Universidad Zaragoza
1542

Trabajo Fin de Grado

Collar con localizador GPS para mascotas

Pet GPS tracker collar

Autor/es

Alfonso Forcén Domínguez

Director/es

Roberto Casas

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Universidad de Zaragoza

2020

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS

ÍNDICE DE ESQUEMÁTICOS Y PCBs

ÍNDICE DE TABLAS

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Alfonso Forcén Domínguez ,en

aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster)

Grado (Título del Trabajo)

Collar con localizador GPS para mascotas

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 27 de Febrero de 2020

Fdo: Alfonso Forcén Domínguez

Collar con localizador GPS para mascotas

RESUMEN

En este trabajo fin de grado se ha desarrollado e implementado una pareja de módulos para localizar de manera remota la posición de una mascota, enfocado a su aplicación en fines particulares. Para ello se han acoplado una serie de sensores, LEDs y módulos de comunicación a un microprocesador que envía, manipula, y recibe los datos de los diferentes componentes.

El proyecto podría enmarcarse como un trabajo de IoT (internet of things), cuyo cometido es la utilización de la tecnología en el ambiente cotidiano, con el fin de evitar un problema concreto, evitar perder a nuestra mascota. En resumen, hacer un gadget útil, que necesite pocos recursos, y fácil de utilizar.

El proyecto necesita de dos módulos, de los cuáles uno irá unido al collar del animal, y el otro deberá quedarse a una “distancia bluetooth” del teléfono del usuario. Ambos se han implementado sobre tecnología Arduino, usando el microprocesador Atmega320p y la placa de prototipado Arduino UNO. Para el envío y recepción de datos entre las partes se ha utilizado tecnología radio, concretamente dos módulos emisor-receptor LoRa XL1278-SMT, de bajo coste y consumo. Para la detección de coordenadas se ha utilizado un módulo GPS NEO-6M-0-001 conectado a su antena por defecto. Finalmente, para la comunicación bluetooth mediante módulo y teléfono Android se ha utilizado un módulo bluetooth HM-05, monitorizado por una aplicación creada a través de la plataforma MIT App Inventor 2.

Además, como último objetivo del trabajo, se planea diseñar un esquemático y una placa de circuito impreso para cada uno de los módulos, que catapultará el proyecto a una versión optimizada y comercial para el collar.

De este modo, en este Trabajo fin de Grado se han desarrollado dos módulos interconectados entre ellos, que manipularán los datos y medidas de los diferentes componentes, hasta enviarlos al teléfono. Una vez los reciba, este los representará de manera visual.

Pet GPS tracker collar

ABSTRACT

In this bachelor's dissertation, a pet GPS tracker collar has been developed with its application aimed to personal and daily purposes. The collar, supported by an Android app, is been designed to allow the user to know where his/her pet is in every moment. It required the use of LED lights, sensors and communication modules to a microprocessor which send, modify, and receive the date from the different components.

The project could be framed in the IoT (internet of things) field. Its main purpose is allow the customer to use technology in a quotidian life and, concretely, to solve a concrete work, which is not losing the pet if it runs away. In summary, the idea was developing a useful gadget, which not many resources, and easy for the user to handle.

The thesis needed two modules: one attached to the dog collar, and another one to commuicate the first one, which will stay next to the user phone (bluetooth range). Both have been implemented based on Arduino technology, using ATmega320p microproccessor and the prototyping board Arduino UNO. For sending and receiving data between modules, it uses radio technology, more specifically two LoRa XL1278-XMT modules, which are low cost and they require low power. For tracking the coordinates it uses a NEO-6M-0-001 GPS module with its default antenna plugged. Finally, the bluetooth communication between the user module and the phone has been developed over a HM-05 module, which at the same time is controlled by an app created with MIT App Inventor 2.

In addition, as the last goal of the project, it is planned to design a schematic and a printed circuit board for each of the modules. This will end creating a commercial and optimized version of the collar.

As a result, this dissertation has consisted on developing two modules which, interconnected, will manipulate the different sorts of data and measures given from the components until make them readable, and send it to the user's phone, that will show them visually.

1. INTRODUCCIÓN

1.1. Origen del mismo

El presente proyecto tiene su origen en haber cursado la asignatura de Laboratorio de diseño electrónico, asignatura optativa de tercero del grado de Ingeniería de Tecnologías y Servicios de Telecomunicación de la Universidad de Zaragoza. Esta asignatura se basa en el desarrollo de un producto que contiene un sistema electrónico en colaboración interdisciplinar con los estudiantes de diseño industrial.

La parte de trabajo de los estudiantes de electrónica se basa en la investigación de tecnologías, elección de componentes, diseño, construcción y puesta a punto de la electrónica e integración en un prototipo funcional.

Durante esta asignatura me inicié como “maker” de prototipado en tecnología Arduino y durante el año de intercambio en Estados Unidos, más en concreto en la Universidad de Northern Arizona University (NAU), Flagstaff, Arizona, me especialicé en el mismo. Allí el professor David Trevas dirige el llamado “Arduino Club” donde a los alumnos con inquietudes en esta tecnología se les proporciona ayuda y material para desarrollar gadgets a su elección usando esta tecnología.

Figura 1. Arduino Club en Northern Arizona University

1. 2. Análisis de la problemática

Lamentablemente, son numerosas las desapariciones de animales de compañía debido a que estos se pierden y no consiguen volver donde sus dueños viven o los dejaron sueltos para que corriesen. Los dueños tratan por casi cualquier medio dar con él/ella, muchas veces en vano. Proporcionar al dueño una herramienta para localizar al animal y así reducir este número es el objetivo principal del trabajo.

Figura 2. Imágenes de carteles de dueños que han perdido sus mascotas

1.3. Motivación y objetivos del proyecto

Este no fue un proyecto en colaboración con ningún otro estudiante, grupo, u otros. Al llegar a la mencionada NAU y conocer el Arduino Club, me

presenté y comenté al profesor Trevas en que consistía el trabajo fin de grado. Tras una lluvia de ideas y descartar varios proyectos, decidimos aventurarnos en este. Las principales razones fueron la simplicidad de comprender el mismo, las facilidad de usarlo, y la inexistencia de un gadget similar en el mercado (explicado más adelante en el apartado de antecedentes).

A nivel personal, el principal objetivo es poder aprender a través de este proyecto y adquirir nuevos conocimientos sobre el diseño electrónico y la programación. Además, yo particularmente seré el primero que utilice el proyecto para trackear a mi mascota y evitar que se pierda.

Para cumplir con estos objetivos generales se han definido una serie de requerimientos y habilidades que se necesitan para llevar a cabo el proyecto:

- Análisis de los requisitos que el proyecto requiere
- Comprensión del funcionamiento de la tecnología Arduino
- Perfecto entendimiento de las tecnologías de comunicación utilizadas: Bluetooth y radio
- Correcta selección de componentes y materiales
- Diseño y desarrollo de un programa que correctamente solicite y muestre los datos obtenidos

1.4. Planificación

Para llevar a cabo los objetivos planteados, se establecen las siguientes etapas en el desarrollo del proyecto:

- Fase de **análisis de la problemática**. En esta se analiza el problema que motivó a realizar este proyecto. Básicamente la contínua pérdida

o extravío de animales domésticos, mayoritariamente perros y gatos. Previamente analizada.

- Fase de **análisis de requisitos**. Donde se estudia, analiza y escogen los requisitos que la aplicación y el dispositivo han de cumplir, además de un ligero planteamiento de las partes.
- Fase de **tecnología y primeros pasos**. Esta es posiblemente la fase más importante y difícil del proyevto. En ella se tratará de escoger las tecnologías a utilizar (Bluetooth, WiFi, radio...) y una vez elegidas, los módulos y componentes que las llevarán a cabo.
- Fase de **pruebas pre-prototipado**. Una vez elegidos los componentes y tecnologías, se han de realizar pruebas exahustivas de los mismos para asegurar que funcionan correctamente, además de aprender a configurarlos y ponerlos en marcha.
- Fase de **prototipado**. Finalmente, tras comprobar que los componentes se han elegido de manera apropiada y que funcionan a la perfección, se unirán todos ellos sobre la placa de prototipado que se haya escogido (Arduino UNO en este caso) con un código común que complete un prototipo funcional del proyecto. Se llama de prototipado porque no será un aparato funcional sino más bien una simulación del mismo, principalmente porque no estará alimentado en sí sino que ambos módulos deberán permanecer conectados al ordenador pues no estarán independientemente alimentados.

2. Análisis de requisitos

Al inicio del proyecto, una vez estudiada la problemática y fijados los objetivos, se establecen la lista de requisitos mínimos que el prototipo ha de cumplir para considerarse funcional y exitoso:

2.1. Simple e intuitivo.

El dispositivo no puede ser difícil de usar. Está destinado a todo público y esto incluye niños y ancianos, los cuáles pueden no estar muy familiarizados con la tecnología y así verse incapacitados para usarlo.

Por ello mismo, se ha diseñado de manera que sea lo más fácil e intuitivo posible:

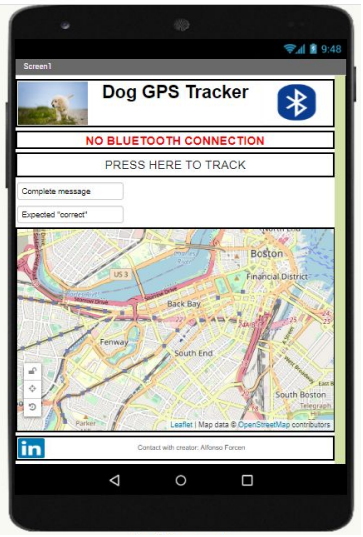


Figura 3. Captura de pantalla de la aplicación móvil tomada desde la página web donde se creó, MIT App Inventor 2 (https://appinventor.mit.edu/)

Como se puede apreciar en la imagen, tan solo son dos los botones que el usuario ha de utilizar, explicados más en profundidad en el apartado de la aplicación.

2.2. Eficiente/Útil.

Obviamente, el dispositivo ha de funcionar y ser de utilidad. Para ello se han escogido tecnología (radio) y componentes (los emisores-receptores de la misma) que permiten comunicarse de manera rápida, pues la mascota está siempre en movimiento, y en amplias distancias, siendo los componentes escogidos capaces de comunicarse entre ellos estando separados hasta 10 kilómetros de distancia.

- **"User module" o "Módulo del usuario"**. Este módulo estará a una distancia máxima de 10 metros del móvil. Esta es la distancia máxima a la que el módulo Bluetooth puede transmitir, aunque siempre que se pueda debe estar más próximo.
  Sus funciones serán conectarse al móvil vía Bluetooth y esperar la solicitud de información, la cual mandará al módulo del animal. Después de esto esperará la respuesta y, una vez recibida esta, la enviará al móvil que la presentará al usuario.

- **"Pet module" o "Módulo del animal"**. Este módulo estará insertado/pegado al collar del animal. Es el encargado de obtener las coordenadas donde se encuentra el animal.
  Sus funciones serán esperar la petición del módulo del dueño. Una vez recibida creará un *string* con las coordenadas y se las enviará de vuelta al primer módulo.

- **Aplicación Android**. Será desde el usuario haga la solicitud de localización, y donde verá la misma.

Esta imagen creada con las herramientas de dibujo "Desygner" [4] y Miro [5] representa el boceto. Las flechas grises no llevan la información de las coordenadas mientras que las azules sí. Nótese que el ciclo empieza y acaba en el SmartPhone (figura Android)

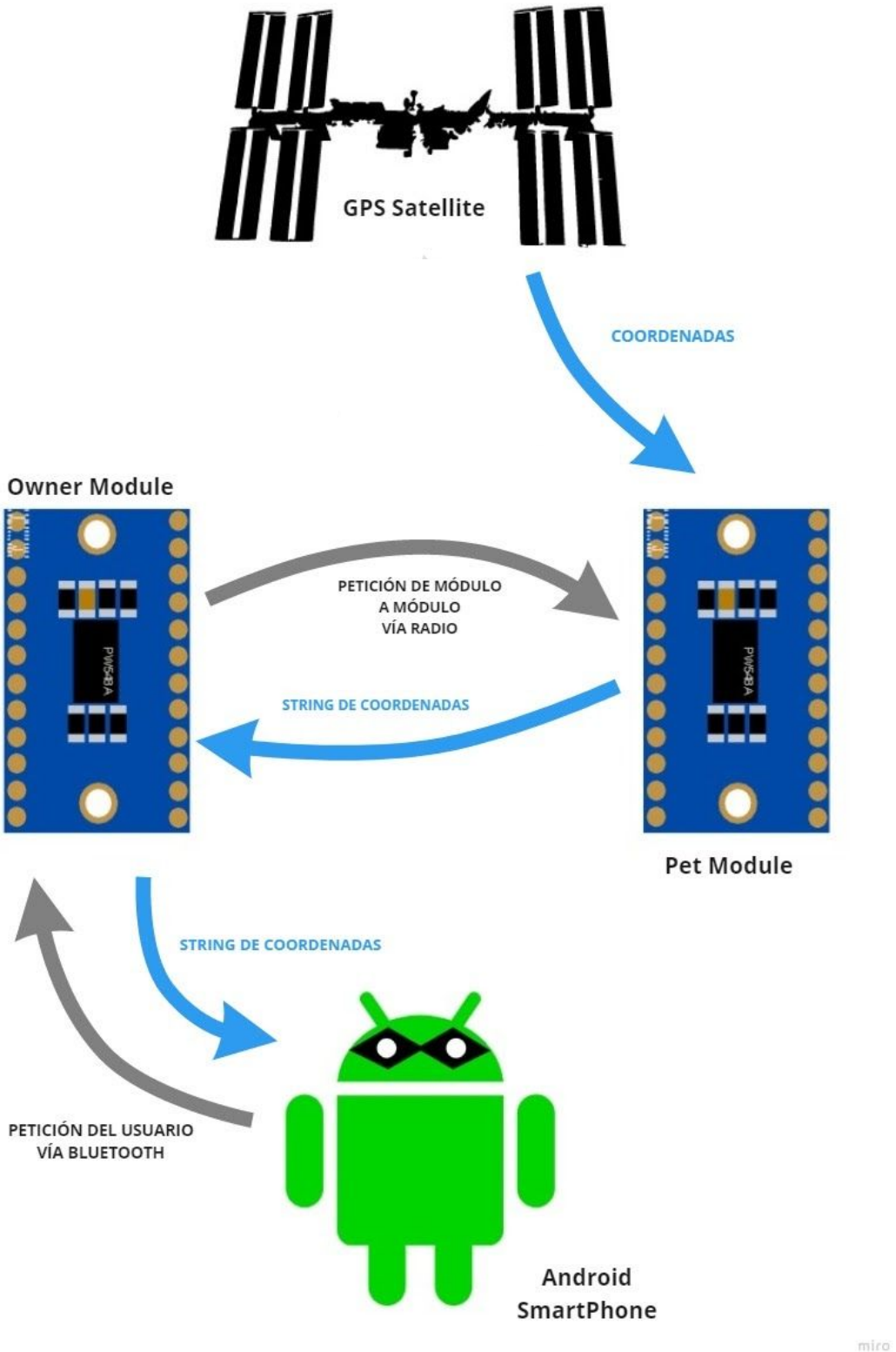


Figura 4. Boceto de la idea inicial

3. Tecnología y primeros pasos

La primer parte del trabajo es enteramente de investigación y comparación de diferentes posibilidades. No se ha de subestimar, una mala decisión en las primeras fases puede condicionar todo el desarrollo del producto y desencadenar en un error fatal y, muchas veces, irreparable. Lo primero es elegir las tecnologías a utilizar:

3.1. Procesador y uso de los datos

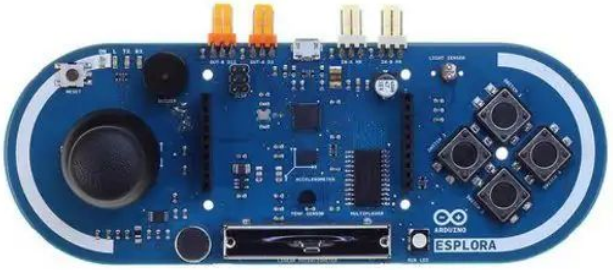
Para el procesador se eligió Arduino frente a otras como Rasperry Pi debido a experiencia previa con el mismo y a que ambas opciones ofrecían los requisitos mínimos necesarios para llevar a cabo el trabajo

Una vez escogido Arduino, tocaba elegir el tipo de microprocesador [6], lo cuál se basaba en unos pocos requisitos:

- Se necesitan al menos 11 pines digitales en el caso del módulo “Owner-Module” (3 LED, 6 para el emisor-receptor radio, y 2 para el módulo Bluetooth) y 10 para el módulo “Pet Module” (2 LED y de nuevo 6 para el emisor-receptor radio, y 2 para el módulo Bluetooth)
- La velocidad de funcionamiento o frecuencia no suponía ningún problema. No se tomó como un factor a tener en cuenta (por lo general fiuncionan a 16MHz pero hay modelos concretos que trabajan a diferentes frecuencias, como el “Arduino Zero”, a 48MHz)
- Voltaje de funcionamiento. El Arduino, como vemos más tarde, ha de ser capaz de devolver salidas de 3.3V para alimentar el LoRa (emisor-receptor radio)

Siguiendo estas especificaciones básicas se filtraron los modelos de Arduino compatibles y resultó que lo cumplían todos, exceptuando

modelos excepcionales como el “Arduino Esplora”. Por tanto, se optó por el modelo Arduino UNO por dos razones fundamentales y meramente prácticas: experiencia en su manipulación (el aprender como trabajar con otro o manipularlo tomaría más tiempo) y comodidad con el sistema de prototipado del mismo. El Arduino UNO es el Arduino más simple y básico que cuenta con un equipo de prototipado completamente funcional. Otros como el Mini, el Mini PRO o el Micro requieren de alimentación extra y una placa blanca donde ubicar todos los componentes y resto de dispositivos.



