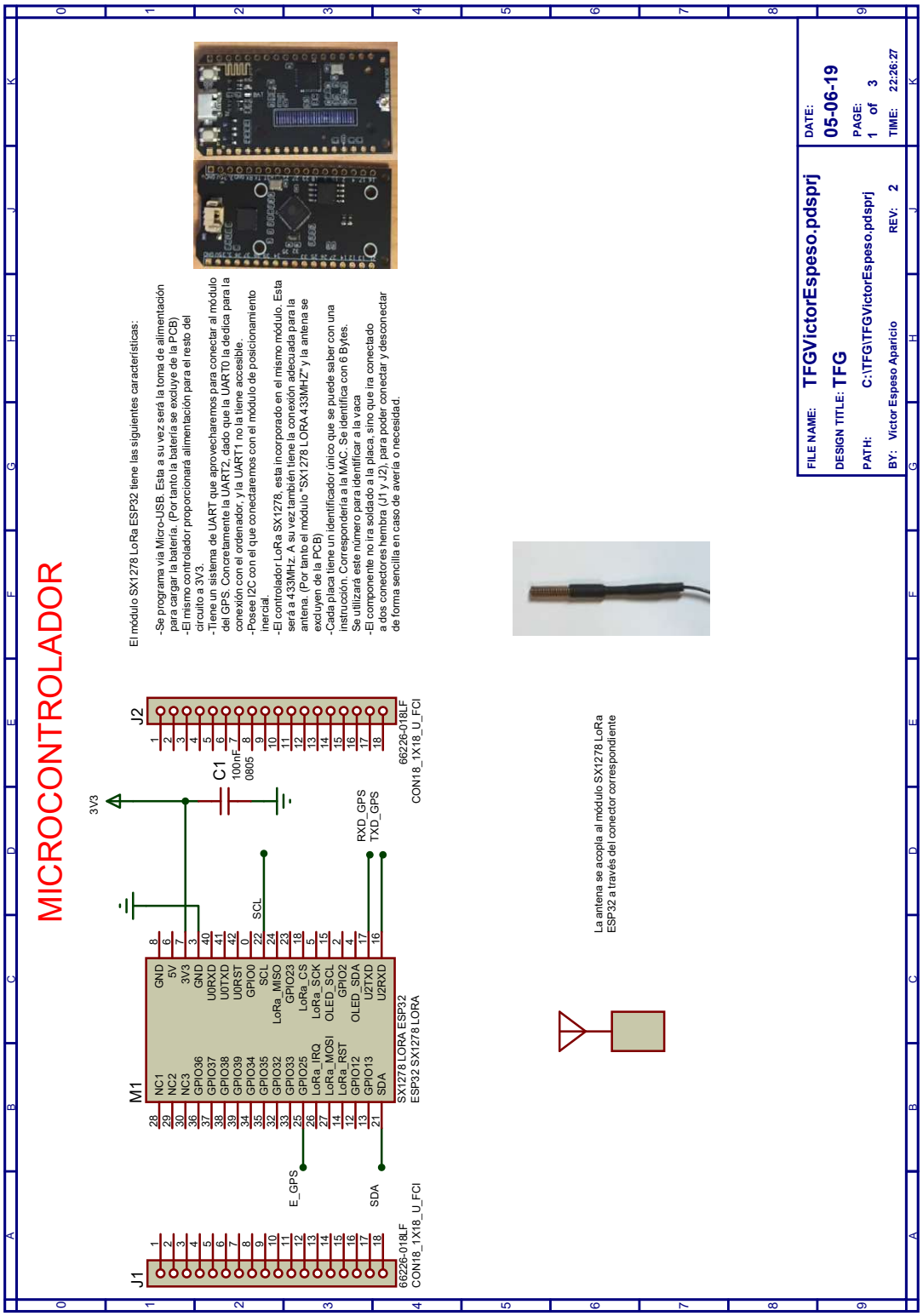
5 Miscellaneous

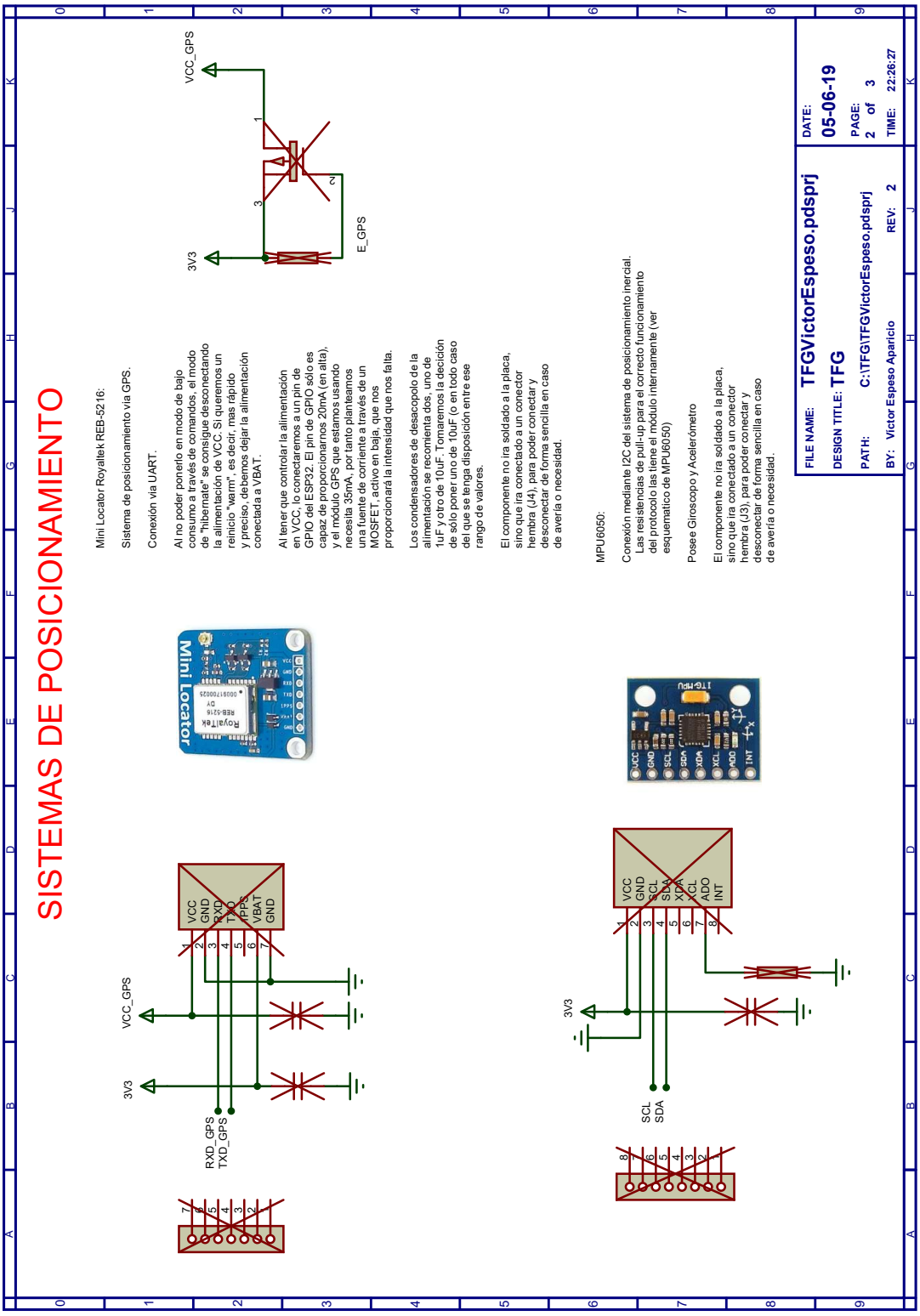
Quantity	References	Value	Unit Cost	PCB Package	LINK
1	B1	BATERY 13AH4	€83,00	NULL	https://goo.gl/XWTfNh
2	J1-J2	66226-018LF	€4,39	CON18_1X18_U_FCI	https://amzn.to/2Z4GLsd
1	J3	66226-008LF	€0,00	CON8_1X8_U_FCI	https://amzn.to/2Z4GLsd
1	J4	66226-007LF	€0,00	CON7_1X7_U_FCI	https://amzn.to/2Z4GLsd
Sub-totals:			€91,78		

Totals: €121,91

viernes, 31 de mayo de 2019 15:59:30

B.2 Esquemático del sistema del ganadero


MICROCONTROLADOR
El módulo SX1278 LoRa ESP32 tiene las siguientes características:
-Se programa via Micro-USB. Esta a su vez será la toma de alimentación para cargar la batería. (Por tanto la batería se excluye de la PCB)
-El mismo controlador proporcionará alimentación para el resto del circuito a 3V3.
-Tiene un sistema de UART que aprovecharemos para conectar al módulo del GPS. Concretamente la UART2, dado que la UART0 la dedica para la conexión con el ordenador, y la UART1 no la tiene accesible.
-Posee I2C con el que conectaremos con el módulo de posicionamiento inercial.
-El controlador LoRa SX1278, esta incorporado en el mismo módulo. Esta será a 433MHz. A su vez también tiene la conexión adecuada para la antena. (Por tanto el módulo "SX1278 LORA 433MHZ" y la antena se excluyen de la PCB)
-Cada placa tiene un identificador único que se puede saber con una instrucción. Correspondería a la MAC. Se identifica con 6 Bytes. Se utilizará este número para identificar a la vaca
-El componente no irá soldado a la placa, sino que irá conectado a dos conectores hembra (J1 y J2), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
La antena se acopla al módulo SX1278 LoRa ESP32 a través del conector correspondiente
J1
J2
M1
C1
100nF
0805
3V3
SCL
SDA
E_GPS
RXD_GPS
TXD_GPS
SX1278 LORA ESP32
ESP32 SX1278 LORA
66226-018LF
CON18_1X18_U_FCI
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio
REV: 2
DATE: 05-06-19
PAGE: 1 of 3
TIME: 22:26:27


