UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DISPOSITIVO WEARABLE DE MEDIDA DE RENDIMIENTO EN DEPORTES DE RAQUETA CON COMUNICACIÓN CON DISPOSITIVO MÓVIL

Diego Pérez Lavarías

Tutor: Marina Alonso Díaz

Cotutor: Salvador Coll Arnau

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 19 de junio de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

Este proyecto tiene como objetivo el desarrollo de una aplicación móvil cuya utilidad sea la de realizar un análisis del rendimiento en deportes de raqueta utilizando los datos medidos en un dispositivo *wearable* para obtener mejoras en la técnica y en la prevención de posibles lesiones, de cara a un grupo de usuarios de tipo aficionado que no tienen disponibles herramientas de este tipo a su alcance.

Para ello, se ha realizado un estudio del mercado actual de dispositivos wearables válidos a adoptar, así como de los sistemas móviles en los que se puede implementar una aplicación de este tipo.

Una vez adoptada la solución más apropiada, se trata el desarrollo de la aplicación que se encarga de tomar los datos de los sensores del wearable usando Bluetooth Low Energy, colocado en la muñeca con la cual se coge la raqueta, y llevar a cabo el análisis correspondiente, mostrando mediante una interfaz gráfica el movimiento registrado, con lo que el usuario podrá realizar comparaciones entre sus golpes, que quedan almacenados en el historial.

Resum

Aquest projecte té com a objectiu el desenvolupament d'una aplicació mòbil amb la qual es puga realitzar una anàlisi del rendiment en esports de raqueta mitjançant les dades mesurades amb un dispositiu wearable per a obtindre millores en la tècnica y en la prevenció de posibles lesions, orientada a un grup d'usuaris aficionats que no tenen al seu abast ferramentes d'aquest tipus.

Amb eixa finalitat, s'ha dut a terme un estudi del mercat actual de dispositius wearables vàlids a utilitzar, així com dels sistemes operatius mòbils en els quals es pot implementar una aplicació d'aquest tipus.

Seguidament a l'elecció de la solució més apropiada, es tracta el desenvolupament de l'aplicació la qual s'encarrega de prendre les dades dels sensors del wearable per Bluetooth Low Energy, col·locat al canell de la mà amb la qual s'agafa la raqueta, i analitzarles, mostrant el movimient enregistrat mitjançant una interfaç gràfica, donant al usuari l'opció de realitzar comparacions entre els seus colps, que s'emmagatzemen a l'historial.

Abstract

This project has the objective of developing a mobile application which performs an analysis of performance in racket sports using data measured by a wearable device, to improve both technically and in injury prevention, and oriented to amateur athletes, who do not usually reach this kind of tools due to money issues.

To accomplish that purpose, a wearable market research has been carried out, as well as a mobile operative system study, to observe which one is the most accurate choice for this app.

Subsequently, the development of the application is introduced. This app will be in charge of taking the data from the wearable via Bluetooth Low Energy, which is placed in the user's wrist, and analyze it, showing the results from the registered movement using a graphic interface, with which the user will be able of making a comparison between this data and his previous registered movements, which get stored.

Índice

Capítulo 1. Introducción

1.1 Motivación

He decidido realizar este proyecto porque reúne varios de mis intereses, el deporte tanto a nivel aficionado como más avanzado, y las nuevas tecnologías que incluyen los dispositivos wearables.

Me considero deportista desde siempre, compitiendo federado desde los 8 años y teniendo en cuenta este tipo de actividades como claves en mi desarrollo personal, y entiendo que un trabajo como éste puede ayudar a personas con actividades similares a las mías.

El interés en la tecnología y concretamente en las tecnologías innovadoras también ha estado presente desde siempre en mi vida, comenzando desde muy joven a utilizar dispositivos como consolas de videojuegos, teléfonos móviles, ordenadores... Considero que es el motivo principal por el que decidí estudiar este Grado, y este proyecto me pareció una buena forma de combinar estos estudios con este tipo de tecnologías, utilizando los conocimientos relacionados con aplicaciones Android y Java y el uso de bases de datos, así como aprendiendo cosas nuevas relacionadas con dispositivos externos con los que no había trabajado nunca.

A continuación se entrará en detalle de la importancia de este proyecto específico para los deportes de raqueta y con el uso de dispositivos *wearable*.

1.1.1 Deportes de raqueta

Es un hecho que en los últimos años los deportes de raqueta, en especial el tenis y el pádel, se han establecido como una parte muy importante de entre los deportes que ocupan el tiempo de ocio de los habitantes de España. Como se puede observar en la Figura 1.1, el número total de federados entre ambos deportes ronda los 150.000 cada año, a los que hay que sumar todas las personas que lo practican de forma amateur, sin licencia. La mayoría de estos deportistas no tienen a su alcance herramientas con las cuales medir y perfeccionar su técnica ni entrenadores que les ayuden, lo que impacta de forma directa en su rendimiento y puede llegar a provocar muchas lesiones.

Este ha sido otro factor importante en mi elección, ya que considero que existe mucho potencial en la existencia de una herramienta así que sea accesible para el mayor número de gente posible.

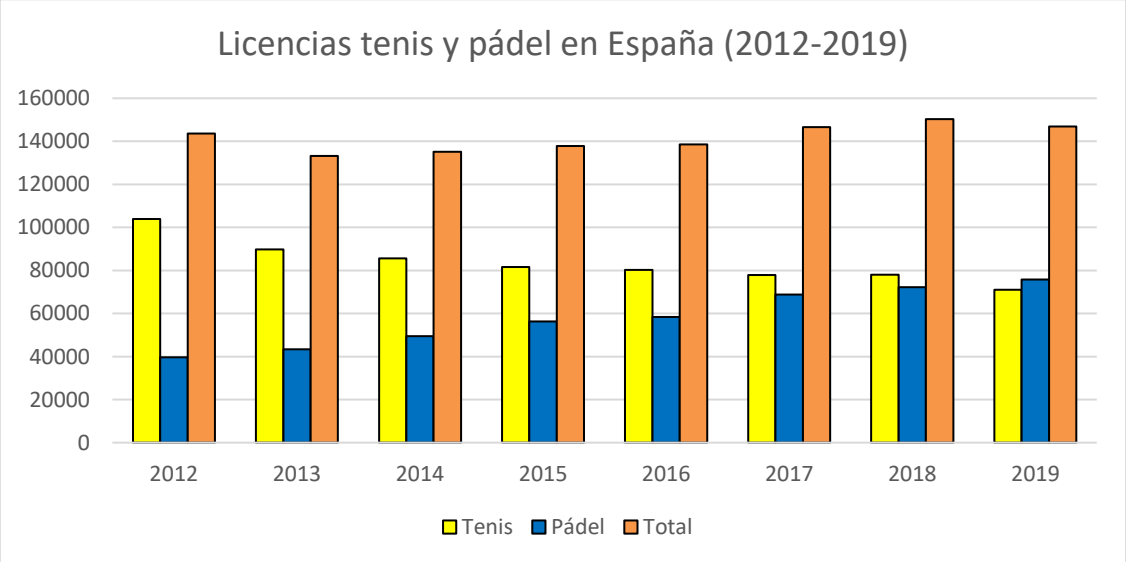

Figura 1.1. Evolución de licencias de tenis y pádel desde 2012. (1)

1.1.2 Tecnología wearable

La tecnología wearable ("vestible") trata de dispositivos electrónicos inteligentes incorporados a la vestimenta, como pueden ser accesorios, y que interactúan con el usuario y otros dispositivos.

Aunque su uso principal es el de la monitorización de actividades, esta tecnología tiene una variedad de aplicaciones que viene creciendo los últimos años, a la par que el campo de conocimiento que se tiene sobre ellos.

Su popularidad ha aumentado en los últimos tiempos y para mucha gente se han convertido en un dispositivo tan necesario como podría ser un teléfono móvil (algunos relojes realizan funciones incluso de llamadas y mensajería) y que resultan de gran ayuda en especial en el terreno deportivo, facilitando el acceso a sistemas de monitorización de la actividad deportiva y de magnitudes físicas como la frecuencia cardíaca, claves en análisis de rendimiento y estado físico.

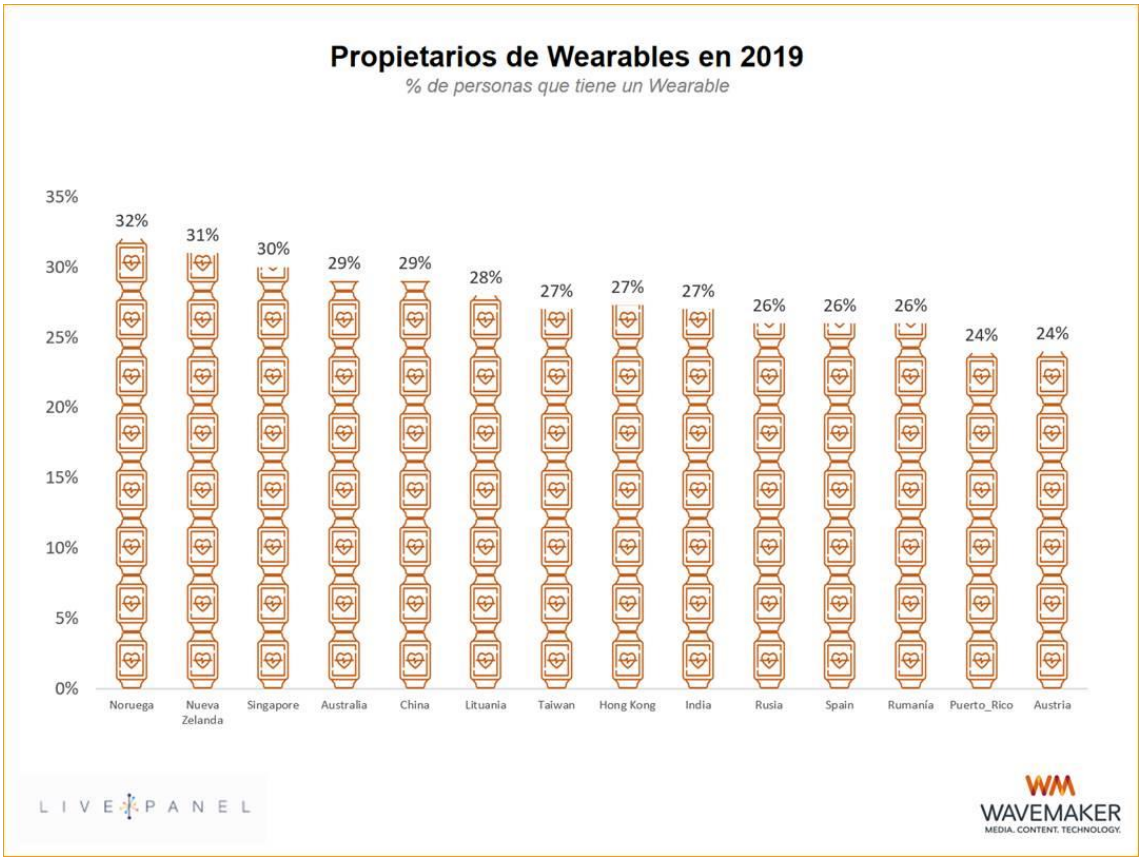

Figura 1.2. Porcentaje de penetración de wearables. (2)

El porcentaje de penetración de este tipo de dispositivos (porcentaje de gente que ha accedido a ellos, que posee uno al menos) rondaba ya el 30% en varios países en 2019, y con tendencia al alza. Esto hace que sea una tecnología muy interesante pero que está aún por descubrir en su mayor parte. Se puede decir que su papel está limitado, aunque tenga esta presencia emergente no se ha desarrollado del todo.