[6]

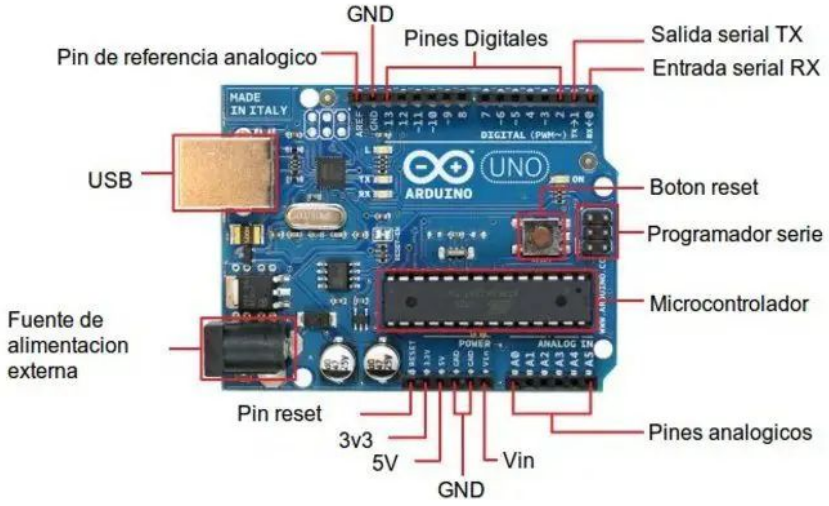
Figura 5. Arduino Esplora. Principalmente destinado a proyectos relacionados con videojugeos y entretenimiento

[6]

Figura 6. Arduino Micro

[6]

Figura 7. Arduino Mini



[6]

Figura 8. Placa de prototipado prefabricada con tecnología Arduino UNO

3.2. Comunicación módulo a módulo

Para esta parte se eligió tecnología radio. ¿Por qué? Para esta decisión nos remitimos a los antecedentes del proyecto.

La inmensa mayoría de proyectos similares a este utilizaban tecnología GSM, utilizando una tarjeta SIM igual que la de los teléfonos móviles para localizar a la mascota. A pesar de su gran precisión y rapidez

(ambas mayores que las proporcionadas por este trabajo), el principal fallo y fuente de rechazo a esta tecnología es la imperativa necesidad de una cuota mensual, pues este tipo de tarjetas lo requieren, mientras que la radio no. En conclusión, la tecnología GSM es superior en cualquier aspecto: precisión, velocidad, rango, menor consumo… exceptuando el económico, pues el dispositivo de este trabajo requiere un único y primer pago, sin mensualidad ni mantenimiento.

[7]

Figura 9. Dispositivo Weenect. Este opera con GSM y requiere de un pago mensual

Una vez decidido utilizar la tecnología radio, el mercado está lleno de diferentes módulos con diferentes especificaciones y funcionalidades, ¿cuál es el más adecuado?

Para empezar se han de establecer los criterios y requerimientos que se necesitan para el proyecto: el módulo ha de consumir poca energía (esto conllevará a una menor batería y esto a un peso inferior), ha de ser pequeño (para que el animal no tenga molestias y se pueda acoplar sin problema), ha de tener una cobertura más o menos grande (radio de unos

pocos kilómetros), y como no, ha de ser lo más barato posible para no disparar el precio del collar.

Tras investigar el mercado y las opciones más populares, se acabó comparando los módulos de comunicación radio más utilizados y "mejor" compatibles con la tecnología Arduino: los módulos LoRa xl, el módulo NRF24L01 y los módulos de transmisión RF a 433MHz emisor (FS1000A) y receptor (XY-MK-5V). Para el análisis y la decisión se hizo una tabla comparativa:

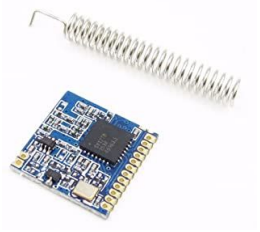
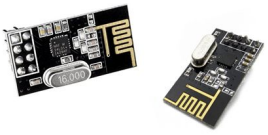
	LoRa xl128-smt [8]	**NRF24L01 [9]**	**FS1000A + XY-MK-5V [10]**
Figura			
Energía **(transmitiendo)**	10mA	15mA	Aproximadamente 7mA cada uno
Radio de alcance	5-7km	700m si alimentado a 12V	3-5m
Tamaño	16mm x 17mm	29mm x 15.2mm	20mm x 12mm 16mm x 15mm
Precio	1.70€ / ud	3.05€ pack de 5	1.70€ el conjunto

Tabla 1. Comparativa de módulos radio

A pesar de ser el más costoso, el módulo LoRa xl128-smt es el único que cumple con los requisitos mínimos, pues el resto tienen una cobertura con distancias muy reducidas, que no serían útiles.

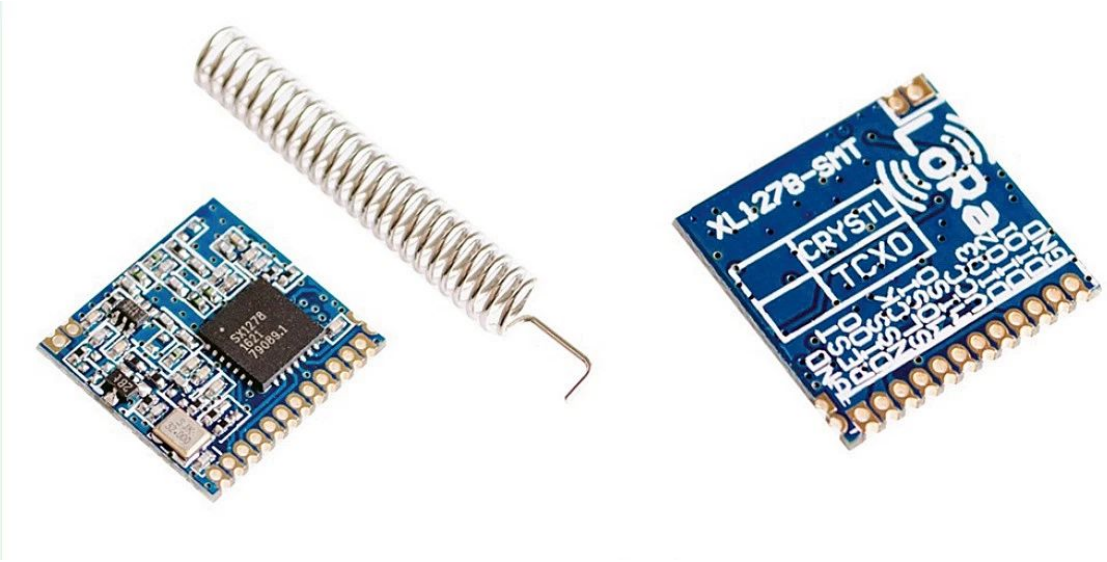


Figura 10. Módulo radio LoRa xl1278-smt

3.3. Comunicación módulo a teléfono

Para esta parte se eligió tecnología Bluetooth. ¿Por qué? Básicamente se decidió entre WiFi y Bluetooth pues eran las únicas funcionales. Otras como quedaron excluidas directamente; por ejemplo, la transmisión de datos via infrarrojos es más complicada y para que esta conexión funcione el dispositivo y el teléfono han de estar "apuntándose" mutuamente.

Entre Bluetooth y WiFi se escogió Bluetooth principalmente por una razón, con WiFi el teléfono y el dispositivo deberían estar conectados a una red WiFi y eso, si por ejemplo el dueño está paseando a su mascota fuera de casa, es altamente ineficaz. Bluetooth por otro lado no necesita de nada pues los aparatos se interconectan entre ellos, sin necesidad de disponer de internet.

El módulo escogido fue el popular HC-05 [11]. Fácil de utilizar, barato (menos de 2€ / unidad) y compacto (aunque no es especialmente necesario el tamaño y peso en este módulo, siempre es mejor cuanto más compacto)

[12]

Figura 11. Módulo Bluetooth HC-05

3.4. Módulo GPS

El módulo debía adecuarse lo máximo posible a unos pocos requisitos: ser compacto (para no molestar demasiado al animal), ser preciso (con un radio de unos pocos metros aproximadamente), un bajo uso de energía (lo que llevará a una batería de menor tamaño) y un precio económico, cuanto más mejor

De nuevo, se usa una tabla comparativa de los modelos más populares para decidir el modelo a utilizar.

El primer módulo a analizar es la familia de módulos GPS NEO [13], que incluye principalmente tres, NEO-6Q, NEO-6M, y NEO-6G. El tercero quedó descartado automáticamente pues requiere de una alimentación muy inferior al resto (1.5 - 2V), lo que requería de un regulador de voltaje adicional (más coste, complicación, y peso). Finalmente se optó por el modelo 6M por ser más comercial, más comunmente utilizado y disponer de más ejemplos y ayudas en la web.

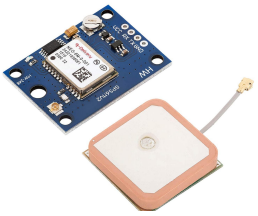
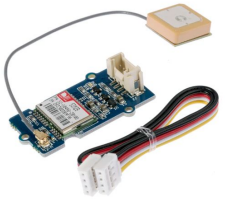
	GY-GPS6MV2 (NEO-6M) [14]	GPS Seeed Studio [15]
Figura		
Energía (transmitiendo)	37mA	42mA
Radio de precisión	< 6m	< 1.8m
Tamaño	36mm x 26mm	13mm x 40mm
Precio	$5.30 (4.5€ aprox)	22.80€

Tabla 2. Comparativa de módulos GPS

Esta vez, quedó elegida la primera opción principalmente por el precio pues, comprar el otro, supondría un enorme aumento del precio del dispositivo, cuyo principal objetivo después de funcionar es ser accesible para todo el público.

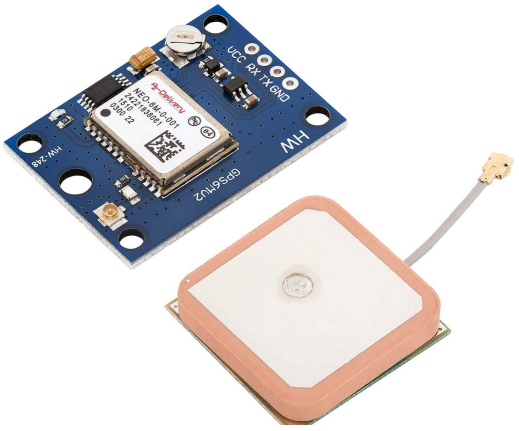


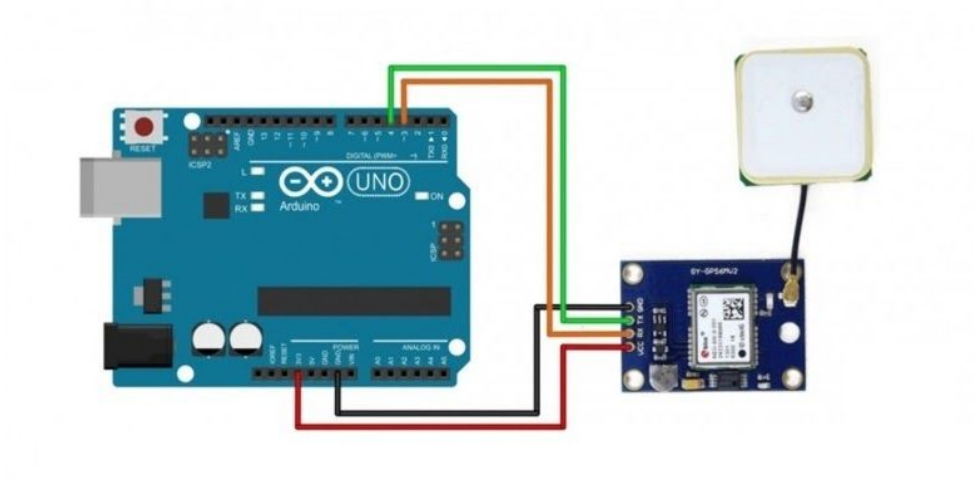
Figura 12. Módulo NEO-6M finalmete escogido

4. Pruebas pre-prototipado

El primer paso es crear ambos módulos alrededor de placas de prototipado Arduino UNO y placas blancas. Para ello, se van analizando y estudiando diferentes funcionamientos de los diferentes componentes mediante códigos sencillos hasta llegar a un final donde se juntan todos y se programa el código definitivo.

4.1. Módulo GPS. Código [1]

Se utilizó la librería Arduino “TinyGPS” [16], con el ejemplo “simple_test” para verificar que el módulo recibía las coordenadas correctas:



[17]

Figura 13. Modelo de cableado para la prueba del GPS

El programa Arduino utilizado, *"simple test"*[1], incluído en los anexos, funciona siguiendo el siguiente diagrama de flujo :

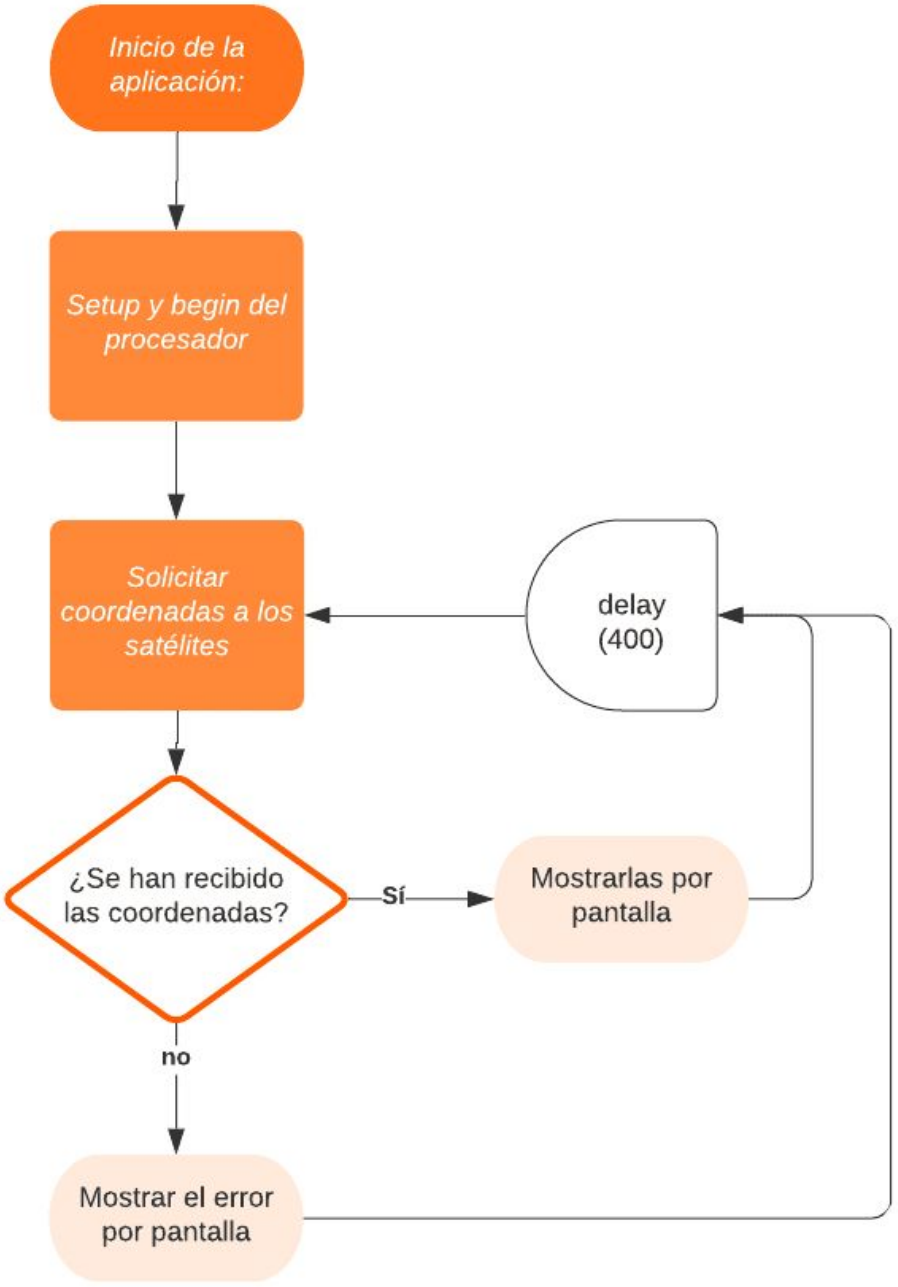


Diagrama 1. "Simple-test"

El resultado del cableado y prueba quedó así:

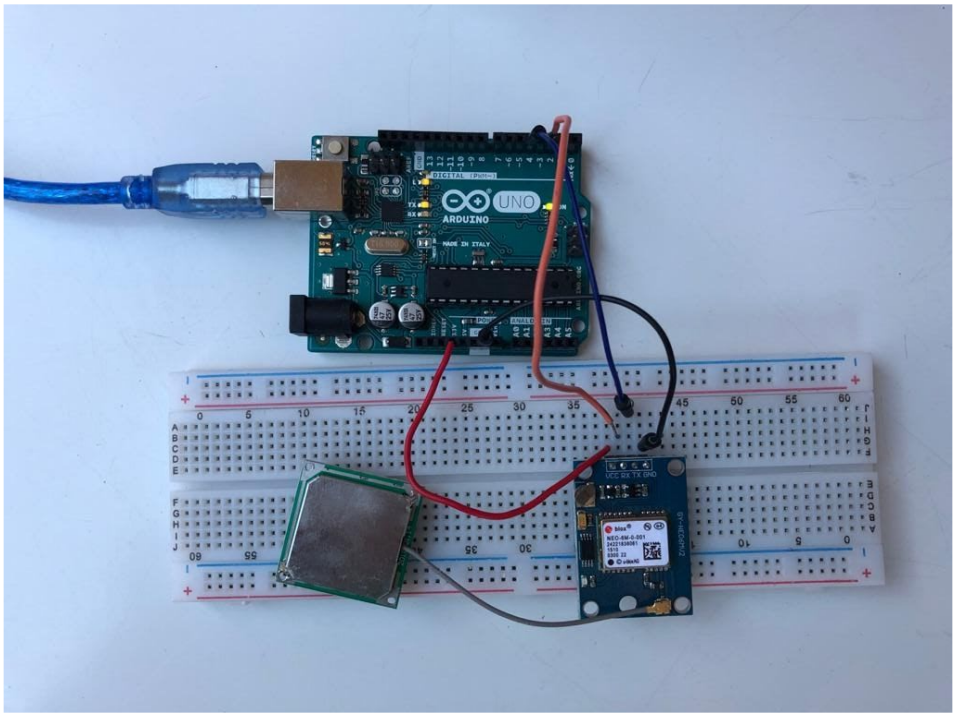


Figura 14. Resultado del cableado para el test del módulo GPS

El programa devolvió estos datos:

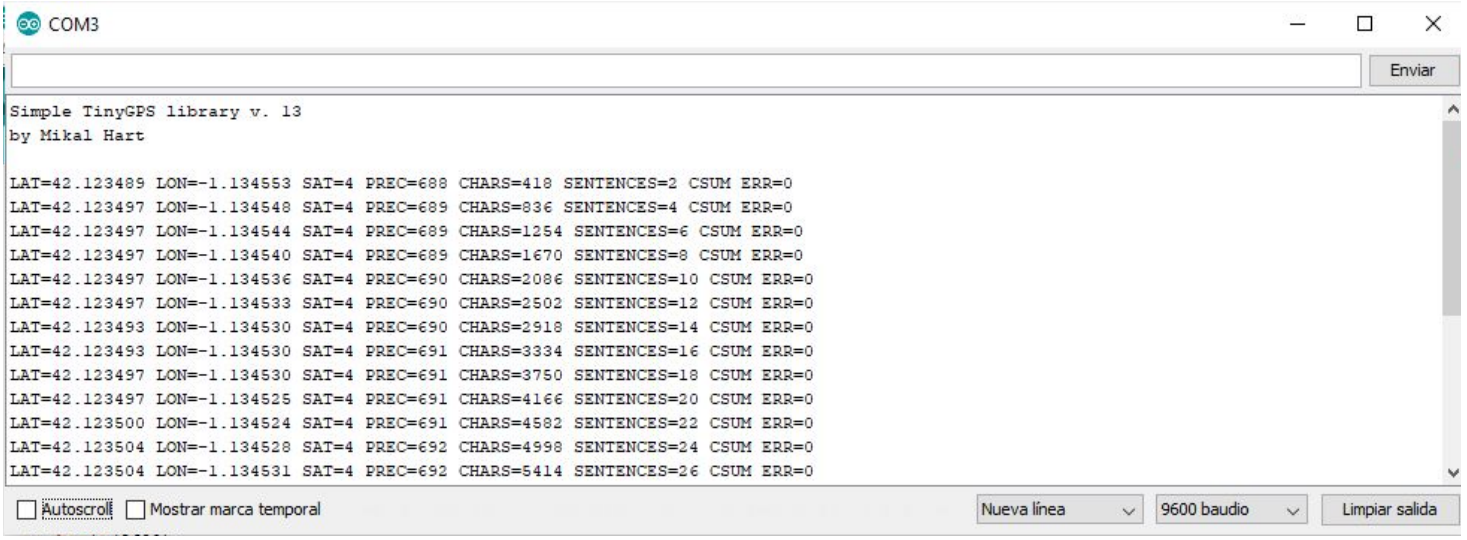


Figura 15. Resultado de la ejecución del programa de testeo

Que tras introducir en Google Maps nos devolvió la coordenada correcta, con muy buena precisión:

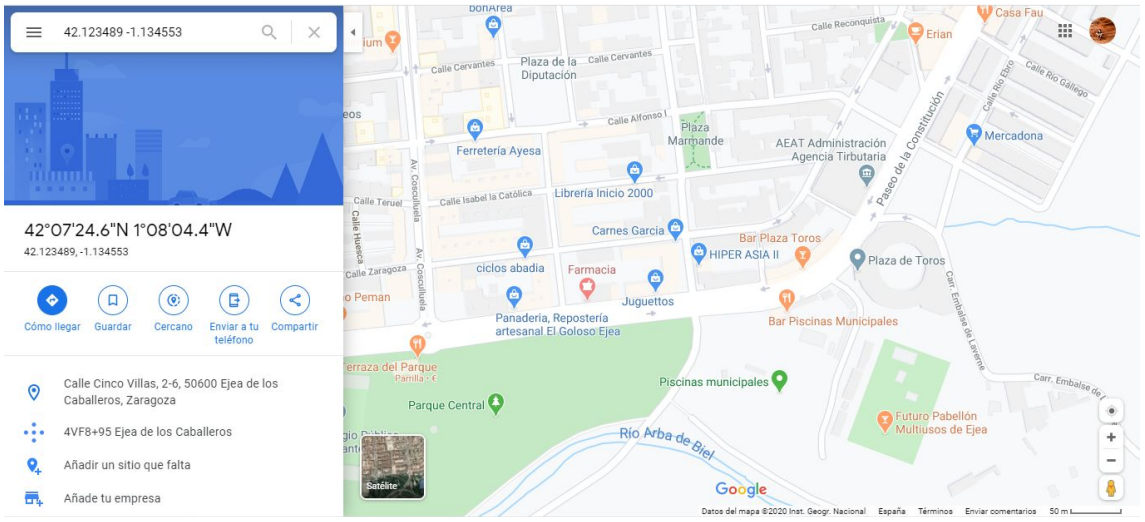


Figura 16. Resultado de introducir las coordenadas obtenidas con el GPS en la plataforma Google Maps

Y con esto, verificando que funciona correctmente, damos por concluida la parte de testeo del módulo GPS. Será en el programa final que, basándonos en la lógica del programa utilizado, codificaremos una función GetCoordinates() que hará el mismo trabajo.

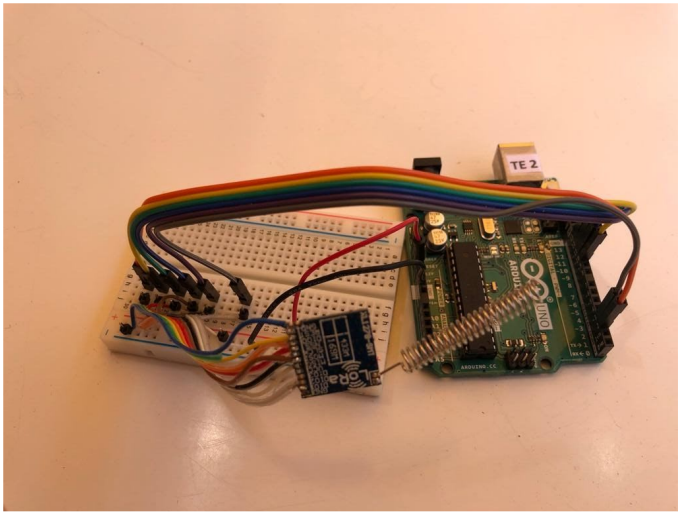
4.2. Módulos radio. Código [2]

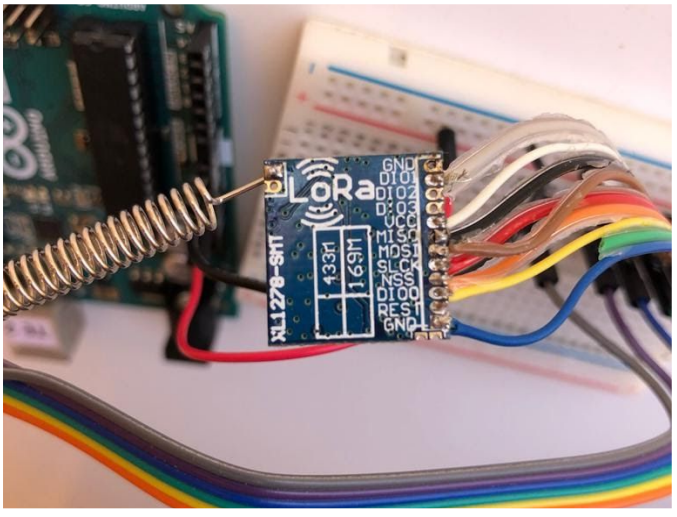
Se utilizaron las librerías Arduino "*receive*" y "*transmit*" de los ejemplos proporcionados por la libreria "LoRaLib" [18]. El testeo se hizo apoyándose en la página web [19]. El cableado utilizado fue:

LoRa Module	Arduino Uno
MISO	D12
MOSI	D11
SLCK	D13
D100	D2
D101	D3
VCC	3.3 V
GND	GND
NSS	D10

Tabla 3. Cableado del módulo LoRa para el testeo

Quedando así el montaje: (más difícil que el resto pues el soldado de los cables era más complicado, estando estos a 1.27mm de separación, cuando la distancia "convencional" es 2.54mm)





Figuras 17 y 18. Resultado del cableado de los módulos radio

Los códigos utilizados son dos, "*receive*":

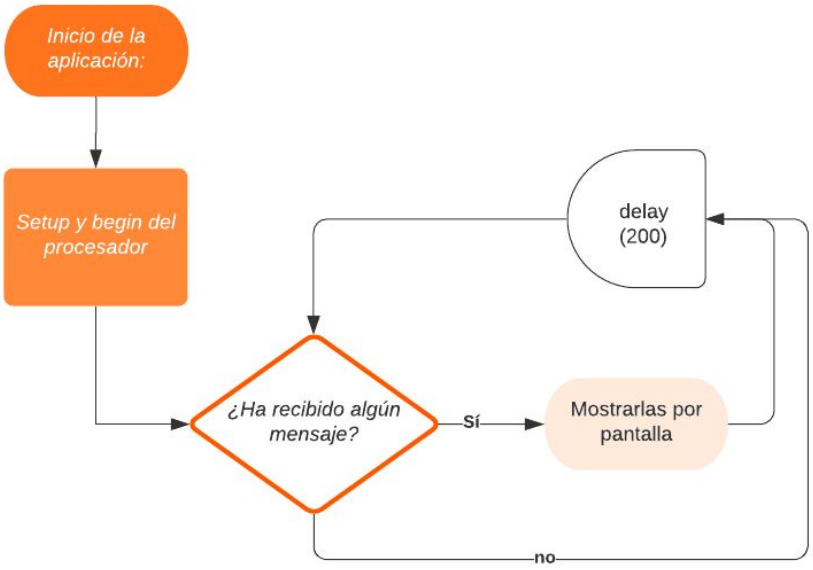


Diagrama 2. "Receive"

Y "*transmit*" :

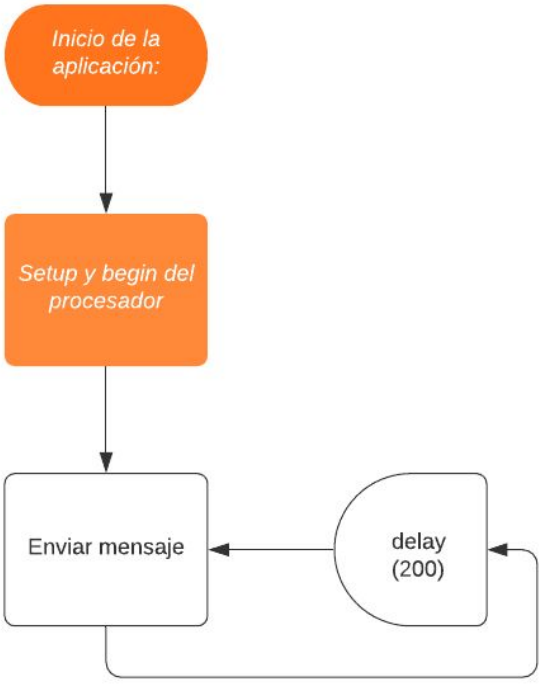


Diagrama 3. "Transmit"

4.3. Módulo Bluetooth. Código [3]

Finalmente, para el testeo del módulo Bluetooth se va a implementar una aplicación móvil usando tecnología Android en la plataforma "*MIT App Inventor 2*". La aplicación ha de comprobar que tanto la emisión como recepción de datos funciona correctamente. Para ello, se va a crear una aplicación simple, con dos funciones principales:

- Por un lado la aplicación tendra un botón que encenderá y apagará un LED, para testear la comunicación Android -> módulo
- Por otro lado, se instalará un pulsador que una vez presionado, envíe un mensaje al teléfono. Así se testeará la comunicación módulo -> Android

Esta parte del testeo resulta la más importante pues es principalmente a raíz de ella a partir de la cuál se construirá la aplicación final, donde esta vez se envíen solicitudes y se reciban las coordenadas del animal.

Empecemos con la aplicación móvil, la cuál se divide en tres bloques principales:

- El primero se repite en todas aplicaciones que necesiten de conexión Bluetooth, y es el encargado de mostrar por pantalla las posibles conexiones, y dar al usuario la opción de conectarse a la que más le interese. Esto no necesita código en Arduino.

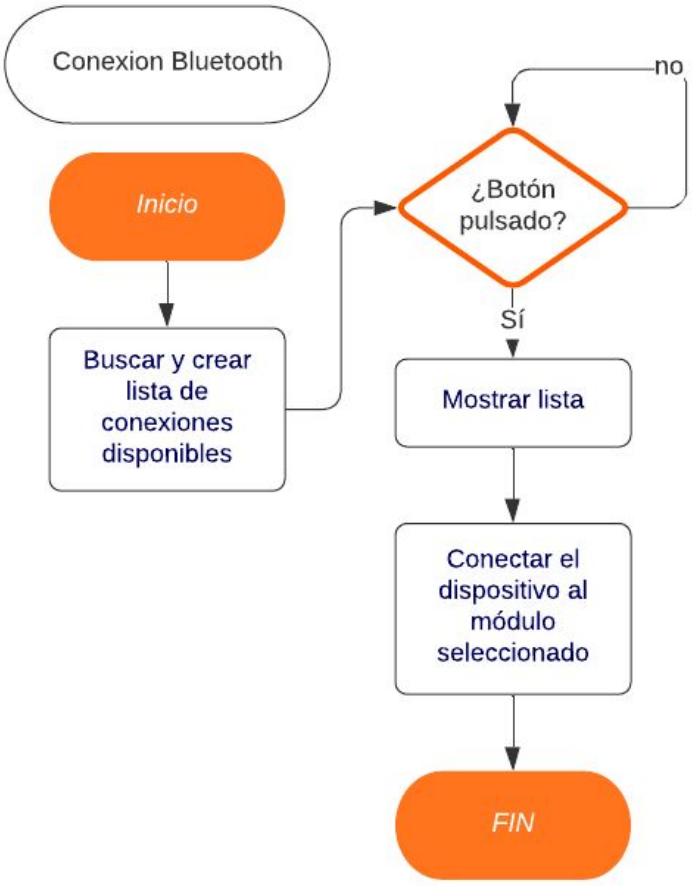


Diagrama 4. Conexión Bluetooth Android <-> Módulo del usuario

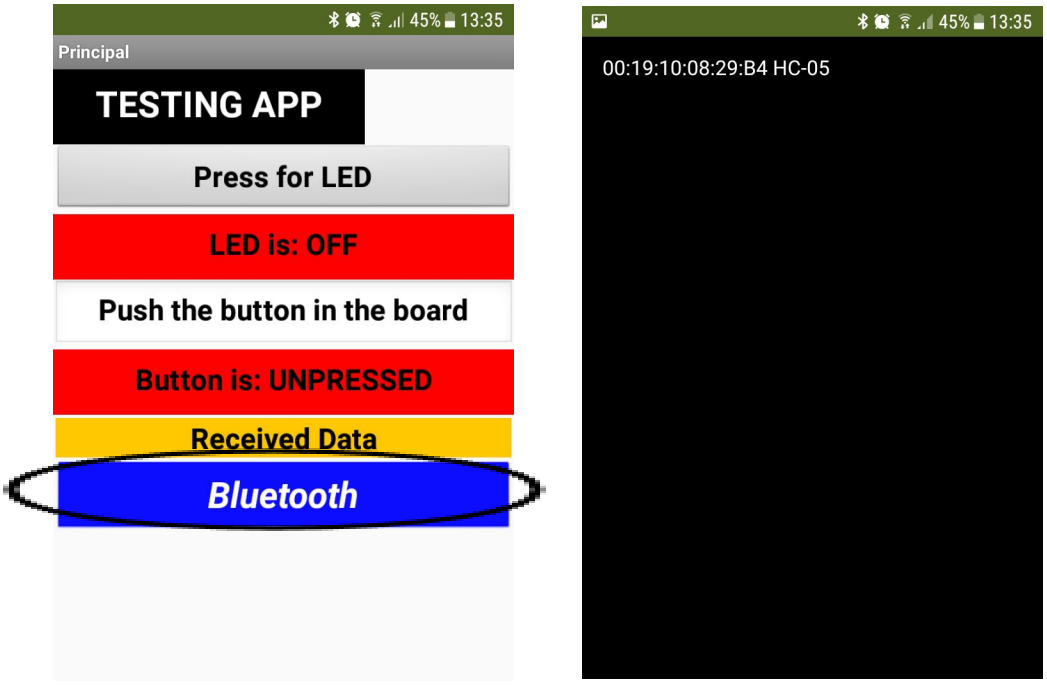


Figura 19. Capturas de pantalla de la aplicación de testeo conectándo al módulo Bluetooth

- Segundo, la aplicación ha de ser capaz de controlar un LED usando el botón de “Press for LED” [20] arriba mostrado. Para ello hace falta código tanto en el Arduino (líneas 26 -> 42) como un nuevo bloque en la aplicación:

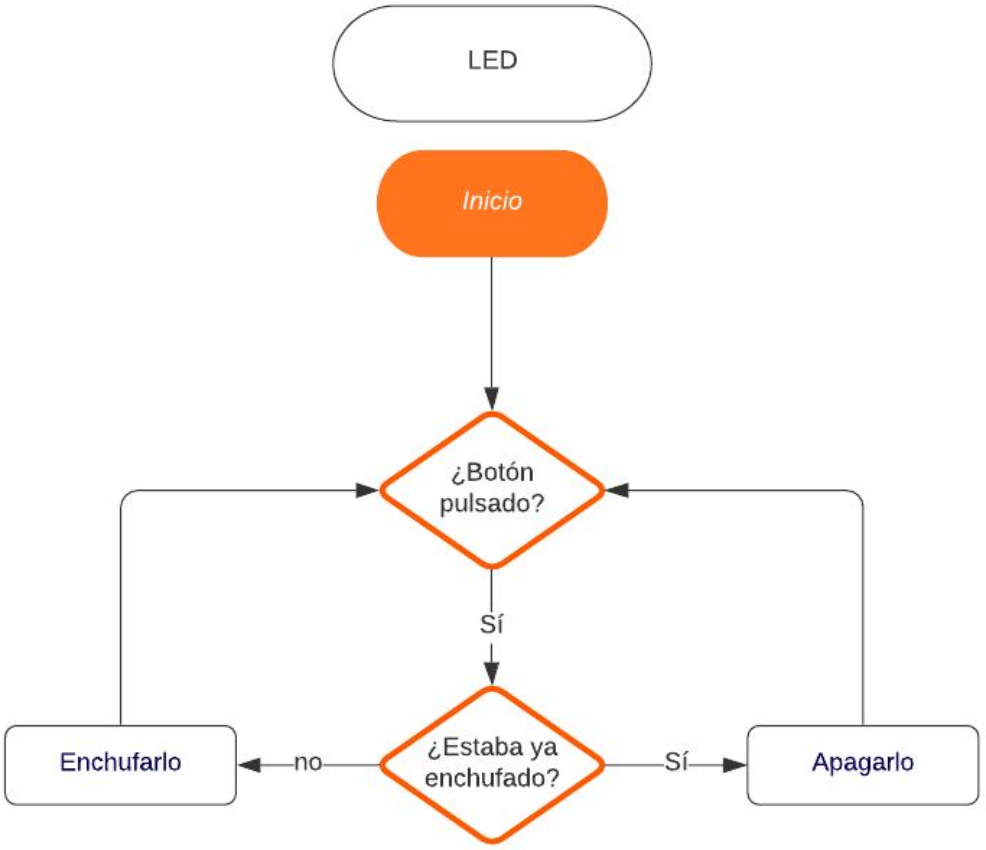


Diagrama 5. Control del LED

- Tercero y último, la aplicación ha de ser capaz de cerciorarse cuando el pulsador está activo [21], cambiando el texto y el color de su respectivo espacio en la app. Esto de nuevo requiere codificación en ambos ámbitos: Arduino (líneas 43 -> 56) y Android:

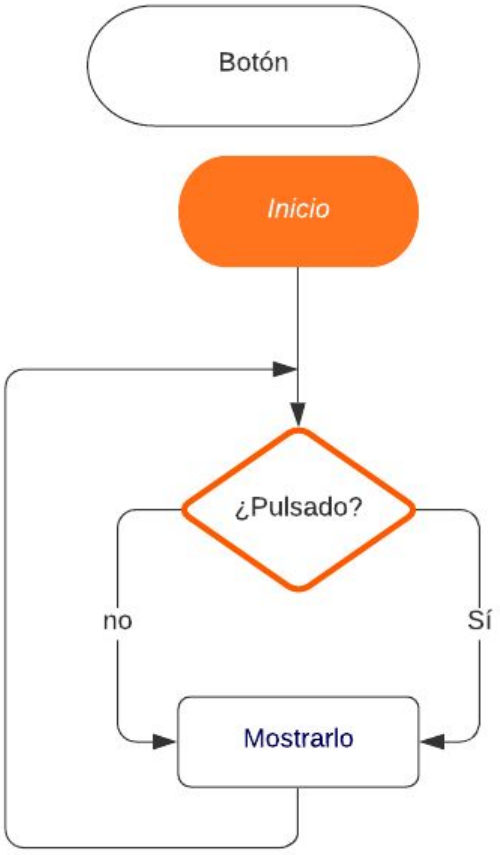


Diagrama 6. Control del pulsador

El programa Arduino utilizado es muy importante porque es la base del que será el final, respetando la estructura e incluso manteniendo algunas partes como la función “boolean” que detecta si se ha recibido una nueva petición señal (líneas 65 -> 77). El programa funciona siguiendo el siguiente diagrama de flujo:

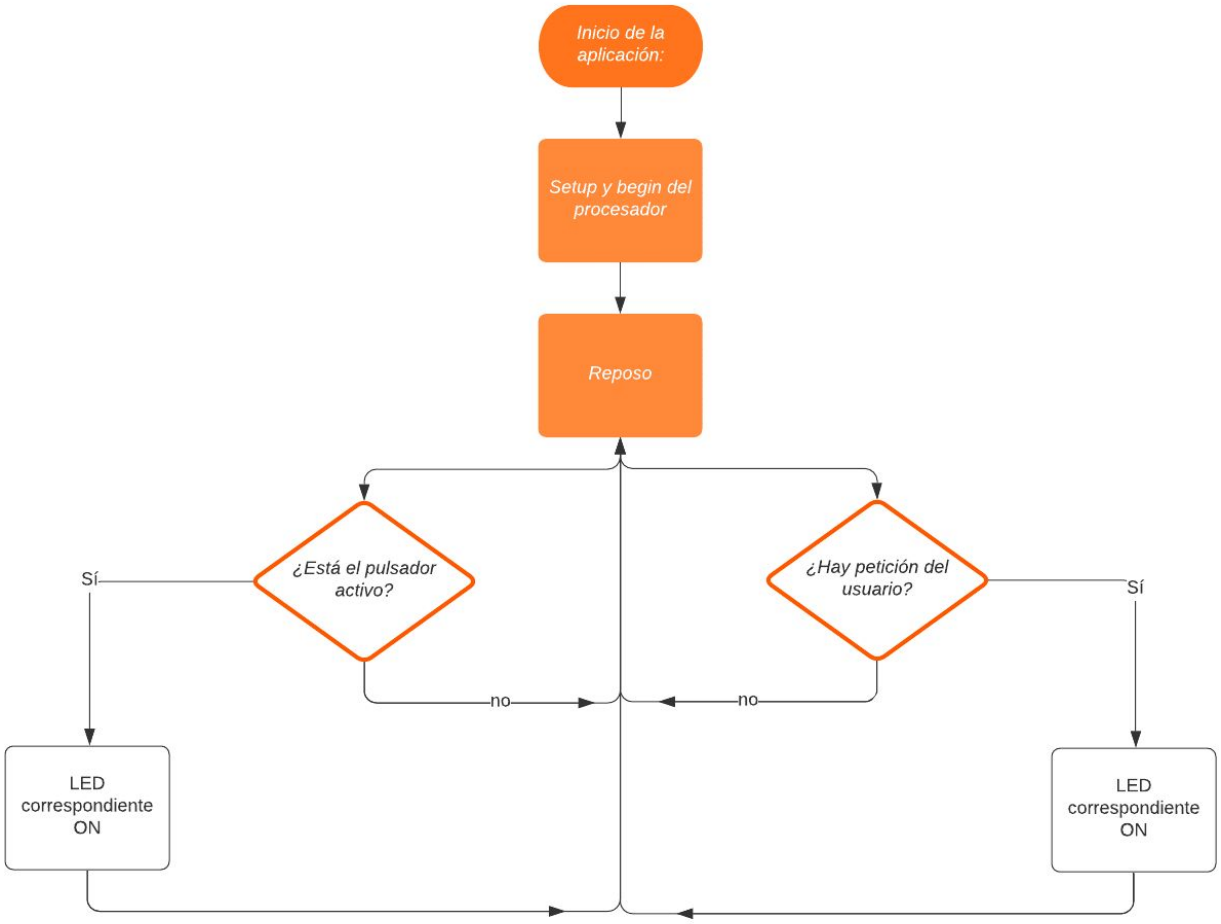


Diagrama 7. Aplicación final de testeo

5. Montaje y primer prototipo

Una vez entendemos y podemos confirmar que todos los diferentes compoentes del collar funcionan, pasamos a realizar el prototipado, siguiendo el modelo de funcionamiento mostrado en la Figura 4.

5.1. Programa Arduino. Código [4]

La estructura del programa Arduino se basará en diferentes funciones a las que el "loop" o ciclo llamará cuando considere necesario, tanto en un módulo como en el otro. Las funciones serán las siguientes:

- `void makeLEDBlink (int LED)`. Esta función no devuelve nada y solo se utiliza al arranque del programa, para asegurar que todos los LEDs funcionan. El funcionamiento es muy básico: introduces el número del pin conectado al LED que quierer hacer parpadear como parámetro, y el programa lo enchufa y apaga iterativamente 3 veces:

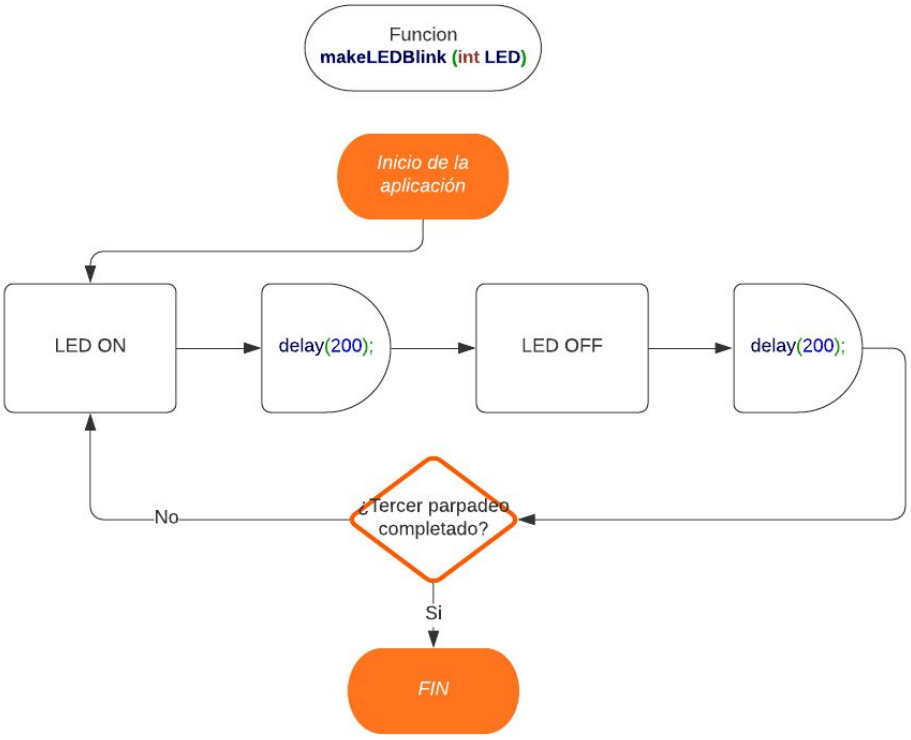


Diagrama 8. makeLEDBlink(int LED)

- `void checkLEDs()`. La cuál solo llama a la anterior tantas veces como LEDs tiene que comprobar (3 para el módulo del usuario y 2 para el del animal). Esta sería la del módulo del usuario:

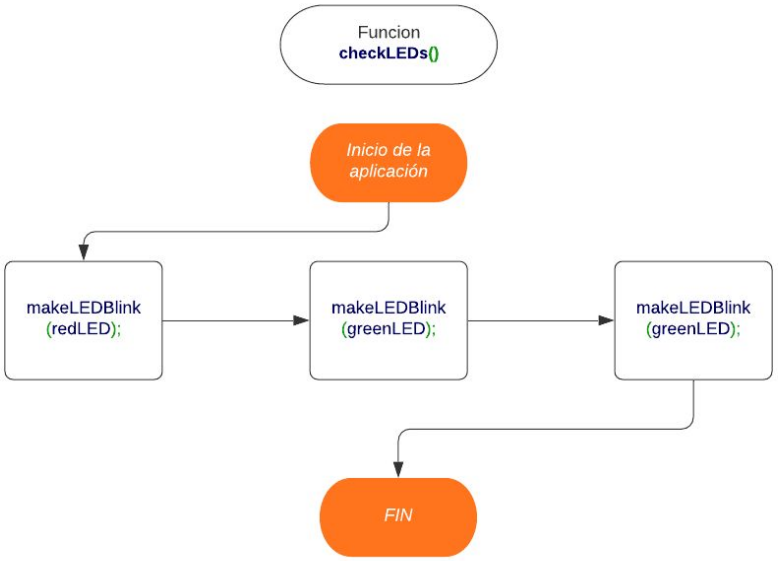


Diagrama 9. checkLEDs()

- Funciones de uso del módulo GPS. Estas incluyen `void initializeLoRa ()`, `void checkLoRa ()`, `bool receiveLoRa ()`, `bool sendLoRa ()`. Estas funciones están incluidas en la librería que se uso anteriormente para comprobar el funcionamiento de los módulos radio y solo incluyen una pequeña modificación: pasan de ser "`void`" a ser "`boolean`" y devolver un estado "true" en caso de que envía o reciba correctamente, dependiendo de si es `receiveLoRa` o `sendLoRa.`

- `bool getGPSCoordinates ()`. Función basada en la incluída en la librería TinyGPS usada para el testeo del mismo módulo. Esta se ha modificado levemente de igual manera que las anteriores, de manera que cuando se requiera de su uso, esta itere repetidamente hasta que los resultados sean positivos. Es decir, hasta que reciba unas coordenadas correctas y reales y las almacene en las variables asiganadas.

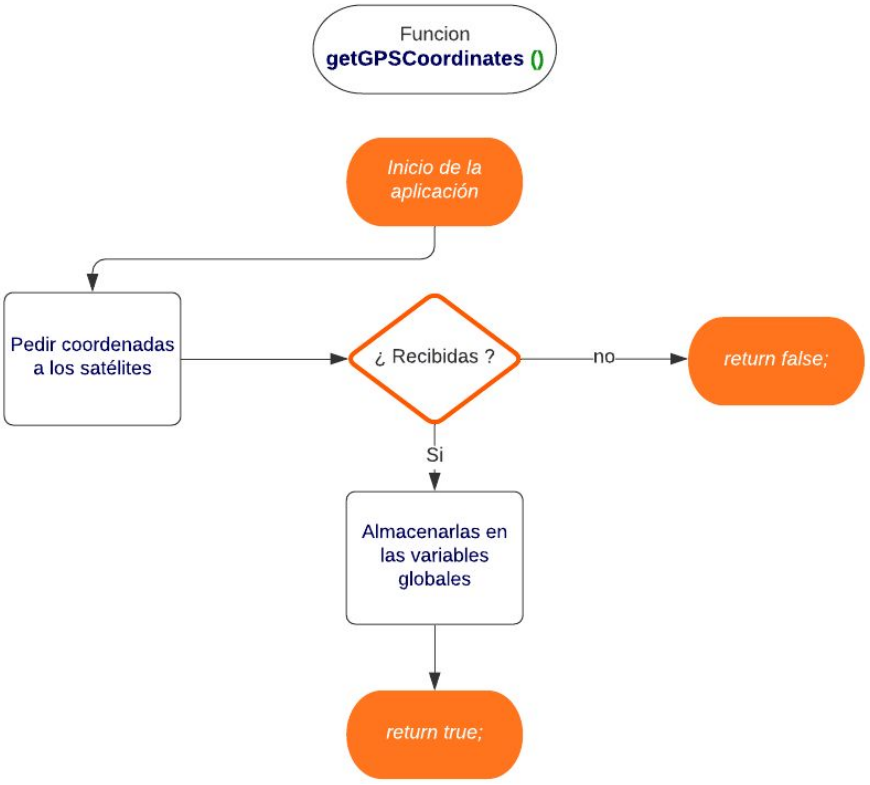


Diagrama 10. getGPSCoordinates()

Finalmente, hubo que programar tanto el “setup” de la aplicación, como el “loop” de la misma:

- `void setup ()`. A pesar de ser diferente para cada módulo, ambos tienen una serie de funciones en común: ambas han de inicializar el sistema con la orden `Serial.begin(9600)` y la conexión Bluetooth o el módulo GPS, respectivamente. Además, se han de establecer las funciones de los diferentes pines, o bien “INPUT” u “OUTPUT”, dependiendo si se encarga de enviar información, o de recibirla. Por último, se llama a la función `void checkLEDs ()` para comprobar el correcto funcionamiento de los mismos. Así quedó la función en el módulo del animal:

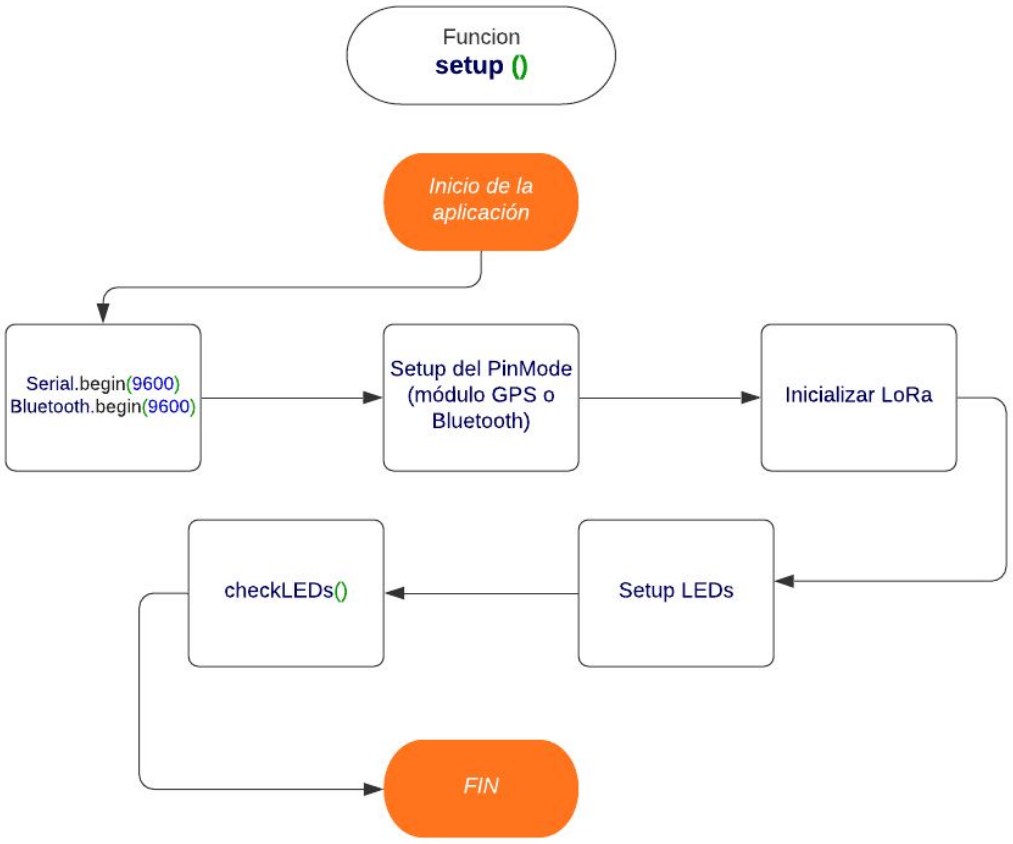


Diagrama 11. setup()

- `void loop ()`. Podemos definir esta función como la encargada del control total del módulo. Este si es bien diferenciado dependiendo del módulo del que se hable, pues cada uno tiene diferentes funciones, ejecutadas a su vez en distinto orden:

    1. Módulo del usuario. Es el que arrancará primero. Empieza esperando a que el móvil le envíe la petición de coordenadas, cuando el usuario presione el botón para ello. Una vez recibida esta, se la enviará vía radio al otro módulo y acto seguido, tras comprobar que este lo ha recibido, se quedará a la espera del siguiente mensaje, que contendrá ya las coordenadas. Una vez tenga estas, las pasará vía Bluetooth al teléfono donde la aplicación los mostrará en el mapa. El código queda así:

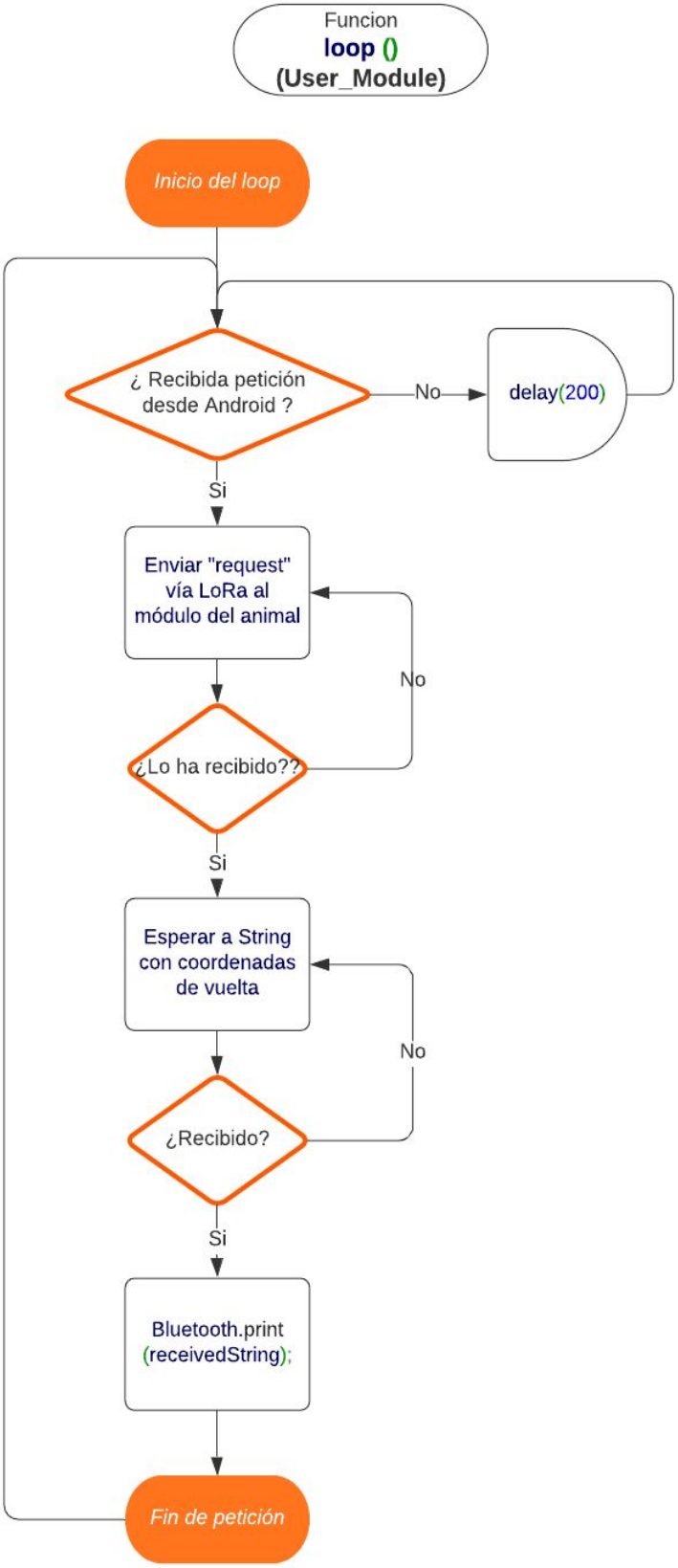


Diagrama 12. User module loop()

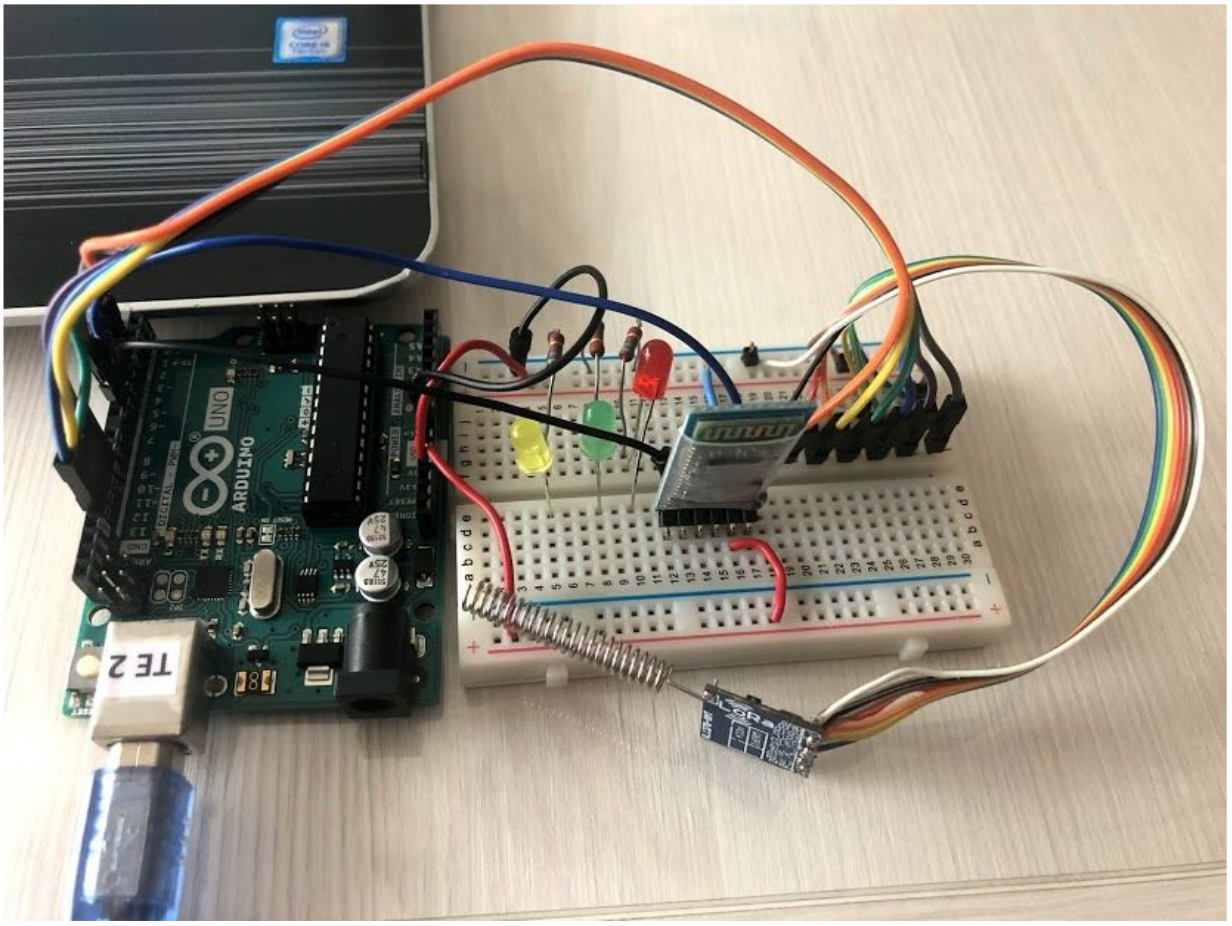


Figura 20. Resultado del módulo del usuario

2. Módulo del animal. Este comenzará igual que el del usuario, esperando la petición de coordenadas. La diferencia es que este la recibirá desde el otro módulo, no desde el teléfono. Una vez la reciba, este se encargará de tomar las coordenadas con el módulo GPS, crear el "string" que las contenga, y enviárselo al otro módulo de vuelta. Cuando pueda confirmar que se han transmitido correctamente, este habrá terminado. El código queda así:

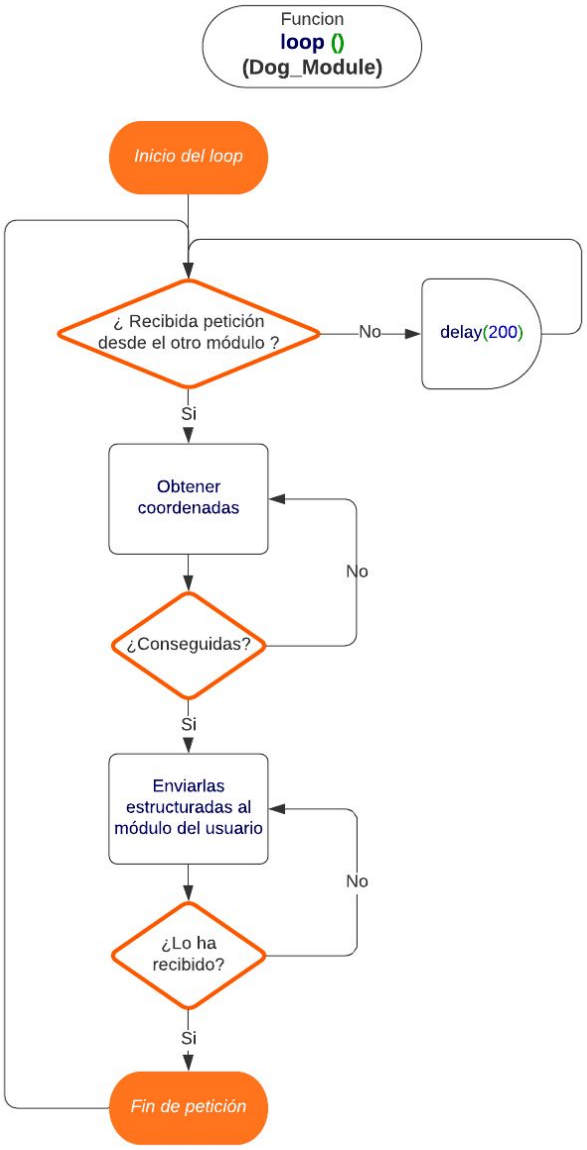


Diagrama 13. Dog module loop()

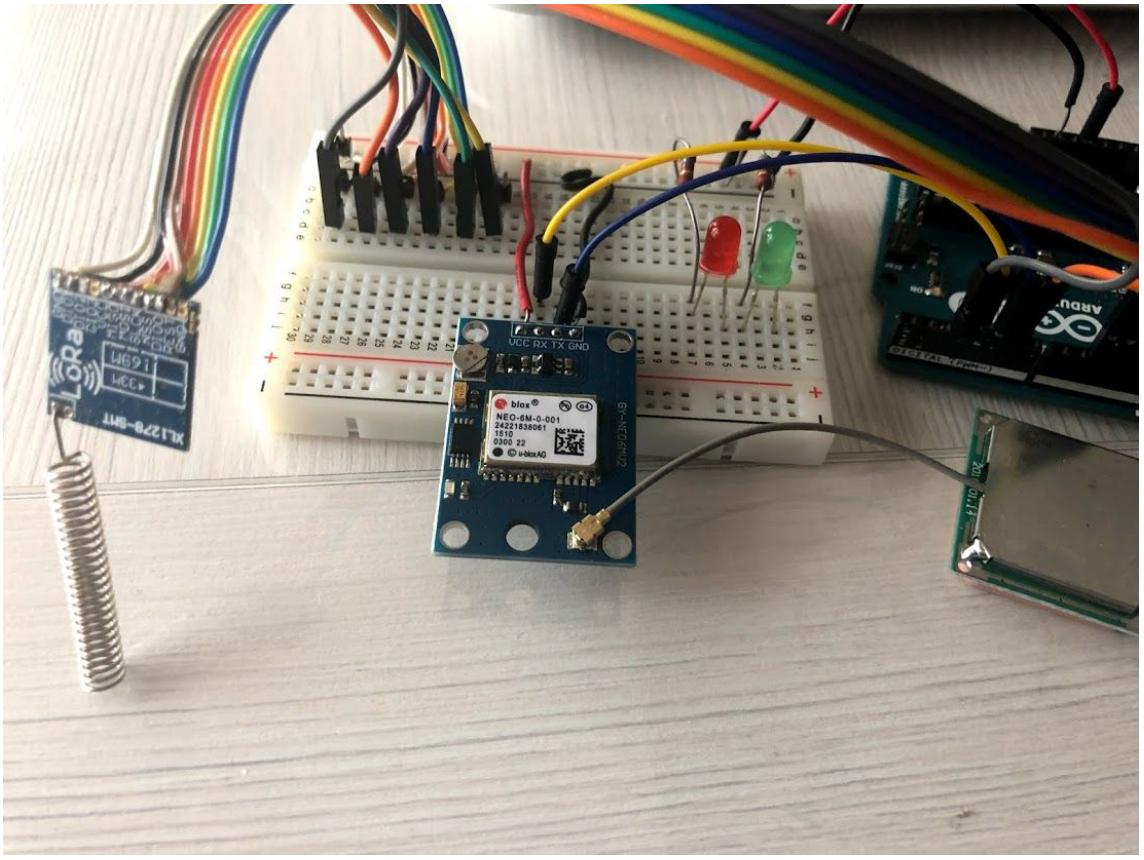


Figura 21. Resultado del módulo del animal

5.2. Aplicación Android. Código [5]

De igual manera, el desarrollo de la aplicaión se puede dividir en partes:

- Configuración Bluetooth. Primeros dos bloques iguales a los de la aplicación de prueba. Se encargan de buscar y mostrar la lista de dispositivos compatibles para conectarse, y de conectar una vez el usuario a seleccionado a cuál le interesa, respectivamente. Además se ha añadido la muestra de un cuadro de error en el caso de que el usuario trate de conectarse con el Bluetooth desactivado.

- Link a la red social "LinkedIn" [22]. En la parte inferior de la pantalla se incluye un link a mi página principal de LinkedIn, donde está registrada toda mi actividad laboral hasta el momento, y donde se me puede consultar por dudas y averías en la aplicación.

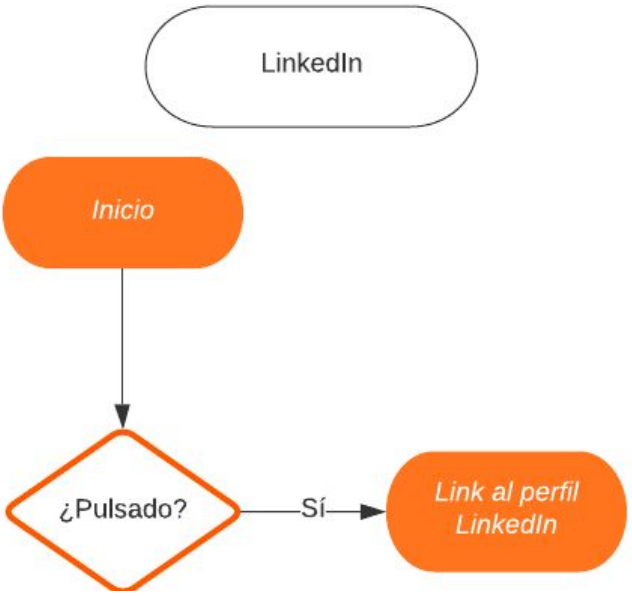


Diagrama 14. LinkedIn link

- Por otro lado, tenemos la configuración del botón “TrackButton”. Este básicamente se encargará de mandar un “request” al módulo del usuario e iniciar todo el proceso una vez el usuario pulse. Además, cuando este pulse, el texto que aparece en el botón pasará de “PRESS HERE TO TRACK” a “Getting the coordinates…” hasta que las reciba.

