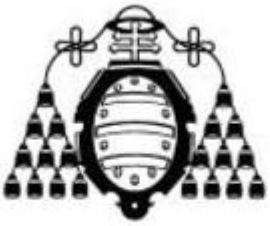
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Aplicación para la ayuda al muestreo de fauna salvaje

DIRECTOR: José Ramón Arias García

AUTOR: Eduardo Lamas Suárez

Vº Bº del Director del Proyecto

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE GRADO

19 de noviembre de 2020

Quién suscribe:

Eduardo Lamas Suárez

Hace constar que es el autor del trabajo:

Aplicación para la ayuda de muestreo de fauna salvaje.

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.: Eduardo Lamas Suárez.

Agradecimientos

Quiero expresar mi agradecimiento a todas y cada una de las personas que de una forma u otra han participado en la realización de este TFG.

En primer lugar, a mi tutor José Ramón Arias García, por todo el apoyo recibido a lo largo de este proyecto, que con su dedicación ha conseguido sacar lo mejor de mí.

Por supuesto, a la profesora de la Universidad de Lancaster, la bióloga Rosa Menéndez Martínez ya que sin sus aportes y consejos no hubiera podido enfocar y desarrollar correctamente este TFG.

A mi familia, por apoyarme desde pequeño para que cumpliera mi vocación, porque sin ellos esto no habría sido posible.

Y por último a todos los magníficos compañeros que he tenido, entre todos hemos conseguido superar las dificultades que encontrábamos en el camino.

Abstract

El ejercicio de muestreo/conteo de fauna salvaje, es una de las actividades más comunes entre los etólogos y aficionados de la fauna y flora, sin embargo, esta actividad sigue realizándose como se hacía hace años, apenas sí se ha actualizado a las nuevas tecnologías. Este proyecto plantea una posible solución a este problema gracias al uso de dispositivo portátil auxiliar de toma de muestras ambientales construido entorno a un microcontrolador ESP32 y una aplicación Android con la que recibir estos datos, almacenarlos durante el registro de un animal en una base de datos y la exportación de la base de datos a un archivo u otra plataforma.

Índice de Contenido

Índice de Figuras

1. Introducción

Hoy en día, quizás agravado por la actual situación de pandemia por la Covid-19, se ha podido observar que la era del papel y lápiz está llegando a su fin. La tendencia actual lleva a una digitalización de las actividades, ya sea por necesidad como es el caso de la pandemia o por eficiencia. Entre los ejercicios que todavía no se han digitalizado, está el de muestreo/conteo de fauna salvaje.

Este muestreo/conteo se realiza sobre una población silvestre concreta que se identifica como un grupo de individuos que pueblan una superficie determinada. La certeza absoluta sobre el tamaño de estas poblaciones solo se consigue con el conteo de todos los individuos que las componen. Sin embargo, esta metodología es inviable por lo que normalmente los biólogos acuden a métodos indirectos que les permiten calcular el tamaño poblacional. Estos métodos varían según las características propias de la especie que se quiere investigar y del hábitat donde se desarrolla el estudio. Así la densidad y la variedad de especies se calculan, normalmente, con las técnicas de transecto [1] y de muestreo.

El muestreo consiste en la toma de una muestra representativa de la población a estudiar de la que se extrapolarán los datos. Es por lo que la muestra debe de estar compuesta de un número adecuado de transectos repartidos de un modo correcto. Así el transecto es la unidad de muestreo donde se va a desarrollar el registro de los datos que debe ser trazado y dimensionado en función, tanto de la fisiografía y extensión del terreno, como de las características concretas de la especie a estudiar. Todo ello debe ser previamente definido.

La elección del transecto es uno de los elementos claves para la correcta ejecución del muestreo, existen dos tipos fundamentales de transectos:

- Transecto de ancho fijo: el ancho del transecto se encuentra fijado de antemano y se van registrando todos los individuos que se van encontrando. Este es el método más adecuado para la determinación de densidades en poblaciones de mamíferos de tamaño mediano o grande ya que se debe estar seguro de que se encontrarán todos los individuos existentes en la banda.

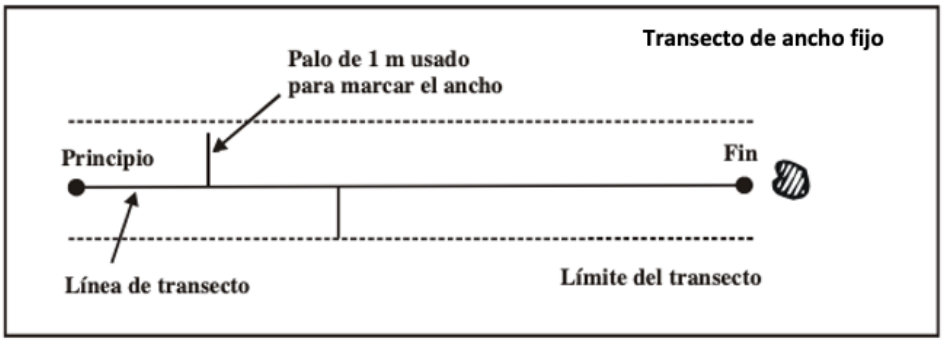


Ilustración 1:Transecto de ancho fijo según el manual de Rabinowitz (2003)

- Transecto lineal: es adecuado para registrar animales difíciles de observar bien por su pequeño tamaño o por ser animales fácilmente asustables. El ancho de la banda se recaba a partir de cada observación tomando las distancias desde el observador al animal avistado o la distancia perpendicular desde el animal al transecto.

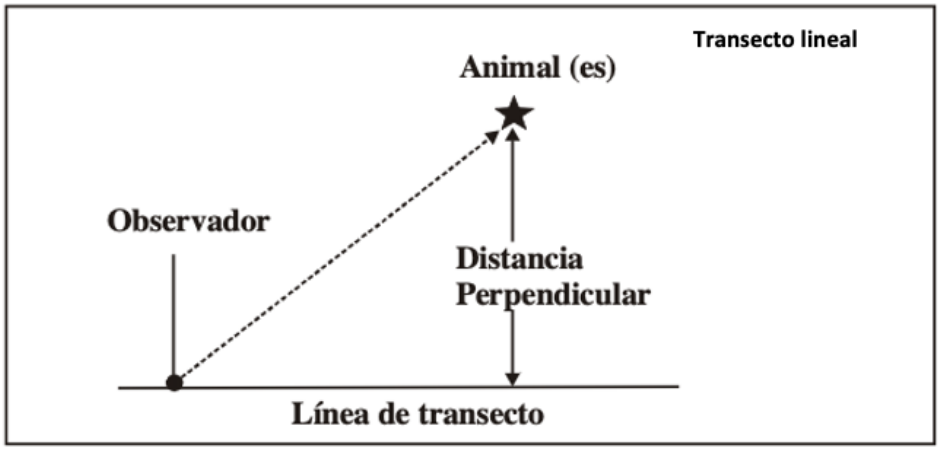


Ilustración 2: Transecto lineal según el manual de Rabinowitz (2003)

Por tanto, toda investigación ecológica se sustenta en la medición de parámetros de los distintos organismos que viven en un determinado medio, el conteo o muestreo. La fiabilidad del conteo se basa en el número de muestras y su distribución tanto espacial como temporal por lo que la digitalización de esta labor simplificaría en gran medida el trabajo no solo de los especialistas sino también de cualquier aficionado que realice esta tarea. Esta actividad, común entre biólogos y amantes de la naturaleza, se realiza siguiendo unas técnicas determinadas, como puede ser, por ejemplo, el conteo por puntos para aves o los transectos para los animales huidizos [2]. Pero, independientemente de la técnica empleada todos siguen un procedimiento común:

1. El observador llega a la zona de muestreo.
2. Aplica la técnica seleccionada.
3. Apunta los datos necesarios en papel o en una nota de móvil.
4. Traspasa los datos a mano a otra aplicación como puede ser Microsoft Excel o Drive.

		Transectos							
Variables		1	2	3	4	5	6	7	8
Fecha									
Localidad									
Hora: (inicio)-(término)									
Hábitat									
Banda transecto	Largo								
	Ancho								
Condiciones meteorológicas									

Ilustración 3:Ejemplo de planilla para el registro de los datos de conteos en bandas transectos.

El equipo usado, hasta ahora, por los aficionados para el registro de los avistamientos se compone básicamente de lápiz y papel. Pero, sin embargo, la lista del equipo aconsejado para los profesionales por el profesor Rabinowitz en su manual es bastante más extensa y da idea de lo complejo de este proceso.

LISTA DEL EQUIPO UTILIZADO EN EJERCICIOS DE CAPACITACIÓN

Equipo de Campo
(en orden alfabético)
Altímetro
Anteojo meridiano (de bolsillo)
Cinta señaladora y cintas de colores
Binoculares
Brújulas
Cámaras
Cintas métricas (1m, 30m, 50m)
Clinómetros
Equipo de radiotelemetría*
Grabaciones de vocalizaciones de animales
Grabadora
Guías de campo/Listas de registro para fauna silvestre local
Lápices/plumas (bolígrafos)
Libretas o cuadernos
Linternas
Mapas de:
- Sitio del ejercicio: topográfico (1:50,000, 1:100,000); de hábitats y vegetación
- País completo o de grandes regiones del mismo
- Mapas generales: mapas satelitales, mapas planimétricos, mapas fotográficos

Olores de animales (o productos sustitutos como aceite de sardina, aceite de hígado de bacalao y esencias de olores frutales)
Redes grandes para capturar animales
Reflectores (para vehículos)
Reloj pulsera
Telémetros (Range finders)
Trampas de cámara
Trampas para mamíferos pequeños*
Yeso

Equipo para salón de clase
Especímenes de animales (colección mixta de pieles, cráneos y especímenes disecados de varios grupos taxonómicos de mamíferos y aves)
Etiquetas de museo/fichas
Generador (para proveer corriente eléctrica)
Pizarra
Proyector de diapositivas
Retroproyector
Reglas/transportadores
Televisor y videocasetera

Nota: ***Únicamente papel y lápiz son absolutamente necesarios para el uso de este manual o la realización de un curso de campo.***

*Únicamente para cursos avanzados o especializados

Ilustración 4: Equipo utilizado para la capacitación según el manual de Rabinowitz (2003)

Por todo lo anterior, se puede concluir que el procedimiento no está eficientemente adaptado a las nuevas tecnologías o directamente no las usa, es por esto que se ha visto la necesidad de realizar este proyecto que pretende subsanar estas limitaciones digitalizando todo el proceso de muestreo. Un sistema que recoja la temperatura, la humedad, la presión, la latitud y longitud, la altitud y los índices UV de manera automática y eficiente que facilite la labor de los equipos de los investigadores de campo con la menor equipación posible. Para ello se ha buscado desarrollar un sistema, llamado **GeoFauna**, compuesto por dos elementos principales:

- Un dispositivo portátil auxiliar de toma de muestras ambientales construido entorno a un microcontrolador ESP32, al que se conectará una serie de sensores que medirán distintos valores como puede ser la presión o temperatura.
- Una aplicación Android en la que se recibir los datos obtenidos por el dispositivo de toma de muestras ambientales y con la que registrar los animales en una base datos y exportar esta base de datos a un archivo o a otra plataforma.

A lo largo de esta memoria, se especificará el proceso de creación de GeoFauna, empezando por la necesidad de este recurso, las diferentes etapas de diseño y posterior evaluación del sistema desarrollado en este Trabajo Fin de Grado.

1.1 Estructura del equipo

El equipo que realizara este trabajo fin de grado está formado por:

- Un desarrollador: Eduardo Lamas Suárez (Universidad de Oviedo)
- El director del TFG: José Ramón Arias García (Universidad de Oviedo)
- La bióloga como consultora externa: Rosa Menéndez Martínez (Universidad de Lancaster)

1.2 Aspectos teóricos

1.2.1 Bluetooth Low Energy

Para la comunicación entre los dos elementos principales, el dispositivo portátil auxiliar de toma de muestras ambientales y la aplicación Android, se ha decidido utilizar el "Bluetooth Low Energy (BLE)" [3] [4], subconjunto del estándar "Bluetooth v4.0" que fue introducido por primera vez en el proyecto "Wibree" desarrollado por Nokia en 2010. Se escogió este método de comunicación debido a los beneficios que aporta al sistema:

- Tiene un bajo consumo para un largo alcance.
- Opera a 2.4Gh.
- El proceso de conexión y transmisión de datos es muy rápido.
- Los dispositivos BLE tiene un precio asequible.

Pero como todas las tecnologías, tiene sus inconvenientes:

- Al utilizar la misma frecuencia que el WiFi y que otros dispositivos bluetooth, puede estar sometido a interferencias y latencia.
- Solo se puede enviar paquetes de datos. Lo que dificultaría la transmisión de imágenes u otro tipo de medida.

Existen dos tipos de roles dentro de estos sistemas de comunicación BLE, los periféricos y los dispositivos centrales:

- Los dispositivos centrales son aquellos dispositivos con alta capacidad de procesamiento que inician los comandos que en nuestro proyecto será la aplicación Android instalada en el móvil o Tablet.
- Los periféricos, son a diferencia de los centrales, los dispositivos que disponen de poca potencia y recursos, este tipo de dispositivo se conecta a un dispositivo central, en este caso el microcontrolador portátil conectado a los sensores que medirán los datos medioambientales.

Estos dos tipos de roles son los que van a ser utilizados durante las dos etapas de este tipo de comunicación la fase del **Generic Access Profile (GAP)** y la del **Generic Access Attribute Profile (GATT).**

Generic Access Profile (GAP)

El Generic Access Profile (GAP) es el encargado de manejar las conexiones y anuncios en BLE, es decir, lo que permite que un dispositivo sea visible para otros y establezca como se van a comunicar dos elementos. Además, es el que define los roles anteriormente descritos.

Dentro del GAP existen dos mecanismos disponibles para transmitir información: el Advertising Data payload y el Scan Response payload.

- El **Advertising Data payload**, es generado y emitido a intervalos regulares ajustables por los periféricos. Este tipo de payload se utiliza para mostrar la presencia y estado del dispositivo. Entre lo más importante de esta información:
  - Es escaneable y conectable o no.
  - También cabe mencionar que cuando el Advertising Data Payload se hace en grandes intervalos de tiempo, la comunicación pierde reactividad, pero gana en ahorro de batería.
- **Scan Response payload**, es creado por los dispositivos centrales con el fin de solicitar información adicional a un periférico si este lo permite.

Ambos Payloads pueden estar formados por un máximo de 31 bits, pero solo el Advertising Data Payload es obligatorio.

Generic Access Attribute Profile (GATT)

El Generic Access Attribute Profile se utiliza después de haber utilizado el GAP, a través de este "profile" dos dispositivos BLE pueden comunicarse entre sí usando Servicios y Características. Este acto de comunicación emplea el conocido Attribute Protocol (ATT).

El ATT es simplemente un protocolo que utiliza unidades de información llamadas Atributos, los cuales se podría decir que están formados por tres elementos:

- Un identificador de 16 bits.
- Un tipo, que en realidad es un UUID de 16, 32 o 128 bits, que determina el tipo de dato que se presenta
- Un valor de un tamaño máximo de 512 bytes.

Utilizando ATT, el GATT organizar los datos jerárquicamente de tal manera que el acceso a los valores es más práctico y reusable. Esta jerarquía está formada por un perfil compuesto de servicios, los cuales a su vez estos están formados por características.