Esta tecnología aporta hoy en día funcionalidades autónomas poco desarrolladas, pero en los próximos años se estima un crecimiento importante.

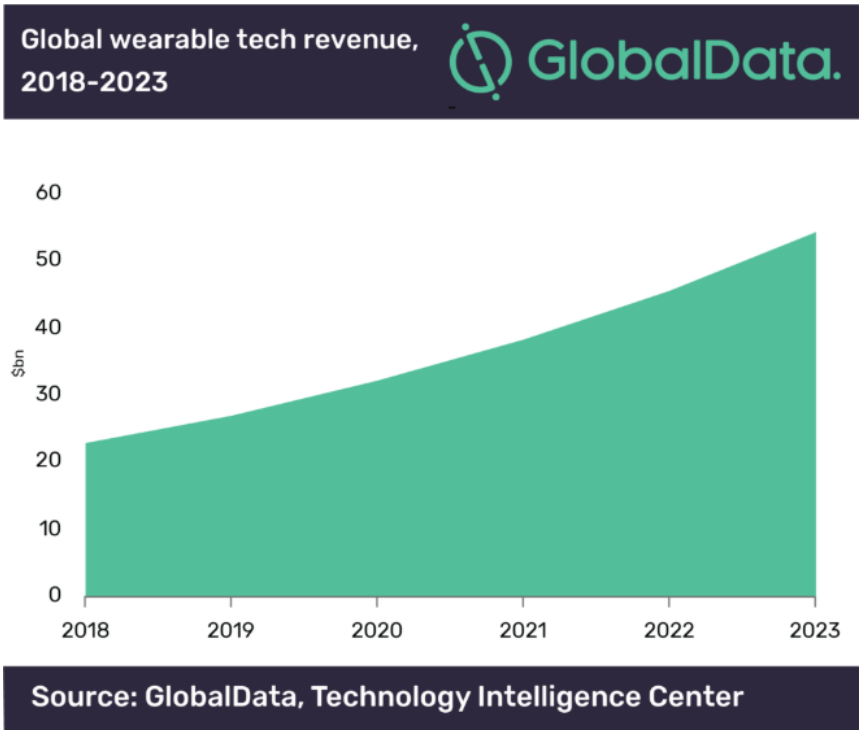

Figura 1.3. Previsión de ingresos por venta de wearables (2018-2023). (3)

Como se aprecia en la figura 1.3, se cerró 2019 con unos ingresos de alrededor de 22.000 millones de dólares por venta de dispositivos wearables. Si bien este estudio es previo a la pandemia de 2020, se cree que se puede mantener el crecimiento de ingresos, que se estimaba podría superar los 50.000 millones de dólares en el año 2023.

En este crecimiento tendrán un papel protagonista los *smartwatches*, que aportarán gran parte de estos ingresos, según indica también este informe.

Esta es la otra razón por la que se ha elegido este tema para realizar el trabajo de fin de grado, me gusta estar informado acerca de las nuevas tecnologías y desde hace un tiempo entiendo los *wearables* como una tecnología que será muy importante en el futuro.

1.2 Objetivos

El principal objetivo de este trabajo de fin de grado es desarrollar una aplicación en Android que permita monitorizar la técnica en el deporte de raqueta realizado para conseguir una mejora de rendimiento y prevención de lesiones en la práctica.

Para ello, se empezará por escoger el dispositivo wearable que se adapte mejor a las necesidades del trabajo sin que se tenga que modificar su *firmware*. Este dispositivo tendrá que permitir tomar medidas con las que se pueda analizar los golpes que efectúe el usuario. También habrá que familiarizarse con el funcionamiento de los sensores integrados.

Otra parte de ese objetivo será conocer y entender el protocolo por el cual se conectan el dispositivo wearable y el smartphone con el que manejaremos la app.

El siguiente objetivo será la implementación de los algoritmos adecuados para la medida precisa de las magnitudes físicas que nos interesan, que serán las que nos aporten los sensores de medida inercial del dispositivo: acelerómetro, giróscopo y magnetómetro. El siguiente paso tras medir estas magnitudes físicas será trabajar con ellas para obtener mejores medidas que se puedan relacionar con la calidad de la actividad física realizada, en este caso los golpes que registre el usuario. El proyecto se ha centrado en golpes de tenis.

Por tanto, el funcionamiento de la aplicación se basará en el registro de golpes por parte del usuario, pudiéndose realizar estos registros de forma individual o en una sesión de múltiples golpes, situando la acción más cerca de lo que sería la práctica del deporte.

El último objetivo será el de cumplir los anteriores sin la necesidad de modificar el firmware del dispositivo de medida, ya que añadiría un punto de complejidad al sistema que se puede evitar con la elección de un dispositivo ya preparado para el uso.

Con esto, se llevará a cabo el desarrollo de la aplicación cumpliendo todos estos pasos y añadiendo funcionalidades básicas que permitan su uso sin dificultad, como puede ser un sistema de usuarios y ayudas gráficas. Todas las funcionalidades que se implementen se explicarán con detalle dentro del capítulo 4.

1.3 Metodología

Se ha definido una estructura clara a la hora de realizar el proyecto paso a paso para trabajar con el diseño y la gestión del proyecto. Esto se conoce como proceso o ciclo de vida del desarrollo de software y es bueno tenerlo claro para facilitar el proceso e incrementar la calidad y productividad del mismo.

En este caso, se ha dividido el proceso en tres bloques, que a su vez están subdivididos en trece módulos. Los tres principales son los estudios previos a la realización de la aplicación, el desarrollo de la propia aplicación y la elaboración de la memoria del proyecto.

Lo que primero se llevó a cabo fue el estudio previo, empezando por la toma de datos de los deportes de raqueta y siguiendo con el estudio del mercado de los dispositivos wearable y de los sistemas operativos de móvil.

En este punto se pudo empezar a trabajar con el bloque de la aplicación, ya que ya se había elegido las herramientas que se iban a utilizar, tanto hardware (el dispositivo wearable) como software (el sistema operativo). Del bloque de estudios quedaba pendiente el módulo de fundamentos teóricos, dentro de lo cual se encuentra la medida y representación de la orientación (con los datos que se obtienen de un sistema de sensores) y el funcionamiento de Bluetooth Low Energy.

De forma paralela, se comenzó a trabajar con el diseño de la aplicación, que consiste en planificar y realizar la estructura que tendrá la aplicación, tanto las actividades donde se ejecutará el código que implemente las funciones deseadas como los *layouts* o pantallas que el usuario verá al utilizar la aplicación y que incluye los botones, imágenes, textos y otros elementos que constituyen las uniones entre actividades.

Una vez terminado este paso, se procedió a crear la base de datos de la aplicación, donde se almacenan los usuarios y datos que registra cada usuario al utilizar la aplicación. Este paso es relativamente sencillo de implementar, pero es necesario realizar retoques una vez se van añadiendo pasos nuevos en el desarrollo del software.

El siguiente paso fue el módulo de usuarios, donde se realiza el inicio de sesión y registro. Se integra con la base de datos, así como con el bloque de menú, donde ya se implementa el borrado de datos y de usuario, así como otras lecturas de la base de datos (vacía en estos momentos, más allá de los usuarios).

Una vez completado el bloque de estudio de los fundamentos teóricos se puede continuar con el bloque de conexión BLE, ya que es necesario conocer el funcionamiento de esta tecnología de conectividad para poder realizar la conexión con el dispositivo wearable. Este bloque va directamente enlazado con el siguiente, en el que se obtienen los datos de los sensores y se almacenan en la base de datos.

Finalmente, con el conocimiento restante del estudio de los fundamentos teóricos (la medida y representación de la orientación) se implementó el módulo de análisis del golpe, que requería de los bloques anteriores de obtención de datos y de acceso a la base de datos.

En el siguiente diagrama se ve de forma clara y estructurada cuánto tiempo se ha dedicado a cada bloque del proyecto.

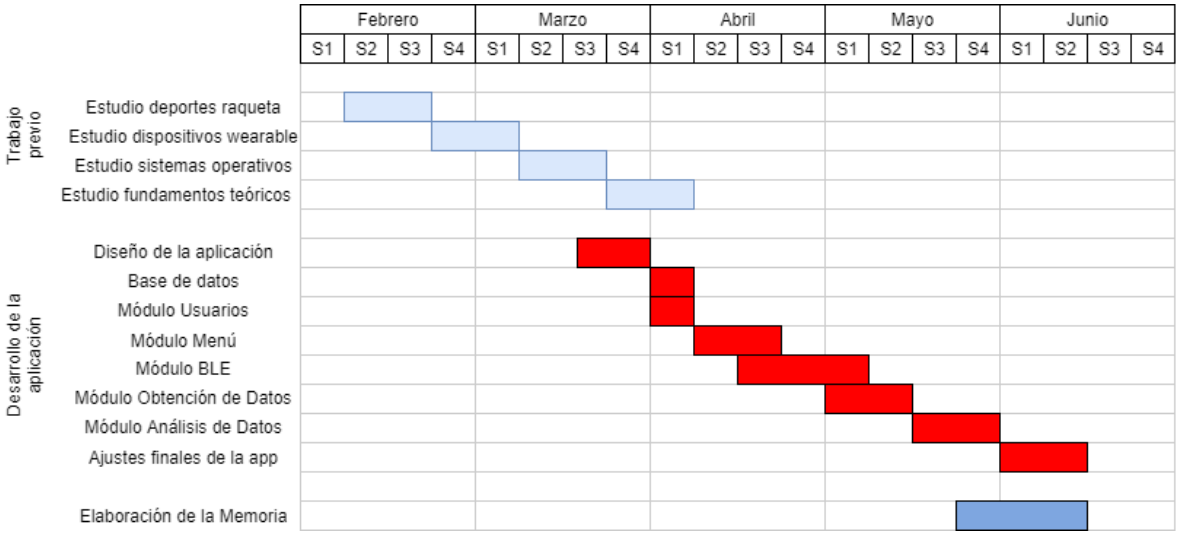

Figura 1.4. Diagrama de Gantt.

Capítulo 2. Identificación y análisis de posibles soluciones

En este capítulo se analizarán las distintas soluciones con las que se podría llevar a cabo el proyecto, tanto de hardware como de software. Se explicarán diferentes herramientas existentes que se podrían utilizar, comparando sus ventajas y sus desventajas.

2.1 Posibles soluciones hardware

A continuación se presentarán los *wearables* disponibles para participar en este proyecto. Se estudiarán sus componentes, con especial énfasis en su conectividad y en sus módulos de sensores de acelerómetro, giroscopio y magnetómetro.

Finalmente, se realiza una comparación entre estos diferentes dispositivos.

2.1.1 MIKROE-2026 (Hexiwear)

El dispositivo de *MikroElectronika* MIKROE-2026 o *Hexiwear* (4) es un smartwatch con formato de kit de desarrollo. Ofrece la posibilidad de crear un dispositivo portátil propio compatible con aplicaciones de móvil. Consta de pantalla OLED de 1.1 pulgadas con 6 botones capacitivos. Conexión Micro USB-B para cargar la batería y para desarrollo del software del dispositivo. Su peso es de 40 gramos.

Contiene sensores de frecuencia cardíaca, temperatura, humedad, presión, luz ambiental y medida de orientación.

Figura 2.1. Dispositivo Hexiwear MIKROE-2026.

El procesador que integra este dispositivo es un NXP-*Kinetis* K64 MCU (5), basado en un ARM Cortex-M4:

- Velocidad de reloj de hasta 120 MHz.
- Memoria SRAM de 256 KB.
- Memoria flash de 1 MB.