SISTEMAS DE POSICIONAMIENTO
Mini Locator Royaltek REB-5216:
Sistema de posicionamiento via GPS.
Conexión via UART.
Al no poder ponerlo en modo de bajo consumo a través de comandos, el modo de "hibernate" se consigue desconectando la alimentación de VCC. Si queremos un reinicio "warm", es decir, mas rápido y preciso, debemos dejar la alimentación conectada a VBAT.
Al tener que controlar la alimentación en VCC, lo conectaremos a un pin de GPIO del ESP32. El pin de GPIO sólo es capaz de proporcionarnos 20mA (en alta), y el módulo GPS que estamos usando necesita 35mA, por tanto planteamos una fuente de corriente a través de un MOSFET, activo en baja, que nos proporcionará la intensidad que nos falta.
Los condensadores de desacopolo de la alimentación se recomienta dos, uno de 1uF y otro de 10uF. Tomaremos la decición de sólo poner uno de 10uF (o en todo caso del que se tenga disposición entre ese rango de valores.
El componente no ira soldado a la placa, sino que ira conectado a un conector hembra (J4), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
MPU6050:
Conexión mediante I2C del sistema de posicionamiento inercial. Las resistencias de pull-up para el correcto funcionamiento del protocolo las tiene el módulo internamente (ver esquematico de MPU6050)
Posee Giroscopo y Acelerómetro
El componente no ira soldado a la placa, sino que ira conectado a un conector hembra (J3), para poder conectar y desconectar de forma sencilla en caso de avería o necesidad.
VCC_GPS
3V3
E_GPS
Q1
VCC
GND
RXD
TXD
1PPS
VBAT
GND
RXD_GPS
TXD_GPS
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
SCL
SDA
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio
REV: 2
DATE: 05-06-19
PAGE: 2 of 3
TIME: 22:26:27

ALIMENTACIÓN Y CONSUMO
Consumo (3V3):
-REB5216
-Normal 35 mA
-Hibernate 50 uA
-MPU 6050 3,9 mA
-Gyr+Ac+DMP 10 uA (1 Hz update)
-LowPowAc 20 uA (5 Hz update)
70 uA (20 Hz update)
-SX1278 LoRa
-Sleep 0,2 uA
-Idle 1,5 uA
-Standby 1,6 mA
-Receive mode 12 mA
-Transmit mode 120 mA (+20 dBm)
87 mA (+17 dBm)
29 mA (+13 dBm)
20 mA (+7 dBm)
-ESP32
-240 MHz 50 mA
-160 MHz 34 mA
- 80 MHz 25 mA (Normal)
- 2 MHz 4 mA (Low-Speed)
-Ligth-Speed 0,8 mA
BATERY
68 mm x 80 mm x 19 mm (+-1 mm)
13400 mAh
Estrategia uso:
LoRa: Modo recepción 12 h/día. 6,00075 mA = (12 mA x 0,5)+(1,5 uA x 0,5)
GPS: Cada 5 minutos funcionando. 4,244 mA = (50uA x 0,88)+(35mA x 0,12)
Acelerómetro: A 40 Hz.
ESP32: 80 MHz sólo cuando hay que hacer algo, y Ligth-Speed el resto del tiempo.
Estará conectado mientras este funcionando el GPS y el Acelerómetro. Alimentará al módulo LoRa para cuando haya que hacer una transmisión que le despierte por interrupciones.
GPS =10368s/día encendido
Acelerómetro = (40(datos/s)/4(datos/Tx))*0,48(ms/Tx)*86400(s/día)
= 414,72 s/día
Total ESP = 10782,72 s/día 12,48% del día
(25mA * 01248) + (0,8mA * 0,8752) = 3,82016 mA
Total 14,20491 mA
5000mAh - 14 días 16 h
10000mAh - 29 días 8 h
13400mAh - 39 días 7 h
*Había una tolerancia de un par de días dado que en el cálculo no se prevee una Tx y el tiempo que esta el procesador activo para dicha Tx.
FILE NAME: TFGVictorEspeso.pdsprj
DESIGN TITLE: TFG
PATH: C:\TFG\TFGVictorEspeso.pdsprj
BY: Victor Espeso Aparicio
REV: 2
DATE: 05-06-19
PAGE: 3 of 3
TIME: 22:26:27

Bill Of Materials for TFG

Design Title	TFG
Author	Víctor Espeso Aparicio
Document Number	1
Revision	2
Assembly Variant	Maestro (Ganadero)
Design Created	jueves, 6 de septiembre de 2018
Design Last Modified	viernes, 31 de mayo de 2019
Total Parts In Design	4

1 Modules

Quantity	References	Value	Unit Cost	PCB Package	LINK
1	M1	SX1278 LORA ESP32	€7,83	ESP32 SX1278 LORA	goo.gl/Ecz42g
Sub-totals:			€7,83		

1 Capacitors

Quantity	References	Value	Unit Cost	PCB Package	LINK
1	C1	100nF	€0,11	0805	https://goo.gl/rQquqm
Sub-totals:			€0,11		

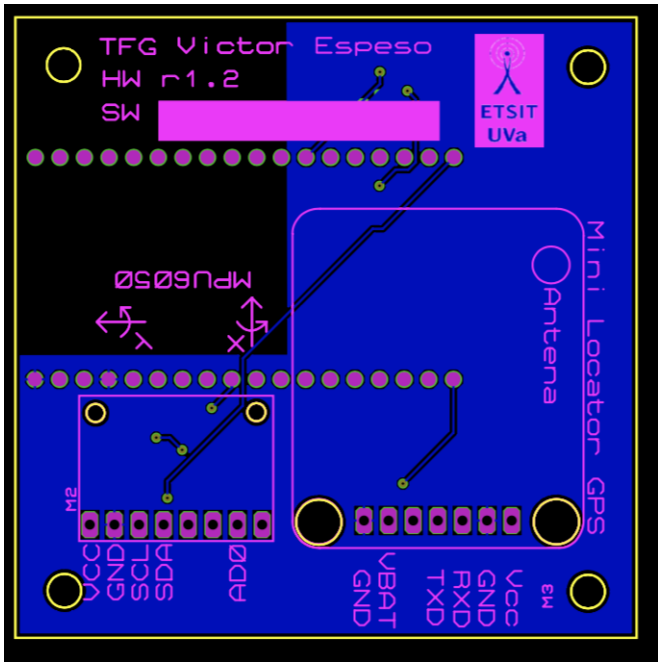
2 Miscellaneous

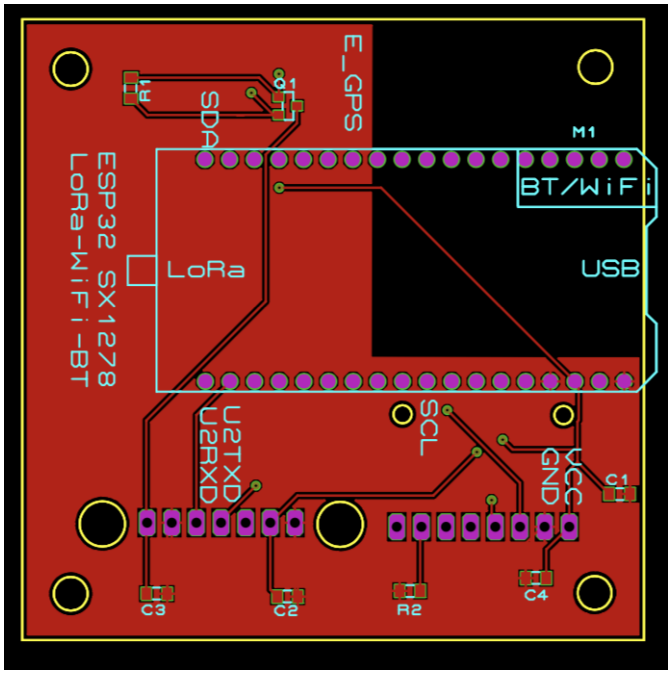
Quantity	References	Value	Unit Cost	PCB Package	LINK
2	J1-J2	66226-018LF	€4,39	CON18_1X18_U_FCI	https://amzn.to/2Z4GLsd
Sub-totals:			€8,78		

Totals: €16,71

viernes, 31 de mayo de 2019 15:54:49

B.3 PCB Final


TFG Victor Espeso
HW r1.2
SW
ETSIT
UVa
Antena
Mini Locator GPS
M2
VCC
GND
SCL
SDA
ADO
VBAT
GND
RXD
TXD
GND
Vcc
M3