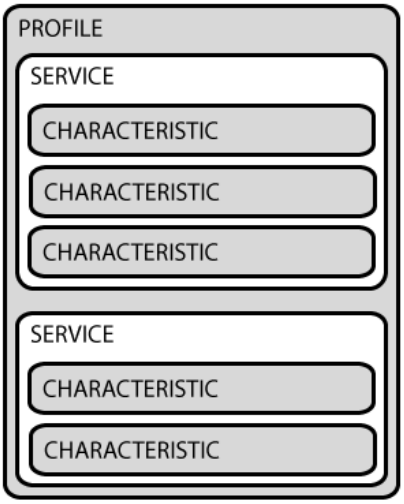


Ilustración 5: Objeto anidado producido en una transacción GATT

Durante esta segunda fase se produce una comunicación cliente/servidor, donde el periférico hace el papel de servidor GATT, el encargado de almacenar todos los datos siguiendo la jerarquía anteriormente descrita, y el cliente GATT es el dispositivo central, aquel dispositivo que realiza peticiones al servidor GATT para el acceso a estos datos.

Siguiendo con la jerarquía anteriormente descrita, quedaría explicar en qué consiste cada componente:

- El perfil (Profile): es simplemente la agrupación de todos los servicios que contiene un periférico, este normalmente viene definido por el fabricante o por Bluetoooth SIG.
- Los servicios (Service): Son divisiones lógicas del perfil, en los que se agrupan características relacionas en una misma sección, son identificados por UUIDs de 16-bits o 128-bits.
- Las características (Characteristics): estos elementos es el nivel donde se almacena un dato concreto de un tamaño máximo de 20 bytes. Al igual que los servicios se identifican por UUIDs de 16-bits o 128-bits.

1.2.2 Programación reactiva

La programación reactiva es un paradigma de programación basado en el uso del patrón de diseño Observer, el paradigma funcional y en el LINQ (Language Integrated Query) [5], con el fin de procesar flujos de eventos/datos de manera asíncrona. Esto la convierte en un instrumento adecuado para desarrollar programas y aplicaciones reactivas, resilientes y fácilmente escalables.

Podemos analizar su utilidad examinando la siguiente fórmula que constituye un típico ejemplo para explicar este tipo de programación:

$$a = b + c$$

Siendo las variables ***a***, ***b*** y ***c*** números, se puede observar que ***a*** depende de las variables ***b*** y ***c***. En un paradigma de programación imperativo el valor de ***b*** y ***c*** podrían ser alterados, pero esto no actualizaría la variable ***a*** a su nuevo valor, sería necesario ordenarlo imperativamente. En cambio, en la programación reactiva, ***a*** podría ser actualizado automáticamente cada vez que se actualiza ***b*** o ***c***. Gracias a este ejemplo se puede entender porque este paradigma constituye el futuro en los entornos de desarrollo web y móviles, donde existen una gran cantidad de eventos (lectura de sensores, pulsaciones en la pantalla, clics en botones, etc.) que necesitan de la actualización de valores o pantallas.

Este paradigma fue creado en 1986 por Glenn Wadden para el lenguaje VTScript [6], pero no fue hasta 2010 cuando Microsoft publica la librería RX.NET que empieza su desarrollo y alcanza su máximo potencial al ser portado a otros lenguajes a través de otras librerías.

Los conceptos fundamentales de este paradigma son:

- Observable: Componente que va a ser observado y, a su vez, va a observar el flujo de datos.
- Observer: Elemento que será notificado por el Observable y que se encargará de realizar las modificaciones necesarias cuando lo notifiquen.
- Flujo de datos/eventos: Conjunto de datos que se modificaran o generaran en tiempo real y que el Observer se encargará de estar vigilando.

Otro concepto importante en este tipo de programación es que entre el Observable y el Observer también existe un flujo de datos que puede ser manipulado mediante un conjunto de operaciones que la práctica totalidad de las librerías que implementan este paradigma incluyen. Las operaciones que se suelen considerar más importantes son las siguientes:

- Flatmap: Transforma un observable en otro observable.

- Map: Transforma el tipo de datos del flujo en otro tipo de datos.
- ObserveOn: Una de las operaciones más importantes que sirve para especificar un hilo de ejecución en el que serán realizadas todas las siguientes operaciones relacionadas con el flujo y el observer a partir de esa especificación.
- SubscribeOn: Junto con el ObserveOn, es también una operación de gran importancia, al especificar un hilo de ejecución en el que este Observable va a ser ejecutado.
- Filter: Filtrar los datos que pasan por el flujo.

En las operaciones de ObserveOn y SubscribeOn es necesario especificar un hilo mediante el elemento llamado Scheduler o Dispatchers (su nombre puede variar dependiendo del framweork) con el fin de no colapsar el hilo principal de ejecución. Existen tres tipos de hilos, dependiendo de su función:

- UI: Se emplea para operaciones relacionadas con la interfaz de usuario.
- I/O: Se utiliza en operaciones de entrada y salida (input/output), como pueden ser accesos a las bases de datos, sensores o peticiones a sitios web.
- Computación: Para acciones que necesiten hacer cálculos pesados.

Este flujo de datos y las operaciones que lo transforman se representan mediante el uso de diagramas como el que sigue a continuación:

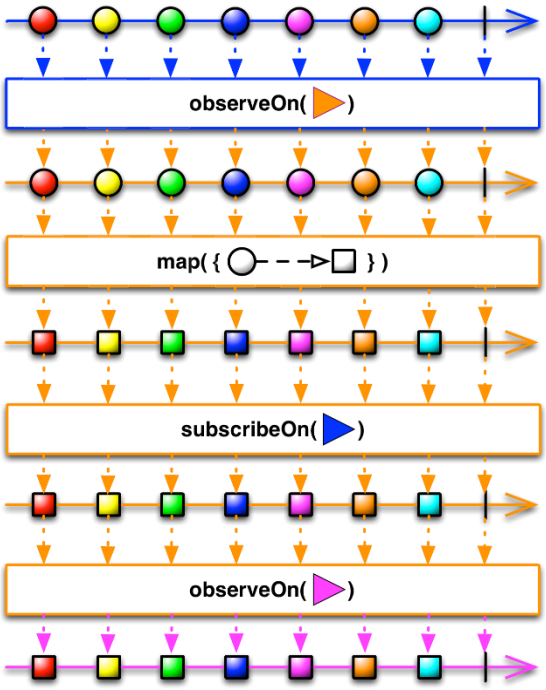


Ilustración 6: Diagrama del flujo de datos y sus transformaciones en la programación reactiva

Por último, se realiza la operación de "Subscribe" para asociar las notificaciones (flujo de eventos) al Observer.

1.2.3 Formulas

Ecuación hipsométrica [7]

Para lograr medir la altura en una posición determinada, es necesario realizar una estimación mediante la ecuación hipsométrica, que permite relacionar dos presiones en dos alturas distintas con la temperatura y dichas alturas.

$$h = z_2 - z_1 = \left(\frac{R * T}{g}\right) * \ln\left(\frac{p_1}{p_2}\right)$$

Donde:

- z_1: Altura en el primer punto.
- z_2: Altura en el segundo punto.
- R: Constante específica para el aire seco (287.06 $Jkg^{-1}K^{-1}$)
- T: Temperatura en kelvin.
- g: Aceleración gravitacional (9.8 m/s^2).
- p_1: Presión en el primer punto.
- p_2: Presión en el segundo punto.
- h: Grosor de la capa atmosférica.

Teniendo esta ecuación en mente, se puede alterar la fórmula para así estimar una segunda altura en función a una altura base:

$$z_2 = \left(\left(\frac{R * T}{g}\right) * \ln\left(\frac{p_1}{p_2}\right)\right) + z_1$$

Índice UV

El sistema de medición de la radiación ultravioleta es conocido como el Índice UV. Este sistema está basado en 11 índices distribuidos en cinco niveles (bajo, moderado, alto, muy alto y extremadamente alto), estos índices están repartidos de la siguiente manera:

- Nivel bajo: Índice 1 y 2.
- Nivel moderado: Índice 3, 4, 5.
- Nivel alto: Índice 6, 7.
- Nivel muy alto: Índice 8, 9, 10.
- Nivel extremadamente alto: Índice 11.

Este índice UV, aunque se utiliza para medir la radiación proveniente del sol, también puede ser utilizada para calcular la radiación producida por distintos tipos de equipos como pueden ser las lámparas o LEDs.

La fórmula oficial [8] para calcular este valor está basada en el espectro de acción de referencia para el eritema provocado por radiación UV establecido por la Comisión Internacional sobre la Iluminación (CIE) en la piel (ISO 17166:1999/CIE S 007/E-1998) es la siguiente:

$$I_{UV} = k_{er} * \int_{250\ nm}^{400\ nm} E_\lambda * S_{er}(\lambda)\, d\lambda$$

Donde:

- E_λ: Irradiancia espectral solar (W/(m2 .nm)) a la longitud de onda.
- $S_{er}(\lambda)$: Espectro de acción de referencia para el eritema.
- k_{er}: Constante igual a 40 m^2/W.

Aunque esta es la fórmula oficial para calcular el índice UV, los fabricantes de los sensores suelen dar ya tabuladas estos valores, relacionándolos así con los índices.

UV Index	0	1	2	3	4	5
Vout(mV)	<50	227	318	408	503	606
Analog Value	<10	46	65	83	103	124
UV Index	6	7	8	9	10	11+
Vout(mV)	696	795	881	976	1079	1170+
Analog Value	142	162	180	200	221	240

Ilustración 7: Tabulación de los Indices UV

1.3 Trabajo previo

Como ya se anticipó, en la actualidad apenas existen proyectos o aplicaciones dedicados al control y muestreo de la fauna. Entre los existentes podemos destacar:

- **Sistema de monitorización de fauna no intrusivo basado en visión por computador (SCOUT**) [9], proyecto financiado por la unión europea y por el Plan de Ciencia, Tecnología e Innovación de Asturias (PCTI), es un sistema en desarrollo para la monitorización en espacios abiertos de la fauna mediante cámaras fijas. Este proyecto está basado en la técnica de fototrampeo, que, junto a los nuevos avances tecnológicos en las áreas de software, comunicación y sensores, forman un sistema fijo, eficiente e inteligente. Mientras que la propuesta de este Trabajo Fin de Grado (TFG) busca la creación de una aplicación móvil que permita recoger datos como las distintas posiciones geográficas, la temperatura, la presiones e índices UV de los transectos.
- **Animal Observer:** [10], es una de las aplicaciones más conocidas entre los etólogos, ya que ayuda al muestreo del comportamiento grupal o individual del animal, se centra en las técnicas de muestreo de barrido y focal [11]. Aunque es un buen ejemplo de integración de los smartphones en el campo de la biología esta aplicación no está orientada al conteo/avistamiento de animales y, además, se presenta en exclusiva para dispositivos iOS frente a **GeoFauna** que se centrará en Android con lo que se pretende abarcar mayor número de dispositivos [12].

También existen gran cantidad de webs y aplicaciones que ayudan a los usuarios a identificar plantas, animales y setas, gracias al empleo de inteligencia artificial, entre las que se pueden destacar:

- **Google lens** [13], aplicación creada por Google que reconoce objetos e identifica algunas especies de animales.

- **Seek** [14], aplicación diseñada por iNaturalist que con ayuda de la cámara del smartphone identifica animales y plantas.
- **BirdNET**, desarrollada por Stefan Kahl, identifica las diferentes especies de aves a través de sus cantos.
- **Picture Insect-Insectos ID**, por Glority LLC, para el reconocimiento de más de mil especies de insectos por medio de fotos.

2. Objetivos

Se estableció para este proyecto una serie de objetivos principales, junto con unos secundarios que se intentarán cumplir durante el desarrollo de este TFG. Estos objetivos se explicarán en los dos siguientes subapartados.

2.1 Objetivos principales

La principal meta de este trabajo es la creación de un sistema, **GeoFauna**, formado por un dispositivo portátil auxiliar para la toma de muestras ambientales y una aplicación Android que ayude a los biólogos y a los aficionados al avistamiento de animales.

Esta meta se puede descomponer en dos secciones de objetivos principales:

La aplicación Android

Los objetivos principales para la aplicación Android son:

- Registrar los avistamientos en una base de datos local.
- Recibir los datos obtenidos por diversos sensores del dispositivo portátil auxiliar.
- Permitir la exportación de esta base datos a un archivo o a otra plataforma como puede ser “Whatsapp” o “OneDrive”.

Dispositivo portátil auxiliar para la toma de muestras ambientales

Los objetivos principales de este dispositivo son:

- Tomar datos del entorno mediante sensores conectados a él.
- Transmitir los datos de los sensores al smartphone.

2.2 Objetivos secundarios

Además de los anteriores objetivos principales hay una serie de objetivos secundarios que se intentaran alcanzar en este trabajo:

3. La aplicación debe ser intuitiva.
4. El dispositivo portátil auxiliar para la toma de muestras ambientales no debe requerir ninguna acción del usuario.
5. La comunicación entre el dispositivo portátil auxiliar y el teléfono debe ser realizada por BLE (Bluetooth Low Energy) [4].

3. Planificación

Para el desarrollo de este proyecto se realizó una planificación al inicio, con el fin de facilitar, estructurar y, en resumen, organizar la labor. En esta estimación, el proyecto tenía una duración de 105 días, con jornadas de tres horas, para poder compaginar el proyecto de este TFG con la jornada laboral de 8h.

La planificación está dividida en seis hitos: Análisis, Diseño, Aprendizaje de nuevas tecnologías, Implementación, Prueba de campo y Memoria.

WBS	Task Name	Duration	Start	Finish
1	**GeoFauna**	**105 days**	**Fri 24/07/20**	**Thu 05/11/20**
1.1	**Análisis**	**6 days**	**Fri 24/07/20**	**Wed 29/07/20**
1.2	**Diseño**	**24 days**	**Thu 30/07/20**	**Sat 22/08/20**
1.3	**Aprendizaje de las nuevas tecnologías**	**26 days**	**Sun 23/08/20**	**Thu 17/09/20**
1.4	**Implementación**	**38 days**	**Fri 18/09/20**	**Sun 25/10/20**
1.5	**Prueba de campo**	**1 day**	**Mon 26/10/20**	**Mon 26/10/20**
1.5.1	Hacer ruta de prueba	1 day	Mon 26/10/20	Mon 26/10/20
1.6	**Memoria**	**10 days**	**Tue 27/10/20**	**Thu 05/11/20**
1.6.1	Hacer la memoria del TFG	10 days	Tue 27/10/20	Thu 05/11/20

Ilustración 8:Planificación general.

De estos seis hitos, los cuatro primeros se han subdividido en varias tareas.

3.1 Análisis

Durante este hito se realizarán las tareas de análisis del sistema, identificación de requisitos y definición de alcance.

WBS	Task Name	Duration	Start	Finish
1.1	**Análisis**	**6 days**	**Fri 24/07/20**	**Wed 29/07/20**
1.1.1	Analisis general del sistema	1 day	Fri 24/07/20	Fri 24/07/20
1.1.2	Definición del alcance	1 day	Sat 25/07/20	Sat 25/07/20
1.1.3	Toma de requisitos funcionales	2 days	Sun 26/07/20	Mon 27/07/20
1.1.4	Toma de requisito no funcionales	2 days	Tue 28/07/20	Wed 29/07/20

Ilustración 9: Planificación desglosada de la etapa de "Análisis".

3.2 Diseño

En el segundo hito, se intentará dar forma al sistema que se va a desarrollar. Esta etapa se estructurará en dos partes: el del ESP32 y el de la aplicación Android.