Para la comunicación, incorpora un procesador NXP-*Kinetis* KW4x (5), basado en un ARM Cortex-M0+ y que proporciona conectividad BLE (Bluetooth Low Energy).

En cuanto a los sensores que incorpora, tiene dos apartados: por una parte el FXOS8700CQ, que integra acelerómetro y magnetómetro, y por otra el FXAS21002, que proporciona medidas del giróscopo. Ambos proporcionados por NXP. (5)

Como dato interesante, el desarrollo de este dispositivo se llevó a cabo gracias a un proceso de crowdfunding y es un proyecto de código libre.

2.1.2 SensorTag CC1350STK EU

Se definen como un entorno de desarrollo fácil de usar con un set de herramientas amplio y común para todos los dispositivos, además de una duración de la batería muy buena.

La idea es la de una familia de dispositivos de precio reducido, inalámbricos y con sensores de baja potencia, con los cuales llevar a cabo proyectos de medición de datos, automatización de sistemas del hogar, sistemas de seguridad y alarmas, sistemas de salud inalámbricos, etc.

Estas familias de dispositivos permiten la modificación del software mediante un pack de desarrollo que aporta el fabricante.

Sus dimensiones son de 5 x 6.7 x 1.4 cm.

Figura 2.2. Dispositivo SensorTag CC1350. (6)

El microprocesador que incorporan está basado en un ARM Cortex-M3 de 32 bits:

- Velocidad de reloj de hasta 48 MHz.
- Memoria flash de 128 KB.
- Memoria SRAM de 20 KB.
- 8 KB de caché/RAM.

También contienen hasta diez sensores, con los cuales se puede medir magnitudes como temperatura, humedad y presión, junto con el módulo MPU-9250 (7) que se define como controlador de sensores de ultra baja potencia, que permite el uso autónomo, con 20 KB de SRAM y que soporta actualización OTA (Over-the-Air).

La conectividad de este dispositivo se basa en dos sistemas: por una parte permite Bluetooth Low Energy, y por otra añade funcionalidad Sub-1GHz, que consiste en usar una frecuencia inferior, por lo tanto aporta mayor rango y menor potencia.

2.1.3 SensorTag CC3200STK - WIFIMK

Es un dispositivo muy similar a los de la familia anterior. Contiene los mismos sensores y la MPU-9250 de *Invensense*. La diferencia se encuentra en el procesador que incorpora, que en este caso está basado en un ARM Cortex-M4 de 32 bits:

- Velocidad de reloj de hasta 80 MHz.
- RAM de 256 KB.

También se encuentra una gran diferencia en la comunicación, realizándose en este caso por Wi-Fi (por estándar 802.11 b/g/n), lo que permite el uso de TCP/IP, TLS/SSL, HTTP y otros protocolos de Internet. Esto es útil porque permite la carga directamente a Internet en el caso de que se utilice un servidor web o en la nube para almacenar los datos.

También es diferente en las dimensiones, ya que es algo más pequeño que el dispositivo anterior: 3.2 x 4.2 x 0.8 cm.

Figura 2.3. Dispositivo SensorTag CC3200. (6)

2.1.4 STEVAL-WESU1

El STEVAL-WESU1 de ST MicroElectronics (8) es un dispositivo pensado para wearable, aplicaciones de medición de datos con un set completo de ejemplos firmware. Es sencillo de programar, mediante el ST-LINK. En este caso contiene cuatro sensores: el IMU habitual más el sensor de presión.

Tiene disponibles apps en iOS y Android que miden y muestran por pantalla el comportamiento de los sensores.

Sus dimensiones son de 3x3.5 cm, y viene en formato de cápsula que se introduce en una correa tipo reloj (incluido en el pack de venta). Peso de 15 gramos.

Figura 2.4. Dispositivo STEVAL-WESU1.

Incorpora un microprocesador STM32L151VEY6 basado en ARM Cortex-M3 de 32 bits:

- Velocidad de reloj de hasta 32 MHz.
- Memoria RAM de 48 KB.
- Memoria Flash de 512 KB.

La comunicación se realiza mediante BLE, gracias al módulo BLUENRG-MS, de acuerdo con Bluetooth 4.1.

Todos los sistemas son de fabricación propia de ST Microelectronics.

La unidad de sensores de orientación no consiste solamente en un módulo como en alguno de los casos anteriores, sino que trata de varios módulos:

- LSM6DS3, que incluye el acelerómetro y el giroscopio, ambos de 3 ejes.
- LIS3MDL, el magnetómetro de 3 ejes también.
- LPS25HB, sensor de presión.

2.1.5 Tabla comparativa

Dispositivo	Procesador	Vel. Reloj (MHz)	Conectividad	Dimensiones (cm x cm x cm)	Precio (€)
Hexiwear	NXP-Kinetis K64 MCU	120	BLE	Sin datos	44,1
CC1350STK	ARM Cortex-M3	48	BLE, Sub-1GHz	5 x 6,7 x 1,4	69
CC3200STK	ARM Cortex-M4	80	Wi-Fi	3,2 x 4,2 x 0,8	46,79
STEVAL-WESU1	ARM Cortex-M3	32	BLE	3 x 3,5 x 1	46,75

Tabla 2.1. Comparación de datos de dispositivos.

En la tabla comparativa podemos observar que el Hexiwear MIKROE-2026 se sale de la norma, teniendo un procesador diferente fabricado por NXP mientras el resto tiene procesadores de fabricación propia pero basados todos en ARM Cortex. Esto hace que su procesador tenga mayores prestaciones de velocidad.

En cuanto a la conectividad, lo establecido es el Bluetooth de baja energía o BLE, que se ha mostrado como un sistema muy útil para este segmento, como se puede observar en el mercado, donde las empresas más grandes como puede ser Xiaomi lo utilizan en sus dispositivos wearable, las pulseras y relojes de actividad de la gama Amazfit. Aun así, destaca el uso de Wi-Fi en el caso del CC-3200STK, lo cual hace que sea una gran opción para aplicaciones que utilicen un servicio web, ya que se puede cargar los datos directamente a un servidor de este tipo.

Fijándonos en las dimensiones, realmente todas las alternativas son de tamaño contenido, con utilidad de reloj de pulsera. No se han encontrado los datos exactos del Hexiwear, pero el peso es de 40 gramos y se vende con pulsera para usarlo de esa manera, así que se asume que es así también.

Por último, en cuanto a precio, vemos que todas las alternativas se sitúan en el mismo rango de entre 40 y 50 euros, excepto el CC-1350STK, que sube de precio debido a su baja disponibilidad en detrimento de dispositivos más modernos. Cabe recordar que este dispositivo salió en 2016.

También se puede realizar una comparación de los sensores que incorporan los dispositivos.

MIKROE-2026: Incluye una unidad de sensores con dos módulos diferentes, por un lado el acelerómetro y magnetómetro y por otro lado el giróscopo. (4)

- Acelerómetro: FXOS8700CQ
  - Rango de ±2g, ±4g, ±8g.
  - Frecuencia de muestreo de hasta 800 Hz.

- Giróscopo: FXAS21002C
  - Rango de ±250º/s, ±500º/s, ±1000º/s, ±2000º/s.
  - Frecuencia de muestreo de hasta 800 Hz.

- Magnetómetro: FXOS8700CQ
  - Rango de ±1200 µT.
  - Frecuencia de muestreo de hasta 800 Hz.

CC1350STK y CC3200STK: Incluyen el mismo sistema, la MPU-9250, de Invensense. (7)

- Acelerómetro:
  - Rango de ±2g, ±4g, ±8g, ±16g.
  - Frecuencia de muestreo de hasta 4 KHz.

- Giróscopo:
  - Rango de ±250º/s, ±500º/s, ±1000º/s, ±2000º/s.
  - Frecuencia de muestreo de hasta 1 KHz.

- Magnetómetro:
  - Rango de ±4800 µT.
  - Frecuencia de muestreo de hasta 8 Hz.

STEVAL-WESU1: Dividen sus sensores de orientación en dos módulos, en este caso, el acelerómetro y el giróscopo en uno y el magnetómetro en otro. (8)

- Acelerómetro: LSM6DS3
  - Rango de ±2g, ±4g, ±8g, ±16g.
  - Frecuencia de muestreo de hasta 1.6 KHz.

- Giróscopo: LSM6DS3
  - Rango de ±125º/s, ±245º/s, ±500º/s, ±1000º/s, ±2000º/s.
  - Frecuencia de muestreo de hasta 1.6 KHz.

- Magnetómetro: LIS3MDL
  - Rango de ±400 µT, ±800 µT, 1200 µT, 1600 µT.[1]
  - Frecuencia de muestreo de hasta 1,6 KHz.

[1] Los datos vienen en Gauss en el *datasheet*. Se ha utilizado la conversión 1 Gs = 10^{-4} T para presentar todos los datos en las mismas unidades.

Con estos datos, podemos observar que en los tres casos las características son bastante similares, teniendo menos rango y algo menos de frecuencia de muestreo en general el dispositivo MIKROE-2026, y estando las otras dos igualadas. Cabe destacar el acelerómetro de la MPU-9250, ya que en rango y en frecuencia de muestreo es la más alta, y también el magnetómetro del STEVAL-WESU1.

2.2 Posibles soluciones software

En este apartado se describirán los diferentes sistemas operativos móviles que se consideran alternativas para realizar el proyecto.

2.2.1 Android

Android es el sistema operativo móvil más utilizado en la actualidad (9). Según datos de Google (que lo desarrolla), se utiliza en 2500 millones de dispositivos activos.

Está basado en el Kernel de Linux y su principal característica es la de ser una plataforma de código abierto y de desarrollo libre por parte de cualquier usuario que quiera ser desarrollador.

Su primera versión fue presentada en 2007 y Android 1 (Apple Pie) salió en septiembre de 2008. Se ha ido desarrollando desde entonces, siendo su última versión Android 10, lanzada en septiembre de 2019.

Figura 2.5. Logotipo de Android.

Los componentes principales del sistema son:

- Aplicaciones: escritas en Java o Kotlin principalmente. Implementan los servicios desarrollados para el usuario.
- Marco de trabajo para apps: la arquitectura está diseñada para que la reutilización de los componentes sea sencilla, de forma que una aplicación publique sus capacidades y cualquier otra pueda hacer uso de ellos, siempre de manera segura (según reglas del '*framework'*.
- Bibliotecas: Android incluye un conjunto de características llamado bibliotecas, donde los desarrolladores podrán acceder a esas características para añadirlas a sus aplicaciones. Hay multitud como SQLite, 3D, etc.
- Núcleo Linux: gestiona los servicios básicos como son la seguridad o la gestión de memoria. Completamente de código abierto.

Con esto, las características que definen a Android son:

1. Código libre para el desarrollo de cualquier usuario con programación en Java/Kotlin.
2. Catálogo de aplicaciones muy amplio a través de su tienda Google Play Store (gracias a la libertad de desarrollo que aporta el código abierto).
3. Diseño adaptable a pantallas de resolución, tamaño y formas muy variables.
4. Soporte multimedia y streaming extensos.
5. Navegación web mediante el uso de HTML.
6. Multitarea real de aplicaciones.
7. Conexiones inalámbricas como Wi-Fi, redes celulares y Bluetooth.
8. Asistente virtual (Google Assistant).
9. Amplia base de usuarios potenciales, válido para todo tipo de gente.

2.2.2 iOS

iOS es el sistema operativo móvil desarrollado por la empresa estadounidense Apple. Fue originalmente pensado para el iPhone, pero más adelante también usado en tabletas y reproductores de música (iPod). Deriva del sistema operativo de Apple para sus dispositivos sobremesa MacOS, y por tanto es un sistema operativo tipo Unix, uno de los primeros S.O. de código libre creado hace décadas en EEUU.