E_GPS
SDA
M1
BT/WiFi
USB
LoRa
ESP32 SX1278
LoRa-WiFi-BT
U2TXD
U2RXD
SCL
VCC
GND
C1
C2
C3
C4
R2

Apéndice C

Código de la caja (OpenSCAD)

```
rad = 1;                //Radio de la esfera del borde
borde = (2 * rad)+1;    //Borde caja
he = (53-(2*rad));      //Altura caja
ae = (89-(2*rad));      //Ancho caja
le = (90-(2*rad));      //Largo caja
fn=100;                 //Segmentos de curvatura
ht=2.5;                 //Altura de la tapa

//Muesca para sujetar la PCB
module muesca(){
    difference(){
    cube([2.5,5,2.5]);
        translate([0,-1,2.5])
        rotate([0,45,0])
        cube([4,7,2.5]);
    }
}

//Agujeros para los tornillos
module screws(){
    translate([2,2, he-5]){
    color( "Aqua", 1.0 ){
        translate([le-4,0,0])
            cylinder(r=1.45, h=10, $fn=fn);
        translate([le-4,ae-4,0])
            cylinder(r=1.45, h=10, $fn=fn);
        translate([0,0,0])
            cylinder(r=1.45, h=10, $fn=fn);
        translate([0,ae-4,0])
            cylinder(r=1.45, h=10, $fn=fn);
        }
    }
}

//Ensanchamiento de esquinas para los tornillos
module corner(){
    translate([le-4,0,0])
        cube([4,4,he+rad]);
    translate([le-4,ae-4,0])
        cube([4,4,he+rad]);
    translate([0,0,0])
        cube([4,4,he+rad]);
    translate([0,ae-4,0])
        cube([4,4,he+rad]);
}

//Separadores verticales
module sepvert(){
    translate([9, 11, 0]){
        //Para apoyyar y separar la bateria
        cube([2, 75, 28+15]);
        //Habitaculo de la PCB
        cube([5, 75, 28]);
        difference(){
            cube([74, 5, 28]);
            translate([29.5, -2.5, -5])
                cube([15, 10, 50]);
            translate([5, 1.25, -5])
                cube([22, 2.5, 50]);
```

```
            translate([47, 1.25, -5])
                cube([25, 2.5, 50]);
        }
    }
    //Para que no se desplaze la antena GPS fuera de su sitio
    translate([53.5, 0, 0])
        cube([2.5,15, 10]);
}

//Separadores horizontales para sujetar las antenas
module sephor(){

    difference(){
        translate([0, 58, 20])
            cube([10,8, 9]);
        translate([5.9, 70, 25])
            rotate([90,0,0])
                cylinder(h=15, r=2.55, $fn=fn);
        }
}

//Modelo de pivote para sujetar la PCB
module mainpivot(){
    cylinder(h=14.5, r=3, $fn=fn);
    color( "red", 1.0 ){
        cylinder(h=17.5, r=1.1, $fn=fn);
    }
}

//Pivote para sujetar el GPS (No se usa)
module gpspivot(){
    cylinder(h=12.5, r=3, $fn=fn);
    cylinder(h=25, r=1.25, $fn=fn);
}

//Agujero para el puerto USB
module usbhole(){
    translate([36.5,84,23.5])
        cube([12,5,9]);
}

//Modulo principal de la caja
module box(){
    //Borde redondeado de la caja
    difference(){
        minkowski() {
            cube([le,ae,he]);
            sphere(rad, $fn=fn);
        }
        translate([rad+2,rad+0.5, rad])
        cube([le-(borde)-3,ae-(borde),2*(he-(2*borde))]);
    }

    //Separaciones
    sepvert();
    sephor();

    //Pivotes de sujeccion
    translate([21,26,0])
        mainpivot();
    translate([21,79,0])//referencia
        mainpivot();
    translate([75,26,0])
        mainpivot();
    translate([75,79,0])
        mainpivot();

    //soporte de sujeccion para los pivotes
    translate([75,24.7,0])
        cube([10,2.5,12.5]);
    translate([75,77.7,0])
        cube([10,2.5,12.5]);
    translate([19.8,78,0])
        cube([2.5,7.5,12.5]);
    translate([19.8,15,0])
        cube([2.5,10,12.5]);

    //Esquinas para los tornillos
    corner();

    //Muesca para sujetar la PCB
    translate([14,50,16])
        muesca();
  }
```

```
//Rail para introducir la tapa
module rail(){
    difference(){
        translate([0,2* borde,0]){
            minkowski() {
                cube([2,75,he+5]);
                sphere(rad, $fn=fn);
            }
        }
        translate([1,4,0])
        cube([borde,80,he+4]);
    }
}

//A la caja se le aplican los tornillos y los agujeros
module sbox(){
    difference(){
        box();
        usbhole();
        screws();
        }
}

//Diseño base de la tapa
module tapabase(){
    cube([63,85,ht]);

    translate([63,20,0])
        cylinder(r=20, h=ht, $fn=fn);
    translate([63,65,0])
        cylinder(r=20, h=ht, $fn=fn);

    translate([0,20,0])
        cube([83,45,ht]);
}

//Agujeros de la tapa
module tapahole(){
    translate([65,62.7,-2])
        cylinder(r=10, h=6, $fn=fn);

    translate([65,20,-2])
        cylinder(r=10, h=6, $fn=fn);

    translate([55,20,-2])
        cube([20,42,40]);
}

//A la base de la tapa se le aplica los agujeros
module mediatapa(){
    rotate([0,0,90]){
        translate([40,-86.5,52]){
            difference(){
                tapabase();
                tapahole();
            }
        }
    }
}

//Se completa la tapa haciendo un espejado y juntando ambas mitades
module tapa(){
    mediatapa();
    translate([0,85,0])
        mirror([0,1,0])
            mediatapa();
}

//A la tapa se le aplican los tornillos para evitar que deslice
module stapa(){
    difference(){
        tapa();
        screws();
    }
}


//Para mostrar los elementos hay que llamar a los modulos
```

Apéndice D

Códigos de los sistemas (Arduino)