WBS	Task Name	Duration	Start	Finish
1.2	**Diseño**	**24 days**	**Thu 30/07/20**	**Sat 22/08/20**
1.2.1	**Diseño del ESP32**	**6 days**	**Thu 30/07/20**	**Tue 04/08/20**
1.2.1.1	Selección de sensores	1 day	Thu 30/07/20	Thu 30/07/20
1.2.1.2	Diseño de diagrama de conexión	3 days	Fri 31/07/20	Sun 02/08/20
1.2.1.3	Diseño de la arquitectura del microc	2 days	Mon 03/08/20	Tue 04/08/20
1.2.2	**Diseño de la aplicación Android**	**12 days**	**Wed 05/08/20**	**Sun 16/08/20**
1.2.2.1	Diseño de la interfaz de usuario	5 days	Wed 05/08/20	Sun 09/08/20
1.2.2.2	Diseño de la base de datos	2 days	Mon 10/08/20	Tue 11/08/20
1.2.2.3	Diseño de la arquitectura de la aplica	5 days	Wed 12/08/20	Sun 16/08/20
1.2.3	Diseño de la comunicación	4 days	Mon 17/08/20	Thu 20/08/20
1.2.4	Compra de materiales	2 days	Fri 21/08/20	Sat 22/08/20

Ilustración 10: Planificación desglosada de la etapa de "Diseño".

3.3 Aprendizaje de las nuevas tecnologías

Se creó este hito ya que para el desarrollo de este TFG se utilizaron varias tecnologías que son nuevas para el desarrollador, así que se calcularon 26 días para familiarizarse con el uso y funcionamiento de estas.

WBS	Task Name	Duration	Start	Finish
1.3	**Aprendizaje de las nuevas tecnologías**	**26 days**	**Sun 23/08/20**	**Thu 17/09/20**
1.3.1	Kotlin	3 days	Sun 23/08/20	Tue 25/08/20
1.3.2	Programación reactiva	5 days	Wed 26/08/20	Sun 30/08/20
1.3.3	BLE	7 days	Mon 31/08/20	Sun 06/09/20
1.3.4	ESP32	2 days	Mon 07/09/20	Tue 08/09/20
1.3.5	BME/BMP280	1 day	Wed 09/09/20	Wed 09/09/20
1.3.6	GUVA-S12SD	1 day	Thu 10/09/20	Thu 10/09/20
1.3.7	GY-NEO6MV2	2 days	Fri 11/09/20	Sat 12/09/20
1.3.8	Android Jetpack	5 days	Sun 13/09/20	Thu 17/09/20

Ilustración 11: Planificación desglosada de la etapa de "Aprendizaje de las nuevas tecnologías".

3.4 Implementación

Durante este último hito de 38 días de duración, el desarrollador creará todo el sistema. Como se hizo en la etapa de diseño, también se dividió siguiendo los mismos dos módulos (microcontrolador y aplicación).

WBS	Task Name	Duration	Start	Finish
1.4	**Implementación**	**38 days**	**Fri 18/09/20**	**Sun 25/10/20**
1.4.1	**Implentación del microcontrolador**	**10 days**	**Fri 18/09/20**	**Sun 27/09/20**
1.4.1.1	Conexión de los sensores	2 days	Fri 18/09/20	Sat 19/09/20
1.4.1.2	Testear sensores	2 days	Sun 20/09/20	Mon 21/09/20
1.4.1.3	Moulo de lectura de sensores	2 days	Tue 22/09/20	Wed 23/09/20
1.4.1.4	Moulo de BLE	4 days	Thu 24/09/20	Sun 27/09/20
1.4.2	**Implementación de la aplicación**	**28 days**	**Mon 28/09/20**	**Sun 25/10/20**
1.4.2.1	Interfaz de usuario	5 days	Mon 28/09/20	Fri 02/10/20
1.4.2.2	BLE	6 days	Sat 03/10/20	Thu 08/10/20
1.4.2.3	Room	2 days	Fri 09/10/20	Sat 10/10/20
1.4.2.4	Localización	3 days	Sun 11/10/20	Tue 13/10/20
1.4.2.5	Rest	2 days	Wed 14/10/20	Thu 15/10/20
1.4.2.6	Tests	10 days	Fri 16/10/20	Sun 25/10/20

Ilustración 12: Planificación desglosada de la fase de "Implementación".

4 Requisitos del sistema

4.1 Requisitos funcionales

Al igual que en los objetivos principales se pueden clasificar los requisitos funcionales en dos categorías: aplicación Android y dispositivo portátil auxiliar.

Requisitos funcionales de la aplicación Android

Código	Nombre del requisito	Descripción del requisito
RF1	Añadir transecto	El usuario podrá añadir nuevos transectos en los que más adelante se guardaran los avistamientos.
RF2	Eliminar transecto	El usuario podrá eliminar los transectos.

RF3	Editar transecto	El usuario podrá editar un transecto.
RF4	Registrar avistamiento	El usuario será capaz de registrar avistamientos en una base de datos local una vez introducido un set de datos obligatorios
RF5	Editar avistamiento	El usuario será capaz de editar las entradas de la base de datos, es decir, de los avistamientos registrados.
RF6	Eliminar avistamiento	El usuario tendrá la posibilidad de borrar un avistamiento después de haberlo añadido.
RF7	Estimación de altura	El usuario será capaz de realizar estimaciones sobre la altura de un avistamiento en base a otra altura y presión.
RF8	Limpiar base de datos	El usuario podrá borrar todos los avistamientos de un transecto pulsando un botón
RF9	Visualización de la base de datos	El usuario tendrá la posibilidad de visualizar el estado de la base de datos, es decir, los avistamientos registrados en un transecto.
RF10	Conexión BLE	El usuario podrá conectarse mediante BLE a un dispositivo portátil auxiliar para la toma de datos concreto.
RF11	Comunicación BLE	El usuario podrá una vez conectado a su dispositivo portátil auxiliar, recibir los datos que este le envía siendo estos datos opcionales en el registro de un avistamiento.
RF12	Exportación a un archivo	El beneficiario podrá exportar a un archivo .csv los avistamientos registrados en un transecto.
RF13	Exportar a otra plataforma	El usuario tendrá la posibilidad de exportar la base de datos de un transecto a otra plataforma como “OneDrive”, “Outlook” o “WhatsApp”.
RF14	Internacionalización	La aplicación estará disponible en varios idiomas (inglés y español).
RF15	Alternativa GPS	En ausencia del microcontrolador, el usuario podrá utilizar el GPS del smartphone para obtener la latitud y longitud

Requisitos funcionales del dispositivo portátil auxiliar para la toma de muestras ambientales

Código	**Nombre del Requisito**	**Descripción del requisito**
RF16	Uso del microcontrolador	El usuario solo tendrá que encender el dispositivo portátil auxiliar para iniciar su funcionamiento.
RF17	Recolección de datos	El dispositivo portátil auxiliar utilizará una serie de sensores para tomar datos del entorno.
RF18	Configuración BLE	El dispositivo auxiliar portátil para la toma de muestras ambientales configurará la comunicación BLE automáticamente después de su inicio.
RF19	Indicadores	El dispositivo mostrará su estado al usuario a través de un LED.

4.2 Requisitos no funcionales

GeoFauna también tendrá una serie de requisitos no funcionales que se siguen dividiendo en dos categorías: los de la aplicación Android y los del dispositivo portátil auxiliar para la toma de muestras ambientales.

Requisitos no funcionales de la aplicación Android

Código	Tipo de requisito	Descripción del requisito
RNF1	De eficiencia	La carga de vistas no puede durar más de 5 segundos.
RNF2	De eficiencia	Los cambios en la base de datos deben de mostrarse instantáneamente en todas las vistas en la que se muestre la información almacenada.
RNF3	De usabilidad	Antes de una acción peligrosa (eliminación permanente de información), un mensaje de aviso se cargará antes de continuar.
RNF4	De usabilidad	Aún con errores, la aplicación debe de poder continuar con su funcionamiento.
RNF5	De mantenibilidad	La aplicación debe de tener un desarrollo limpio y con alta cohesión, para así facilitar su modificación y extensibilidad.
RNF6	De portabilidad	La exportación de la información de la base de datos debe de ser realizada en un formato estándar, entendible por otras plataformas.

Requisitos funcionales del dispositivo portátil auxiliar para la toma de muestras ambientales

Código	Tipo de requisito	Descripción del requisito
RNF7	De eficiencia	La configuración debe tardar menos de 10 segundos.
RNF8	De eficiencia y de usabilidad	El dispositivo portátil auxiliar debe mostrar su estado al usuario en menos de dos segundos después de ser encendido.
RNF9	De portabilidad	La parte del sistema relacionada con el dispositivo portátil auxiliar para la toma de muestras ambientales (microcontrolador y sensores) debe de poder ser transportada y almacenada en un recipiente de menor o igual tamaño que una caja de zapatos.

5. Presupuesto

El coste de este Trabajo de Fin de Grado fue calculado basándose en la planificación presentada en la sección “3. Planificación” con un coste del esfuerzo/hora del desarrollador de 8.33 €/h (equivaldría a 2000€ al mes si fuera a jornada completa), un coste eléctrico de 0.13€/kWh[1] y una potencia de 220W[2] para el ordenador usado en este proyecto.

Teniendo en cuenta estos datos iniciales, se calculó el siguiente coste de los materiales

[1] Coste de kWh basado en la información obtenida en https://tarifaluzhora.es/info/precio-kwh

[2] Valor basado en la información obtenida en https://ganaenergia.com/blog/que-consumo-nos-supone-utilizar-el-ordenador/

:

Hardware necesario	Precio (€)
BME/BMP280	8,29 €
GY-NEO6MV2	9,99 €
GUVA-S12SD	7,00 €
1x LED	0,07 €
1x Resistencia 220Ω	0,03 €
15x Cables jumper	1,49 €
ESP-32	8,99 €
Total	35,86

Tabla 1: Presupuesto TFG

Con este valor obtenido, solo queda calcular el coste general del proyecto, el cual se computará como la suma del coste de cada hito formado por la suma del coste esfuerzo/día del desarrollador más el coste eléctrico del equipo de desarrollo, salvo en la prueba de campo que solo se contabilizará el coste esfuerzo/día al no utilizarse el ordenador.

Hitos	Días	Coste (€)
Análisis	6	150,4548
Diseño	24	601,8192
Aprendizaje	38	952,8804
Implementación	1	25,0758
Prueba de campo	1	24,99
Memoria	10	250,758
Materiales		35,83
Coste Total		2041,8082

Datos extra	
Coste de kWh (€/kWh)	0,13
Coste esfuerzo/h (€/h)	8,33
Jornada (h/día)	3
Consumo ordenador (kW)	0,22

Ilustración 13: Presupuesto general del proyecto

6. Tecnologías utilizadas

Durante el desarrollo de este TFG se pueden destacar la siguiente lista de tecnologías:

6.1 Android

En la actualidad son dos los sistemas operativos que se reparten el mercado de los dispositivos móviles, IOS y Android, por lo que a la hora de desarrollar una aplicación es necesario comenzar decidiendo cuál de los dos sistemas se va a utilizar. ¿A qué va a estar dirigido esta aplicación?

Para la anterior pregunta hay dos posibles respuestas, o bien se utiliza una tecnología multiplataforma o, por el contrario, se emplea solo uno de los sistemas. Aunque, en un principio, parece más interesante abarcar los dos sistemas operativos gracias a la utilización de un entorno

multiplataforma, como puede ser React Native, esta solución se ha descartado para minimizar así los posibles problemas surgidos al intentar utilizar los distintos sensores del móvil.

Frente a los dos sistemas operativos se ha considerado más acertado el uso del sistema Android sobre el IOS por las siguientes razones:

- Android es el sistema operativo líder en el mercado que ha sido desarrollado por la compañía de Google, es usado en los dispositivos móviles (teléfonos y tabletas). Además, desde su lanzamiento es de código abierto y distribución libre, por lo que llega a un gran número de usuarios gracias a su flexibilidad y precio.
- Otro aspecto para tener en cuenta es la gratuidad de las herramientas de desarrollo y los diferentes manuales disponibles en la página oficial Android.
  También, utiliza Java y Kotlin como lenguajes de desarrollo, ya conocidos y utilizados por el desarrollador

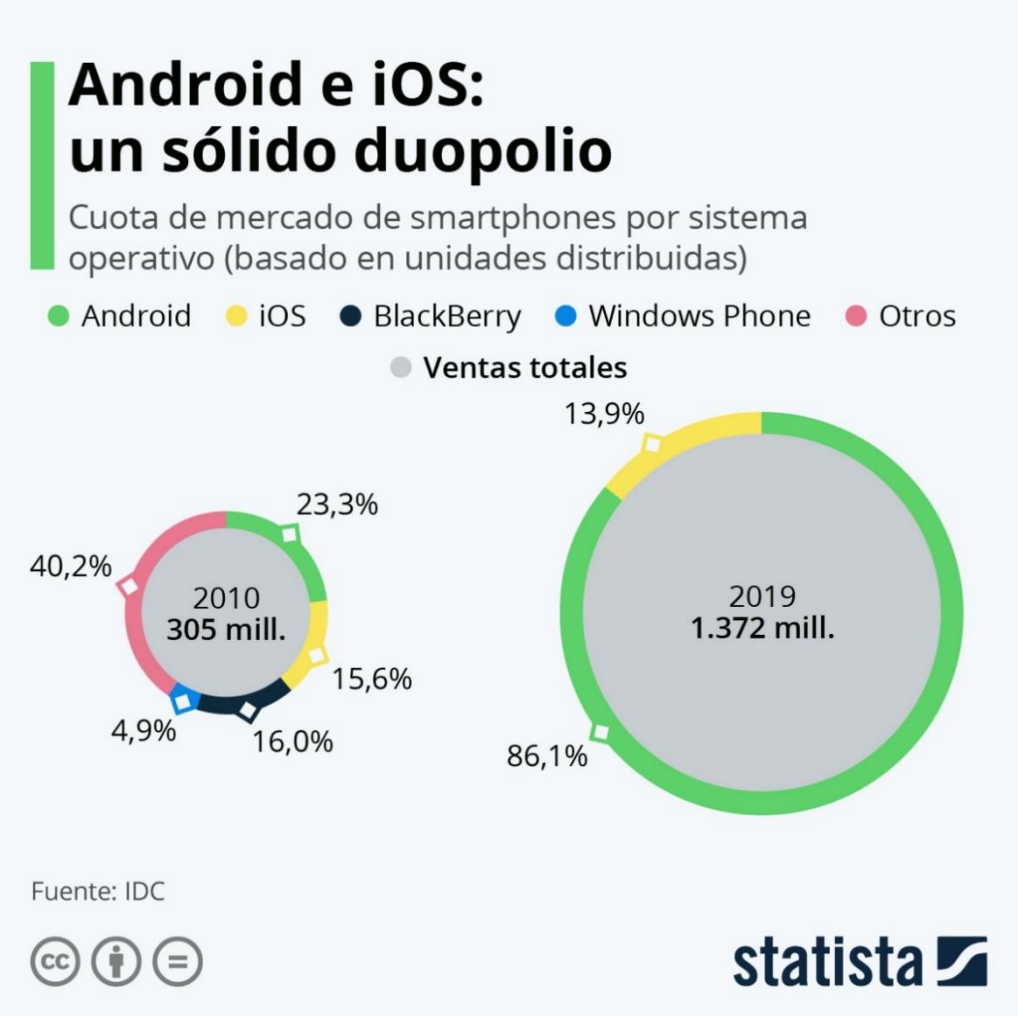


Ilustración 14: Gráfica realizada por Statista sobre la cuota de mercado de cada sistema operativo movil

6.2 Koltin

Lenguaje de programación diseñado por JetBrains, cuya primera aparición fue en 2016. Este lenguaje, en sus inicios, estuvo influenciado por C#, Scala, Java, Groovy y Gosu. No solo es el

lenguaje oficial de Android, sino que además aúna lo mejor de los lenguajes Java y C# todo esto hace que haya sido escogido para el desarrollo del sistema de este TFG.

6.3 Android Jetpack

Conjunto de librerías proporcionadas por Android, que apoyan a los desarrolladores en la realización de aplicaciones para distintas versiones de Android de manera uniforme y eficiente.