Figura 2.6. Logotipo de iOS.

Su lanzamiento inicial data de 2007 (iOS 1) y la última versión salió en septiembre de 2019, siendo actualizado constantemente, por última vez en abril de 2020 (iOS 13.4.1). (10)

Es el segundo sistema operativo en número de usuarios. Apple no da datos en su web como hace Google, pero se estima que, a nivel mundial, el número de usuarios de iOS es en torno a un tercio del de Android. Esto situaría este número en cientos de millones de usuarios.

Las características de iOS son similares a las de Android, con la mayor diferencia de la adaptabilidad del sistema y la libertad del código. En este caso, el sistema no se adapta a diferentes dispositivos porque no es libre, solamente lo utiliza Apple para sus terminales. En el resto, funciona de manera similar, con aplicaciones implementadas por desarrolladores mediante Swift, el lenguaje creado por la misma empresa para este medio, y puestas en el mercado mediante la tienda App Store.

También tiene soporte multimedia y streaming, navegación web mediante HTML y un asistente virtual, Siri.

2.2.3 Otros sistemas operativos

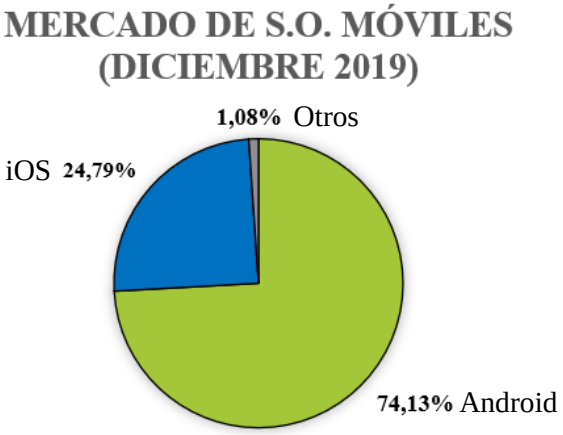

Figura 2.7. Mercado de S.O. móviles. (11)

En este gráfico se presentan los datos de cuota de mercado de los sistemas operativos de móvil, a fecha de diciembre de 2019. Como se ha mencionado anteriormente, Android es el sistema más usado de forma amplia, con un 74% del mercado, seguido de iOS con casi un 25%.

En este apartado se presentarán algunos de ellos.

Kai OS

Es un sistema operativo basado en Linux cuyo objetivo es el de acercar lo mejor de los teléfonos móviles inteligentes a dispositivos asequibles. Esto es, se trata de un sistema diseñado para terminales tecnológicamente accesibles, como los que utiliza la gente mayor. Es una plataforma de código libre surgida de la comunidad de Firefox.

Sus principales características son similares a las de los sistemas dominantes del mercado, pero orientados a teléfonos no táctiles, con una interfaz totalmente optimizada y necesidades bajas de energía y memoria. Es capaz de funcionar en dispositivos con solamente 256 MB de memoria y con duración de la batería de semanas.

Acerca las aplicaciones y el uso de redes celulares para la conexión a Internet a dispositivos como los que se utilizaban en la pasada década.

Se estima que su cuota de mercado es de un 0,35%. (11)

Figura 2.8. Logotipo de Kai OS. (12)

Otros

El resto de sistemas operativos que se utilizan en la actualidad son sistemas instalados en terminales antiguos que se siguen usando, por comodidad o por no querer cambiar. Por ejemplo, el sistema Windows Phone que fue descontinuado en 2015 tras fracasar en ventas, teniendo máximos de un 3-4% a principios de la década de los 2010.

Otros sistemas aún existentes pero minoritarios son Blackberry OS, Symbian y Nokia OS.

En el futuro se piensa que es posible la irrupción de Huawei con su propio sistema debido a los problemas que hubo en 2019 entre esta empresa y los Estados Unidos[2], que provocaron que no pudieran acceder al sistema Android como venían haciendo los últimos años. De todas maneras, en la fecha en que se realizó esta memoria, el problema seguía en pie y Huawei no había desarrollado todavía su sistema, que teniendo en cuenta que Huawei tiene una cuota de mercado de alrededor del 15% de dispositivos del mundo sería uno de los grandes sistemas del panorama.

2.3 Descripción de la solución adoptada

En este apartado se presentará la solución adoptada para la realización del proyecto y se expondrán las razones que han llevado a esta elección, tanto del hardware como del software.

2.3.1 Hardware

Se ha decidido elegir el dispositivo STEVAL-WESU1 de ST Microelectronics.

En primer lugar, se ha elegido debido al protocolo de comunicación que incorpora. Se considera que, en vista de que la mayoría de dispositivos wearables del mercado (como los smartwatch de Xiaomi) utilizan BLE para la comunicación con el smartphone que lo acompaña, es coherente utilizar un dispositivo con este tipo de conexión para nuestro proyecto. Con este protocolo, podremos realizar una conexión de bajo rango de distancia (móvil – wearable) y con un consumo de energía mínimo.

Otro motivo por el cual se ha elegido este dispositivo es su tamaño. Se trata de la alternativa con menores dimensiones de entre las que se han considerado para realizar el proyecto. La idea es la de colocar el dispositivo en la muñeca con la que se utiliza la raqueta para poder medir las magnitudes físicas en el golpeo, y un tamaño pequeño y un peso mínimo es algo básico para poder realizar el deporte sin ninguna molestia o intromisión por parte del dispositivo de medición.

El tercer motivo que se ha tenido en cuenta ha sido los sensores que incorpora. Junto con la IMU de los dispositivos de Texas Instruments, tienen mayor rango que los sensores del Hexiwear, pero hay una gran diferencia en la frecuencia de muestreo de los de TI: su magnetómetro solamente llega hasta los 8 Hz de manera nativa. Los dispositivos de TI nos añaden ese problema, que habría que solucionar mediante la modificación del firmware, mientras que con el STEVAL-WESU1 nos ahorramos este paso y podemos centrarnos desde el inicio en la implementación de nuestro proyecto.

En último lugar, elegimos este dispositivo porque se encuentra disponible en la UPV para su uso inmediato, sin tener que realizar la compra. El precio no se ha considerado demasiado, ya que todos los dispositivos que se han tomado como alternativas presentan un precio semejante, pero la disponibilidad es algo muy apreciado, puesto que poder trabajar desde el principio con el dispositivo facilita esa familiaridad con él de cara a implementar la aplicación.

[2] https://rpp.pe/tecnologia/moviles/huawei-esto-es-lo-que-debes-saber-del-caso-huawei-con-google-android-y-estados-unidos-noticia-1198645

Detalles técnicos

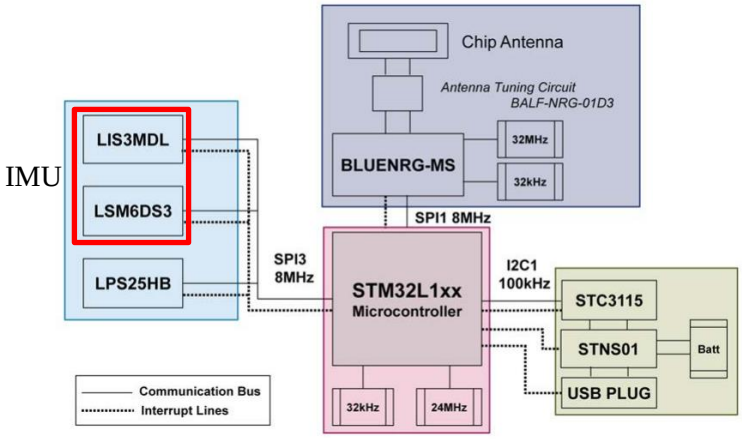

Figura 2.9. Diagrama de bloques funcional del dispositivo.

Como observamos en la figura 2.9, este dispositivo tiene 4 bloques funcionales: el Microcontrolador, el sistema de gestión de energía, el módulo de conexión BLE y los sensores. Todos los diagramas e imágenes de este apartado han sido obtenidos del manual (8).

· **Microcontrolador**

Figura 2.10. Microcontrolador STM32L151VEY6.

Microcontrolador de muy baja potencia basado en el ARM Cortex-M3, un estándar en la industria con núcleo de 32 bits, diseñado para aplicaciones de baja potencia. El bajo consumo es clave para aplicaciones de wearables que necesitan iniciarse periódicamente para ejecutar tareas de software.

· **Gestión de energía**

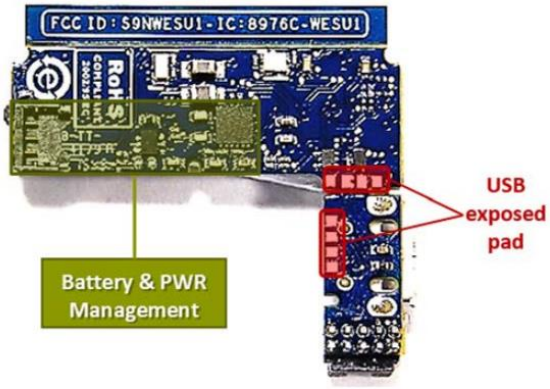

Figura 2.11. Subsistema de gestión de energía y batería.

Se compone de varios módulos: la batería de ion-litio de 100 mAh, el sistema STNS01 que la carga y el módulo STC3115 que implementa la monitorización del nivel de batería, que se encuentran en el reverso de la placa. En el lado principal encontramos también el conector Micro-USB con el que se carga la batería.

El sistema cargador STNS01 es un cargador lineal para baterías de ion-litio de una celda. En este dispositivo, está configurado como cargador de batería y como *switch* entre la fuente de energía de la batería y del conector USB. Incluye un LED que nos indica si carga correctamente (color rojo fijo) o si hay problemas (parpadeo).

El módulo STC3115 incluye las funciones hardware de monitorización de SOC (state-of-charge, nivel de batería). Se mide por corriente y voltaje.

· **Conexión BLE**

Figura 2.12. Módulo BLE.

Se trata de un microprocesador BLE de baja energía, el BLUENRG-MS, implementado de acuerdo con la especificación de Bluetooth v4.1. Puede actuar tanto de maestro como de esclavo en la conexión BLE. Contiene una memoria Flash no volátil.

· **Sensores**

Figura 2.13. Módulo de sensores.

Integra sensores cuyo funcionamiento es ideal para algoritmos de medición de movimientos en wearables, con capacidades de baja energía y alto rendimiento en cuanto a precisión. Contiene 3 sistemas, de los cuales nos centraremos en dos, los que incluyen los sistemas que nos interesan, dejando aparte el LPS25HB que mide presión.

El LSM6DS3 incluye los sensores de acelerómetro y giróscopo. Son ambos digitales y de 3 ejes o 3D, con un consumo de corriente de 1,25 mA. Los rangos, ya expuestos anteriormente, son:

- Acelerómetro: ±2g, ±4g, ±8g, ±16g.
- Giróscopo: ±125º/s, ±245º/s, ±500º/s, ±1000º/s, ±2000º/s.

La frecuencia de muestreo de ambos es de hasta 1,6 kHz.

El otro sistema sensor es el LIS3MDL, que incluye el magnetómetro, también 3D y digital. Puede ser configurado para generar señales al detectar campo magnético. El rango del magnetómetro es de ±4, ±8, ±12, ±16 gauss, que es igual a ±400 µT, ±800 µT, 1200 µT, 1600 µT.

Todos estos módulos van encapsulados en una pequeña caja de plástico que hace la función de corona del reloj. Sumado a esto, en la caja viene una pulsera de goma donde introducir la cápsula y colocarlo en la parte del cuerpo donde se quiera realizar la medición, la muñeca o el tobillo.

Figura 2.14. Cápsula y correa del reloj.