D.1 Sistema de la Vaca

```
/*
  Este programa es capaz de:
  1. Contabilizar los pasos de una vaca y la localiza.
  2. Almacenar la información en la Flash
  3. Envíar los datos bajo demanda de LoRa
*/

#include "HardwareSerial.h"
#include "Adafruit_GPS.h"
#include "MPU6050.h"
#include "Wire.h"
#include "SPIFFS.h"
#include "FS.h"
#include "stdlib.h"
#include "stdio.h"
#include "uart.h"
#include "time.h"
#include "heltec.h"


//Para RTC
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;
struct tm timeinfo;
/*
  Struct tm have this param
    tm_sec  int seconds after the minute  0-61*
    tm_min  int minutes after the hour  0-59
    tm_hour int hours since midnight  0-23
    tm_mday int day of the month  1-31
    tm_mon  int months since January  0-11
    tm_year int years since 1900
    tm_wday int days since Sunday 0-6
    tm_yday int days since January 1  0-365
    tm_isdst  int Daylight Saving Time flag
*/


//Variables que guardan el ID(MAC) en distintos formatos
uint64_t chipidint;
char chipidchar[12];
String chipidstring;


//Variables para LoRa
#define BAND  868E6     //Bandas 433E6,868E6,915E6
String outgoing;
String Broadcast = "FFFFFFFFFFFF";
String origenmsgLora;   //Origen del Mensaje Lora
String destmsgLora;     //Destino del Mensaje Lora
byte destination;
byte msgLft = 0;        //Mensajes que quedan de Lora
char bufferenvio[50];   //Buffer para el envio

const int bytesLoraPacket = 64; //El chip sx127x
String incoming;                //Buffer para leer lo que se recibe
String masterAdress;            //ID del maestro
```

```
String slaveAdress;             //ID del esclavo
int pot = 0;                    //Potencia del paquete recibido
bool recibido = false;          //Flag que denota si hay un dato recibido
bool recibiendoLora = false;    //Flag que denota si esta recibiendo

//Contadores
int i = 0;

#define FORMAT_SPIFFS_IF_FAILED true

// Nombres para los archivos de la Flash
const char* MPUFlashMod = "/MpuFlashMod.txt";
const char* MPUFlashZ = "/MpuFlashZ.txt";
const char* PasosFlash = "/MpuPasos.txt";
const char* GPSFlash = "/GpsFlash.txt";

//Buffers para la Flash
char buffin[50];
String buffout;
int salidaFlash;


//Variables para MPU6050
MPU6050 mpu;
#define TMUESTREOMPU 200        //tiempo cada cual se consulta el mpu
#define NUMLECTRAT 8            //numero de lecturas que se haran para hacer el tratamiento de datos
#define TGUARDARDATOS 150
/*Cada cuantos datos cogidos del mpu se guardan los datos tratados en la memoria FLASH
   Esto será la unidad de medida para luego subirlo y almacenarlo. "pasos / t"
   Tabla de la variable para TMUESTREOMPU 200

  t       |  TGUARDARDATOS
  ------------------------
  30 s    |       150
  1 min   |       300
  5 min   |      1500
  15 min  |      4500
  30 min  |      9000
  1 h     | .    18000
  24 h    | .  432000
*/

int contadorLecturasMPU = 1;  //contador de las lecturas que se hacen al mpu
int salidaMPU[NUMLECTRAT];
int bufferint[1000];
int ansVal = 0;
int pasos = 0;
int counter = 0;
bool primLec = true;
Vector dAcel;                 //Vectores del Acelerómetro
Vector Ceros;                 //Offset del Acelerómetro

//variables para el procesado de datos
bool pasoActual = false;
int minMod;
int maxMod;
int ansMod;
bool sube;
bool ansSube;


//Variable para GPS
#define SETUPGPS 60000 //teoricamente tarda 35 segundos en coger datos fiables. Pongamos 1 min
#define GPSSerial Serial2
#define MUESTREOGPS  (5 * 60000) * (1000 / TMUESTREOMPU)
/*
    Al tener solo un temporizador que nos incrementa el numero de lecturas del MPU,
    dependen las unidades del muestreo que se haga

    5 min/muestreo * 60000 ms/1min * (1000 ms/1s / TMUESTREOMPU ms)

    MUESTREOMPU (ms)   |   MUESTREOGPS (???????????)
    ---------------------------------------------
    25                 | 12000000
    50                 | .6000000
    200                |  1500000
    800                |   375000


*/

bool GPSencendido = false;
int fallosGPS = 0;
int startGPS = 0;
int counterGPS = 0;
```

```
Adafruit_GPS GPS(&GPSSerial);
float Lat, Lng, Alt;
int Ano, Mes, Dia, Hora , Min , Seg;


/***************Funciones del Lora*********************/
/*
  Función que fragmenta la carga si esta sobrepasa el tamaño máximo permitido

    Se calcula el numero de envios que hay que hacer
      Si hay que hacer mas de uno:
        Se diferencia entre numero entero de paquetes completos o el último incompleto:
          Asumimos 4 los bytes maximos de envio:

          Si es numero entero de paquetes completo, hay que hacer correción -1
          8 / 4 = 2 - 1 = 1
          es decir
          1º msg -> envio 4 -> msgleft = 1 -> quedan 4 por enviar
          4 = 4 por tanto entra en la siguiente condición
          2º msg -> envio 4 -> msgleft = 0;

          Si el último ira incompleto, el truncamiento de (int) ya lo corrije
          9 / 4 = 2
          es decir
          1º msg -> envio 4 -> msgleft = 2 -> quedan 5 por enviar
          5 / 4 = 1
          2º msg -> envio 4 -> msgleft = 1 -> queda 1 por enviar
          1 < 4 por tanto entra en la siguiente condición
          2º msg -> envio 1 -> msgleft = 0;
      Una vez enviado un mensaje hay que borra del buffer de envio la parte
      correspondiente y esperar un pequeño tiempo para que no se solapen
      mensajes y dejar al receptor interpretarlo.
    Cuando se envia el último mensaje se activa el flag correspondiente
*/
void sendLora(String paquete, String destination) {

  bool envioTerminado = false;
  int tiempoEnvio = 200;

  while (!envioTerminado) {

    if (paquete.length() > bytesLoraPacket) {
      if (!(paquete.length() % bytesLoraPacket))
        msgLft = (paquete.length() / bytesLoraPacket) - 1;
      else
        msgLft =  (paquete.length() / bytesLoraPacket);

      sendMessage(paquete.substring(0, bytesLoraPacket), destination);
      paquete.remove(0, bytesLoraPacket);

    } else {

      msgLft = 0;
      sendMessage(paquete, destination);
      envioTerminado = true;
    }
    delay(tiempoEnvio);
  }
}


void sendMessage(String outgoing, String destination)
{
  LoRa.beginPacket();                   // start packet
  LoRa.print(destination);              // add destination address
  LoRa.print(chipidstring);             // add sender address
  LoRa.write(msgLft);                   // add nº msg left
  LoRa.write(outgoing.length());        // add payload length
  LoRa.print(outgoing);                 // add payload
  LoRa.endPacket();                     // finish packet and send it

}

/*
  Función que detecta y lee si ha llegado un mensaje LoRa
  Se va decodificando en función de la trama Lora modificada que se usa

  Destino | Origen  | Mensajes restantes del paquete  | tamaño de la carga  | carga
  6 bytes | 6 bytes |             1 byte              | 1 byte              | 64 bytes

  6 bytes -> 12 caracteres hexadecimales

  Se comprueba si el tamaño que se recibe es el que se dice que se recibe
```

```
  Si se denota que hay mas mensajes se espera para rellenar el buffer de recepcion
*/

void onReceive(int packetSize)
{

  if (packetSize == 0) return;          // if there's no packet, return
  recibido = true;

  destmsgLora = "";
  for (i = 0; i < 12; i++) {
    destmsgLora += (char)LoRa.read();
  }

  origenmsgLora = "";
  for (i = 0; i < 12; i++) {
    origenmsgLora += (char)LoRa.read();
  }

  byte incomingMsgLft = LoRa.read();
  byte incomingLength = LoRa.read();

  while (LoRa.available())
  {
    incoming += (char)LoRa.read();
  }

  if (incomingLength != incoming.length())
  {
    String respuesta = "Error de tamaño en el mensaje LoRa";
    sprintf(bufferenvio, "%s", respuesta.c_str());
    publishFeedback();
    return;
  }

  if (incomingMsgLft != 0 ) {
    while (1) {
      onReceive(LoRa.parsePacket())
    }
  }

  //  Muestra por el Serial la info del msg recibido
  //  Serial.println("Message ID: " + String(incomingMsgLft));
  //  Serial.println("Message length: " + String(incomingLength));
  //  Serial.println("Origen: " + origenmsgLora);
  //  Serial.println("Destino: " + destmsgLora);
  //  Serial.println("Message: " + incoming);
  //  Serial.println("RSSI: " + String(LoRa.packetRssi()));
  //  Serial.println("Snr: " + String(LoRa.packetSnr()));
  //  Serial.println();

  pot = LoRa.packetRssi();
  dato = true;
}

/***************Funciones del SPIFFS*********************/

/*
  Muestra el directorio pedido, con el nivel de profundidad deseado
*/
void listDir(fs::FS &fs, const char * dirname, uint8_t levels) {
  Serial.printf("Listing directory: %s\r\n", dirname);

  File root = fs.open(dirname);
  if (!root) {
    Serial.println("- failed to open directory");
    return;
  }
  if (!root.isDirectory()) {
    Serial.println(" - not a directory");
    return;
  }

  File file = root.openNextFile();
  while (file) {
    if (file.isDirectory()) {
      Serial.print("  DIR : ");
      Serial.println(file.name());
      if (levels) {
        listDir(fs, file.name(), levels - 1);
      }
    } else {
      Serial.print("  FILE: ");
      Serial.print(file.name());
```

```
      Serial.print("\tSIZE: ");
      Serial.println(file.size());
    }
    file = root.openNextFile();
  }
}

/*
  Lee el archivo pedido, y almacena su valor en buffout
*/

void readFile(fs::FS &fs, const char * path) {
  Serial.printf("Reading file: %s\r\n", path);

  File file = fs.open(path);
  if (!file || file.isDirectory()) {
    Serial.println("- failed to open file for reading");
    return;
  }

  Serial.println("- read from file:");
  while (file.available()) {
    salidaFlash = file.read();
    buffout += char(salidaFlash);
  }
}

/*
  Crea y escribe el archivo mencionado.
  Pisa el anterior archivo si ya existia uno con ese nombre
*/

void writeFile(fs::FS &fs, const char * path, const char * message) {
  //Serial.printf("Writing file: %s\r\n", path);

  File file = fs.open(path, FILE_WRITE);
  if (!file) {
    Serial.println("- failed to open file for writing");
    return;
  }
  if (file.print(message)) {
    //  Serial.println("- file written");
  } else {
    Serial.println("- frite failed");
  }
}

/*
  Añade al archivo indicado, la cadena de texto que se introduzca
*/
void appendFile(fs::FS &fs, const char * path,/* const*/ char* message) {
  Serial.printf("Appending to file: %s\r\n", path);

  File file = fs.open(path, FILE_APPEND);
  if (!file) {
    Serial.println("- failed to open file for appending");
    return;
  }
  if (file.print(message)) {
    Serial.println("- message appended");
  } else {
    Serial.println("- append failed");
  }
}

/*
  Renombra un archivo
*/
void renameFile(fs::FS &fs, const char * path1, const char * path2) {
  Serial.printf("Renaming file %s to %s\r\n", path1, path2);
  if (fs.rename(path1, path2)) {
    Serial.println("- file renamed");
  } else {
    Serial.println("- rename failed");
  }
}


/*
  elimina un archivo
*/
void deleteFile(fs::FS &fs, const char * path) {
  Serial.printf("Deleting file: %s\r\n", path);
  if (fs.remove(path)) {
    Serial.println("- file deleted");
```

```
    } else {
      Serial.println("- delete failed");
401 }
}

/*
  Inicializa el sistema de almacenamiento flash file system a traves de SPI
406 Crea dos archivos, uno para almacenar los datos de cada sensor (GPS y MPU)
*/

void initFlash() {

411 if (!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)) {
    Serial.println("SPIFFS Mount Failed");
    return;
  }

416 writeFile(SPIFFS, PasosFlash, "PASOS");
  writeFile(SPIFFS, GPSFlash, "Lat Lng Alt Ano Mes Dia Hora Min Seg");
}

/***************Funciones del MPU6050*********************/
421
/*
  Inicializa el MPU
    Inicialmente iniciamos el módulo en:
      Gyr: 250 degrees per second
426     Accel: +-2g/s
     Si no consigue inicializarle se queda en el bucle, típicos problemas
     - Por fecto AD0=GND(0V) -> I2Cadrres= 0x68
     - Check SCL y SDA