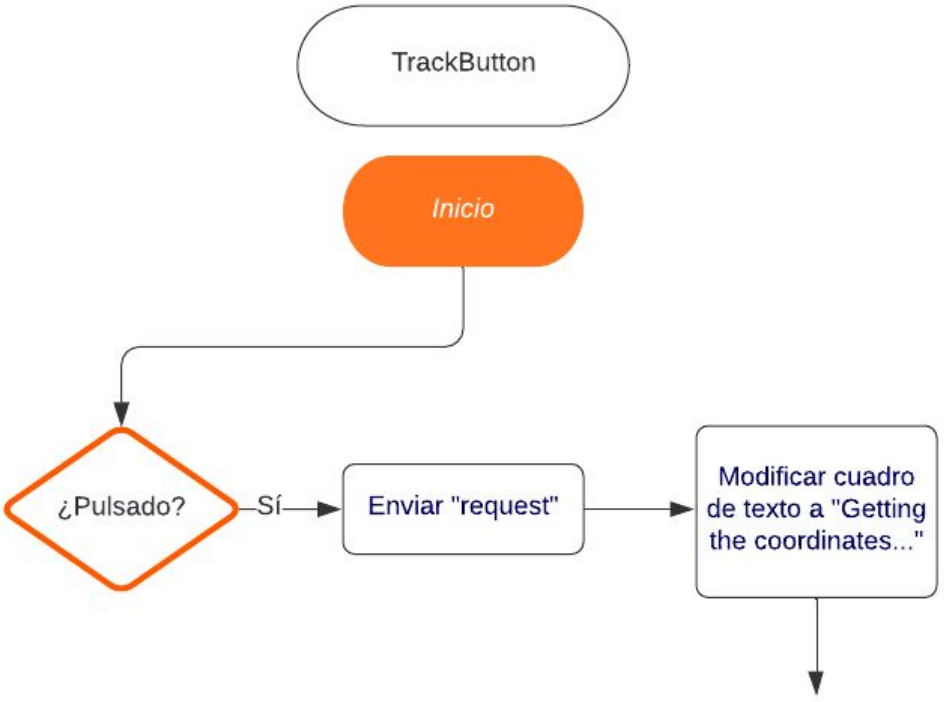


Diagrama 15. Track Button

- Finalmente, la parte más importante de la aplicación, la obtención, manejo y muestra por pantalla de datos. Cada ciclo de reloj la aplicación guarda los datos recibidos desde el módulo del usuario, los cuáles deberían ser nulos. Pero no siempre es así por lo que se ha codificado de manera que el string que este espera y que tomará como bueno tendrá la estructura: “correct lat*1000000 alt*1000000”. Será solo despues de dividir el mismo en tres, usando los espacios como

delimitadores, y comprobando que el primero es “correct”, el programa pasará a mostrar los datos. Este mostrará el string completo en un cuadro de texto situado debajo del botón en caso de que el usuario prefiero usar una aplicación de mapas externa. Y además, incluye un mapa donde se marcará la localización del animal. Este funciona siguiendo el siguiente diagrama de flujo:

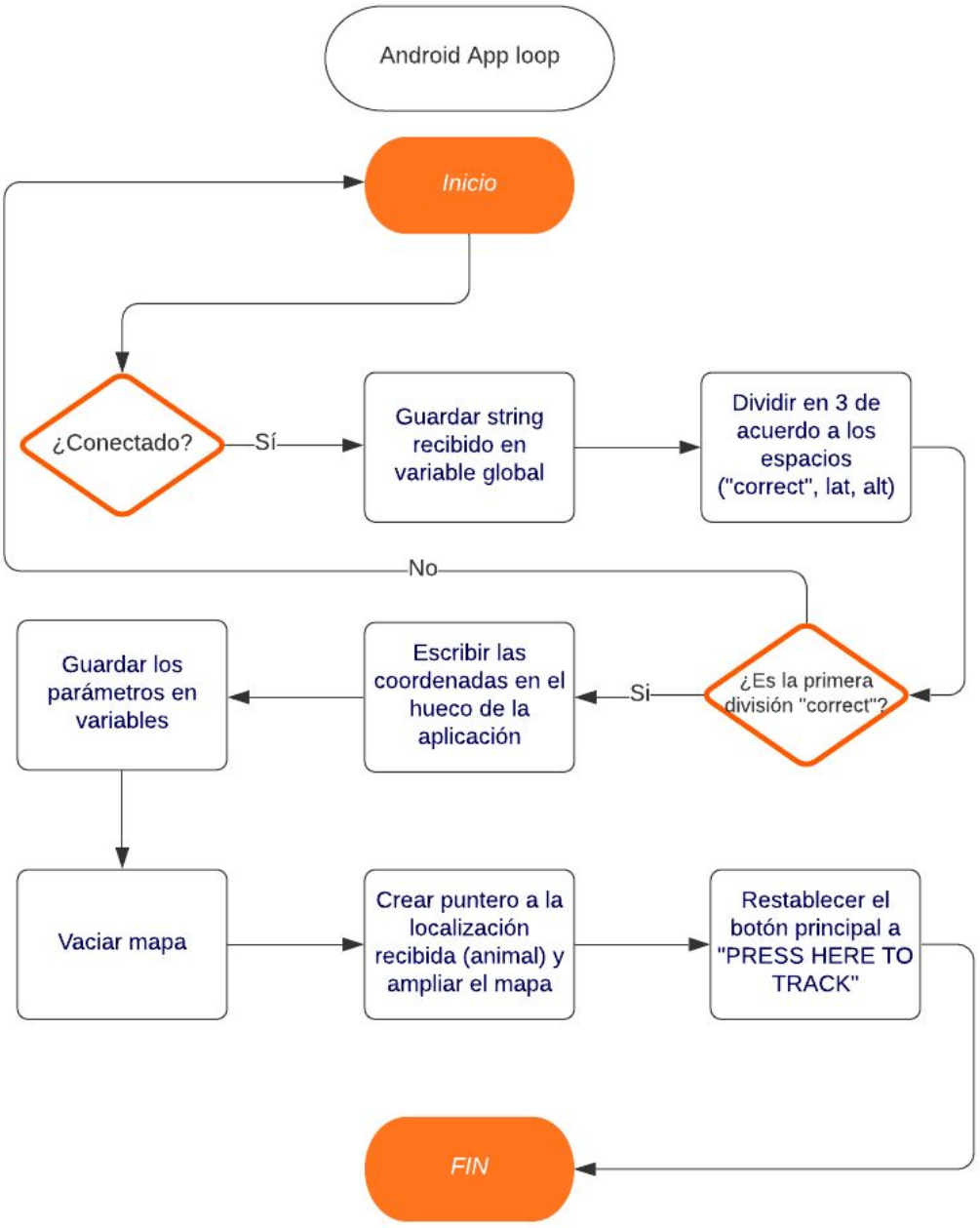


Diagrama 16. Android App loop

Y así queda la aplicación finalmente, antes y después de solicitar las coordenadas respectivamente:

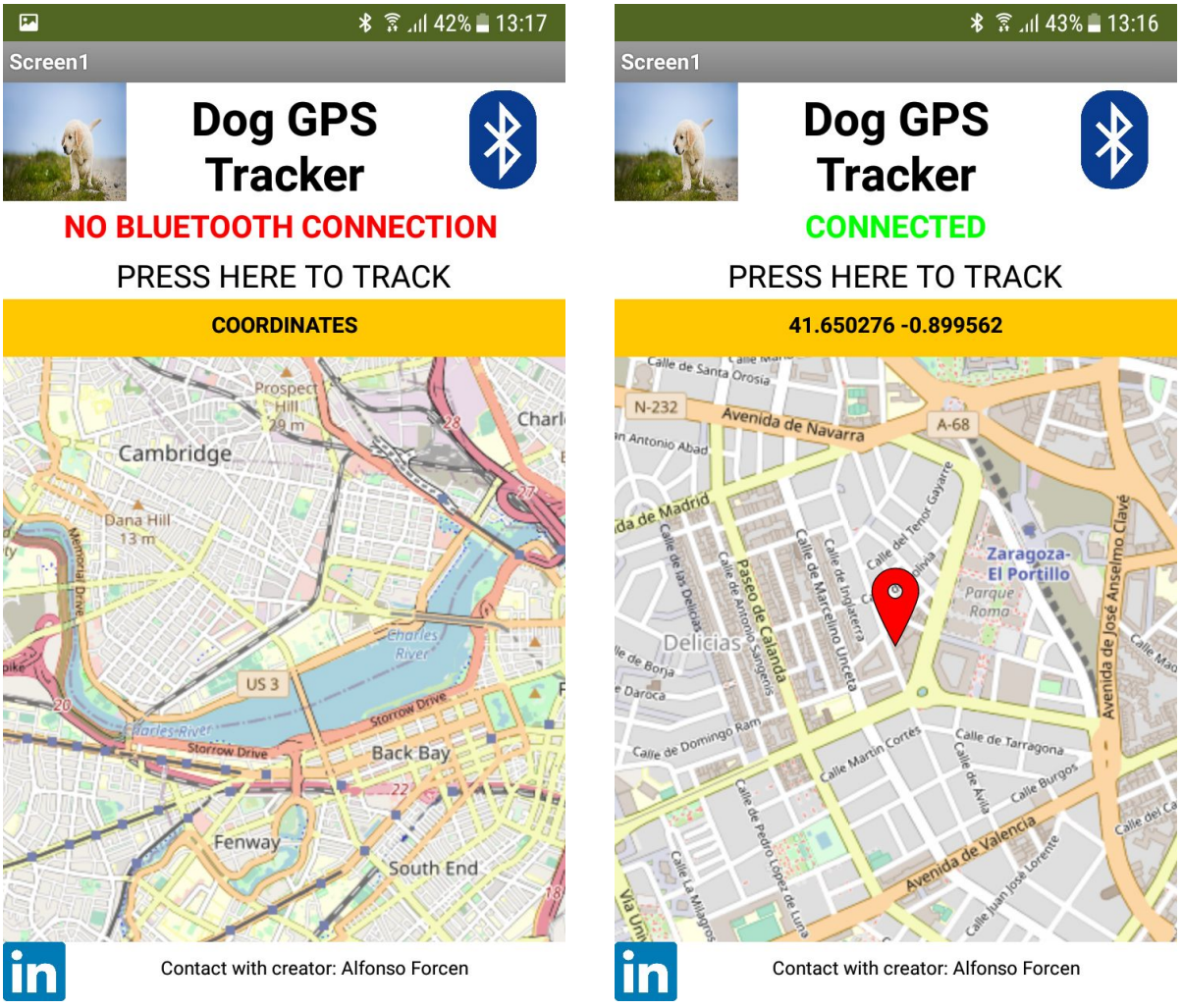


Figura 22. Capturas de pantalla de la aplicación final antes y después de solicitar las coordenadas

6. Diseño de PCBs

Una vez desarrollada la aplicación, tanto Android como configurados ambos microprocesadores atmega328p, el siguiente paso es diseñar una placa PCB (“printed circuit board” o placa de circuito impreso en español). Los beneficios de estas placas son variados [23]:

- El uso de PCBs puede reducir considerablemente el volumen y el peso de los productos electrónicos, y es adecuado para el desarrollo de productos electrónicos en la dirección de alta densidad, miniaturización y alta confiabilidad.
- El precio no es demasiado elevado y apenas aumenta el valor del dispositivo o gadget que implementa.
- La PCB también tiene una buena función en la disipación de calor y la capacidad de soldadura, y es fácil de instalar.

Obviamente esta ténica no está libre de desventajas:

- Es difícil cambiar y reparar el circuito flexible. Una vez que se fabrica el PCB flexible, se debe cambiar desde el mapa base o el programa de litografía programado, para que no sea fácil cambiarlo.
- El tamaño es limitado. Los FPC generalmente se fabrican mediante un proceso por lotes en ausencia de un proceso convencional, y por lo tanto están limitados por el tamaño del equipo de producción y no pueden hacerse largos y anchos.
- Funcionamiento inadecuado y daños. El funcionamiento incorrecto del personal de instalación puede causar daños en el circuito blando. La soldadura y el retrabajo deben ser operados por personal capacitado.

Sin embargo, para el desarrollo de este proyecto sólo hay que preocuparse por la primera desventaja pues, la placa va a ser muy reducida pues apenas incorpora componentes y ha de ser llevable por el animal y cómoda por el usuario; y en segundo lugar, a la hora de rutear el resultado, se han aplicado normas de distancias que, a pesar de no suponer la mayor optimización de tamaño, hacen a una persona no experta capaz de soldar sin problema los componentes a las vías correspondientes.

En resultado a esto, una placa de circuito impreso parecía una buena elección y se pasó a su diseño y creación. Para ello se utilizó la herramienta Circuit Maker, la cuál ya se usó en la asignatura que inspiró este proyecto [24] [25]

6.1. Diseño

6.1.1. Microprocesador

Se estructuraron por partes, rodeando el centro de la PCB el cuál debía ser el microprocesador [26]. Este necesita unos determinados componentes por defecto para su correcto funcionamiento:

- Dos de sus pines han de estar conectados a los extremos de un cristal de 16MHz. los cuáles además están conectados a dos condensadores de 22pF.
- Tiene dos pines Vcc y AVcc que han de ser alimentados a un voltaje de entre 3.3v y 5v. En función del voltaje al que esté alimentado de igual intensidad serán sus respuestas. Por tanto, lo alimentaremos a 3.3v para no quemar el transmisor radio, que funciona a ese voltaje como máximo.
- Dos de sus pines son tierra o “ground”
- Se va a instalar también un botón de reseteo o “reset”. Lo que este hará será reiniciar el programa del microprocesador siempre que el

usuario lo presione. Este es activo en bajo, lo que significa que está siempre alimentado (3.3v) excepto cuando se pulsa el botón.

6.1.2. Módulo LoRa

Se ha modificado el cableado con respecto a la prueba de acuerdo a obtener una mayor eficiencia en el ruteo posterior, adjudicando a los pines del microprocesador una posición estratégica con respecto a los del módulo. Quedó así:

LoRa Module	Arduino Uno
MISO	D9
MOSI	D10
SLCK	D11
D100	D13
D101	D8
VCC	3.3 V
GND	GND
NSS	D12

Tabla 4. Cableado del módulo LoRa para el diseño de la PCB

6.1.3. Regulador de tensión

La alimentación que alimentará la PCB, todavía por decidir, si es superior a 5V deberá quedar reducida a 3.3V, por tanto se decidió incluir un regulador de tensión LM3480IM3-3.3/NOPB [27] con salida de 3.3V y una entrada de hasta 30V. Si finalmente el circuito se alimenta directamente a 3.3V este se podrá eliminar.

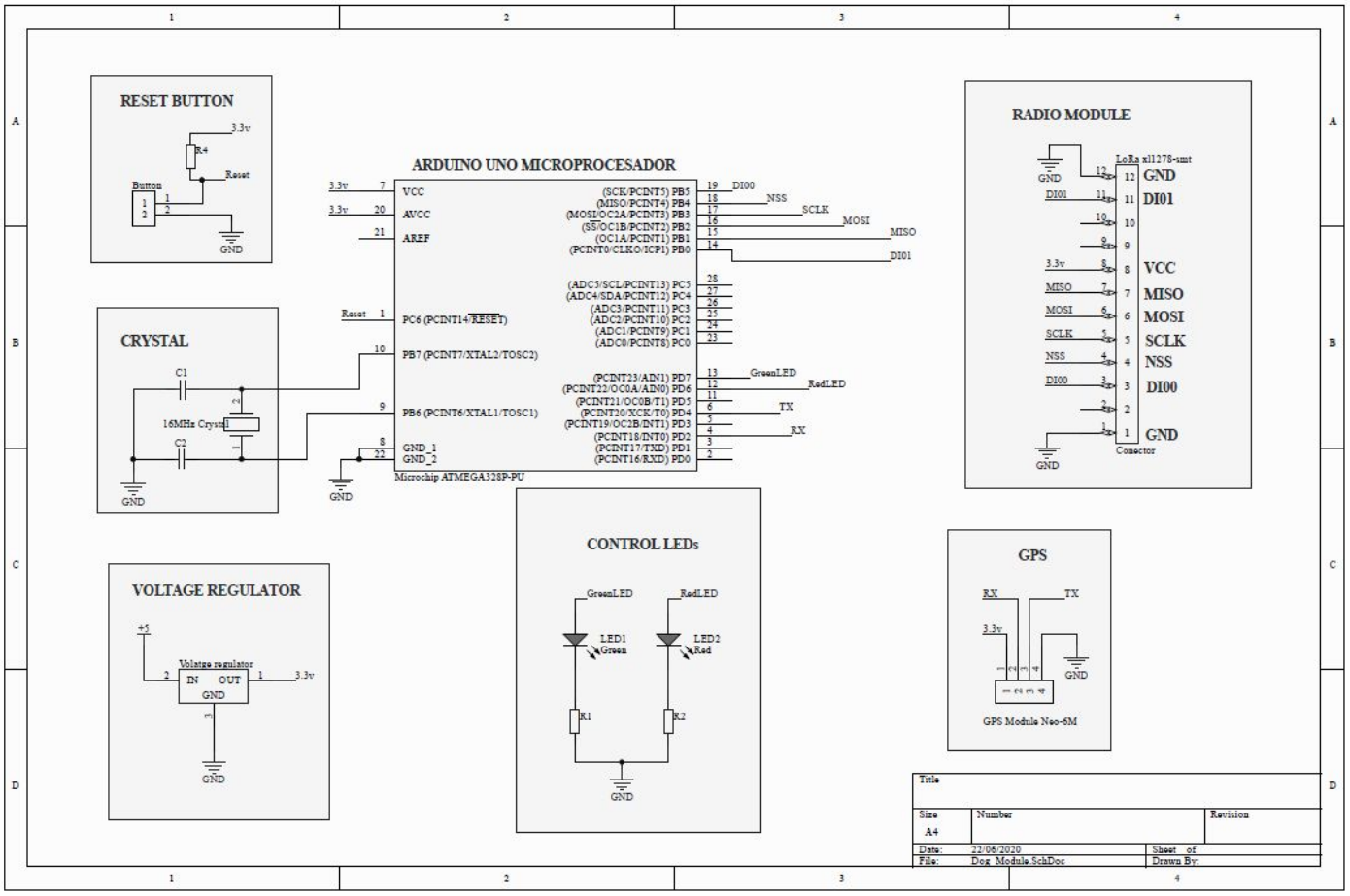
6.1.4. Resto de componentes

El resto de componentes se han ruteado por defecto, con la singularidad de que en el módulo del usuario, la alimentación deberá ser obligatoriamente 5V pues el módulo Bluetooth no funciona por debajo de esa tensión, lo cuál hace el regulador indispensable. El problema es mínimo pues el módulo que se ha de optimizar al máximo es el del animal, que lo ha de llevar en el collar y mucho peso lo haría molesto e incluso imposible de llevar por el mismo.

Los LEDs se han unido a unas resistencias de 100 ohmios para evitar quemaduras pero a su vez no perder potencia de luz, pues las resistencias son muy pequeñas.