2.3.2 Software

Se ha elegido el sistema operativo Android para el desarrollo de la aplicación que implemente nuestro proyecto.

Se ha elegido por varios motivos. El principal ha sido la característica de este S.O. de ser código libre. Esto hace que la comunidad de desarrolladores sea muy amplia, con mucha documentación accesible en Internet y con un entorno de desarrollo oficial que la misma empresa otorga para cualquiera que quiera introducirse en el desarrollo de apps y con posibilidades para cualquier nivel de usuario, desde el principiante hasta el experto. Personalmente, este es un entorno de desarrollo con el que he trabajado durante el grado, lo que se ha tenido en cuenta ya que estaba familiarizado con él. Eso, sumado al hecho de que se utilice Java (también conocido y usado en el grado) ha provocado esta decisión en mayor medida.

Otro motivo ha sido la cuota de mercado de Android. Como se ha visto en la Figura 2.7, este sistema ocupa tres cuartas partes del mercado de los dispositivos móviles del mundo. Si nos centramos en el panorama nacional, esta predilección de los usuarios por este sistema es aún mayor. Según un informe del Centro Criptológico Nacional (Figura 2.15), en España este sistema alcanza más del 90% del mercado. Esta gran popularidad ha sido importante en la elección de este sistema para nuestro proyecto, ya que tanto a la hora de usarlo como de probarlo, la disponibilidad de un terminal con este S.O. instalado es mucho mayor que uno con iOS.

Figura 2.15. Cuota de mercado de S.O. móviles, España. (14)

Por tanto, se utilizará el entorno de desarrollo Android Studio (15). La aplicación y sus características se implementarán de forma íntegra en este IDE, que permite tanto la programación (en Java o Kotlin, se usará Java) como el uso de bases de datos SQLite.

Para el uso de Android Studio se utilizará un PC con Windows 10, sistema compatible con este entorno de desarrollo, y un teléfono con Android 9 para realizar las pruebas y testear el uso de la app.

También se ha utilizado la página web *draw.io* para realizar los esquemas que se incorporen a esta memoria.

Capítulo 3. Fundamentos teóricos

En este capítulo se tratarán los fundamentos teóricos sobre los cuales se realizará el proyecto. En primer lugar se explica la medida de la orientación de un cuerpo, después se desarrolla la representación de dicha orientación, y finalmente se estudia el protocolo de comunicación mediante el cual se envían dichos datos al dispositivo móvil en el cual se implementa la aplicación.

3.1 Medida de la orientación

A la hora de determinar la orientación de un cuerpo es necesario el uso de unos sensores para obtener ciertos datos que, al combinarlos, orienten este cuerpo en un espacio.

Para este proyecto, se destacan dos sistemas:

- Unidad de Medición Inercial (IMU)
- Sistemas de Referencia de Actitud y Rumbo (AHRS)

3.1.1 IMU

Una unidad de medición inercial es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando para ello una combinación de acelerómetros y giroscopios.

Normalmente son utilizados en dispositivos como móviles, para los sistemas de navegación donde se requiera este tipo de mediciones y no haya posibilidad de uso de referencias externas. También se utilizan en sistemas de aviones y naves espaciales para sus maniobras.

Su uso se basa en la detección de la aceleración que mide el sensor, y a su vez los cambios rotacionales detectados por el giróscopo, como el cabeceo (pitch), alabeo (roll) y guiñada (yaw).

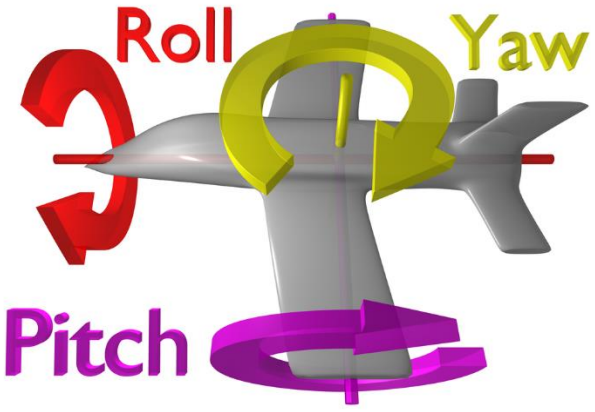

Figura 3.1. Cambios rotacionales en un avión.

El problema de este tipo de sistemas es que suelen estar afectados por un error acumulativo. Esto es, como el sistema va agregando los cambios detectados a las posiciones que se han detectado previamente, cualquier error en la medición se va añadiendo y acumulando. Por tanto, con el paso del tiempo y acumulando estos pequeños errores, se encuentra una diferencia cada vez mayor entre la posición real y la que el sistema determina.

3.1.2 AHRS

Los Sistemas de Referencia de Actitud y Rumbo son otro tipo de sistemas capaces de proporcionar los mismos datos de orientación de la figura 3.1, pero con un cambio respecto de las IMU. En este caso, al uso de acelerómetro y giróscopo se le suma la utilización del magnetómetro. Gracias a este sensor se puede evitar el error acumulativo de las IMU.

El posible problema que puede surgir de usar tantos sensores (3 sensores de 3 ejes cada uno, mínimo) es que se obtienen diversas fuentes y por tanto hay que elegir. Para solventar esto es normal que se utilice un Filtro de Kalman.

3.1.3 Filtro de Kalman

El Filtro de Kalman (16) es un algoritmo desarrollado por R. E. Kalman en 1960 cuya utilidad es la de estimar variables de estado no observables partiendo de variables previamente medidas. Para ello, requiere dos tipos de ecuaciones: las que relacionan las variables de estado con las observables (principales) y las que determinan la estructura temporal del estado (de estado).

Las estimaciones del estado se realizan de acuerdo con la dinámica de ellas mismas (dimensión temporal) y con la de las mediciones que se van obteniendo (dimensión transversal). Por tanto, la dinámica se resume en:

1. Estimar las variables de estado utilizando su misma dinámica (predicción).
2. Mejorar esa estimación utilizando la información obtenida en las variables observables (corrección).

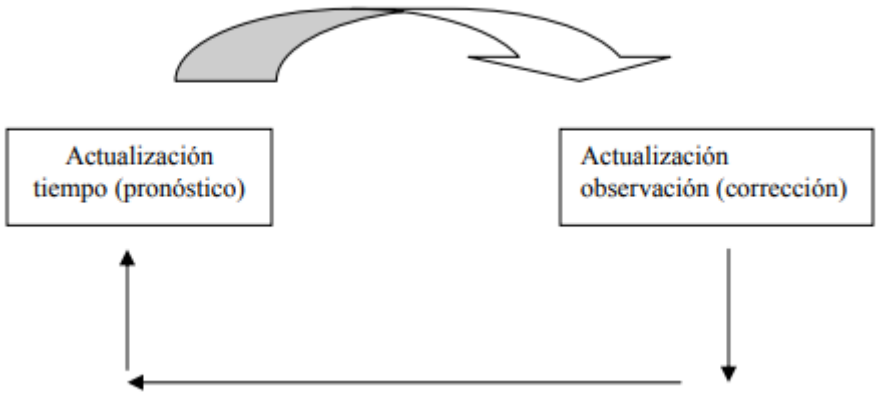

Figura 3.2. Ciclo del filtro de Kalman.

Un aspecto interesante de este Filtro es su característica de recursividad. La estimación de las variables de estado no requiere que se almacenen los datos anteriores para volverlos a procesar con la llegada de nuevas muestras.

Tiene numerosas aplicaciones, como la guía y navegación de vehículos, especialmente espaciales, además de campos como la econometría y procesado de señal.

3.1.4 Algoritmo de Madgwick

Este algoritmo (17) trata de un paso más a la idea del Filtro de Kalman para su aplicación en IMU y AHRS. Toma su nombre de su autor, Sebastian O.H. Madgwick (investigador de la Universidad de Bristol) y utiliza el cuaternión como método para representar la orientación. Esto soluciona problemas en la representación que veremos en el apartado siguiente.

Este método nos ofrece buenos resultados gracias a ese uso del cuaternión que hemos comentado y está optimizado para su implementación en microcontroladores de pequeñas prestaciones, además de estar ideado para su uso con AHRS con 9 grados de medición, como es nuestro caso.

Los beneficios que obtenemos de usar este algoritmo son:

1. Buena efectividad en aplicaciones con frecuencia de muestreo relativamente baja, como es este caso, donde la frecuencia estará entre los 100 y los 1000 Hz.
2. Bajo coste computacional. Requiere solamente 109 operaciones aritméticas en su variante IMU y 277 en su variante AHRS, que utilizaremos.
3. Contiene parámetros que se pueden ajustar para obtener mejores resultados en función del sistema en el que se usará. Concretamente, se puede modificar un parámetro en IMU (periodo de muestreo) y dos en AHRS (el periodo de muestro y la beta).
4. Es un algoritmo muy usado en este campo y se puede encontrar fácilmente en bibliotecas fiables de Internet, ya que está implementado en muchos sistemas de programación y es de código abierto.

Se utilizará este algoritmo en su variante AHRS, con el mismo periodo de muestreo que utilicemos en los sensores. La beta quedará como parámetro ajustable para obtener el resultado óptimo que buscamos. Este parámetro se interpreta como la ganancia del algoritmo, es decir, el tiempo que necesitará para corregir errores. A mayor β, menos errores pero más tiempo necesita para converger.

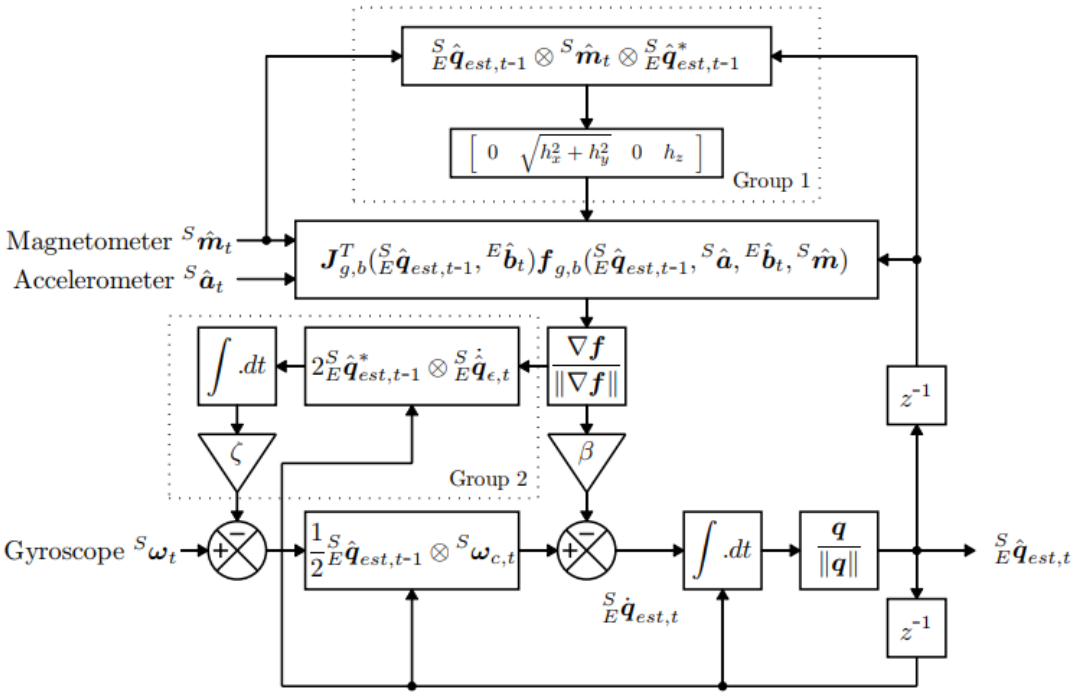

Figura 3.3. Diagrama de bloques del algoritmo de Madgwick, variante AHRS.