    Después calibra el MPU
431 Por último lo establece en Cycle Mode
      MPU6050_Cycle_1_25  -> 1,25Hz
      MPU6050_Cycle_5     -> 5Hz
      MPU6050_Cycle_20    -> 20 Hz
      MPU6050_Cycle_40    -> 40 Hz
436
*/
void initMPU6050() {
  Serial.println("Initialize MPU6050");
  while (!mpu.begin(MPU6050_SCALE_250DPS, MPU6050_RANGE_2G))
441 {
    Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
    delay(500);
  }
  calibrar();
446 mpu.setCycleMode(MPU6050_Cycle_5);
}

/*
  Lee los datos del sensor y los normaliza en relación al offset almacenado
451 Genera también el módulo del vector.
*/


void readMPUData() {
456
  //Coge un dato de
  Vector Acel = mpu.readRawAccel();
  dAcel.XAxis = Ceros.XAxis - C2_2BN(Acel.XAxis);
  dAcel.YAxis = Ceros.YAxis - C2_2BN(Acel.YAxis);
461 dAcel.ZAxis = Ceros.ZAxis - C2_2BN(Acel.ZAxis);
  dAcel.XYZmod = sqrt(pow(dAcel.XAxis, 2) + pow(dAcel.YAxis, 2) + pow(dAcel.ZAxis, 2));
}

/*
466 Calibra el MPU y genera el valor de offset al rededor del cual
  se realizaran todas las medidas del MPU.



471 */

void calibrar() {

  // Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
476 int16_t ax, ay, az;

  //Variables usadas por el filtro pasa bajos
  long f_ax, f_ay, f_az;
  int p_ax, p_ay, p_az;
481
  uint8_t counter = 0;
```

```
    //Valor de los offsets
    int16_t ax_o, ay_o, az_o;

    //confirmación de los offset
    bool fox = false;
    bool foy = false;
    bool foz = false;

    // Leer los offset los offsets anteriores
    ax_o = mpu.getAccelOffsetX();
    ay_o = mpu.getAccelOffsetY();
    az_o = mpu.getAccelOffsetZ();

    while ((!fox) && (!foy) && (!fox)) {
      // Leer las aceleraciones y velocidades angulares
      Vector Acel = mpu.readRawAccel();

      // Filtrar las lecturas mientras no hayamos calibrado el eje
      if (!fox) {
        f_ax = f_ax - (f_ax >> 5) + Acel.XAxis;
        p_ax = f_ax >> 5;
      }

      if (!foy) {
        f_ay = f_ay - (f_ay >> 5) + Acel.YAxis;
        p_ay = f_ay >> 5;
      }

      if (!foz) {
        f_az = f_az - (f_az >> 5) + Acel.ZAxis;
        p_az = f_az >> 5;
      }

      //Cada 100 lecturas corregir el offset
      if (counter == 100) {

        if (!fox) {
          if (p_ax > 0) ax_o -= 1;
          else {
            ax_o += 1;
          }
        }

        if (!foy) {
          if (p_ay > 0) ay_o -= 1;
          else {
            ay_o += 1;
          }
        }

        if (!foz) {
          if (p_az - 16384 > 0) az_o--;
          else {
            az_o++;
          }
        }

        mpu.setAccelOffsetX(ax_o);
        mpu.setAccelOffsetY(ay_o);
        mpu.setAccelOffsetZ(az_o);

        mpu.readRawAccel();

        if ((NormX < 0,001) && (NormX > -0,004))
          fox = true;

        if ((NormY < 0,001) && (NormY > -0,004))
          foy = true;

        if ((NormZ < 9,85) && (NormZ > 9,75))
          foz = true;

        counter = 0;
        //delay(1);
      }
      counter++;
    }

    Vector Acel = mpu.readRawAccel();

    Ceros.XAxis = C2_2BN(Acel.XAxis);
    Ceros.YAxis = C2_2BN(Acel.YAxis);
    Ceros.ZAxis = C2_2BN(Acel.ZAxis);
}
```

```
571 /***************Funciones del GPS*********************/
    /*
      Inicia el GPS
        Se controla a través del GPIO25.
        Una vez se baja el pin, el transistor pasa a conducir
576     y se alimenta.

    */

    void initGPS() {
581
      pinMode(25, OUTPUT);
      digitalWrite(25, LOW);

      GPS.begin(9600);
586   //uart_set_pin(UART_NUM_2, 17, 16,18,19);
      delay(1000);
    }


591 /*
        Convierte la información de 2 bytes en complemento a dos
        a un unidades de aceleración (g) en formato float.
    */

596 float C2_2BN(int in) {
      if (in > 32767)
        return ((-((in ^ 0xFFFF) + 1)) * (9.81 / 16384.0));
      return (in * (9.81 / 16384.0));
    }
601
    /*
      Introduce un número flotante en un buffer modo char*
    */

606 void conversorFloattoBuff(float flotante) {

      char signo = 1;
      //si es negativo le meto el modulo y lo guardo en el signo
      if (flotante < 0) {
611     flotante = flotante - (flotante * 2);
        signo = '-';

      }

616   int parteDecimal;
      int parteEntera = (int)(flotante);

      if (!parteEntera)
        parteDecimal = (int)((flotante) * 100);
621   else
        parteDecimal = (int)((flotante - parteEntera) * 100);

      if (0 == (int)(parteDecimal / 10)) {
        if (signo)
626       sprintf(buffin, "%d.0%d " , parteEntera, parteDecimal );
        sprintf(buffin, "%c%d.0%d " , signo, parteEntera, parteDecimal );
      } else {
        if (signo)
          sprintf(buffin, "%d.%d " , parteEntera, parteDecimal );
631     sprintf(buffin, "%c%d.%d " , signo, parteEntera, parteDecimal );
      }

    }

636


    /*
        Esta función nos consigue el ID o MAC del chip, que será a su vez el identificador del animal.
        Nos devuelve dos variables globales:
641       chipidint  -> variable en formato hexadecimal en uint16_t
          chipidchar -> variable en formato hexadecimal en char*, mas sencillo para ser tratado
    */

    void getChipID() {
646   chipidint = ESP.getEfuseMac();
      sprintf(chipidchar, "%04X%08X", (uint16_t)(chipidint >> 32), (uint32_t)chipidint);
      chipidstring = String(chipidchar);
      delay(2000);
      Serial.println(chipidstring);
651 }
```

```
/*
  Primera funcion del programa:

  Inicializa el sistema heltec, con la pantalla desactivada, y LoRa y Serial activados

  Adquirimos el ID del dispositivo, que usaremos para su identificación en LoRa.

  Llamamos a las funciones que inicializan el MPU6050 y la Flash.
  (El GPS se activa bajo demanda)

  Se configura el Lora para que emita:
      -a max potencia.
      -con factor de Spread 9
      -con ancho de banda 250KHz

*/


void setup() {


  Heltec.begin(false /*DisplayEnable Disable*/, true /*Heltec.LoRa Enable*/, true /*Serial Enable*/, true
      /*PABOOST Enable*/, BAND /*long BAND*/);
  delay(250);

  getChipID();
  delay(250);

  initMPU6050();
  delay(250);

  //  initGPS();
  //  delay(250);

  initFlash();
  delay(250);

  LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setSpreadingFactor(9);
  LoRa.setSignalBandwidth(250E3);
  delay(250);
}

void loop() {

  // parse for a packet, and call onReceive with the result:
  onReceive(LoRa.parsePacket());
  if (recibido) {
    if ((!destmsgLora.compareTo(Broadcast)) && (!incoming.compareTo("GetData")))
    {

      readFile(SPIFFS, "/MpuPasos.txt");
      buffout.remove(0, 5);
      buffout += "fin";

      delay(1000);
      sendLora(buffout, origenmsgLora);
      buffout = "";
      deleteFile(SPIFFS, "/MpuPasos.txt");

      writeFile(SPIFFS, PasosFlash, "PASOS");
    }
    incoming = "";
  }

  /*
  Temporizador para el muestreo del MPU6050 y GPS
    TMUESTREOMPU -> variable definida como el tiempo necesario para pedir datos (ver define para más
        detalles)
    readMPUData -> Recoge los datos del Sensor Inercial
      A continuación trata los datos directamente sin almacenalos.
      Les almacena en función de la unidad de medida que deseemos.
      Es decir, si queremos mas precisión: pasos/min; para menos pasos/h
      esto lo regularemos con TGUARDARDATOS (ver define para más detalles)
      El tratamiento que se realiza es el siguiente:

    1   Fija todas las variables al primer valor
    2   Se guarda el flag de subida.
        Se compara Si el modulo anterior es mayor que el actual
          Lo es A3 no lo es B3
    A3  Se fija el flag en subida y se actaliza el máximo
    B3  En este caso se fijan los máximos y los mínimos al valor actual.
        Se compara si antes subia y ahora baja.
        Si se da ese caso se compara para ver si la diferencia entre
```

```
        el mínimo y el máximo supera un margen para que se pueda
        considerar un paso
    */
741 if (!(millis() % TMUESTREOMPU))
    {

      readMPUData();
      if (primLec) {
746
        maxMod = dAcel.XYZmod;
        minMod = dAcel.XYZmod;
        ansMod = dAcel.XYZmod;
        sube = false;
751     primLec = false;
        Serial.println("Primera lectura");

      } else {
        ansSube = sube;
756     if (ansMod <= dAcel.XYZmod) {
          sube = true;
        } else {
          sube = false;
        }
761
        ansMod = dAcel.XYZmod;
        if (sube) {
          maxMod = dAcel.XYZmod;

766     } else {
          if (sube != ansSube) {
            if (maxMod > (1.5 * minMod)) {
              pasos++;
              Serial.println("PasoPillado");
771         }
          }
          maxMod = dAcel.XYZmod;
          minMod = dAcel.XYZmod;
        }
776   }
      contadorLecturasMPU++;
    }

    /*
781   Cada TGUARDARDATOS ms se guarda los datos tratados en el archivo
      ver define para mas información.
      Se resetea el contador de pasos actual
    */
    if (!(int)(contadorLecturasMPU % TGUARDARDATOS))
786 {
      sprintf(buffin, "%d " , pasos);
      appendFile(SPIFFS, PasosFlash, buffin);
      pasos = 0;
      contadorLecturasMPU++;
791 }

    /*
      Entra a activar el GPS cuando se cumplen las siguientes condiciones:
        1. Resto lecturas entre muestreo del GPS es 0. (Se temporiza a partir de MUESTREOGPS)
796     2. Si el GPS no esta encendido todavía.
        3. Si el GPS no ha sobrepasado el número de fallos.

      Una vez dentro activa el GPS, marca el flag de GPSencendido y guarda en que instante se ha
      iniciado el GPS (en startGPS).
801
    */

    if ( (!(contadorLecturasMPU % MUESTREOGPS)) && (!GPSencendido) && (fallosGPS < 3)) {

806   Serial.print("Lecturas MPU: ");
      Serial.println(contadorLecturasMPU);
      Serial.print("Muestreo GPS: ");
      Serial.println(MUESTREOGPS);
      Serial.print("Resto de contadorLecturasMPU % MUESTREOGPS ");
811   Serial.println(contadorLecturasMPU % MUESTREOGPS);


      GPSencendido = true;
      initGPS();
816   startGPS = millis();

    }

    /*
821   Entra a muestrear el GPS si se cumplen las siguientes condiciones:
```

```
        1. El flag muestra que el GPS esta encendido
        2. El tiempo de set up del GPS ha sido superado. (Programable en SETUPGPS)

      Una vez dentro comprueba que se detecten satélites:

        Si NO se consiguen, se escribe en FLASH que no se ha conseguido
        las efemerides y se incrementa el contador de fallos del GPS (fallosGPS)

        Si SÍ se consiguen, se adquieren todos los datos que proporciona
        y los almacena en FLASH.

      Por último, se desactiva el GPS y se desmarca el flag de encendido.

    */

    if (GPSencendido && ((startGPS + SETUPGPS) < millis())) {
      if (!(int)GPS.satellites) {
        Serial.println("No se encuentran efemerides");
        appendFile(SPIFFS, GPSFlash, "No se encuentran efemerides");
        appendFile(SPIFFS, GPSFlash, "\n");
        fallosGPS++;
      } else {
        Lat = GPS.latitude;
        Lng = GPS.longitude;
        Alt = GPS.altitude;
        Ano = GPS.year;
        Mes = GPS.month;
        Dia = GPS.day;
        Hora = GPS.hour;
        Min = GPS.minute;
        Seg = GPS.seconds;

        appendFile(SPIFFS, GPSFlash, "\n");
        conversorFloattoBuff(Lat);
        appendFile(SPIFFS, GPSFlash, buffin);
        conversorFloattoBuff(Lng);
        appendFile(SPIFFS, GPSFlash, buffin);
        conversorFloattoBuff(Alt);
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Ano );
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Mes );
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Dia );
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Hora );
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Min );
        appendFile(SPIFFS, GPSFlash, buffin);
        sprintf(buffin, "%d " , Seg );
        appendFile(SPIFFS, GPSFlash, buffin);
        counterGPS = 0;
      }
      counterGPS++;
      GPSencendido = false;
      digitalWrite(25, HIGH);
      Serial.println("Apago el GPS");
    }
  }
```