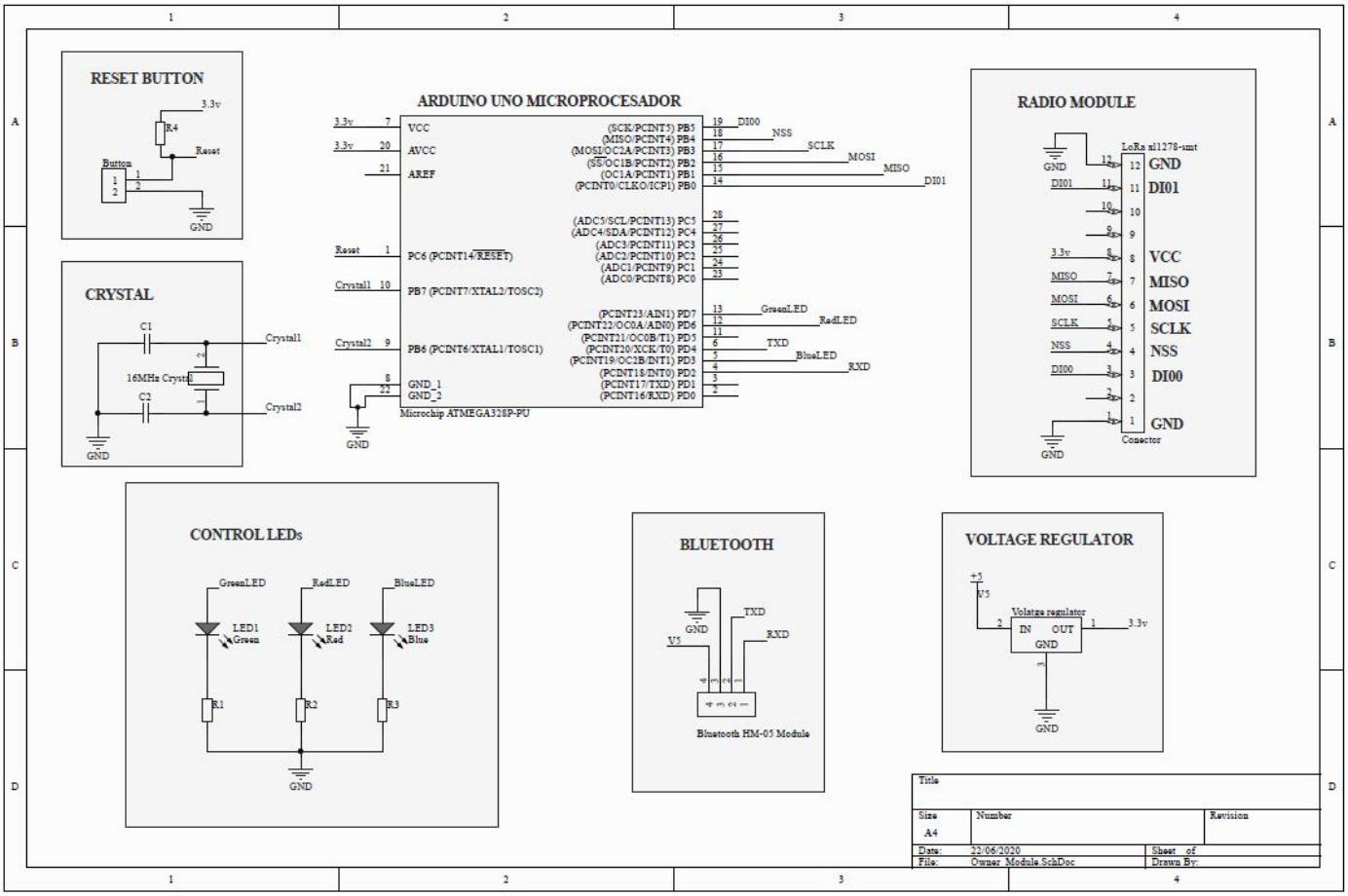
6.2. Esquemáticos

6.2.1. Esquemático del módulo del animal



Esquemático 1. Esquemático para el módulo del animal

6.2.2. Esquemático del módulo del usuario



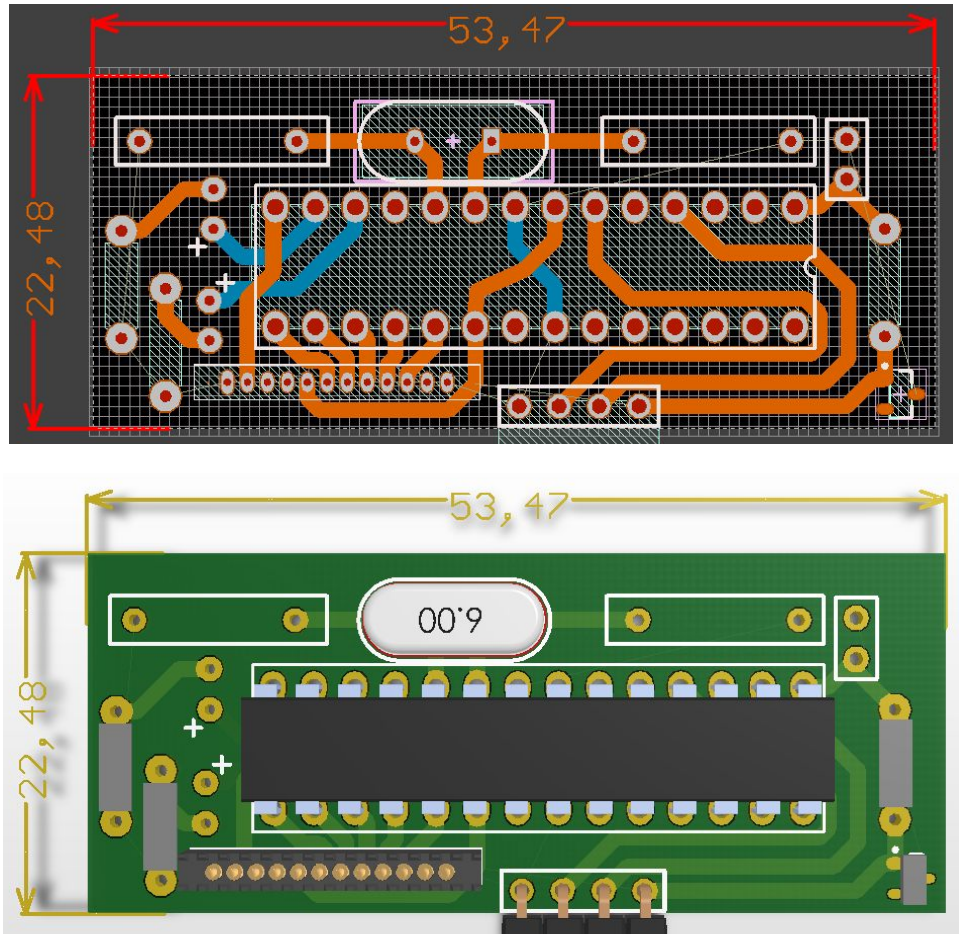
Esquemático 2. Esquemático para el módulo del usuario

6.3. PCBs

Como se ha comentado anteriormente, se han utilizado (siempre en la medida de lo posible) las mayores distancias entre rutas y medidas de las mismas como medidas de seguridad.

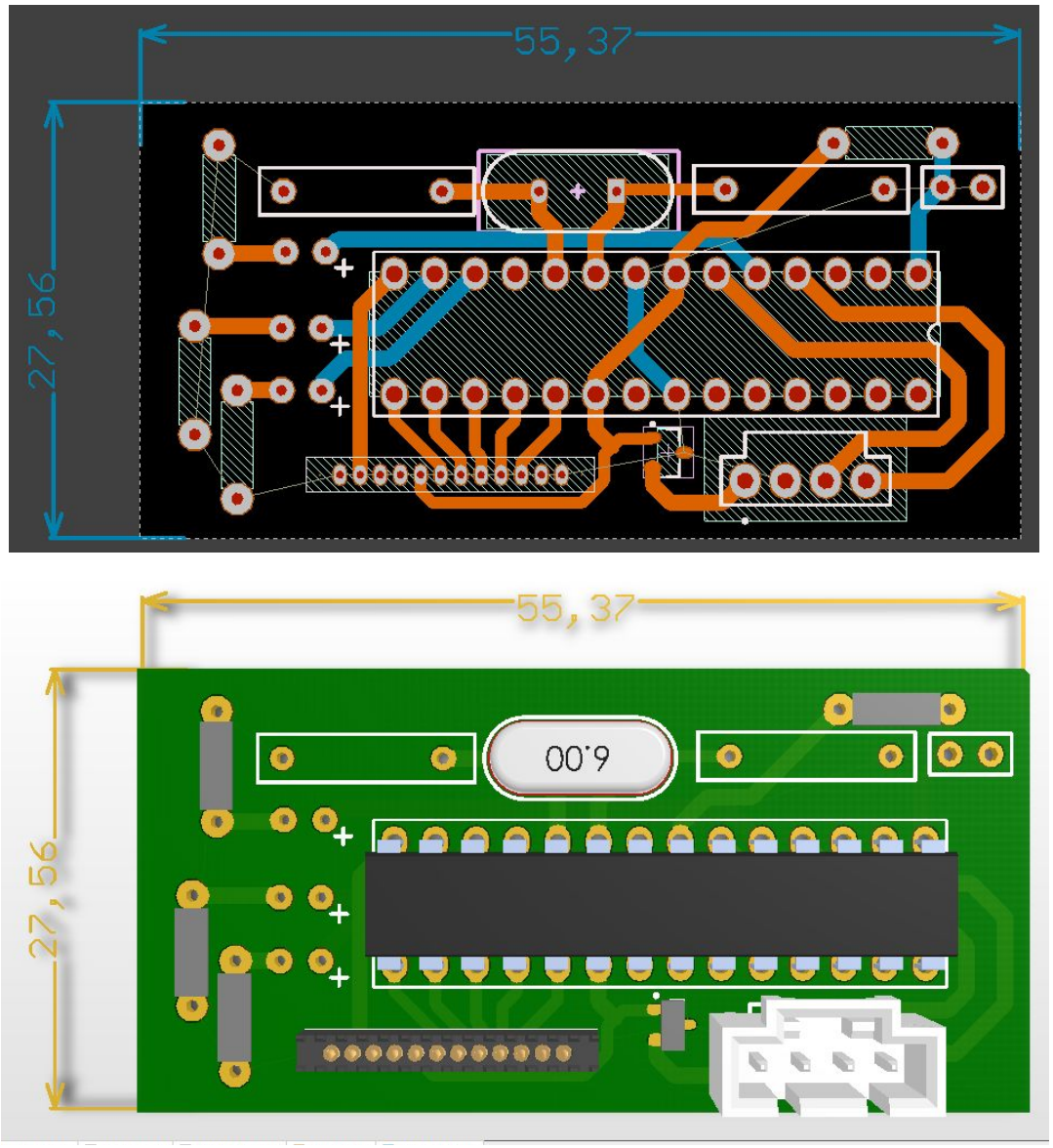
Las medidas finales fueron 1.016mm para todas las rutas exceptuando las que conectan el módulo LoRa pues este tiene sus pines a 1.27mm de separación entre ellos, en lugar del normalizado 2.54mm. Para esas rutas la anchura utilizada pasó a ser 0.762mm. Las PCBs resultantes quedaron con unas dimensiones de 55.37 x 27.56mm (la del usuario) y de 53.47 x 22.48mm (la del animal).

6.3.1. PCB del módulo del animal



PCB 1. Vistas 2D y 3D de la PCB del módulo del animal

6.3.2. PCB del módulo del usuario



PCB 2. Vistas 2D y 3D de la PCB del módulo del usuario

7. Conclusiones y trabajo futuro

Una vez terminado el trabajo, se pasa a analizar el cumplimiento de los objetivos establecidos al principio del mismo. Estos eran tres principalmente:

En primer lugar, se estableció como objetivo el esarrollo de dos programas, este caso en lenguaje Arduino, que sean capaces de controlar ambos módulos. De manera que, como se dibujó en la figura 4 o boceto inicial. Estos programas se han creado y comprobado que funcionan de manera eficiente y relativamente robusta o capaz de resolver errores como por ejemplo, el no envío o recepción de un mensaje en un primer intento.

Después, una vez el objetivo primero estaba cumplido, se pasó al desarrollo de la aplicación Android. Esta debía tener dos funciones principales: permitir al usuario la petición de coordenadas, y una vez recibidas mostrarlas; estas se muestran como texto por si el usuario prefiere usar otro sistema de navegación al que provee la misma, donde también las muestra. Además se ha añadido una función extra. Esta es básicamente un link al perfil LinkedIn del creador.

Por último, se diseñaron dos placas de circuito impreso para ambos módulos, utilizando la herramienta “Circuit Maker”. Estas se diseñaron siguiendo los esquemáticos también implementados en la misma herramienta.

De cara al futuro, se buscará crear un primer modelo funcional, que pasará por la creación de las PCBs y soldadura de los componentes en las mismas. Para ello, se deberá prestablecer la alimentación de las mismas. Una vez el dispositivo haya sido testado y comprobado su eficiencia, se pasará a la última optimización de tamaño y peso del mismo.

Como principal conclusión, puede que este proyecto abra un nuevo campo en el desarrollo de dispositivos de localización, pues hasta el momento ninguno de los existentes utilizaban la gratis tecnología radio en lugar de la convencional tecnología GSM.

8. ANEXOS

8.1. Fuentes de información / Referencias

Figura 2. Carteles de animales perdidos

[1]https://www.facebook.com/413718910520/photos/sebusca-perro-pug-perdido-losperrosdelcamino-pudahuel/10160705382500521/

[2]https://www.taringa.net/+mascotas/como-buscar-a-un-gato-perdido_12uy3q

[3]https://super4patas.com/5-consejos-practicos-cuando-se-busca-un/

Herramienta “Desygner”

[4] https://desygner.com/es/

Herramienta “Miro”

[5] https://miro.com/

Comparación de Arduinos y figuras 5, 6, 7, 8.

[6]https://comohacer.eu/analisis-comparativo-placas-arduino-oficiales-compatibles/

Figura 9. Modelo *Weenect* collar localizador para mascotas

[7]https://www.weenect.com/es/localizador-gps-para-perros-weenect-dogs.html

Módulo radio LoRa xl128-smt datasheet

[8]http://www.hr-wt.com/html_products/XL1278-SMT-59.html

Módulo radio NRF24L01 datasheet

[9]https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

Módulos radio FS1000A + XY-MK-5V datasheet
[10]http://www.mantech.co.za/Datasheets/Products/433Mhz_RF-TX&RX.pdf

Figura 11. Módulo Bluetooth datasheet / imagen
[11]https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf
[12]https://electronperdido.com/shop/comunicaciones-modulos-expansion/bluetooth-hc-05-con-pines/

Módulos GPS NEO-6x
[13]https://www.luisllamas.es/localizacion-gps-con-arduino-y-los-modulos-gps-neo-6/

Módulo GPS NEO-6M (GY-GPS6MV2) datasheet
[14]https://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/GY-NEO6MV2-GPS-Module-Datasheet.pdf

GPS Seeed Studio datasheet
[15]https://www.mouser.com/datasheet/2/744/Seeed_113020003-1217502.pdf

“TinyGPS”
[16]https://github.com/mikalhart/TinyGPS

Tutorial Módulo GPS
[17]https://naylampmechatronics.com/blog/18_Tutorial-M%C3%B3dulo-GPS-con-Arduino.html

“LoRaLib”
[18] https://github.com/jgromes/LoRaLib

Módulos LoRa testeo
[19] https://www.electroschematics.com/rf-radio-frequency/

Control de LED con MIT App Inventor
[20]https://www.youtube.com/watch?v=mUOLsNQ8Q1k&t=336s

Control de un potenciómetro con MIT App Inventor
[21]https://www.youtube.com/watch?v=j4Raxn5Fqxc

Link a la red social “LinkedIn”
[22]https://www.youtube.com/watch?v=onTsjOWNJw8

Placas PCB
[23] http://es.fastturnpcbs.com/blog/las-ventajas-y-desventajas-de-fpc_b23

Circuit Maker
[24] https://circuitmaker.com/

Tutorial Circuit Maker
[25]https://www.youtube.com/watch?v=AbKhfyLyh04

Atmega328p datasheet
[26] https://www.microchip.com/wwwproducts/en/ATmega328p

Regulador de tensión
[27]https://www.digikey.es/product-detail/es/texas-instruments/LM3480IM3-3.3-NOPB/LM3480IM3-3.3-NOPBCT-ND/270750

BOOM del proyecto CircuitMaker
https://drive.google.com/drive/folders/1qZmheD9MGWPYjNYzBJXY5IHBDJoVDmKG?usp=sharing

Enlace CircuitMaker para acceder al proyecto

https://workspace.circuitmaker.com/Projects/Details/Alfonso-Forcen/Dog-collar

8.2. Códigos

[1] Código “simple_test” ligeramente modificado de la librería TinyGPS by Mikal Hart. Se puede ver y descargar desde http://arduiniana.org/libraries/tinygps/

```
#include <SoftwareSerial.h>

#include <TinyGPS.h>

/* This sample code demonstrates the normal use of a TinyGPS object.
   It requires the use of SoftwareSerial, and assumes that you have a
   4800-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
*/

TinyGPS gps;
SoftwareSerial ss(4, 3);

void setup()
{
  Serial.begin(9600);
  ss.begin(9600);

  Serial.print("Simple TinyGPS library v. ");
  Serial.println(TinyGPS::library_version());
  Serial.println("by Mikal Hart");
  Serial.println();
}

void loop()
{
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;

  // For one second we parse GPS data and report some key values
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (ss.available())
    {
      char c = ss.read();
      // Serial.write(c); // uncomment this line if you want to see the GPS data flowing
      if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }

  if (newData)
  {
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
```

```
    Serial.print("LAT=");
    Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flat, 6);
    Serial.print(" LON=");
    Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
flon, 6);
    Serial.print(" SAT=");
    Serial.print(gps.satellites() ==
TinyGPS::GPS_INVALID_SATELLITES ? 0 : gps.satellites());
    Serial.print(" PREC=");
    Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 :
gps.hdop());
  }

  gps.stats(&chars, &sentences, &failed);
  Serial.print(" CHARS=");
  Serial.print(chars);
  Serial.print(" SENTENCES=");
  Serial.print(sentences);
  Serial.print(" CSUM ERR=");
  Serial.println(failed);
  if (chars == 0)
    Serial.println("** No characters received from GPS: check
wiring **");
}
  delay(400);
```

[2] Códigos “receive” (en primer lugar) y “transmit” (en segundo lugar) ligeramente modificados de la librería LoRaLib. Se puede descargar desde https://github.com/jgromes/LoRaLib y ver su documentación en https://jgromes.github.io/LoRaLib/index.html.