El uso de estás librerías es siempre recomendable porque ayudan a programar con una mayor eficiencia, facilitan la reutilización de códigos oficiales, y economizan a la vez que ponen a disposición del usuario las últimas tendencias en arquitectura Android. Entre las librerías que se han utilizado para este proyecto caben destacar las siguientes:

- Room: librería cuya finalidad es la creación y administración bases de datos SQLite que permite la traducción automática de los datos de la base de datos a objetos Kotlin.
- Fragment: facilita la división de la aplicación en pantallas independientes tratadass como componentes alojados en un Activity.
- Material desing components: conjunto de componentes que ayudan con la implementación de la biblioteca “Material Design” desarrollada por Google. Estos componentes otorgan un aspecto moderno a la Interfaz de Usuario (IU).
- Navigation: biblioteca que simplifica y unifica la navegación entre fragmentos y actividades.
- Lifecycle: conjunto de componentes que facilitan el manejo del ciclo de vida y los estados de los fragmentos o actividades de la aplicación.
- Paging: se ocupa de la carga y presentación de los datos en un RecyclerView.

6.4 Reactive Kotlin

El Reactive Kotlin, es una librería que simplifica el uso del paradigma de programación reactivo cuyo uso está señalado para el procesamiento asíncrono de flujo de datos de circulación constante. Este proyecto pertenece al área del Internet de las Cosas (IoT) y por tanto tendrá una circulación constante de datos entre un smartphone y un microcontrolador para lo que es RXKotlin está indicado.

Para aplicar la programación reactiva en este proyecto se utilizará la librería RxKotlin [15] de ReactiveX [16] , esta librería incluye además de las distintas maneras de crear Observables y Observers, las operaciones descritas en el apartado “1.2.2 Programación reactiva” con las que transformar y procesar un flujo de datos y establecer un hilo de ejecución.

6.5 RxAndroidBLE [17]

Como la configuración de una comunicación BLE desde un dispositivo Android suele ser problemática y caótica se decidió utilizar la librería RxAndroidBLE, que simplifica esta configuración, ayuda con el manejo de hilos de ejecución, proporciona operaciones de lectura, escritura y notificación asíncrona además de estar desarrollada para ser utilizada junto Kotlin o Java reactivo.

El beneficio que aporta esta simplificación se aprecia por primera vez cuando la aplicación se dispone a buscar el dispositivo auxiliar de este proyecto entre todos los dispositivos BLE de su entorno. Si se compara el fragmento de código del ejemplo que aparece en la documentación de Android [18] con el código de este proyecto, donde se hace uso de esta librería, se puede

apreciar que además de ser más compacto, filtra, de entre todos los dispositivos encontrados, el que necesitamos:

```
private fun scanLeDevice(enable: Boolean) {
    when (enable) {
        true -> {
            // Stops scanning after a pre-defined scan period.
            handler.postDelayed({
                mScanning = false
                bluetoothAdapter.stopLeScan(leScanCallback)
            }, SCAN_PERIOD)
            mScanning = true
            bluetoothAdapter.startLeScan(leScanCallback)
        }
        else -> {
            mScanning = false
            bluetoothAdapter.stopLeScan(leScanCallback)
        }
    }
}

private val leScanCallback = BluetoothAdapter.LeScanCallback { device, rssi, scanRecord ->
    runOnUiThread {
        leDeviceListAdapter.addDevice(device)
        leDeviceListAdapter.notifyDataSetChanged()
    }
}
```

Ilustración 15: Fragmento de código de escaneo del entorno proporcionado por Android Developers

```
fun scanDevicesAndConnect(): Observable<String>? {

    return RxBleClient.create(getApplication()).scanBleDevices(ScanSettings.Builder().build(),
            ScanFilter.Builder()
                    .setServiceUuid(SERVICE_UUID)
                    .build()).take(1).observeOn(Schedulers.io()
    ).flatMap {
        macAddress = it.bleDevice.bluetoothDevice.toString()
        return@flatMap Observable.just(macAddress)}
}
```

Ilustración 16: Fragmento de Código de GeoFauna con el que se escanea buscando el microcontrolador

También se apreciar en los siguientes fragmentos de código como esta librería simplifica la conexión y comunicación entre dispositivos (notificación de cambios en las características) donde el código de GeoFauna vuelve a ser más compacto y eficiente frente al proporcionado por la documentación Android que es mucho más extenso.

```
var bluetoothGatt: BluetoothGatt? = null
...

bluetoothGatt = device.connectGatt(this, false, gattCallback)

private val gattCallback = object : BluetoothGattCallback() {
    override fun onConnectionStateChange(
            gatt: BluetoothGatt,
            status: Int,
            newState: Int
    ) {
        val intentAction: String
        when (newState) {
            BluetoothProfile.STATE_CONNECTED -> {
                intentAction = ACTION_GATT_CONNECTED
                connectionState = STATE_CONNECTED
                broadcastUpdate(intentAction)
                Log.i(TAG, "Connected to GATT server.")
                Log.i(TAG, "Attempting to start service discovery: " +
                        bluetoothGatt?.discoverServices())
            }
            BluetoothProfile.STATE_DISCONNECTED -> {
                intentAction = ACTION_GATT_DISCONNECTED
                connectionState = STATE_DISCONNECTED
                Log.i(TAG, "Disconnected from GATT server.")
                broadcastUpdate(intentAction)
            }
        }
    }

  // New services discovered
  override fun onServicesDiscovered(gatt: BluetoothGatt, status: Int) {
      when (status) {
          BluetoothGatt.GATT_SUCCESS -> broadcastUpdate(ACTION_GATT_SERVICES_DISC
          else -> Log.w(TAG, "onServicesDiscovered received: $status")
      }
  }

  // Result of a characteristic read operation
  override fun onCharacteristicRead(
          gatt: BluetoothGatt,
          characteristic: BluetoothGattCharacteristic,
          status: Int
  ) {
      when (status) {
              BluetoothGatt.GATT_SUCCESS -> {
                  broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic)
              }
      }
  }
```

```
private fun broadcastUpdate(action: String, characteristic: BluetoothGattCharacteristic) {
    val intent = Intent(action)

    // This is special handling for the Heart Rate Measurement profile. Data
    // parsing is carried out as per profile specifications.
    when (characteristic.uuid) {
        UUID_HEART_RATE_MEASUREMENT -> {
            val flag = characteristic.properties
            val format = when (flag and 0x01) {
                0x01 -> {
                    Log.d(TAG, "Heart rate format UINT16.")
                    BluetoothGattCharacteristic.FORMAT_UINT16
                }
                else -> {
                    Log.d(TAG, "Heart rate format UINT8.")
                    BluetoothGattCharacteristic.FORMAT_UINT8
                }
            }
            val heartRate = characteristic.getIntValue(format, 1)
            Log.d(TAG, String.format("Received heart rate: %d", heartRate))
            intent.putExtra(EXTRA_DATA, (heartRate).toString())
        }
        else -> {
            // For all other profiles, writes the data formatted in HEX.
            val data: ByteArray? = characteristic.value
            if (data?.isNotEmpty() == true) {
                val hexString: String = data.joinToString(separator = " ") {
                    String.format("%02X", it)
                }
                intent.putExtra(EXTRA_DATA, "$data\n$hexString")
            }
        }

    }
    sendBroadcast(intent)
}
```

```
lateinit var bluetoothGatt: BluetoothGatt
lateinit var characteristic: BluetoothGattCharacteristic
var enabled: Boolean = true
...
bluetoothGatt.setCharacteristicNotification(characteristic, enabled)
val uuid: UUID = UUID.fromString(SampleGattAttributes.CLIENT_CHARACTERISTIC_CONFIG)
val descriptor = characteristic.getDescriptor(uuid).apply {
    value = BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
}
bluetoothGatt.writeDescriptor(descriptor)
```

```
// Characteristic notification
override fun onCharacteristicChanged(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic
) {
    broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic)
}
```

Ilustración 17: Grueso del código relacionado con la conexión BLE y notificación de cambios en las características proporcionado por Android Developers

```
fun startTalking() {
    disposable?.dispose()
    disposables.clear()
    disposable = RxBleClient.create(getApplication()).getBleDevice(macAddress).establishConnection(false).observeOn(Schedulers.io()).subscribe { bleConnection ->
        setupCharacteristic(TEMPERATURE_UUID, BluetoothManager.TEMPERATURE_SENSOR, bleConnection)
        setupCharacteristic(HUMIDITY_UUID, BluetoothManager.HUMIDITY_SENSOR, bleConnection)
        setupCharacteristic(UV_UUID, BluetoothManager.UV_SENSOR, bleConnection)
        setupCharacteristic(LATITUDE_UUID, BluetoothManager.LATITUDE_SENSOR, bleConnection)
        setupCharacteristic(LONGITUDE_UUID, BluetoothManager.LONGITUDE_SENSOR, bleConnection)
        setupCharacteristic(ALTITUDE_UUID, BluetoothManager.ALTITUDE_SENSOR, bleConnection)
        setupCharacteristic(PRESSURE_UUID, BluetoothManager.PRESSURE_SENSOR, bleConnection)
    }

}

private fun setupCharacteristic(characteristicUUID: UUID, sensorPosition: Int, bleConnection: RxBleConnection) {
    disposables.add(bleConnection.readCharacteristic(characteristicUUID).observeOn(AndroidSchedulers.mainThread()).subscribe { byteArray ->
        myData[sensorPosition]?.value = String(byteArray)
        bleConnection.setupNotification(characteristicUUID, NotificationSetupMode.COMPAT).flatMap { it }.observeOn(AndroidSchedulers.mainThread()).subscribe {
            myData[sensorPosition]?.value = String(it)
        }
    }
    )
}
```

Ilustración 18: Fragmento de código de GeoFauna en el que usando RxAndroidBle se conecta y configura las notificaciones de las características

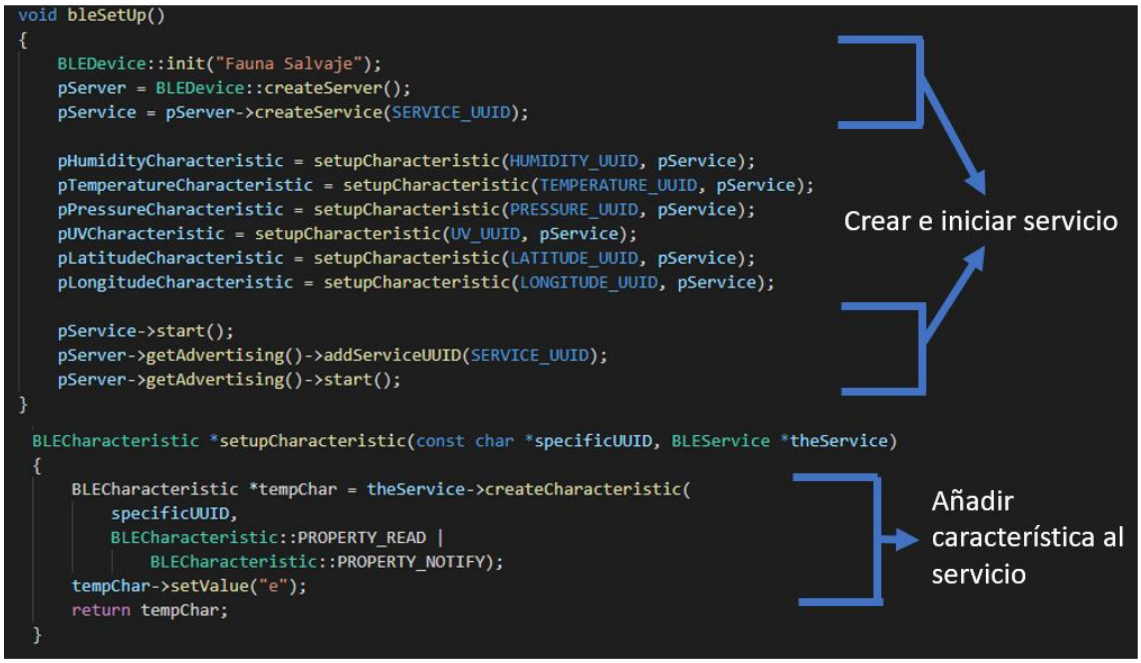
Aunque esta librería es de gran utilidad, como se puede observar en las anteriores cuatro capturas, su utilización conlleva también algunas desventajas como es el difícil manejo de errores y la prácticamente imposibilidad de realizar *Unit testing* y *Behavioural testing* (por lo que se depende casi en su totalidad de los tests manuales).

6.6 ESP32

El ESP32, introducido por primera vez a finales del 2016, es uno de los microcontroladores más utilizados en el área de la electrónica y del Internet de las cosas (IoT), esto es debido a que tiene

módulos wifi y bluetooth (compatible con BLE) integrados, un pequeño tamaño y un escaso precio. Todo ello hace que sea el microcontrolador seleccionado para este TFG.

La cualidad determinante para esta decisión fue el bluetooth incorporado, que ahorró la compra de un módulo externo para este propósito. Además de su fácil configuración, que como se puede observar en la siguiente ilustración, con tan solo ocho líneas de código se pudo crear un servicio y una característica e iniciar dicho servicio.




```
void bleSetUp()
{
    BLEDevice::init("Fauna Salvaje");
    pServer = BLEDevice::createServer();
    pService = pServer->createService(SERVICE_UUID);

    pHumidityCharacteristic = setupCharacteristic(HUMIDITY_UUID, pService);
    pTemperatureCharacteristic = setupCharacteristic(TEMPERATURE_UUID, pService);
    pPressureCharacteristic = setupCharacteristic(PRESSURE_UUID, pService);
    pUVCharacteristic = setupCharacteristic(UV_UUID, pService);
    pLatitudeCharacteristic = setupCharacteristic(LATITUDE_UUID, pService);
    pLongitudeCharacteristic = setupCharacteristic(LONGITUDE_UUID, pService);

    pService->start();
    pServer->getAdvertising()->addServiceUUID(SERVICE_UUID);
    pServer->getAdvertising()->start();
}
 BLECharacteristic *setupCharacteristic(const char *specificUUID, BLEService *theService)
 {
     BLECharacteristic *tempChar = theService->createCharacteristic(
         specificUUID,
         BLECharacteristic::PROPERTY_READ |
             BLECharacteristic::PROPERTY_NOTIFY);
     tempChar->setValue("e");
     return tempChar;
 }
```

Ilustración 19: Fragmento de GeoFauna, en el que se crea el servicio y características

Tras esta configuración inicial, ya solo sería actualizar el valor de las características con el de las lecturas de los sensores en forma de array de chars.

```
void updateData(const char *sensorUUID, double newData, int numberOfDecimals)
{

    char x[20];
    if (numberOfDecimals > 10 || numberOfDecimals < 0)
        dtostrf(newData, 10, 10, x);
    else
        dtostrf(newData, 10, numberOfDecimals, x);

    pService->getCharacteristic(sensorUUID)->setValue(x);
    pService->getCharacteristic(sensorUUID)->notify();
}
```

Ilustración 20: Actualización de una característica.

```
void updateBME()
{

  // BME208: Temperatura - Presion - Humedad

  //Leyendo temperatura en  ºC
  updateData(TEMPERATURE_UUID, bme.readTemperature(), 3);

  //Leyendo presion en hPa
  updateData(PRESSURE_UUID, bme.readPressure() / 100.0F, 3);
  //Manera alternativa para leer altitud en m, pero muy imprecisa ya que la pression a nivel del mar
  //cambia dependiendo del dia y lugar, mejor usar gps
  //updateData(ALT_POS, bme.readAltitude(SEALEVELPRESSURE_HPA));
  //Read Hummidity in %
  updateData(HUMIDITY_UUID, bme.readHumidity(), 3);
}
```