D.2 Sistema del Ganadero

```
/*
    Este programa es capaz de:
    1. Lanzar la app html para captar los datos de la WiFi
       conectada a Internet.
    2. Vincularse a un dashboard de io.adafruit siendo capaz de:
       2a. Pedir datos por LoRa
       2b. Enviar los datos vía MQQT al servidor
       2c. Enviar retroalimentaciones del sistema
*/

//Te dice por el serial el ip al que conectarte, 192.168.4.1
// Load Wi-Fi library
#include "WiFi.h"
#include "WiFiAP.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "time.h"
#include "heltec.h"
#include "stdlib.h"
#include "stdio.h"


//Variables que guardan el ID(MAC) en distintos formatos
uint64_t chipidint;
char chipidchar[12];
String chipidstring;

//Variables para LoRa
#define BAND  868E6      //Bandas 433E6,868E6,915E6
String outgoing;
String Broadcast = "FFFFFFFFFFFF";
String origenmsgLora;    //Origen del Mensaje Lora
String destmsgLora;      //Destino del Mensaje Lora
byte destination;
byte msgLft = 0;         //Mensajes que quedan de Lora
char bufferenvio[50];    //Buffer para el envio


const int bytesLoraPacket = 64; //El chip sx127x
String incoming;                //Buffer para leer lo que se recibe
String masterAdress;            //ID del maestro
String slaveAdress;             //ID del esclavo
int pot = 0;                    //Potencia del paquete recibido
bool recibido = false;          //Flag que denota si hay un dato recibido
bool recibiendoLora = false;    //Flag que denota si esta recibiendo

int steps[500];          //Buffer para guardar los pasos recibidos


//Variables para punto de acceso y WiFi

//Datos del punto acceso del ESP32
const char* ssid     = "ESP32Ganadero";
const char* password = "TFGVictorEspeso";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP
String htmlpayload =
  "<!DOCTYPE␣html>\n"
  "<html>\n"
  "<head>\n"
  "<meta␣http-equiv=\"Content-Type\"␣content=\"text/html;␣charset=utf-8\"/><meta␣name=\"author\"␣content
      =\"Víctor␣Espeso␣Aparicio\">\n"
  "<meta␣name=\"description\"␣content=\"Instrucciones␣y␣formulario␣para␣meter␣los␣datos␣de␣la␣WLAN␣a␣la␣
      que␣nos␣vamos␣a␣conectar\">␣␣\n"
  "<title>APP␣TFG␣Víctor␣Espeso</title>\n"
  "</head>\n"
  "<h1><font␣size=\"7\"␣FACE=\"times␣new␣roman\">APP␣TFG␣Víctor␣Espeso</font></h1>\n"
  "<form␣action=\"/http://mine.es\"␣method=\"get\">\n"
  "<h2><font␣size=\"5\"␣FACE=\"times␣new␣roman\">1º␣Paso</font></h1>\n"
  "<p>Rellene␣los␣apartados␣siguientes␣con␣el␣SSID␣(nombre)␣del␣WiFi␣y␣el␣Password␣(contraseña)<br></p>\n"
  "␣␣␣␣<div>\n"
  "␣␣␣␣␣␣␣␣<label␣for=\"SSIDin\">SSID:</label>\n"
  "␣␣␣␣␣␣␣␣<input␣name=\"SSIDin\"␣type=\"text\"␣id=\"SSIDin\"␣placeholder=\"Escriba␣la␣SSID\"␣/>\n"
  "␣␣␣␣</div>\n"
  "␣␣␣␣<div>\n"
  "␣␣␣␣␣␣␣␣<label␣for=\"PASSin\">Password:</label>\n"
```

```
    "␣␣␣␣␣␣␣␣<input␣name=\"PASSin\"␣type=\"password\"␣id=\"PASSin\"␣placeholder=\"Escriba␣la␣contraseña\"␣
        />\n"
    "␣␣␣␣</div>\n"
    "␣␣␣␣<div>\n"
    "␣␣␣␣␣␣␣␣␣␣<input␣type=\"hidden\"␣name=\"Dummy\"␣value=\"Dummy\">␣␣␣␣</div>\n"
    "<h2><font␣size=\"5\"␣FACE=\"times␣new␣roman\">2º␣Paso</font></h1>\n"
    "<p>Copie␣el␣siguiente␣link␣antes␣de␣dar␣al␣botón␣o␣abralo␣en␣una␣nueva␣pestaña.\n"
    "<br><br>Una␣vez␣de␣al␣botón,␣usted␣deberá␣poner␣este␣link␣en␣el␣navegador␣para␣acceder␣al␣control␣de␣
        los␣datos␣subidos␣a␣la␣nube<br></p>\n"
    "<a␣href=\"https://io.adafruit.com/VictorEspeso/dashboards/configuracion\">https://io.adafruit.com/
        VictorEspeso/dashboards/configuracion</a>\n"
    "<h2><font␣size=\"5\"␣FACE=\"times␣new␣roman\">3º␣Paso</font></h1>\n"
    "<p>\n"
    "<br>Pulse␣el␣botón␣para␣conectarse␣a␣la␣red␣con␣los␣datos␣introducidos<br></p>\n"
    "<!--␣<a␣href=\"https://io.adafruit.com/VictorEspeso/dashboards/configuracion\"><input␣type=\"submit\"␣
        value=\"volver\"></a>␣-->\n"
    "␣␣␣␣<div␣class=\"button\">\n"
    "␣␣␣␣␣␣␣␣<button␣type=\"submit\"␣>Enviar␣información␣</button>\n"
    "␣␣␣␣</div>\n"
    "</html>\n"
    "<h2><font␣size=\"5\"␣FACE=\"times␣new␣roman\">SI␣EN␣10␣SEGUNDOS␣NO␣SE␣HA␣CONECTADO,␣VOLVER␣A␣EMPEZAR</
        font></h1>\n"
    "";

//Cabecera html
String header;

//Indices para leer el request
int indice1;
int indice2;
int indice3;

//Para almacenar los datos de entrada
String inSSID;
String inPASS;

//Flags
bool okSSID = false;
bool okPASS = false;
bool primerIteracion = true;


// Adafruit.io Setup

#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "VictorEspeso"
#define AIO_KEY         "96019f46443e402b926a49152335e242"


WiFiClient client;

// Configura MQTT
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

//Feeds
// Notifica los feeds y los inicializa
Adafruit_MQTT_Publish pasosfeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Pasos");
Adafruit_MQTT_Publish Feedbackfeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Feedback");
Adafruit_MQTT_Publish RSSIfeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/rssi");

// Inicializa las subscripciones
Adafruit_MQTT_Subscribe onoffbutton = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/PedirLora");


//Para RTC
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;
struct tm timeinfo;
/*
  Struct tm have this param
    tm_sec  int seconds after the minute  0-61*
    tm_min  int minutes after the hour  0-59
    tm_hour int hours since midnight  0-23
    tm_mday int day of the month  1-31
    tm_mon  int months since January  0-11
    tm_year int years since 1900
    tm_wday int days since Sunday 0-6
    tm_yday int days since January 1  0-365
    tm_isdst  int Daylight Saving Time flag
*/

bool hayFecha = true;
```

```
/*
  Función que fragmenta la carga si esta sobrepasa el tamaño máximo permitido

    Se calcula el numero de envios que hay que hacer
      Si hay que hacer mas de uno:
        Se diferencia entre numero entero de paquetes completos o el último incompleto:
          Asumimos 4 los bytes maximos de envio:

          Si es numero entero de paquetes completo, hay que hacer correción -1
          8 / 4 = 2 - 1 = 1
          es decir
          1º msg -> envio 4 -> msgleft = 1 -> quedan 4 por enviar
          4 = 4 por tanto entra en la siguiente condición
          2º msg -> envio 4 -> msgleft = 0;

          Si el último ira incompleto, el truncamiento de (int) ya lo corrije
          9 / 4 = 2
          es decir
          1º msg -> envio 4 -> msgleft = 2 -> quedan 5 por enviar
          5 / 4 = 1
          2º msg -> envio 4 -> msgleft = 1 -> queda 1 por enviar
          1 < 4 por tanto entra en la siguiente condición
          2º msg -> envio 1 -> msgleft = 0;
      Una vez enviado un mensaje hay que borra del buffer de envio la parte
      correspondiente y esperar un pequeño tiempo para que no se solapen
      mensajes y dejar al receptor interpretarlo.
    Cuando se envia el último mensaje se activa el flag correspondiente
*/

void sendLora(String paquete, String destination) {

  bool envioTerminado = false;
  int tiempoEnvio = 200;

  while (!envioTerminado) {

    if (paquete.length() > bytesLoraPacket) {
      if (!(paquete.length() % bytesLoraPacket))
        msgLft = (paquete.length() / bytesLoraPacket) - 1;
      else
        msgLft =  (paquete.length() / bytesLoraPacket);

      sendMessage(paquete.substring(0, bytesLoraPacket), destination);
      paquete.remove(0, bytesLoraPacket);

    } else {

      msgLft = 0;
      sendMessage(paquete, destination);
      envioTerminado = true;
    }
    delay(tiempoEnvio);
  }
}


void sendMessage(String outgoing, String destination)
{
  LoRa.beginPacket();                   // start packet
  LoRa.print(destination);              // add destination address
  LoRa.print(chipidstring);             // add sender address
  LoRa.write(msgLft);                   // add nº msg left
  LoRa.write(outgoing.length());        // add payload length
  LoRa.print(outgoing);                 // add payload
  LoRa.endPacket();                     // finish packet and send it

}

/*
  Función que detecta y lee si ha llegado un mensaje LoRa
  Se va decodificando en función de la trama Lora modificada que se usa

  Destino | Origen  | Mensajes restantes del paquete  | tamaño de la carga  | carga
  6 bytes | 6 bytes |             1 byte              | 1 byte              | 64 bytes

  6 bytes -> 12 caracteres hexadecimales

  Se comprueba si el tamaño que se recibe es el que se dice que se recibe

  Si se denota que hay mas mensajes se espera para rellenar el buffer de recepcion
*/

void onReceive(int packetSize)
{
```

```
  if (packetSize == 0) return;          // if there's no packet, return
  recibido = true;

  destmsgLora = "";
  for (i = 0; i < 12; i++) {
    destmsgLora += (char)LoRa.read();
  }

  origenmsgLora = "";
  for (i = 0; i < 12; i++) {
    origenmsgLora += (char)LoRa.read();
  }

  byte incomingMsgLft = LoRa.read();
  byte incomingLength = LoRa.read();

  while (LoRa.available())
  {
    incoming += (char)LoRa.read();
  }

  if (incomingLength != incoming.length())
  {
    String respuesta = "Error de tamaño en el mensaje LoRa";
    sprintf(bufferenvio, "%s", respuesta.c_str());
    publishFeedback();
    return;
  }

  if (incomingMsgLft != 0 ) {
    while (1) {
      onReceive(LoRa.parsePacket())
    }
  }

  //  Muestra por el Serial la info del msg recibido
  //  Serial.println("Message ID: " + String(incomingMsgLft));
  //  Serial.println("Message length: " + String(incomingLength));
  //  Serial.println("Origen: " + origenmsgLora);
  //  Serial.println("Destino: " + destmsgLora);
  //  Serial.println("Message: " + incoming);
  //  Serial.println("RSSI: " + String(LoRa.packetRssi()));
  //  Serial.println("Snr: " + String(LoRa.packetSnr()));
  //  Serial.println();

  pot = LoRa.packetRssi();
  dato = true;
}
/*
  Función que pone el modo punto de acceso y lanza la APP al movil una vez conectado
  Básicamente lanza una un form request y cuando se rellena el formulario
  se lee el codigo de retorno para detectar los datos introducidos por el usuario
*/
void getWLANsettings() {

  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  //Serial.print("AP IP address: ");
  //Serial.println(IP);
  server.begin();

  while (!(okSSID && okPASS)) {

    WiFiClient client = server.available();   // Listen for incoming clients