3.2 Representación de la orientación

Una vez tomados los datos de la orientación del dispositivo, la idea es la de realizar una representación a partir de la cual podamos realizar un análisis de la técnica de golpeo de tenis, como puede ser detectar el golpe que se ha realizado.

Para realizar esta representación, hay que tener en cuenta que no todos los métodos son óptimos. Se van a explicar dos: representación por ángulos de Euler y por cuaterniones.

3.2.1 Ángulos de Euler y ángulos de navegación

Este tipo de ángulos (18) constituyen un conjunto de tres coordenadas angulares cuya utilidad es la de especificar la orientación de un sistema móvil de ejes ortogonales respecto a otro sistema de ejes ortogonales fijos. Fueron introducidos por Leonhard Euler precisamente para describir la orientación de un sistema de referencia.

Teniendo dos sistemas de coordenadas (xyz, el móvil y XYZ, la referencia), se puede determinar la posición de uno en términos de otro utilizando tres ángulos (por convención se utilizan los símbolos α, β, γ). Se basa en escoger dos planos, uno en el sistema de referencia (xy) y otro en el triedro rotado (XY). La intersección entre estos dos planos se llama línea de modos, y es lo que se utiliza para definir los tres ángulos, que se pueden observar en la figura 3.4:

1. α: Ángulo entre eje x y la línea de nodos.
2. β: Ángulo entre el eje z y el Z.
3. γ: Ángulo entre la línea de nodos y el eje X.

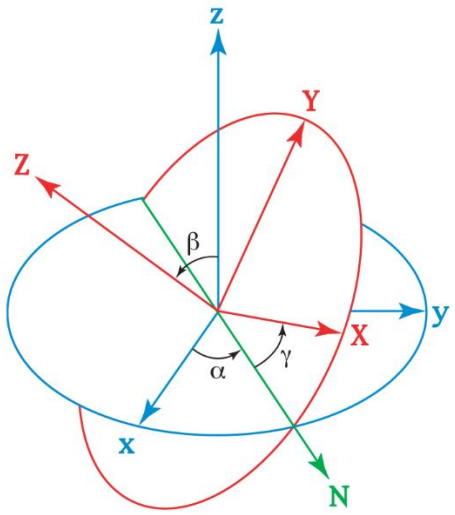

Figura 3.4. Representación de los ángulos de Euler.

En otras ocasiones se utiliza como línea de nodos dos planos no homólogos (en este caso, xy e YZ, o xy y XZ. Cuando se utiliza este último método, hablamos de ángulos de navegación o Tait-Bryan.

Son los que se usan en navegación espacial y ya han sido nombrados anteriormente en el apartado de Medida de la Orientación. Se trata de los ángulos φ (roll o alabeo), θ (pitch o cabeceo) y ψ (yaw o guiñada), y su uso en un avión puede verse en la Figura 3.1.

En este caso, como ya se ha mencionado, la línea de nodos es la intersección entre los planos coordenados xy e YZ (se puede ver en la figura 3.4), y se definen los ángulos:

1. φ: Ángulo entre la línea de nodos y eje Y.
2. θ: Ángulo entre el plano xy y eje X.
3. ψ: Ángulo entre eje y y línea de nodos.

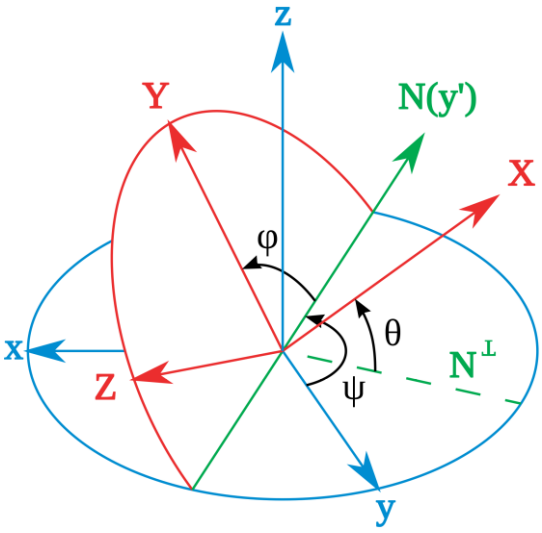

Figura 3.5. Representación de los ángulos de navegación.

No obstante, este sistema de representación tiene un problema, y se trata del Gimbal Lock.

3.2.2 Gimbal Lock

El Gimbal Lock (o bloqueo del cardán en castellano) es un problema que aparece en la representación de rotaciones en un sistema cuando se utilizan ángulos de Euler. Consiste en la pérdida de un grado de libertad cuando dos de los ejes se colocan en paralelo.[3]

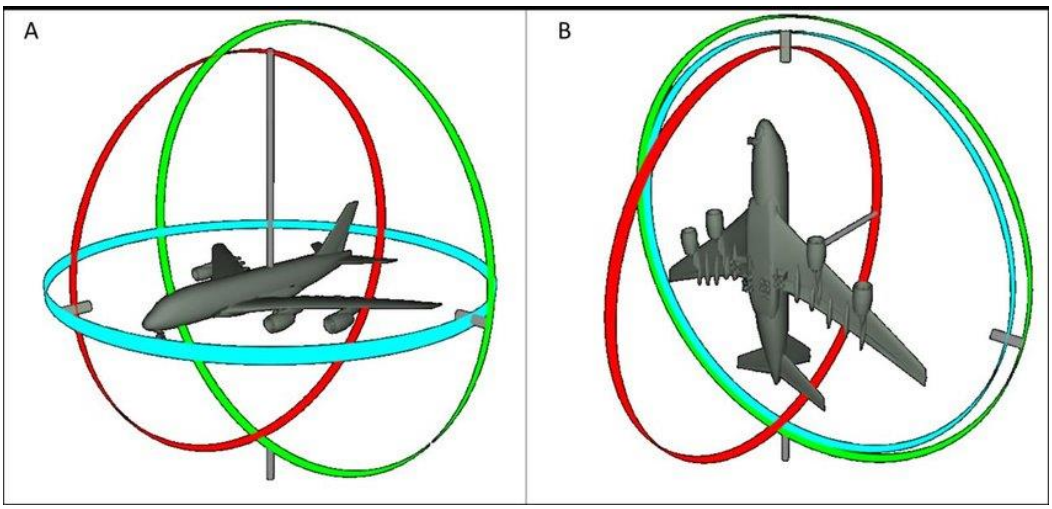

Figura 3.6. Representación del gimbal lock. En la situación A se sitúa perfectamente la orientación del avión, en la situación B tenemos dos ejes en paralelo y se da gimbal lock.

[3] Vídeo donde se observa el problema del *gimbal lock*: https://www.youtube.com/watch?v=N5PDboNJwks

En esta situación, uno de los dos ejes queda dependiente del otro y no puede realizar sus propios movimientos. Para solucionarlo, lo mejor es introducir un método que trate la orientación como un propio valor e incluya un cuarto eje o cardán para mantener el ángulo entre los ejes que se utilizan, impidiendo que se alineen.

Para esto, se introduce el concepto de cuaternión.

3.2.3 Cuaterniones

Los cuaterniones son una extensión de los números reales con cuatro dimensiones. Su utilidad en este campo es la de proporcionar una notación matemática para representar las orientaciones y rotaciones de objetos en un espacio tridimensional. Son más sencillos de componer que los Ángulos de Euler y evitan el problema del Gimbal Lock, ya que tienen cuatro ejes.

Normalmente, se representan de la siguiente manera:

$$q = a + bi + cj + dk \tag{3.1}$$

donde a, b, c, d son números reales y i, j, k, son la parte imaginaria del cuaternión.

Los cuaterniones son lo que se llama un cuerpo asimétrico, esto es, parecido a un cuerpo pero no conmutativo en la multiplicación, sin embargo sí que satisfacen el resto de propiedades de un cuerpo.

Son muy útiles en aplicaciones de gráficos por computadora y robótica, así como en navegación.

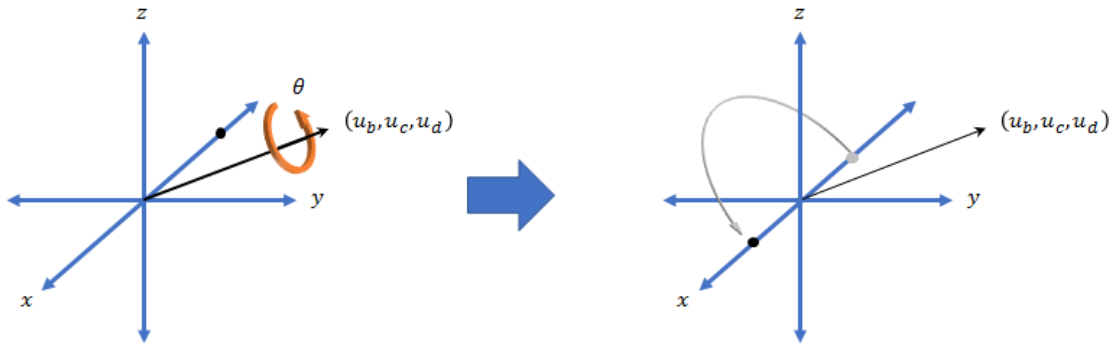

Figura 3.7. Representación de un cuaternión.

El algoritmo de Madgwick que vamos a utilizar en nuestro proyecto implementa el uso de cuaterniones, por lo tanto evitamos el Gimbal Lock y usamos este sistema. En caso de observar que la representación con cuaterniones es demasiado compleja, hay una forma simple de obtener los Ángulos de Euler a partir de cuaterniones, según obtuvo Madgwick en su informe (17).

$$\psi = Atan2(2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_2^2 - 1) \tag{3.2}$$

$$\theta = -sin^{-1}(2q_2q_4 + 2q_1q_3) \tag{3.3}$$

$$\phi = Atan2(2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1) \tag{3.4}$$

3.3 Bluetooth Low Energy (BLE)

En este apartado se explicará el protocolo de comunicación utilizado para la conexión entre el dispositivo *wearable* y el *smartphone* en el que se implementará la aplicación.

Trata de una tecnología de Red de Área Personal (PAN) e inalámbrica, dirigida especialmente a aplicaciones orientadas a la salud, ejercicio físico, seguridad y domótica. Sus orígenes provienen de Nokia en 2006, bajo el nombre *Wibree*[4] e ideado como una forma de rivalizar con Bluetooth. Finalmente, fue integrado en la especificación de Bluetooth 4.0 en 2009, bajo el nombre de Bluetooth Low Energy.

Su principal característica, como su nombre indica, es la de funcionar con un consumo de energía muy reducido, manteniendo un rango de comunicaciones similar. Utiliza la misma banda de frecuencias que el Bluetooth clásico, la de 2.4 GHz. En cuanto a seguridad, soporta el sistema de cifrado AES. La mayoría de sistemas operativos móviles, así como Windows desde Windows 8, MacOS y Linux ya soportan BLE de forma nativa.

Estas características de bajo consumo y soporte nativo en los principales sistemas, así como su bajo coste de implementación, son las que nos han hecho decidirnos a utilizar un dispositivo conectado por BLE para nuestro proyecto.

Entrando más en detalle en su funcionamiento, se observa que esta comunicación se basa en el uso de GATT (19).

3.3.1 GATT: General Attribute Profile

GATT define el modo en que los dispositivos BLE transfieren datos, utilizando los conceptos de Servicios y Características. La conexión entre dos dispositivos se realiza mediante una estructura de Master - Slave, o Cliente – Servidor. El Servidor (en este caso el smartphone) podrá conectarse a varios Clientes o periféricos.