Ilustración 21: Lectura del sensor BME/BMP280 y actualización de las características de temperatura, presión y humedad

```
void updateGPS()
  {
    if (serial_conection.available())
    {
      gps.encode(serial_conection.read());
    }
    if (gps.location.isUpdated())
    {
      //Leyendo latitude
      updateData(LATITUDE_UUID, gps.location.lat(), 10);
      //Leyendo longitud
      updateData(LONGITUDE_UUID, gps.location.lng(), 10);
      //Leyendo altiud manera de leer la altura
      updateData(ALTITUDE_UUID, gps.altitude.meters(), 3);
    }
  }
```

Ilustración 22: Lectura del GPS y actualización de las características de latitud y longitud

```
void updateUV()
{
  float rvVoltage = analogRead(32);
  rvVoltage = rvVoltage / 4095 * 3.3;
  int switchStatement = rvVoltage * 1000;
  //Switch para determinar el indice UV
  switch (switchStatement)
  {
  case 0 ... 49:
    updateData(UV_UUID, 0, 1);
    break;
  case 50 ... 226:
    updateData(UV_UUID, 1, 1);
    break;
  case 227 ... 317:
    updateData(UV_UUID, 2, 1);
    break;
  case 318 ... 407:
    updateData(UV_UUID, 3, 1);
    break;
  case 408 ... 502:
    updateData(UV_UUID, 4, 1);
    break;
  case 503 ... 605:
    updateData(UV_UUID, 5, 1);
    break;
  case 606 ... 695:
    updateData(UV_UUID, 6, 1);
    break;
  case 696 ... 794:
    updateData(UV_UUID, 7, 1);
    break;
  case 795 ... 880:
    updateData(UV_UUID, 8, 1);
    break;
  case 881 ... 975:
    updateData(UV_UUID, 9, 1);
    break;
  case 976 ... 1078:
    updateData(UV_UUID, 10, 1);
    break;
  default:
    if (switchStatement >= 1079)
    {
      updateData(UV_UUID, 11, 1);
    }
    else
    {
      updateData(UV_UUID, 0, 1);
    }
    break;
  }
```

Ilustración 23: Lectura del GUVA-S12SD y actualización de la característica del índice UV

6.7 Visual Studio Code (PlatformIO) y Android Studio

Para la parte del desarrollo de software se utilizaron los siguientes entornos de desarrollo:

- Visual Studio Code (PlatformIO): entorno de desarrollo open source y multiplataforma, creado por Microsoft que se utilizará junto con la extensión de PlatformIO, que favorece la programación en C/C++ de microcontroladores y simplifica bastante el manejo del código.
- Android Studio: este otro entorno de desarrollo multiplataforma oficial de aplicaciones Android, puede ser utilizado tanto para programar en Java como en Kotlin y está basado en el IntelliJ IDEA de JetBrains.

7. Diseño

Para el desarrollo de este proyecto de creación de un sistema de muestreo que facilite la labor de muestreo/conteo de la fauna silvestre compuesto por un dispositivo portátil auxiliar para la toma de muestra ambientales y una aplicación Android se organizó el diseño desde tres perspectivas diferentes o niveles de concreción. Un primer nivel dónde se trabaja desde un punto de vista general del sistema, un segundo nivel centrado en la Aplicación Android, para seguir con el dispositivo auxiliar (ESP32 y sensores), y a continuación analizar las comunicaciones que se dan entre ambos elementos y finalmente se realiza el diseño de las tablas de base de datos.

7.1 Diseño general del sistema

A continuación, se explicará el diseño general del sistema desarrollado en este Trabajo de Fin de Grado.

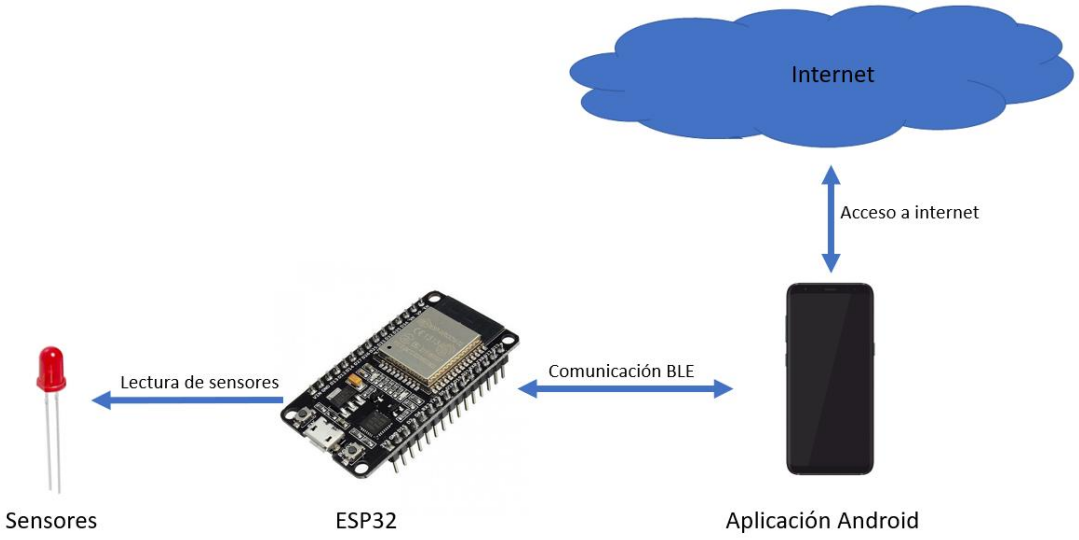


Ilustración 24: Diseño general del sistema

Como se puede observar en el anterior diagrama, el sistema estará formado por dos componentes principales: el dispositivo auxiliar para la toma de muestras ambientales, formado por el ESP32 y los sensores, y la aplicación Android los cuales se comunicarán entre sí por Bluetooth de baja energía. Además de estos dos elementos principales, se contará con APIs residentes en la Nube entre las que estarán la Google API, que permitirá obtener la posición geográfica en caso del incorrecto funcionamiento del GPS y la Geonames API, que ayudará a conocer en cada momento el nombre de la localización más próxima al usuario. A

todo esto, se añadirán las informaciones proporcionadas con el uso del ESP32 gracias a los distintos sensores con los que se obtendrán diferentes lecturas del entorno medioambiental.

7.2 Arquitectura de la aplicación Android

A la hora de desarrollar esta aplicación para dispositivos móviles Android se plantea seguir una arquitectura de capas con la que se buscaba la independencia entre los diferentes niveles.

Por un lado, se pretende independizar el código de la interfaz de usuario de la lógica (Viewwmodels y otra lógica) y, por otro lado, a la vez que se unifica el acceso a la base de datos a través del uso de un Repositorio, se separa del resto del código.

Dentro del segundo nivel, se realiza una distinción entre "Viewmodels", y el resto de la lógica, ya que los Viewmodels tienen la función específica de proteger la información volátil que reside en la interfaz de usuario mientras mantiene sincronizada la información de la base de datos cuando hay que mostrarla.

Se eligió una arquitectura en capas para facilitar el posterior desarrollo y extensión de la aplicación.

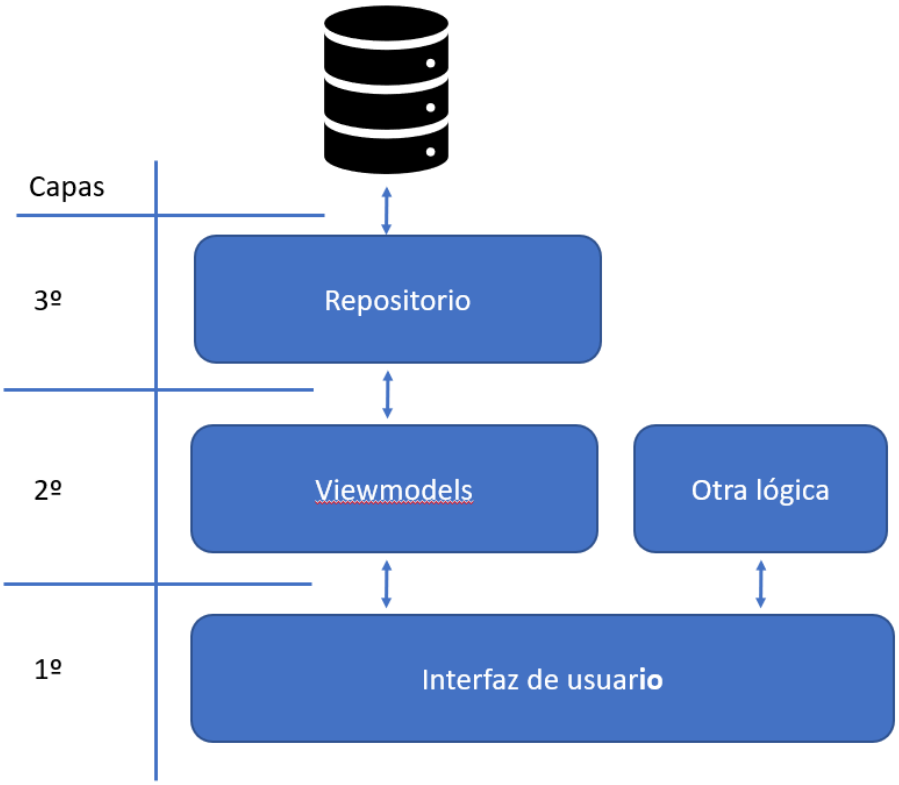


Ilustración 25: Arquitectura de la aplicación Android.

Concretando aún más el anterior esquema de la arquitectura y mostrando las conexiones que existe entre capas, quedaría el siguiente diagrama de componentes, donde las cajas amarillas son los componentes más importantes de cada capa y las líneas los caminos disponibles entre unas y otras.

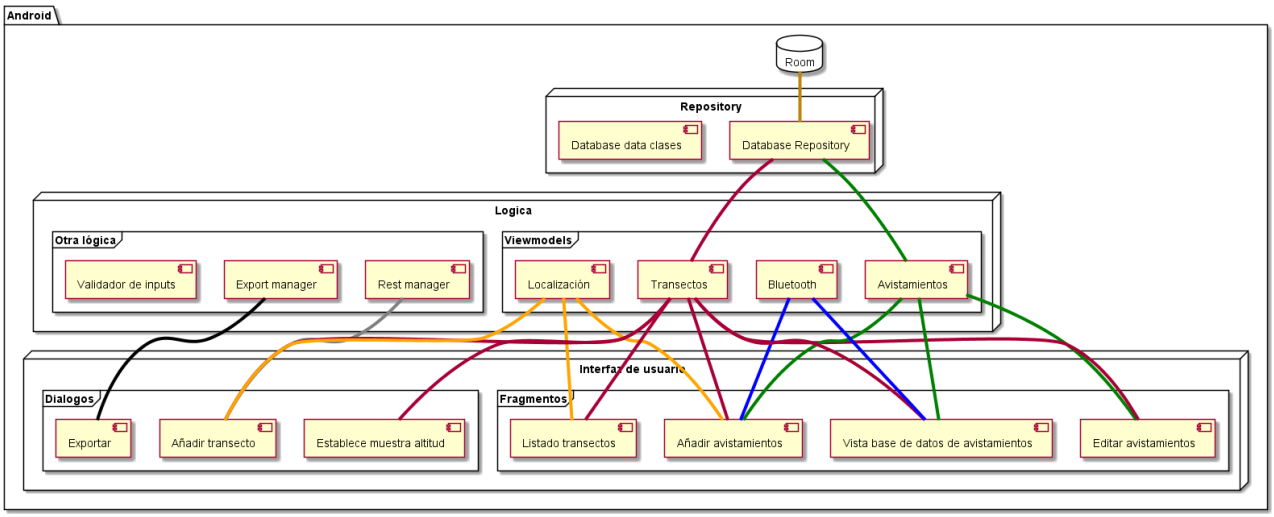


Ilustración 26: Diagrama de componentes y conexiones de la arquitectura de la aplicación Android.

7.3 Dispositivo auxiliar para la toma de muestras ambientales

Para alcanzar la meta, el otro objetivo principal además del desarrollo de la aplicación, es la creación de un sistema portátil formado por un microcontrolador (ESP32) y una serie de sensores.

7.3.1 Diagrama de conexiones de los sensores al ESP32

Una parte importante de este TFG se basa en la conexión de los sensores al microcontrolador, para lo que se siguió el siguiente esquema de conexión. En él se puede observar como el microcontrolador ESP32 se encarga de las lecturas de los diferentes sensores utilizados:

- GY-NEO6MV2 con el que se obtiene la posición geográfica a través de satélites.

Ilustración 27: Fotografía del GY-NEO6MV2.

- GUVA-S12SD que proporciona un sistema de valores que se utiliza para calcular el índice de radiación UV.

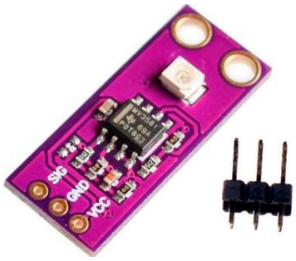


Ilustración 28: Fotografía del GUVA-S12SD

- BME/BMP280 toma las medidas de presión, humedad y temperatura, siendo estas dos últimas medidas utilizadas para estimar la altura.

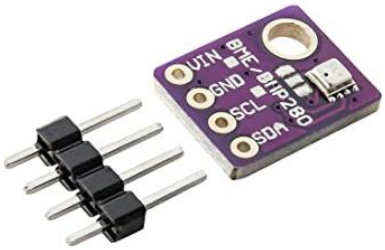


Ilustración 29: Fotografía del BME/BMP280

- Led, muestra el estado, apagado/encendido (off/on), del microcontrolador.

Ilustración 30: Fotografía de un led

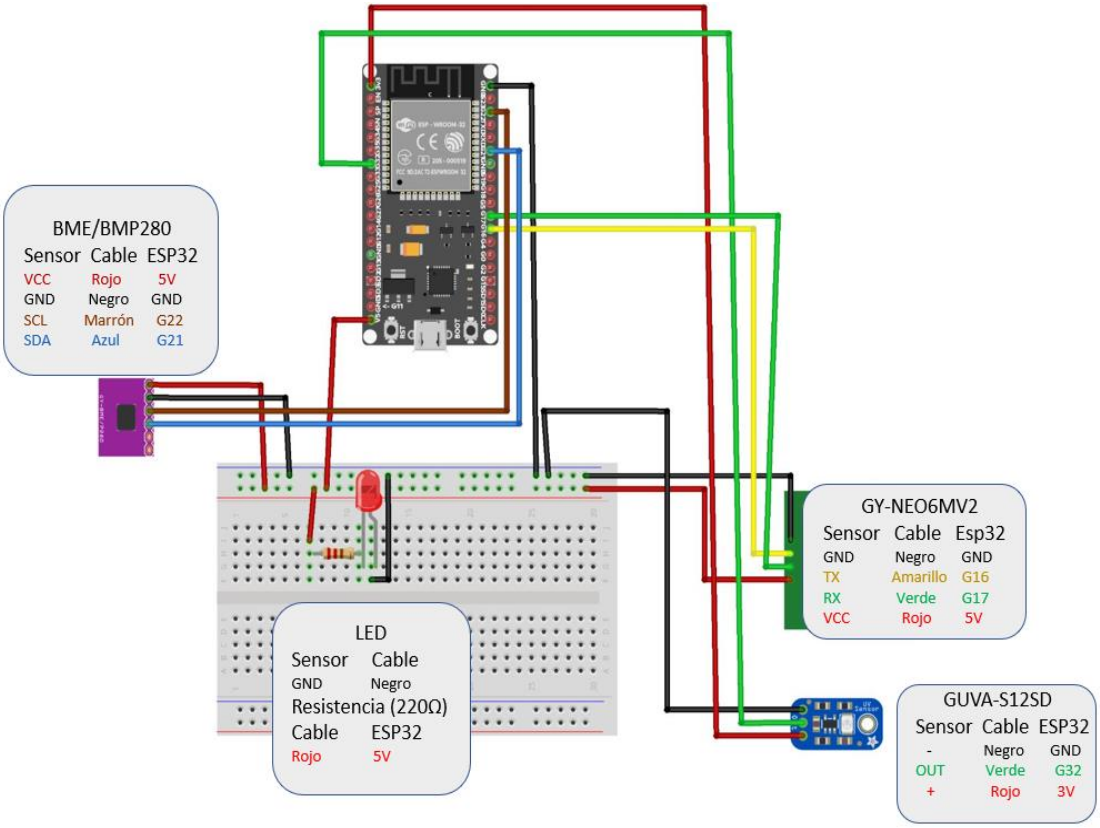


Ilustración 31: Diagrama de conexión al ESP32

7.3.2 Arquitectura del Microcontrolador ESP32

La arquitectura seguida en esta parte del proyecto es muy simple y consta solamente de dos módulos. El primero de los cuales está encargado de la lectura de los sensores anteriormente citados y el segundo módulo se ocupa de la configuración de la comunicación BLE y el posterior envío de datos a la aplicación.