    if (client) {                             // If a new client connects,
      Serial.println("New Client.");          // print a message out in the serial port
      String currentLine = "";                // make a String to hold incoming data from the client
      while (client.connected()) {            // loop while the client's connected
        if (client.available()) {             // if there's bytes to read from the client,
          char c = client.read();             // read a byte, then
          Serial.write(c);                    // print it out the serial monitor
          header += c;
          if (c == '\n') {                    // if the byte is a newline character
            // if the current line is blank, you got two newline characters in a row.
            // that's the end of the client HTTP request, so send a response:
            if (currentLine.length() == 0) {
              // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
              // and a content-type so the client knows what's coming, then a blank line:
              client.println("HTTP/1.1 200 OK");
              client.println("Content-type:text/html");
              client.println("Connection: close");
              client.println();
```

```
                indice1 = header.indexOf("SSIDin=");
330             indice2 = header.indexOf("PASSin=");
                indice3 = header.indexOf("Dummy=");
                if ((indice2 - 8) > indice1) {
                  Serial.println("SSID encontrada:");
                  inSSID = header.substring(indice1 + 7, indice2 - 1);
335               Serial.println(inSSID);
                  okSSID = true;
                } else {
                  Serial.println("No se encuentra SSID");
                  okSSID = false;
340               okPASS = false;
                }

                if ((indice3 - 8) > indice2) {
                  Serial.println("PASS encontrada:");
                  inPASS = header.substring(indice2 + 7, indice3 - 1);
345               Serial.println(inPASS);
                  okPASS = true;
                } else {
                  Serial.println("No se encuentra PASS");
                  okSSID = false;
350               okPASS = false;
                }

                client.println(htmlpayload);
355
                // The HTTP response ends with another blank line
                client.println();
                // Break out of the while loop
                break;
360           } else { // if you got a newline, then clear currentLine
                currentLine = "";
              }
            } else if (c != '\r') {  // if you got anything else but a carriage return character,
              currentLine += c;      // add it to the end of the currentLine
365         }
          }
        }
        // Clear the header variable
        header = "";
370     // Close the connection
        client.stop();
        Serial.println("Client disconnected.");
        Serial.println("");
      }
375   }
}

/*
  Función para conectarse a un WiFi en modo estación.
380 Lo intentara durante 20 segundos aproximadamente sino reinicia el ESP32
  para volver a lanzar el cuestionario
*/


385 void conectWifi() {

  int contador = 0;
  //Serial.print("Connecting to ");
  //Serial.println(inSSID);
390
  WiFi.begin(inSSID.c_str(), inPASS.c_str());
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    // Serial.print(".");
395 contador++;//si no encuentra esque esta mal y se resetea
    if (contador == 40)
      ESP.restart();
  }
  //Serial.println();
400
  //Serial.println("WiFi connected");
  //Serial.println("IP address: "); Serial.println(WiFi.localIP());
}
/*
405 Función para tener conectado mediante el protocolo MQTT
  el ESP32 al servidor de Adafruit
*/

void MQTT_connect() {
410 int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
```

```
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 5;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
    Serial.println(mqtt.connectErrorString(ret));
    Serial.println("Retrying MQTT connection in 5 seconds...");
    mqtt.disconnect();
    delay(5000);  // wait 5 seconds
    retries--;
    if (retries == 0) {
      // basically die and wait for WDT to reset me
      while (1);
    }
  }
  Serial.println("MQTT Connected!");
}

/*
  Mantiene las subscripciones activas
*/

void setupMQTT() {
  // Setup MQTT subscription for onoff feed.
  mqtt.subscribe(&onoffbutton);
}

/*
    Esta función nos consigue el ID o MAC del chip, que será a su vez el identificador del animal.
    Nos devuelve dos variables globales:
      chipidint     -> variable en formato hexadecimal en uint16_t
      chipidchar    -> variable en formato hexadecimal en char*, mas sencillo para ser tratado
      chipidstring  -> variable en formato hexadecimal en String
*/

void getChipID() {

  chipidint = ESP.getEfuseMac(); //The chip ID is essentially its MAC address(length: 6 bytes).
  sprintf(chipidchar, "%04X%08X", (uint16_t)(chipidint >> 32), (uint32_t)chipidint);
  chipidstring = String(chipidchar);
}

/*
  Funcion generica que publica en el feed de retroalimentaciones
  la informacion que este en el buffer de envio
*/
void publishFeedback() {
  if (! Feedbackfeed.publish(bufferenvio)) {
    Serial.println(F("Failed"));
  } else {
    Serial.println(F("OK!"));
  }
}

/*
  Primera función del programa
  1º Inicializa el ESP32 y LoRa
  2º Lee el ID del chip
  3º Lanza el formulario en modo punto de acceso
  4º Intenta conectarse a los datos obtenidos
    4.2 -> Se resetea y volvemos a 1º
  5º Se conecta a MQTT
  6º Lee la hora y la almacena
  7º Fija los valores de LoRa
*/
void setup() {
  Heltec.begin(false /*DisplayEnable Enable*/, true /*Heltec.LoRa Enable*/, true /*Serial Enable*/, true
      /*PABOOST Enable*/, BAND /*long BAND*/);
  delay(250);

  getChipID();
  delay(250);

  getWLANsettings();
  delay(250);

  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
  delay(3000);

  conectWifi();

  setupMQTT();
```

```
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  while ((!getLocalTime(&timeinfo)) || 3 == contador ) {
    Serial.println("Failed to obtain time");
    delay(2000);
    contador++;
    if (contador == 3)
      hayFecha = false;
  }

  LoRa.setTxPower(25, PA_OUTPUT_PA_BOOST_PIN);
  LoRa.setSpreadingFactor(9);
  LoRa.setSignalBandwidth(250E3);
}

/*
  Bucle principal del programa

  En la primera iteración Se envia un feedback de que se ha conectado al WiFi

  Se esta leyendo constantemente la subscripción
    Se mide el tamaño de el feed.
      Si es 3 pedirLora es igual a OFF no se hace nada
      Si es 2 pedirLora es igual a ON se inicia toda la secuencia de comunicación:

  1º  Se envia una trama pidiendo datos a quien lo reciba
      Se espera 5 segundos, si no se recibe nada se vuelve a empezar (1º)
  2º  Se van demodulando los pasos de string a un vector de int
      hasta que se detecta un codigo de fin
  3º  Una vez almacenado todos los pasos, se envian por MQTT
      los datos de mas reciente a mas antiguo enviando
      una retroalimentación cada 12 datos (1 hora)




*/

void loop() {

  MQTT_connect();
  Adafruit_MQTT_Subscribe *subscription;

  if (primerIteracion) {
    sprintf(bufferenvio, "Conectado a %s", inSSID.c_str());
    publishFeedback();
    primerIteracion = false;
  }

  while (subscription = mqtt.readSubscription(50)) {
    if (subscription == &onoffbutton) {
      Serial.print(("Got: "));
      Serial.println((char*)onoffbutton.lastread);
      //sabemos que si el tamaño de los datos es 3 es un OFF y si el tamaño de los datos es 2 es un ON

      while (2 == onoffbutton.datalen) {

        if (!(millis() % 5000)) {

          sendLora("GetData", Broadcast);
          int  setTime1 = millis();
          Serial.println("Esperando la respuesta");
          String respuesta = "Esperando Respuesta";
          sprintf(bufferenvio, "%s", respuesta.c_str());
          publishFeedback();

          while ( setTime1 + 5000 > millis()) {
            onReceive(LoRa.parsePacket());
            if (recibido) {
              i = 0;
              bool completo = false;
              while (!completo) {

                if (incoming.compareTo("fin") != 0) {
                  if (0 == incoming.indexOf("0")) {
                    steps[i] = 0;
                    incoming.remove(0, 2);

                  } else {
                    steps[i] = incoming.toInt();
                    if (steps[i] < 10) {
                      incoming.remove(0, 2);

                    } else if (steps[i] < 100) {
```

```
                    incoming.remove(0, 3);
585                 } else {
                      incoming.remove(0, 4);
                    }
                  }
                  i++;
590
                } else {
                  if (i == 0)
                    steps[0] = 0;
                  completo = true;
595               }
              }

              int contador = 0;
              int dia = timeinfo.tm_mday;
600           int mes = timeinfo.tm_mon;
              int anio = timeinfo.tm_year;
              int hora = timeinfo.tm_hour;
              int minuto = timeinfo.tm_min;
              int segundo = timeinfo.tm_sec;
605
              while (i != 0) {
                i--;
                if (((!(contador % 12)) || (contador == 0)) && hayFecha) {
                  sprintf(bufferenvio, "␣%d/%d/%d␣%d:%d:%d", dia, mes, anio, hora, minuto, segundo);
610               publishFeedback();
                  if (hora == 0) {
                    dia--;
                    hora = 23;
                  } else {
615                 hora--;
                  }
                  delay(3000);
                }
                contador++;
620
                if (! pasosfeed.publish(steps[i])) {
                  Serial.println(F("Failed"));
                } else {
                  Serial.println(F("OK!"));
625               }
                delay(2700);
              }

              String respuesta = "Fin␣del␣mensaje␣LoRa";
630           sprintf(bufferenvio, "%s", respuesta.c_str());
              publishFeedback();

              recibido = false;
              incoming = "";
635           subscription = mqtt.readSubscription(50);
            }
          }
        }
      }
640 }
  }
}
```