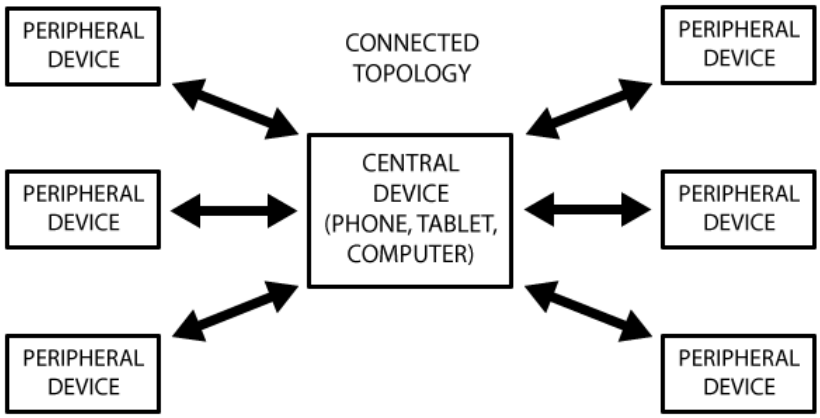

Figura 3.8. Topología de red de conexión BLE. (20)

Se hace uso de un protocolo de datos llamado ATT (Attribute Protocol), en el cual se almacenan los Servicios, Características y datos relacionados en una tabla de consulta (*Lookup Table)* simple. Cada atributo se identifica mediante un Identificador Único Universal (UUID), que es estándar y tiene un tamaño de 128 bits.

Una vez que se ha realizado la conexión dedicada, comienzan las transacciones GATT.

[4] *Is Wibree going to rival Bluetooth?*, diciembre 2006. https://electronics.howstuffworks.com/wibree.htm

El Máster BLE actuará ahora de Cliente GATT. Esto es, actúa como Máster de la comunicación y decide cuándo realiza las peticiones GATT al Servidor situado en el Esclavo de la comunicación BLE.

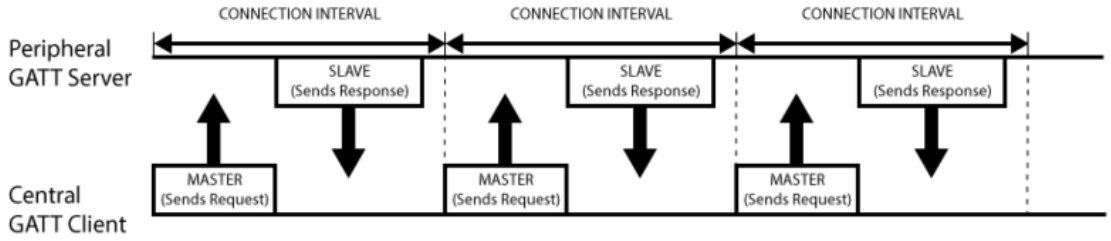

Figura 3.9. Transacción GATT

Los Servicios incluidos en un dispositivo BLE periférico (como los datos de los sensores del dispositivo WeSU) van incluidos en Perfiles. Un Perfil incluye varios Servicios, que a su vez incluyen las Características, como se puede ver en la figura 3.9:

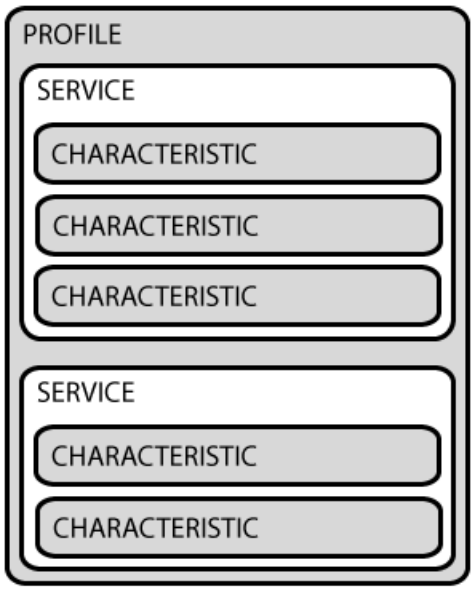

Figura 3.10. Estructura de perfiles GATT.

Las Características son el concepto de nivel más bajo, donde se encapsulan los datos finales, como datos de sensores. Para identificar una característica, se utilizan Descriptores, que describen el valor de dicha característica.

Adaptándolo a nuestro proyecto, los datos de los sensores de acelerómetro, giróscopo y magnetómetro se encuentran en características, por lo tanto tendremos que acceder al perfil adecuado, buscar el servicio que necesitamos, y encontrar el descriptor que acompaña a la característica de cada sensor. Todo esto, utilizando las UUID, los identificadores de cada objeto. Finalmente, estas características se pasan mediante la conexión BLE utilizando Atributos GATT.

3.3.2 Identificadores UUID

UUID (*Universally Unique Identifier)* (21) es un número de 16 bytes utilizado como identificador. El número de posibles UUID es de 16^{32}, por lo que son buenos para dicha utilidad. A la hora de representarlos, lo habitual es hacerlo con 32 dígitos hexadecimales separados en cinco grupos de la forma: 8-4-4-4-12.

Este tipo de identificadores se utiliza en multitud de aplicaciones, desde nuestro caso para identificar los distintos servicios y características Bluetooth como en otro tipo de usos tributarios, de bases de datos, etc.

En nuestro proyecto, para acceder a los servicios vamos a necesitar los identificadores de dos servicios diferentes: el servicio de configuración de los sensores (que contiene la característica de acceso a los registros y para configurar el parámetro de frecuencia de muestreo que queremos) y el servicio de lectura de los sensores (que incluye la característica donde se colocan los datos de lectura de los sensores).

Los UUID de dichos servicios se han extraído del manual de usuario del dispositivo WeSU (8) y son los siguientes:

- CONFIG SERVICE: 00000000-000F-11E1-9AB4-0002A5D5C51B
- REGISTER ACCESS: 00000001-000F-11E1-AC36-0002A5D5C51B
- SERVICE SENSORS DATA: 00000001-0001-11E1-9AB4-0002A5D5C51B
- CHARACTERISTIC SENSORS DATA: 00E0-0001-11E1-AC36-0002A5D5C51B
- DESCRIPTOR: 00002902-0000-1000-8000-00805F9B34FB

Capítulo 4. Desarrollo de la aplicación

En este capítulo se describirá el desarrollo de la aplicación en la que implementaremos nuestro proyecto, comenzando con el diagrama de bloques en el que nos hemos basado, siguiendo con el diseño de la aplicación y acabando con la descripción del código.

4.1 Diagrama de bloques del proyecto

Figura 4.1. Esquema general del proyecto.

La figura 4.1. muestra el esquema general del proyecto. El dispositivo wearable WeSU toma los datos de los sensores de acelerómetro, giróscopo y magnetómetro. Se realiza una conexión por BLE entre este dispositivo y el smartphone, mediante la cual se transportarán estos datos para su análisis en la aplicación.

4.2 Diagrama de bloques software

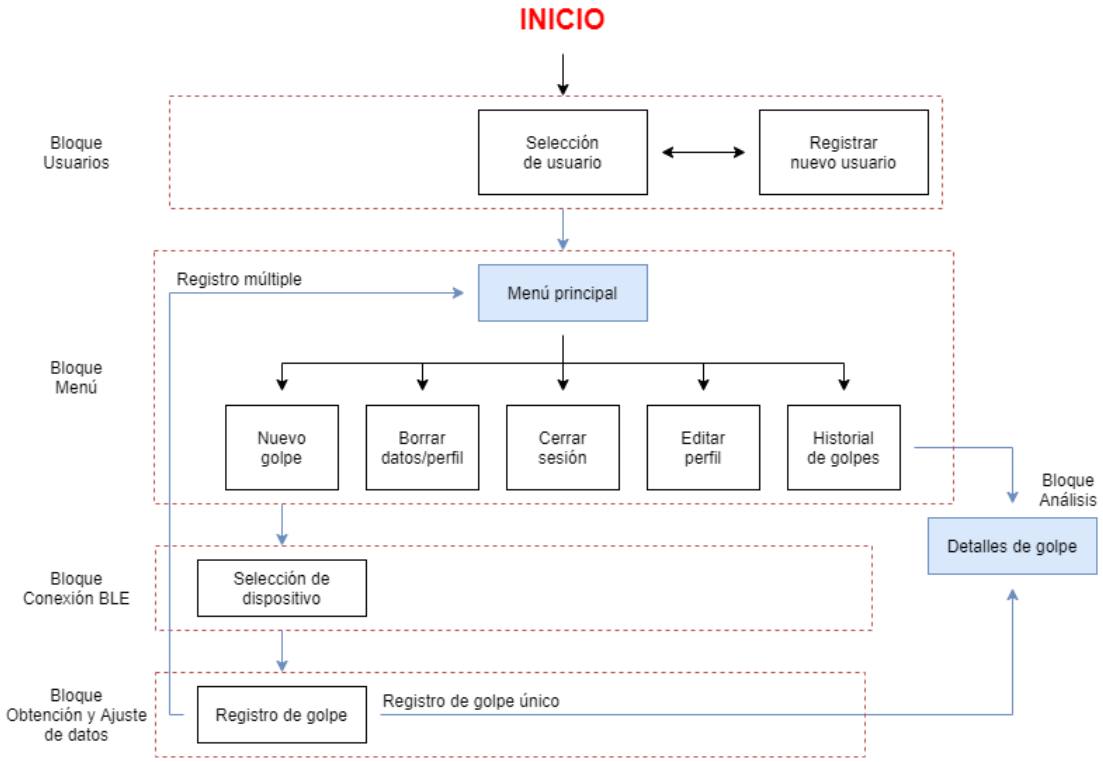

Figura 4.2. Diagrama de bloques del software

Se ha dividido la implementación en cinco bloques diferenciados.

Se inicia la aplicación con el bloque de gestión de usuarios. En este bloque, se da las opciones de iniciar sesión con uno de los usuarios creados o crear uno propio.

Al seleccionar el usuario deseado, se avanza al menú principal, donde se muestran diferentes opciones, tanto propias del usuario (como gestionar los datos o editar el perfil) como de la app (ver el historial del golpes o iniciar el proceso de registrar uno nuevo). Todas estas opciones se llevarán a cabo en este bloque excepto dos: el de registrar un golpe y el de ver los detalles de un golpe, al que se puede acceder desde el historial.

El proceso de registro de un nuevo golpe consta de varios pasos, comenzando por la conexión BLE con el wearable, que se lleva a cabo en el siguiente bloque. Desde aquí ya solamente se puede pasar al siguiente bloque o regresar al anterior si hay algún fallo o se cancela el proceso.

En el bloque de Obtención y Ajuste se procede al registro del golpe, obteniendo los datos de los sensores del dispositivo WeSU, realizando unos pequeños ajustes y conversiones para poder trabajar con ellos más adelante. Cuando acaba este bloque se pasa a los detalles del golpe que se acaba de registrar (en caso de golpe individual) o al menú principal de usuario si es un registro de golpe múltiple.

A continuación, se explicará el funcionamiento y el código de cada uno de los bloques implementados, además de la base de datos creada para el tratamiento de usuarios y golpes.

4.3 Base de datos

La Base de Datos es uno de los componentes del proyecto, y realiza múltiples funciones en la aplicación. Almacena todos los datos que involucran a los usuarios que se creen, así como los golpes que el usuario registre. Para registrar los golpes se toman datos cada 20ms (según se ha elegido la frecuencia de los sensores), por tanto es necesaria otra tabla que almacene todos los datos que forman un golpe. Esta última es la tabla Movimientos, que guarda los datos ya mencionados (datos de sensores del wearable, cada uno con 3 dimensiones), además del identificador Mov y un *Timestamp* para seguir el orden de los datos almacenados y poder representarlo en una gráfica.