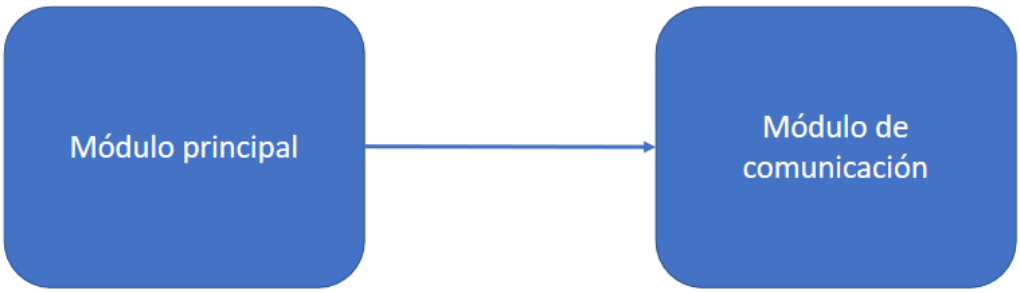


Ilustración 32: Arquitectura del microcontrolador ESP32.

7.3.3 Jerarquía en GATT

Dentro del ESP32 se tuvo que crear un servidor GATT, para el que se diseñó el siguiente perfil, compuesto por un servicio con seis características.

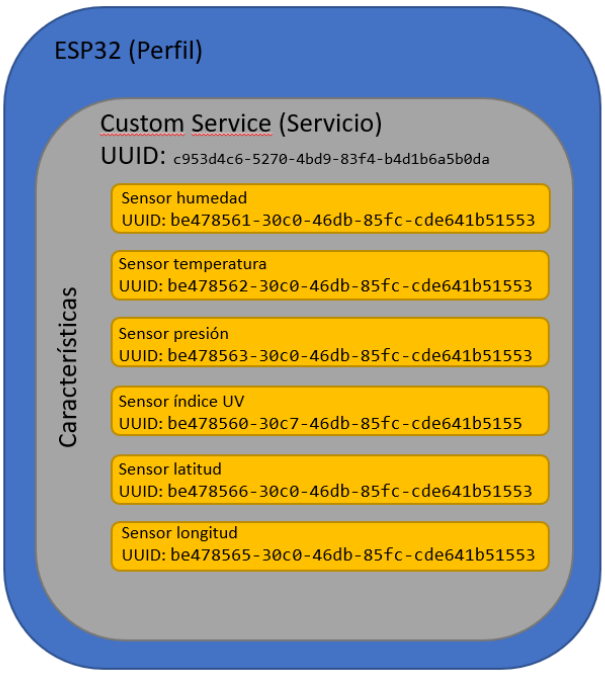


Ilustración 33: Jerarquía en GATT

7.4 Comunicaciones

La aplicación Android no es algo aislado del mundo, ya que esto haría que todo este proyecto fuera inútil, con lo que la comunicación dentro de este proyecto funciona de la siguiente manera.

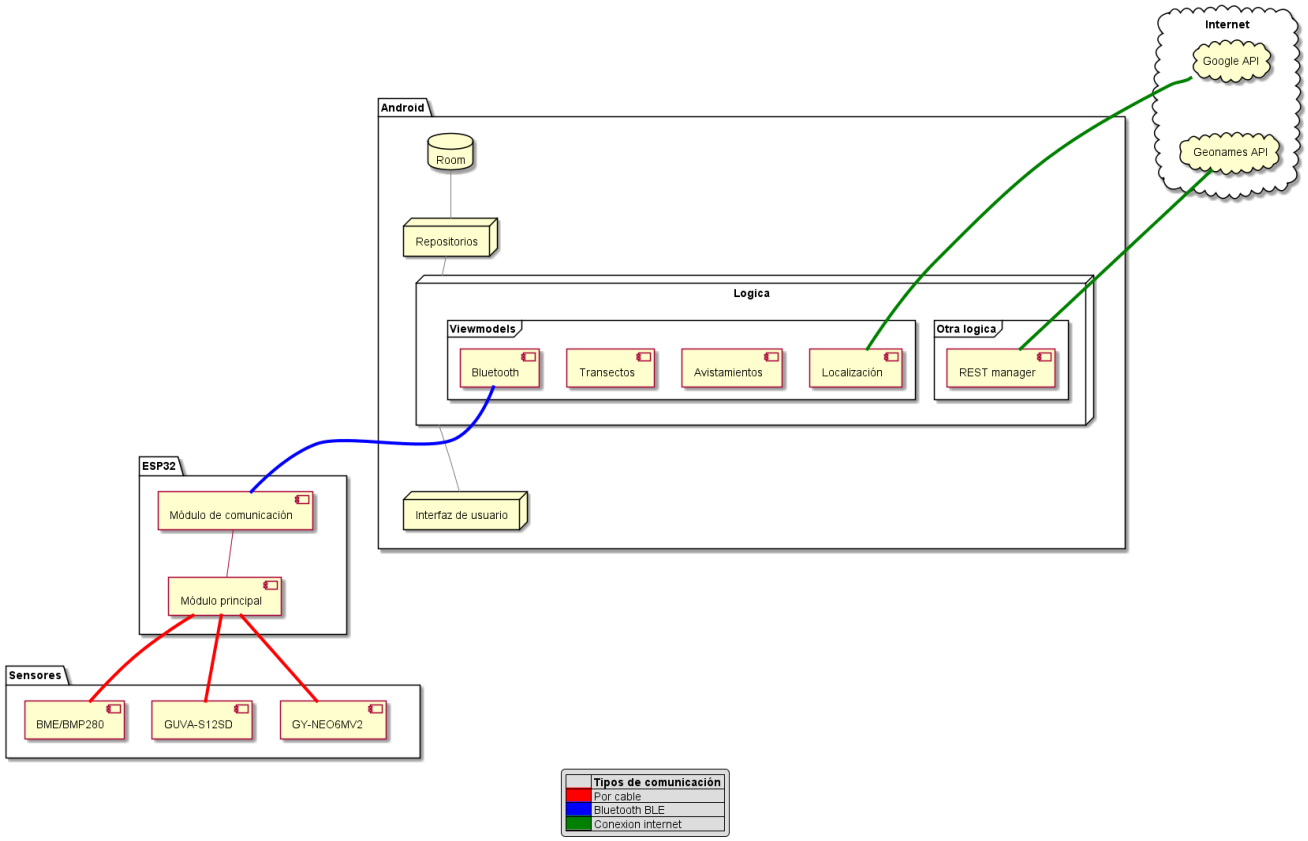


Ilustración 34: Diagrama de la comunicación

Siguiendo el anterior diagrama, los datos son tomados por los sensores del ESP32 por medio de una conexión cableada, para más tarde ser enviados a la aplicación móvil a través de una conexión bluetooth de baja energía. Además, ciertos componentes como el Viewmodel de Localización y el Rest Manager acceden a dos APIs (Aplication programming Interface) de internet, que permiten, siempre que sea posible, obtener tanto la posición geográfica en caso de que el GPS del microcontrolador no pudiera conectarse a un satélite (Google API), como el lugar más cercano a la posición geográfica del usuario (Geonames API REST).

7.5 Diseño de las tablas de base de datos

Una de las cuestiones más importantes a tratar en el sistema es el almacenamiento de los distintos avistamientos y organizarlo. Para este propósito se utilizó la nueva librería de “Room”, ya introducida en el apartado “6.2 Tecnologías utilizadas”, que se usó para implementar las tablas siguientes.

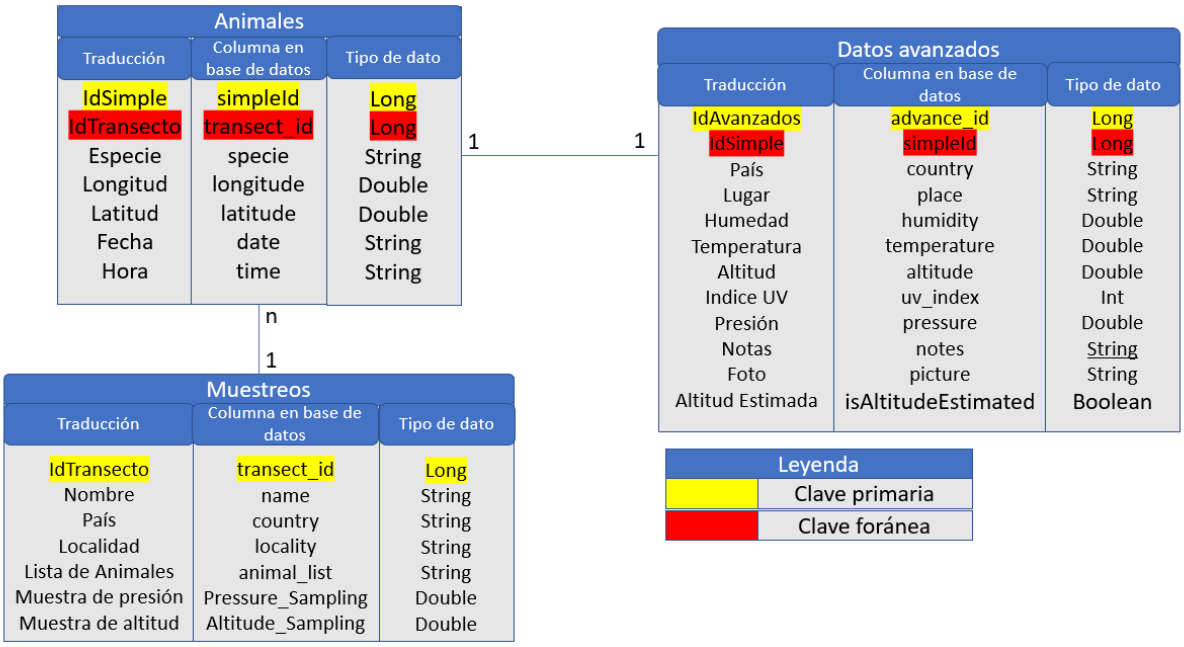


Ilustración 35:Estructura de las tablas de la base de datos

8. Manual de uso

GeoFauna se maneja de la siguiente manera:

8.1 Creación y administración de transectos

Para crear un nuevo transecto, es necesario seguir los siguientes pasos:

1. Pulsar el botón “+” situado en la parte inferior de la pantalla inicial.
2. Se cargará un diálogo donde se podrá introducir datos relacionados con el transecto (nombre, lugar, lista de animales a buscar), siendo el nombre del transecto el único campo obligatorio.
3. Finalmente pulsar el botón “Guardar Transectos.

Una vez creado el transecto aparecerá en la pantalla inicial, donde se podrá eliminar o editar simplemente deslizando la ruta hacia la derecha o izquierda respectivamente.

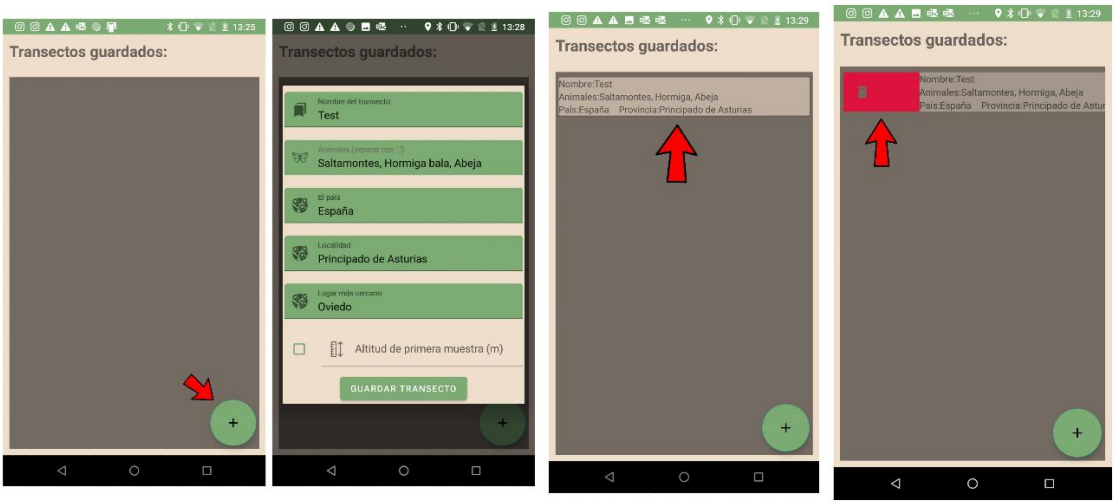


Ilustración 36:Proceso de creación y eliminación de transectos

8.2 Conexión al sistema auxiliar para la toma de muestras ambientales

Para conectar la aplicación con el ESP32 hay que realizar lo siguiente:

1. Es indispensable conectar una batería al ESP32 para encender el dispositivo portátil auxiliar.
2. Dentro de la aplicación pulsar en el transecto que se vaya a realizar.
3. Por último, clicar en el símbolo de bluetooth y esperar que aparezca una ventana emergente “CONECTADO” que indica que el proceso es correcto.

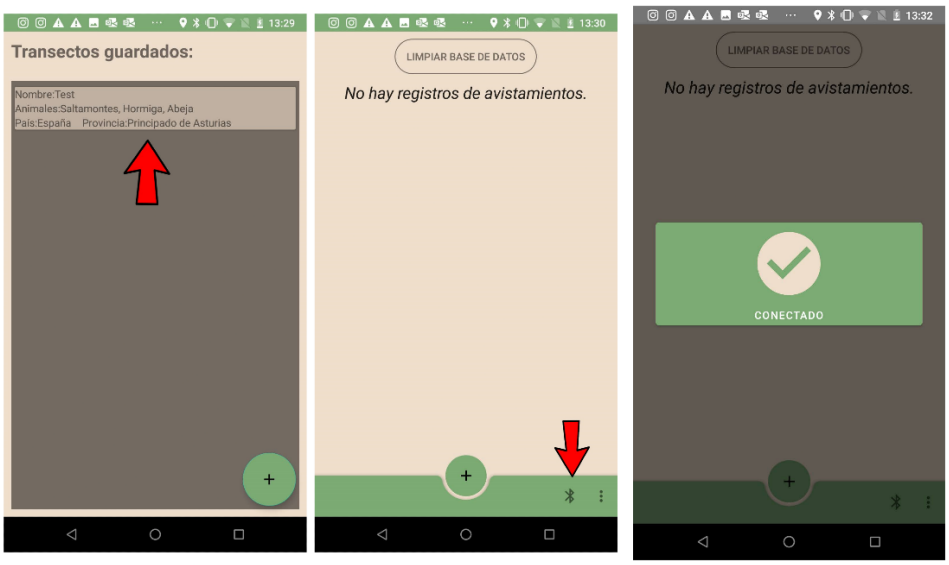


Ilustración 37: Conexión con el ESP32.