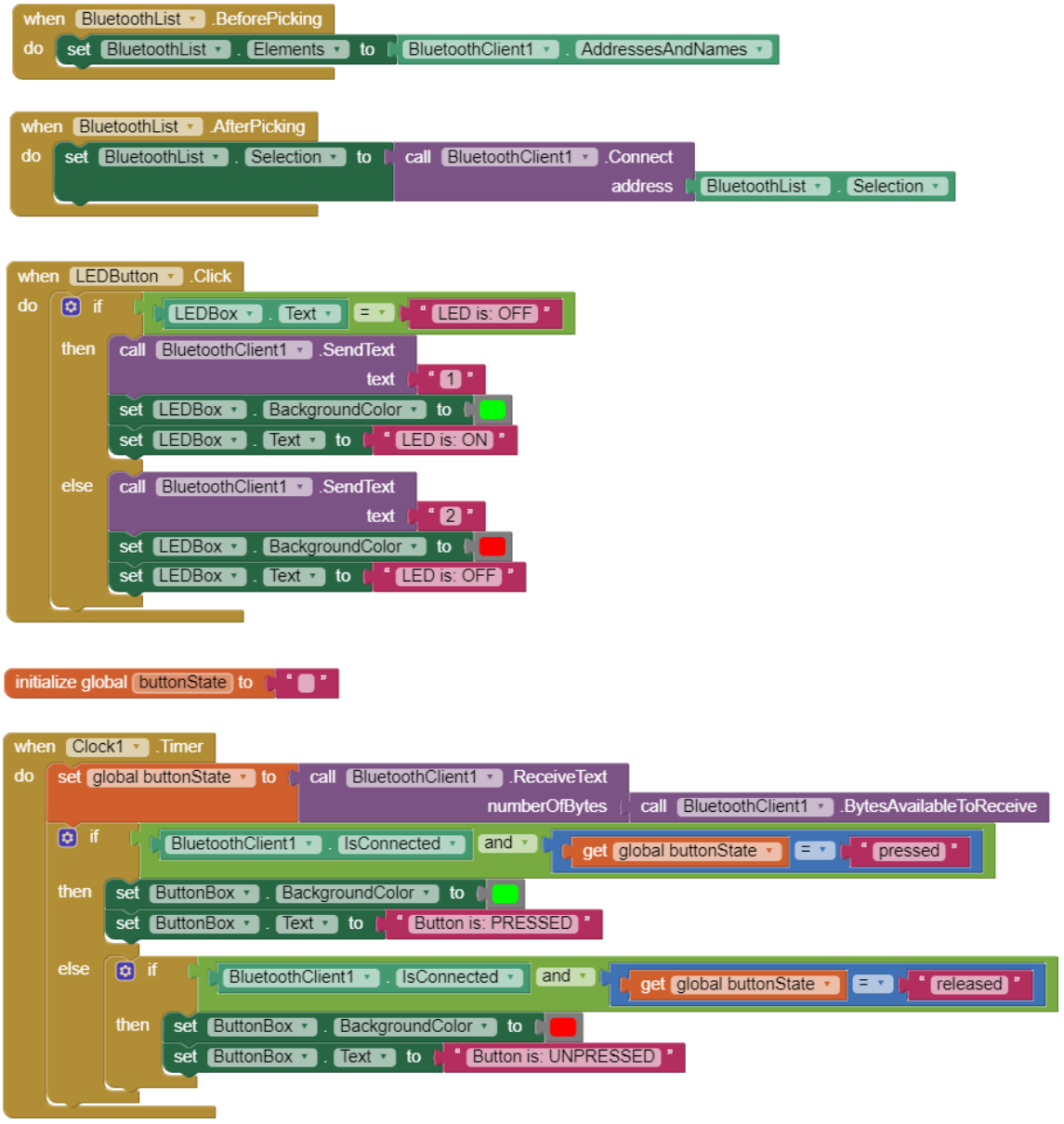
```
// include the library
#include <LoRaLib.h>

// create instance of LoRa class using SX1278 module
// this pinout corresponds to RadioShield

SX1278 lora = new LoRa;

void setup() {
  Serial.begin(9600);

  // initialize SX1278 with default settings

  Serial.print(F("Initializing ... "));
  int state = lora.begin();
  if (state == ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
```

```
    while (true);
  }
}

void loop() {
  Serial.print(F("Waiting for incoming transmission ... "));

  String str;
  int state = lora.receive(str);

  if (state == ERR_NONE) {
    // packet was successfully received
    Serial.println(F("success!"));

    // print data of the packet
    Serial.print(F("Data:\t\t\t"));
    Serial.println(str);

    // print RSSI (Received Signal Strength Indicator)
    // of the last received packet
    Serial.print(F("RSSI:\t\t\t"));
    Serial.print(lora.getRSSI());
    Serial.println(F(" dBm"));

    // print SNR (Signal-to-Noise Ratio)
    // of the last received packet
    Serial.print(F("SNR:\t\t\t"));
    Serial.print(lora.getSNR());
    Serial.println(F(" dB"));

    // print frequency error
    // of the last received packet
    Serial.print(F("Frequency error:\t"));
    Serial.print(lora.getFrequencyError());
    Serial.println(F(" Hz"));

  } else if (state == ERR_RX_TIMEOUT) {
    // timeout occurred while waiting for a packet
    Serial.println(F("timeout!"));

  } else if (state == ERR_CRC_MISMATCH) {
    // packet was received, but is malformed
    Serial.println(F("CRC error!"));

  }
 delay(200);
}
```

```
// include the library

#include <LoRaLib.h>

// create instance of LoRa class using SX1278 module
// this pinout corresponds to RadioShield
SX1278 lora = new LoRa

void setup() {
  Serial.begin(9600);
```

```
10.
11.  // initialize SX1278 with default settings
12.
13.  Serial.print(F("Initializing ... "));
14.  int state = lora.begin();
15.  if (state == ERR_NONE) {
16.    Serial.println(F("success!"));
17.  } else {
18.    Serial.print(F("failed, code "));
19.    Serial.println(state);
20.    while (true);
21.  }
22.}
23.
24.void loop() {
25.  Serial.print(F("Sending packet ... "));
26.
27.  int state = lora.transmit("Hello World!");
28.
29.  if (state == ERR_NONE) {
30.    // the packet was successfully transmitted
31.    Serial.println(F(" success!"));
32.
33.    // print measured data rate
34.    Serial.print(F("Datarate:\t"));
35.    Serial.print(lora.getDataRate());
36.    Serial.println(F(" bps"));
37.
38.  } else if (state == ERR_PACKET_TOO_LONG) {
39.    // the supplied packet was longer than 256 bytes
40.    Serial.println(F(" too long!"));
41.
42.  } else if (state == ERR_TX_TIMEOUT) {
43.    // timeout occurred while transmitting packet
44.    Serial.println(F(" timeout!"));
45.
46.  }
47.
48.  // wait before transmitting again
49.  delay(200);
50.}
```

[3] Aplicación Android (en primer lugar) y código Arduino (en segundo lugar) para el testeo básico realizado con pulsador y botón para enchufar los LEDs respectivos. La aplicación fue creada con la herramienta “MIT app Inventor 2” (https://appinventor.mit.edu/)




```
1. // Pin definitions
1. const int LED = 3;
2. const int BUTTON = 4;
3.
4.
```

```
5. //Setup
6. void setup() {
7.
8.   Serial.begin(9600);
9.
10.  pinMode(LED, OUTPUT);
11.  pinMode(BUTTON, INPUT);
12.
13.}
14.
15.//Variables
16.int buttonState = 0;
17.int serial = 0, oldserial = 0;
18.
19.//Loop
20.void loop() {
21.
22.  //We receive data from the app
23.
24.  serial = Serial.available();
25.
26.  //LED control -> phone to module
27.  if ((serial > 0) && (newSerial(serial, oldserial))) {
28.
29.    serial = serial%2;
30.
31.    if (serial == 0) {
32.
33.      digitalWrite(LED, LOW);
34.
35.    } else {
36.
37.      digitalWrite(LED, HIGH);
38.
39.    }
40.
41.  }
42.
```

```
43.  //Button tracker -> module to phone
44.  if ((buttonState == 0) && (digitalRead(BUTTON) == HIGH)) {
45.
46.    Serial.print("pressed");
47.    buttonState = 1;
48.
49.  }
50.
51.  else if ((buttonState == 1) && (digitalRead(BUTTON) == LOW)) {
52.
53.    Serial.print("released");
54.    buttonState = 0;
55.
56.  }
57.
58.
59.  delay(400);
60.}
61.
62.
63.
64.//Function to detect if there's a new input
65.boolean newSerial (int serial, int oldserial) {
66.
67.  if (serial != oldserial) {
68.
69.    oldserial = serial;
70.    return true;
71.
72.  }
73.
74.  return false;
75.
76.}
```

[4] Código Arduino definitivo, creado basada en un sistema con funciones y un loop reducido.

1. Función void makeLEDBlink (int LED)

```
1. void makeLEDBlink (int LED) {
2.
3.
4.      //Make green LED blink to show success
5.      int i = 0;
6.      digitalWrite(redLED, LOW);
7.      for(i = 0; i < 3; i++) {
8.
9.        digitalWrite(LED, HIGH);
10.       delay(200);
11.       digitalWrite(LED, LOW);
12.       delay(200);
13.
14.     }
15.     digitalWrite(redLED, HIGH);
16.
17.}
```

2. Función void checkLEDs()

```
1. void checkLEDs() {
2.
3.      makeLEDBlink(redLED);
4.      makeLEDBlink(greenLED);
5.      makeLEDBlink(blueLED);
6.
7. }
```

3. Función void setup()

```
1. void setup() {
2.    //Serial
```

```
3.    Serial.begin(9600);
4.
5.    //Bluetooth
6.    Bluetooth.begin(9600);
7.    pinMode(RxD, INPUT);
8.    pinMode(TxD, OUTPUT);
9.
10.   //Initialize LoRa
11.   initializeLoRa ();
12.
13.   //Pins
14.   pinMode(redLED, OUTPUT);
15.   pinMode(greenLED, OUTPUT);
16.   pinMode(blueLED, OUTPUT);
17.   digitalWrite(redLED, HIGH);
18.   digitalWrite(greenLED, LOW);
19.   digitalWrite(blueLED, LOW);
20.
21.   //Be sure every single LED is working
22.   checkLEDs();
23.
24. }
```

4. Función void loop() en el módulo del usuario

```
1. void loop() {
2.
3.    //Check if phone is requesting and save the value
4.    serial = Bluetooth.available();
5.
6.    //If so
7.    if (newSerial(serial, oldserial)) {
8.
9.      //Blink blue LED -> received
10.     makeLEDBlink(blueLED);
11.
12.     //Send the request to collar
13.     sentString = "request";
14.     success = false;
15.     Serial.println("Sending radio request.");
16.     while (!success) {
17.
18.       success = sendLoRa();
19.
20.     }
21.
22.     Serial.print("Message sent: ");
23.     Serial.println(sentString);
24.
25.     //Wait to receive the answer from the collar
26.     success = false;
27.     Serial.println("Waiting for receive GPS.");
28.     while (!success) {
29.
30.       success = receiveLoRa();
```

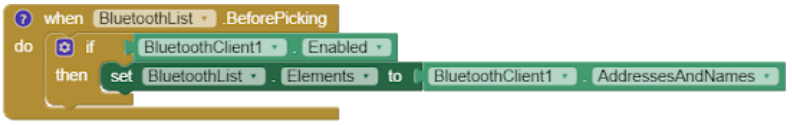
```
31.
32.     }
33.
34.     Serial.print("Coordinates received: ");
35.     Serial.println(receivedString);
36.
37.     //Send the coordinates to the Android app
38.     Bluetooth.print(receivedString);
39.
40.     //Blink blue LED
41.     makeLEDBlink(blueLED);
42.
43.   }
44.
45.   delay(200);
46.
47.}
```

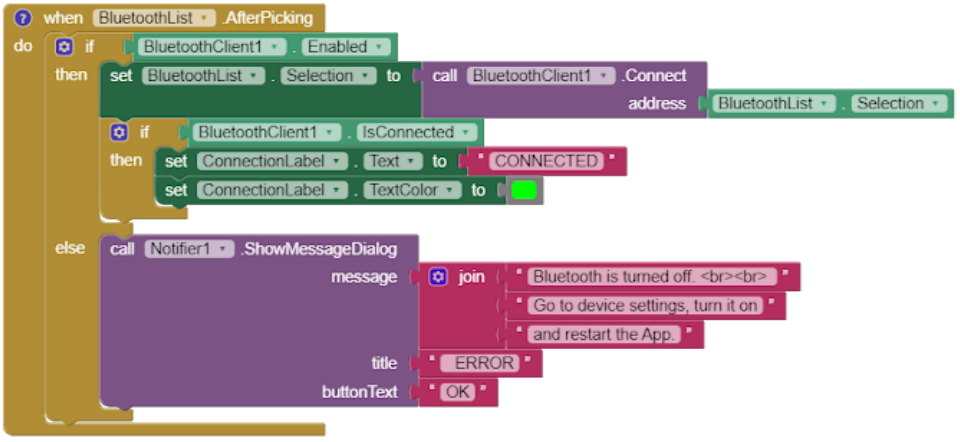
5. Función void loop() en el módulo del animal

```
1. void loop() {
2.
3.   //Turn red LED on to sshow that the module is working
4.   digitalWrite(redLED, HIGH);
5.
6.   //Check if user is requesting
7.   success = false;
8.   while (!success) {
9.
10.    success = receiveLoRa();
11.    delay(400);
12.
13.  }
14.
15.  Serial.print("Received String: ");
16.  Serial.println(receivedString);
17.
18.  //If it gets a request, then it looks for and send the
   coordinates
19.  if (receivedString.equals("request")) {
20.
21.    //Get the GPS coordinates
22.    success = false;
23.    while (!success) {
24.
25.      success = getGPSCoordinates();
26.
27.    }
28.
29.    //Create the String response (sentString)
30.    //For that, it multiplies by one million to not reduce the
   number of decimals
31.    sentString.concat(flat * 1000000);
32.    sentString.concat(" ");
33.    sentString.concat(flon * 1000000);
34.
35.    //And send it after waiting 1.5seconds to be sure it is ready
```

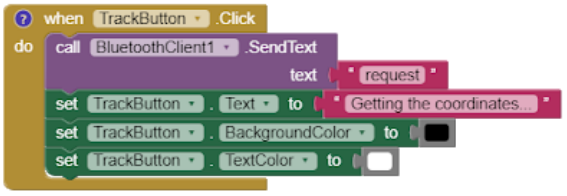
```
    delay(1500);
    Serial.print("Sending String: ");
    Serial.println(sentString);

    success = false;
    while (!success) {

      success = sendLoRa();
      delay(400);

    }

  }

  delay(200);

}
```

[5] Aplicación Android definitiva

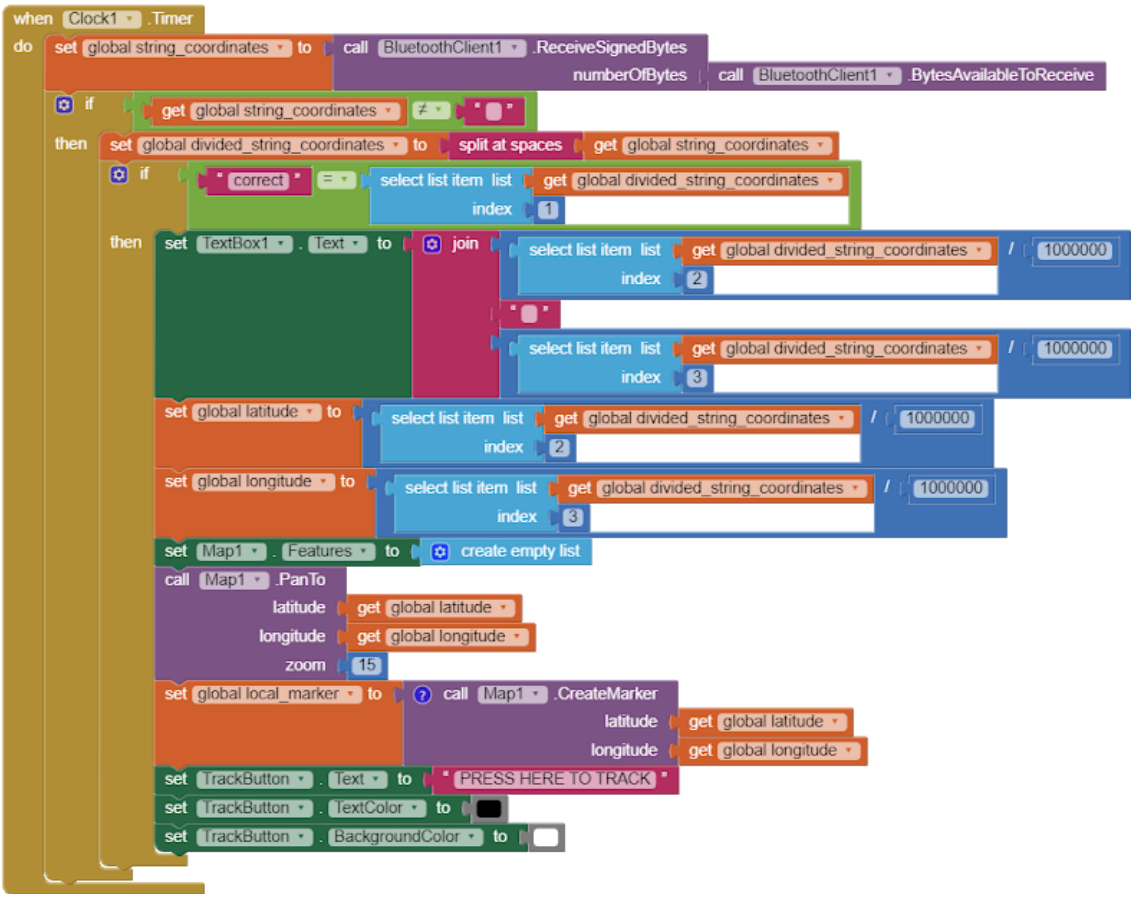





when ContactButton .Click
do set ActivityStarter1 . DataUri to " https://www.linkedin.com/in/alfonso-forc%C3%A9n-... "
set ActivityStarter1 . Action to " android.intent.action.VIEW "
call ActivityStarter1 .StartActivity


when TrackButton .Click
do call BluetoothClient1 .SendText
text " request "
set TrackButton . Text to " Getting the coordinates... "
set TrackButton . BackgroundColor to
set TrackButton . TextColor to


initialize global latitude to 0
initialize global local_marker to " "
initialize global string_coordinates to " "
initialize global longitude to 0
initialize global dog_marker to " "
initialize global divided_string_coordinates to " "


when Clock1 .Timer
do set global string_coordinates to call BluetoothClient1 .ReceiveSignedBytes
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
if get global string_coordinates ≠ " "
then set global divided_string_coordinates to split at spaces get global string_coordinates
if " correct " = select list item list get global divided_string_coordinates
index 1
then set TextBox1 . Text to join select list item list get global divided_string_coordinates / 1000000
index 2
" "
select list item list get global divided_string_coordinates / 1000000
index 3
set global latitude to select list item list get global divided_string_coordinates / 1000000
index 2
set global longitude to select list item list get global divided_string_coordinates / 1000000
index 3
set Map1 . Features to create empty list
call Map1 .PanTo
latitude get global latitude
longitude get global longitude
zoom 15
set global local_marker to call Map1 .CreateMarker
latitude get global latitude
longitude get global longitude
set TrackButton . Text to " PRESS HERE TO TRACK "
set TrackButton . TextColor to
set TrackButton . BackgroundColor to