Apéndice E

Código de la aplicación móvil

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/><meta name="author" content="Víctor Espeso Aparicio">
<meta name="description" content="Instrucciones y formulario para meter los datos de la WLAN a la que nos vamos a conectar">
<title>APP TFG Víctor Espeso</title>
</head>
<h1><font size="7" FACE="times new roman">APP TFG Víctor Espeso</font></h1>
<form action="/http://mine.es" method="get">
<p>Esta app sirve para introducir los datos para poder conectarse el ESP32 maestro a una red WiFi.
<br>Desde la red WiFi que nos conectemos, el ESP32 enviará a través de MQTT a la nube.<br></p>
<h2><font size="5" FACE="times new roman">1º Paso</font></h1>
<p>Rellene los apartados siguientes con el SSID (nombre) del WiFi y el Password (contraseña)<br></p>
    <div>
        <label for="SSIDin">SSID:</label>
        <input name="SSIDin" type="text" id="SSIDin" placeholder="Escriba la SSID" />
    </div>
    <div>
        <label for="PASSin">Password:</label>
        <input name="PASSin" type="password" id="PASSin" placeholder="Escriba la contraseña" />
    </div>
    <div>
        <input type="hidden" name="Dummy" value="Dummy">    </div>
<h2><font size="5" FACE="times new roman">2º Paso</font></h1>
<p>Copie el siguiente link antes de dar al boton o abralo en una nueva pestaña.
<br>Una vez de al botón, usted deberá poner este link en el navegador para acceder al control de los datos MQTT<br></p>
<a href="https://io.adafruit.com/VictorEspeso/dashboards/configuracion">https://io.adafruit.com/VictorEspeso/dashboards/configuracion</a>
<h2><font size="5" FACE="times new roman">3º Paso</font></h1>
<p>
<br>Pulse el boton para conectarse a la red con los datos introducidos<br></p>
<!-- <a href="https://io.adafruit.com/VictorEspeso/dashboards/configuracion"><input type="submit" value="volver"></a> -->
    <div class="button">
        <button type="submit" >Enviar información </button>
    </div>
</html>
<h2><font size="5" FACE="times new roman">SI EN 10 SEGUNDOS NO SE HA CONECTADO, VOLVER A EMPEZAR</font></h1>
```