8.3 Añadir un nuevo avistamiento

El proceso que hay que seguir para añadir un nuevo avistamiento es el siguiente:

1. Una vez dentro de la ruta que se va a seguir, clicar en el botón “+” de la parte inferior.
2. Una nueva pantalla se cargará con varios campos disponibles, de los cuales solo son obligatorios el nombre de la especie, la hora, el día y la posición geográfica.
3. En esta vista también hay una serie de valores opcionales a los que se puede acceder pulsando en el botón de “MÁS DATOS”. Estos valores serán automáticamente rellenados (junto con la latitud y longitud) gracias al ESP32 en el caso de que esté conectado por bluetooth.
4. Una vez rellenado todo, para terminar y almacenar el animal, solo habrá que pulsar el botón con el logo “disquete”.

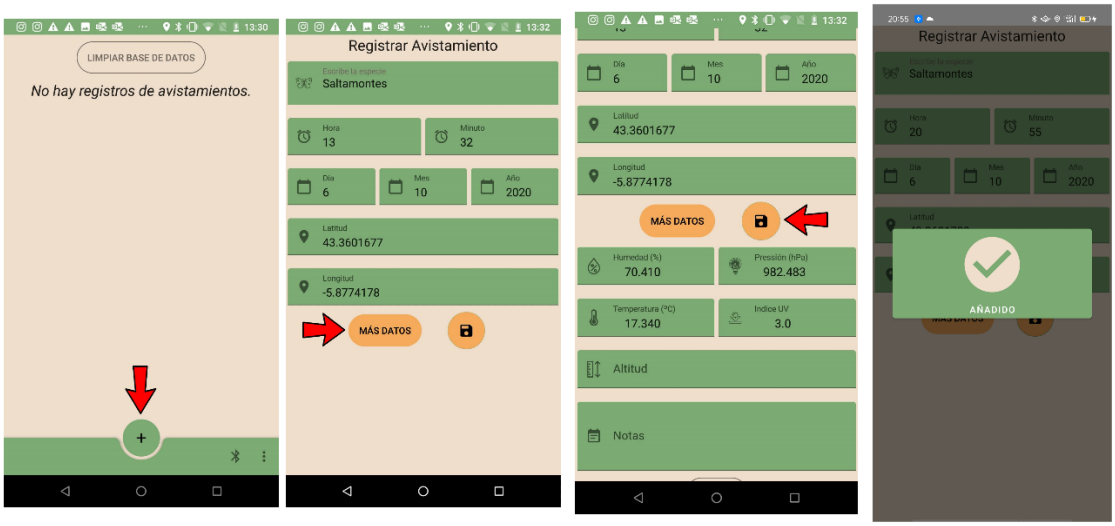


Ilustración 38: Proceso para añadir un avistamiento.

8.4 Configuración de la altura

Hay dos posibles formas de fijar una altura base con la que poder estimar las diferentes alturas de los distintos avistamientos,

- La primera consiste en establecer, a priori, la altura del primer avistamiento para lo que hay que seguir los pasos siguientes:
    1. El primer paso se puede realizar de dos maneras distintas:
        a. Bien añadiendo en el dialogo de creación de un transecto, una altura en la sección de “Altitud de primera muestra”.
        b. O bien, pulsando en los tres puntos, dentro de la ruta que se va a seguir, para luego seleccionar la opción de “Establecer siguiente altitud” y finalmente introducir la altura de la siguiente captura.
    2. Añadir un avistamiento nuevo a esa altura con el ESP32 conectado para así establecer la presión necesaria.

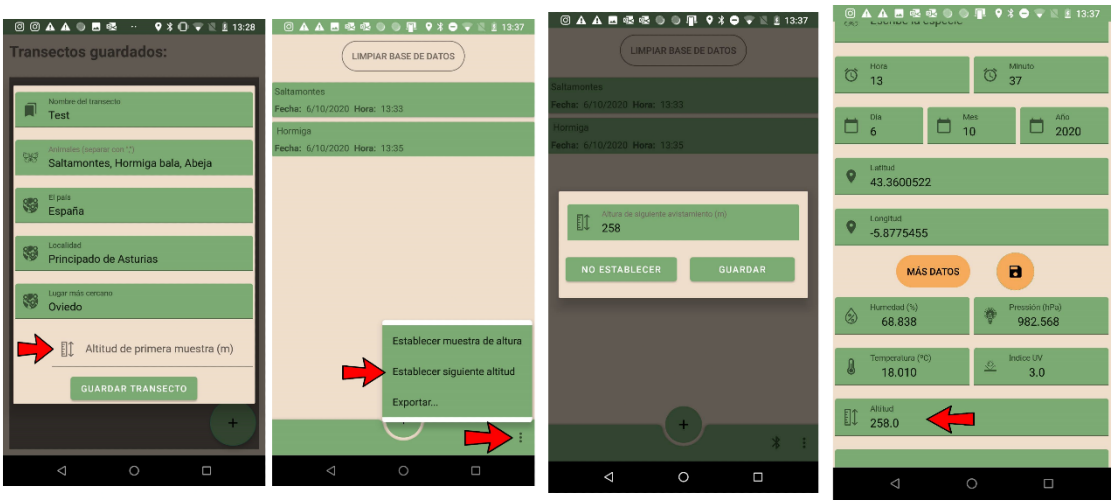


Ilustración 39: Configuración de altura de la siguiente captura.

- La segunda manera de configurar la altura es más fácil y directa:
    - Con el ESP32 conectado, el usuario pulsa en los mismo tres puntos que antes, pero esta vez selecciona “Establecer muestra de altura”.

- Se cargará un dialogo en el que se tendrá que introducir la altura.
- Para finalizar el proceso se seleccionará “MEDIR PRESIÓN Y GUARDAR”.
- También se podrá desactivar la estimación de la altura desde este dialogo.

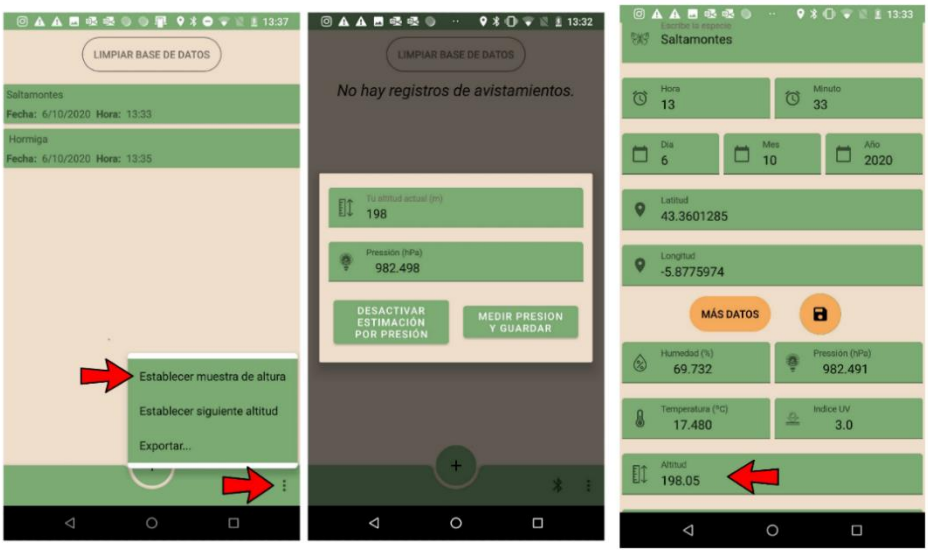


Ilustración 40: Configuración de la estimación de la altura

La principal diferencia entre estos dos métodos es que mientras el primero necesita prever la altura del siguiente avistamiento, el segundo solo necesita saber la altura actual donde se encuentre el usuario y estar conectado al microcontrolador (para obtener la presión).

Una vez configurado esto, para poder estimar las distintas alturas, se necesitan los valores de presión y temperatura, que se autocompletarán directamente si se está conectado al ESP32.

8.5 Administración de avistamientos

Junto con los avistamientos creados en el transecto, están disponible una serie de acciones que facilitan el manejo de entradas en la base de datos:

- El animal, al igual que el transecto se puede eliminar deslizándolo hacia la derecha
- Está disponible la edición de un animal si se desliza hacia la izquierda.
  - Con lo que cargara una vista parecida a la de añadir un nuevo avistamiento, pero con todos los datos con los que fue añadido precargados.
  - Después de editar los valores necesarios, habrá que pulsar en el disquete para guardar los cambios.
- Clicando en el animal, se visualizarán todos los datos del avistamiento.

Ilustración 41: Administración de avistamientos.

8.6 Exportación de base de datos

Dentro de la aplicación se otorga al usuario la posibilidad de exportar los datos obtenidos de una ruta a un archivo en la memora local o a otra plataforma.

- Para exportar a un archivo CSV local, hay que seguir los siguientes pasos:
    1. Pulsar los tres botones y después la opción de “Exportar…”
    2. En el diálogo que aparece pulsar en “MOVIL”.
- Ir a la ubicación en la que se quiere almacenar y seleccionar un nombre si se le quiere dar uno especifico.

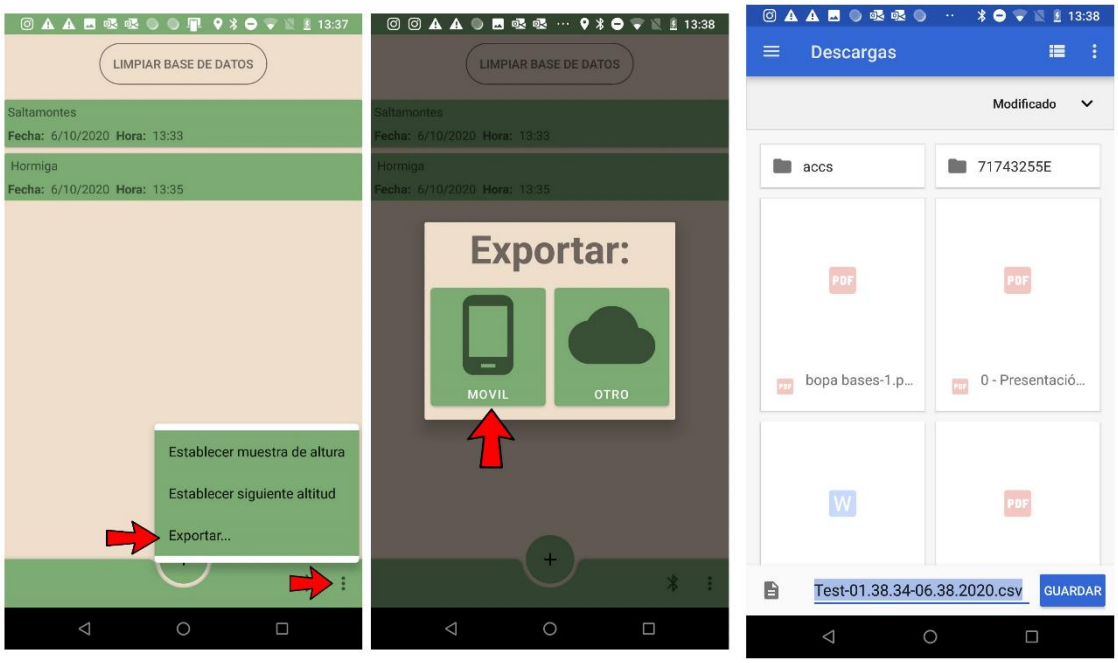


Ilustración 42: Exportar a archivo CSV local

- Para exportar a otra plataforma:
    1. Pulsar los tres botones y después la opción de “Exportar…”
    2. En el diálogo que aparece pulsar en “OTRO.”
    3. Seleccionar la plataforma.

4. Este paso varía dependiendo de la plataforma, por ejemplo, si seleccionamos exportar a “Gmail”, se enviará el archivo CSV por correo electrónico

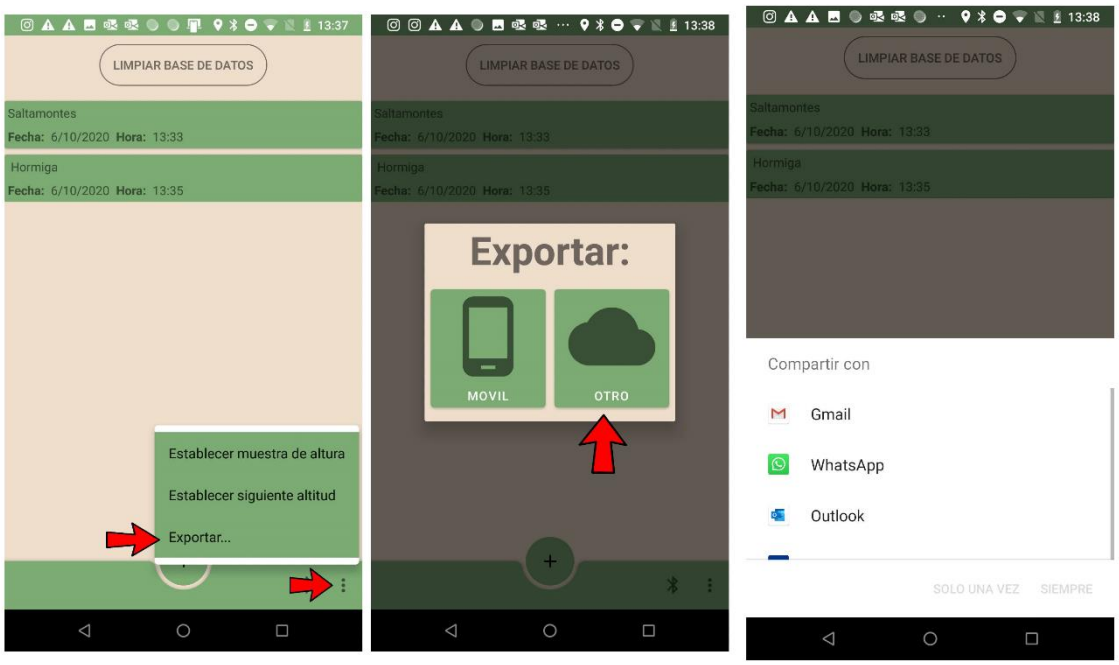


Ilustración 43: Exportar archivo CSV a otra plataforma

9. Pruebas de campo

Una vez desarrollado el sistema se realizó una prueba de campo siguiendo la ruta que aparece en la siguiente captura.

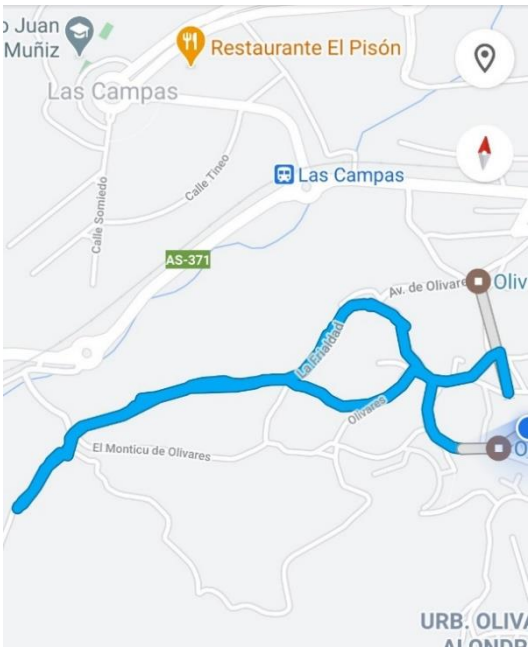


Ilustración 44: Ruta seguida durante la prueba de campo.

Para poder realizar el anterior transecto, se transportó el dispositivo auxiliar para la toma de muestras ambientales en el interior de una pequeña caja. Como se puede observar en las

siguientes fotografías, una vez se conecta una batería, el indicador led se ilumina, mostrando así el inicio de la toma de datos.

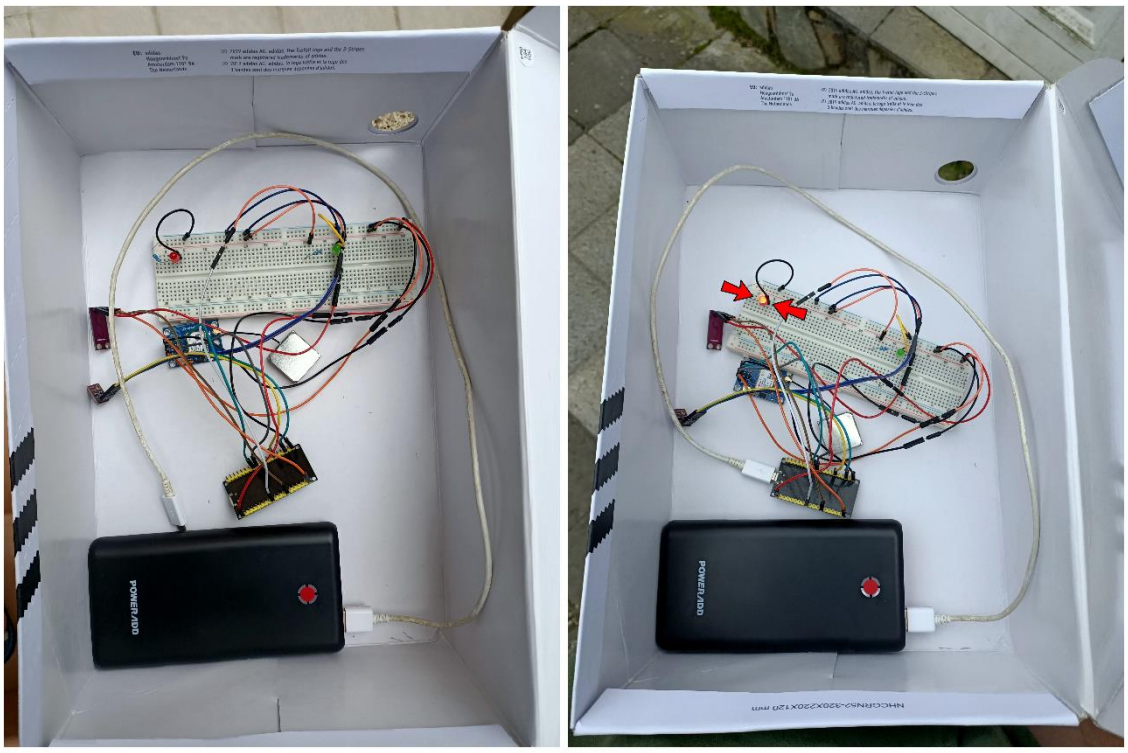


Ilustración 45: Caja con el dispositivo auxiliar ESP32 apagado y encendido en el interior.

Una vez encendido, se creó un nuevo transecto al que se le dio un nombre reconocible y una serie de posibles animales que podrían aparecer a lo largo de la ruta.

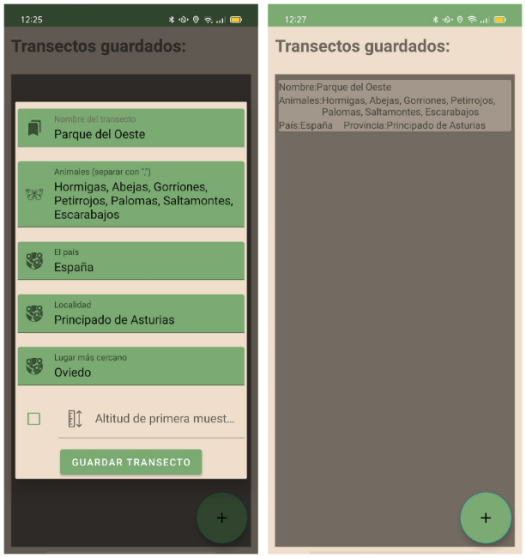


Ilustración 46: Creación del transecto de la prueba de campo.

A continuación, se conectó por conexión BLE el microcontrolador y se tomó una muestra de presión a una altitud conocida, con el fin de poder estimar, más adelante, la altura de los distintos avistamientos.

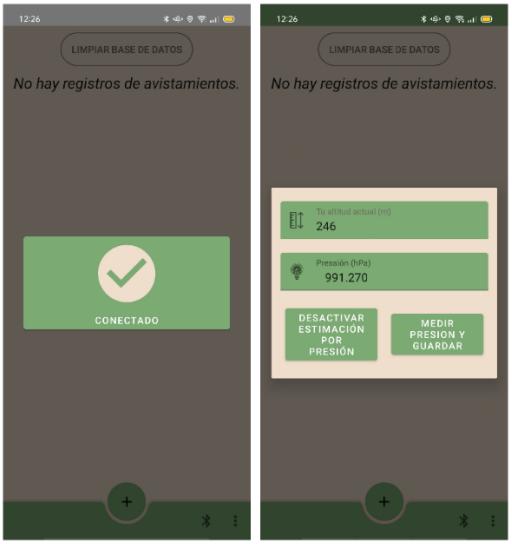


Ilustración 47: Conexión BLE y muestra de altura de la prueba de campo.

Tras realizar estos pasos, dio comienzo la ruta durante la que se registraron los distintos animales que fueron avistados. Todo el proceso trascurrió sin incidencias en ninguna de las partes de GeoFauna (Android y dispositivo auxiliar portátil).

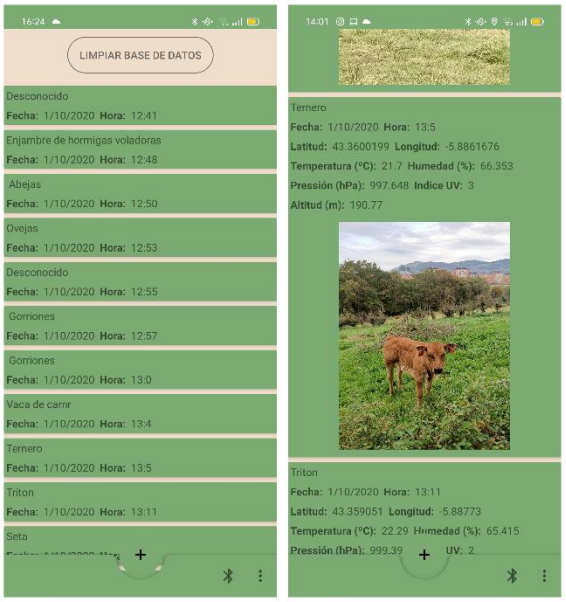


Ilustración 48: Resultado final de la prueba de campo.

Concluido el transecto, y ya de vuelta, se exportaron los datos obtenidos durante el trascurso de la ruta a un ordenador, primero a un archivo CSV y posteriormente a un Excel.

Especie	Longitud	Latitud	Fecha	Hora	Humedad	Temperatura	Es altitud estimada con presiÃ³n	Altitud	Presion	Indicie UV	Lugar	Pais	Notas
Desconocido	-5878073	433609473	01/10/2020	12:41:00	64625	2213	TRUE	21495	994854	2	Principado de Asturias	EspaÃ±a	
Enjambre de hormigas voladoras	-5878073	433609473	01/10/2020	12:48:00	65169	2205	TRUE	20972	995457	3	Principado de Asturias	EspaÃ±a	
Abejas	-58813195	433618505	01/10/2020	12:50:00	63458	2228	TRUE	20999	995422	2	Principado de Asturias	EspaÃ±a	
Ovejas	-58820807	433613942	01/10/2020	12:53:00	65921	2188	TRUE	21353	99502	3	Principado de Asturias	EspaÃ±a	
Desconocido	-58823943	433609532	01/10/2020	12:55:00	6517	2174	TRUE	21397	994971	3	Principado de Asturias	EspaÃ±a	Platero
Gorriones	-58827433	433606336	01/10/2020	12:57:00	63214	2222	TRUE	21402	994959	3	Principado de Asturias	EspaÃ±a	
Gorriones	-5883995	433603833	01/10/2020	13:00:00	62092	2204	TRUE	20465	996041	3	Principado de Asturias	EspaÃ±a	
Vaca de carnr	-58858056	433600871	01/10/2020	13:04:00	6521	2193	TRUE	19269	997421	3	Principado de Asturias	EspaÃ±a	
Ternero	-58861676	433600199	01/10/2020	13:05:00	66353	217	TRUE	19077	997648	3	Principado de Asturias	EspaÃ±a	
Triton	-588773	43359051	01/10/2020	13:11:00	65415	2229	TRUE	17552	999395	2	Principado de Asturias	EspaÃ±a	
Seta	-58879362	433589222	01/10/2020	13:12:00	64664	2231	TRUE	17635	999299	2	Principado de Asturias	EspaÃ±a	
Petirrojo	-58886803	433580728	01/10/2020	13:16:00	66874	2213	TRUE	18416	998401	3	Principado de Asturias	EspaÃ±a	
Avispa asiÃ¡tica	-58886757	433580783	01/10/2020	13:18:00	65704	2249	TRUE	1836	998457	3	Principado de Asturias	EspaÃ±a	
Mariposa	-58883832	433583632	01/10/2020	13:19:00	67397	2224	TRUE	18268	998569	3	Principado de Asturias	EspaÃ±a	Monocolor oscuro y de pequeÃ±o tamaÃ±o
Lagartija	-58851758	433602607	01/10/2020	13:28:00	66228	2261	TRUE	19901	99668	3	Principado de Asturias	EspaÃ±a	
TÃ³rtola	-58846193	433603966	01/10/2020	13:29:00	61631	2288	TRUE	20244	996281	3	Principado de Asturias	EspaÃ±a	
Hormigas	-5880216	433608682	01/10/2020	13:36:00	60373	2389	TRUE	22768	993378	3	Principado de Asturias	EspaÃ±a	Parecen hormigas bala

Ilustración 49: Tabla Excel con los datos de la prueba de campo

Por tanto, se puede concluir tras verificar por medio de las pruebas funcionales que se han cumplido los objetivos iniciales del proyecto. No solo se cumplen todos los todos los requisitos funcionales que debía tener el sistema creado llamado GeoFauna , sino también otros como la facilidad de uso al ser totalmente intuitiva

10. Conclusiones

Con el proyecto terminado y después de una larga reflexión sobre todo el desarrollo, se puede afirmar que se cumplieron no solo la principal meta, desarrollar un sistema capaz de ayudar a los biólogos con la realización de transectos, sino también los objetivos principales y secundarios propuestos en el apartado “2 Objetivos”.

Aunque el proyecto concluyó de manera satisfactoria, cabe destacar que muchos de los problemas que surgieron durante el transcurso de este TFG fueron a raíz de la inexperiencia general del desarrollador y de un diseño inicial pobre debido a la falta conocimiento en las nuevas tecnologías utilizadas como fueron el Bluetooth Low Energy y la programación reactiva. Estas carencias fueron subsanadas con esfuerzo y autoaprendizaje lo que ha supuesto un importante y motivador reto.

Por este motivo se puede afirmar que en el desarrollo de este TFG se han aprendido no solo los conceptos técnicos señalados anteriormente sino otros como planificación y análisis que son base de los desarrollos de cualquier software. Por todo esto, se concluye señalando que el resultado de este TFG ha sido muy provechoso no solo a nivel técnico y profesional sino, y sobre todo, a nivel personal.

Teniendo en cuenta estas conclusiones, se espera que con este sistema se ayude a los etólogos y aficionados al avistamiento de animales al simplificar los procesos de conteo/muestreo de transectos.

11. Ampliaciones

Este proyecto podría tener una serie de posibles líneas de ampliación:

- Aunque, ahora mismo se puede utilizar el sistema actual para el control de flora en una zona, se podría adaptar y especializar la aplicación para este uso concreto. Esta ampliación tendría que llevarse a cabo tanto en la interfaz de usuario como en los valores opcionales (sensores) que permiten almacenar los registros, ya que para GeoFauna solo se tuvo en cuenta el avistamiento de animales.
- Otra posible línea de ampliación sería una mejora general de la interfaz de usuario, para que fuera aún más intuitivo.

- Una posible ampliación más técnica al sistema sería la incorporación de una red neuronal que ayudara a identificar los animales que se vayan encontrando a través de una fotografía.
- Siguiendo con el anterior caso y con la inteligencia artificial, esta aplicación podría también ayudar a entrenar una red neuronal común para el anterior propósito a través de fotografías que se obtuvieran con la aplicación.

Referencias

[1] A. R.Rabinowitz, Manual de capacitación para la investigación de campo y la conservación de la vida silvéstre, New York: WCS, 2003.

[2] FMCN, CONAFOR, USAID y USFS, «Manual para trazar la Unidad de Muestreo en bosques, selvas, zonas áridas y semiáridas,» Comisión Nacional Forestal-Fondo Mexicano para la Conservación de la Naturaleza, México, 2018.

[3] Bluetooth SIG, INC., «Bluetooth,» [En línea]. Available: https://www.bluetooth.com/. [Último acceso: 14 11 2020].

[4] J. Macias, «Solid Gear,» 24 5 2017. [En línea]. Available: https://solidgeargroup.com/bluetooth-ble-el-conocido-desconocido/. [Último acceso: 11 09 2020].

[5] Microsoft, «Microsoft docs,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries. [Último acceso: 13 11 2020].

[6] VTScada, «VTScada,» [En línea]. Available: https://www.vtscada.com/about-trihedral/. [Último acceso: 2020 11 13].

[7] American Meteorological Society, «Glossary of Meteorology,» 9 12 2015. [En línea]. Available: https://glossary.ametsoc.org/wiki/Hypsometric_equation. [Último acceso: 6 11 2020].

[8] World Health Organization, «World Health Organization,» 2003. [En línea]. Available: https://www.who.int/uv/publications/en/uvispa.pdf. [Último acceso: 6 11 2020].

[9] «Biosfera consultoria Medioambiental,» [En línea]. Available: http://www.biosfera.es/sistema-monitorizacion-fauna-no-intrusivo-scout/. [Último acceso: 10 09 2020].

[10] d. FosseyFund, «Animal Observer,» [En línea]. Available: https://fosseyfund.github.io/AOToolBox/index.html. [Último acceso: 10 09 2020].

[11] C. M.-C. Marina Haynes, «TerpConnect,» 1996. [En línea]. Available: http://terpconnect.umd.edu/~wrstrick/secu/ansc455/lab1pt2.htm. [Último acceso: 10 09 2020].

[12] «PcWorld,» 16 25 2019. [En línea]. Available: https://www.pcworld.es/articulos/smartphones/iphone-vs-android-cuota-de-mercado-3692825/. [Último acceso: 10 09 2020].

[13] Google, «Google Lens,» [En línea]. Available: https://lens.google.com/. [Último acceso: 10 09 2020].

[14] abhasm, «iNaturalist,» [En línea]. Available: https://www.inaturalist.org/pages/seek_app. [Último acceso: 10 09 2020].

[15] ReactiveX, «RxKotlin-Github,» [En línea]. Available: https://github.com/ReactiveX/RxKotlin. [Último acceso: 30 09 2020].

[16] ReactiveX, «ReactiveX,» [En línea]. Available: http://reactivex.io/.

[17] Polidea, «RxAndroidBle,» [En línea]. Available: https://github.com/Polidea/RxAndroidBle. [Último acceso: 30 09 2020].

[18] Android , «Android developers,» [En línea]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth-le. [Último acceso: 14 11 2020].

[19] K. Townsend, «Adafruit,» 20 03 2014. [En línea]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. [Último acceso: 29 09 2020].

[20] Espressif System, «Espressif,» 2020. [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Último acceso: 4 11 2020].