La BD que se ha creado es del tipo SQLite, con lo cual es una base de datos local, relacional, organizada en tablas con filas (registros) y columnas (campos de cada registro). Uno de estos campos se debe elegir como campo clave. El término relacional implica que se pueden establecer relaciones entre las tablas, concretamente entre un campo de cada tabla. Estas relaciones permiten una mejor organización de los datos almacenados físicamente sin perder información. Pueden ser de tres tipos diferentes: de uno a varios (1-n), de uno a uno (1-1) y de varios a varios (n-n), pero en este proyecto solamente se necesitará el primer tipo. De esta manera, a cada registro de la primera tabla le pueden corresponder varios casos de la segunda.

Se ha utilizado este tipo de base de datos por estar integrado en el software Android Studio, teniendo a nuestra disposición las API que necesitaremos usar en el paquete *android.database.sqlite*[5] y creando la clase *DatabaseSQLHelper.java*. En esta clase se define la base de datos, sus tablas y los métodos que se deben utilizar para crear nuevos registros, actualizar los existentes o acceder a la información almacenada.

Esos métodos implementan consultas SQL bastante simples, debido a que nuestra aplicación no necesita una base de datos compleja y se ha optado por realizar una simple para acelerar las operaciones que se realicen.

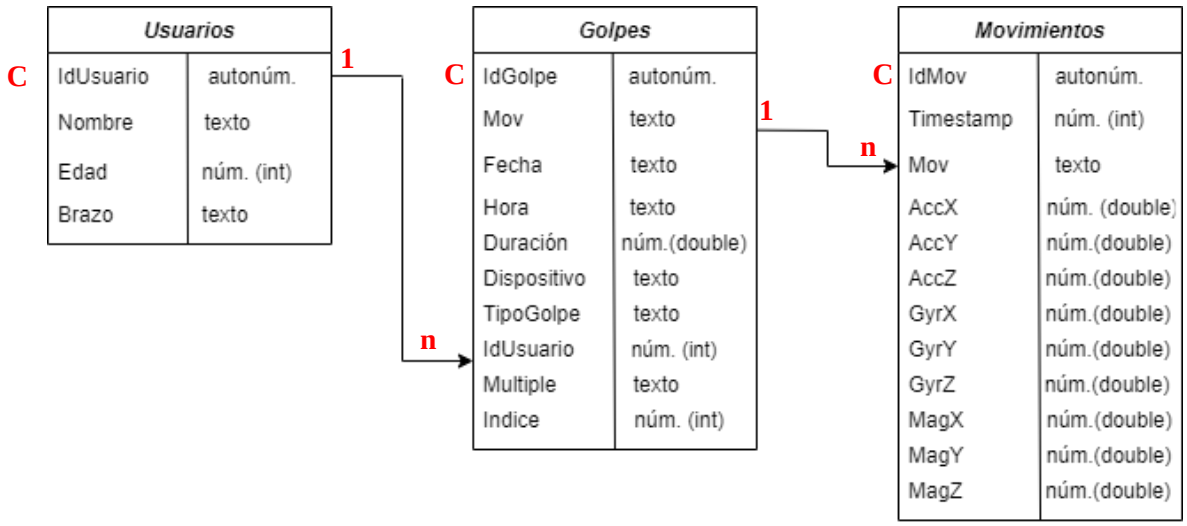

Figura 4.3. Estructura de la base de datos

En la figura 4.3 se observa la estructura de la base de datos que se ha implementado.

[5] Documentación SQLite: https://developer.android.com/training/data-storage/sqlite?hl=es-419

Se trata de tres tablas, relacionadas entre sí por identificadores, excepto en el caso de Golpes y Movimientos, que se relacionan por una cadena de texto, que hemos llamado Mov. Este identificador consiste en la fecha del día donde se registra el golpe, en el formato YYmmDDHHmmss, es decir: año-mes-día, hora-minuto-segundo. Inicialmente no se usaba los segundos, pero eso llevaba a problema cuando se registraban dos golpes dentro del mismo minuto, lo que hacía que los dos golpes se mezclaran y se obtuvieran datos extraños.

La tabla de Usuarios incluye los datos que se introducen al registrarse, es decir, el nombre del usuario, su edad y su brazo dominante (que se tiene en cuenta al detectar el tipo de golpe que se realiza, no salen los mismos datos de orientación con un *drive* con la mano derecha que con la izquierda). También se incluye un identificador que actúa como campo clave de la tabla. El campo clave de una tabla es el que identifica únicamente un registro y no se repetirá, además de cumplir con que tenga el menor tamaño posible.

La tabla de Golpes contiene el identificador Mov con el cual podremos acceder a sus movimientos asociados, así como datos generales de un golpe, como la duración, el usuario que lo ha registrado, el tipo de golpe que se ha detectado y el dispositivo utilizado. Cabe destacar los dos últimos campos, 'Múltiple' e 'Índice', que se utilizarán cuando se realice el registro múltiple de golpes.

Aparte, se han creado varias clases más para simplificar esta clase *DatabaseSQLHelper.java*, como son las clases *Definitions.java*, *Usuario.java*, *Movimiento.java* y *Golpe.java*. Con la primera definimos los nombres de las columnas en variables para poder realizar cambios sin tener que tocar el código SQL, que puede quedar inutilizado por un espacio o una coma mal situada y es más complicado de localizar, al contrario del código Java que sí que avisa al compilarlo (se puede ver un ejemplo en la figura 4.4). Las clases creadas para cada una de las tablas de la base de datos son los lugares donde se implementan los métodos utilizados para acceder a la información almacenada en cada tabla, dejando así la clase principal de la base de datos para los métodos que realizan operaciones de escritura (creación de registros y actualización de campos).

```
4
5   public class Definitions {
6
7       public static abstract class MovimientoEntry implements BaseColumns {
8
9           // Nombre
10          public static final String MOVS_TABLE = "MOVEMENTS";
11
12          // Columnas
13          public static final String MOV = "mov";
14          public static final String TIMESTAMP = "timestamp";
15          public static final String ACC_X= "ACC_X";
16          public static final String ACC_Y= "ACC_Y";
17          public static final String ACC_Z= "ACC_Z";
18          public static final String GYR_X= "GYR_X";
19          public static final String GYR_Y= "GYR_Y";
20          public static final String GYR_Z= "GYR_Z";
21          public static final String MAG_X= "MAG_X";
22          public static final String MAG_Y= "MAG_Y";
23          public static final String MAG_Z= "MAG_Z";
24      }
```

Figura 4.4. Ejemplo de la clase *Definitions*.java

4.4 Desarrollo software

En este apartado se procederá a explicar cada uno de los módulos de los que se compone la aplicación y se han visto en la Figura 4.2.

Es importante mencionar los distintos tipos de clases que se han utilizado. Concretamente son dos: Actividades y Fragments. La mayor diferencia es que los Fragments se utilizan dentro de Actividades para ampliar sus funcionalidades sin tener que crear una nueva Actividad, lo que implicaría salir de la actual, es decir, un Fragment es una clase que se incrusta en una Actividad para completarla.

Asociados a estas clases, tenemos los layouts, que son los contenedores donde se diseña el aspecto de la aplicación. Gestionan los elementos que ponemos, que pueden ser desde bloques de texto hasta imágenes, y establecen el orden y posición entre ellos. Hay varios tipos de layouts, desde los que vienen ya creados y solamente hay que agregar la funcionalidad a cada elemento, hasta los que vienen vacíos completamente. La ventaja de este tipo de objetos es que existen varias formas de editarlos, tanto desde la clase Java mediante código como en el propio objeto, dado que el entorno de desarrollo nos aporta una herramienta gráfica para hacerlo de forma sencilla.

4.4.1 Bloque Usuarios

Figura 4.5. Estructura del módulo de usuarios.

Es el primer bloque al que se accede al iniciar la aplicación. Consiste en dos partes, una donde vemos los usuarios existentes en la base de datos y seleccionamos el que queremos para iniciar sesión, y otra donde registramos un usuario que se añadirá a la base de datos.

La primera parte, de selección de usuario, se basa en una pantalla donde se van colocando en una lista de botones los usuarios registrados en la base de datos. Al pulsar el deseado, se inicia la actividad de menú. La otra posibilidad es la de pulsar el botón de Registro.

Las clases que se implementan son:

- *UsersActivity.java:* Contiene el Fragment *UsersFragment.java* y tiene asociado el *layout activity_users.xml*. Su utilidad es la de ser el contenedor en el que se irán colocando los usuarios que se creen. Al pulsar uno de estos usuarios, se inicia sesión con él. También contiene el botón de Registrar Usuario, que nos lleva a la Actividad *CreateNewUser*.
- *UsersFragment.java*: Tiene dos funcionalidades: por un lado accede a la base de datos y obtiene los datos de los usuarios existentes, y por otra parte crea las pequeñas tarjetas en las que se muestra cada usuario y que funcionan como botones. El layout que utiliza para ello es *fragment_users.xml*.

El método que se ejecuta al pulsar en un usuario es el siguiente:

```
109     private void showMainActivity(String userId,String userName) {
110         Intent intent = new Intent(getActivity(), MainActivity.class);
111         intent.putExtra(MainActivity.EXTRA_USER_ID, userId);
112         intent.putExtra(MainActivity.EXTRA_USER_NAME, userName);
113         startActivityForResult(intent, REQUEST_USER_MAIN);
114     }
```

Figura 4.6. Método que inicializa la actividad de menú al pulsar un usuario

Observamos la utilización del método *putExtra*[6], que se utiliza en Android para el paso de parámetros entre actividades. En este caso, cogemos los datos de identificador y nombre del usuario elegido y los pasamos a la siguiente actividad para iniciar la sesión.

En cambio, si pulsamos el botón de registro no pasamos ningún parámetro:

```
101     private void showAddScreen() {
102         Intent intent = new Intent(getActivity(), CreateNewUser.class);
103         startActivityForResult(intent, CreateNewUser.REQUEST_ADD_USER);
104     }
```

Figura 4.7. Método que lanza la actividad de registro

La segunda parte implementada en este bloque es la de registrar un usuario. Trata de una pantalla de registro típica, con campos en los cual se introduce la información, que se escribe en la base de datos al pulsar Aceptar. El campo de Nombre es de texto, el de edad es de número (int) y en cuanto al brazo dominante, se utiliza un *Spinner*[7] en el que se dan dos opciones (derecho e izquierdo). Se ha hecho así para evitar la introducción de texto no deseado y facilitar el uso más adelante, ya que es un campo que se utiliza.

Las clases que se han implementado son:

- *CreateNewUser.java:* Actúa de manera similar a la actividad anterior. Funciona de contenedor del fragment asociado, en el que se implementan las acciones de registrar un nuevo usuario. Su layout *activity_new_user.xml* contiene el botón de registrar, con el cual se ejecuta el proceso.
- *CreateNewUserFragment.java:* Define los campos en los que el usuario pondrá sus datos, y a continuación, al pulsar el botón de registrar en la actividad *CreateNewUser*, accede a la base de datos para crear el registro correspondiente. También se reutiliza para el caso de editar perfil, que se explicará más adelante.

Cuando se pulsa el botón de aceptar, se llama a un método implementado en la base de datos para crear un usuario:

```
69  @   public long saveUser(Usuario user) {
70          SQLiteDatabase sqLiteDatabase = getWritableDatabase();
71
72          return sqLiteDatabase.insert(
73                  Definitions.UsuarioEntry.USER_TABLE,
74                  nullColumnHack: null,
75                  user.toContentValues());
76      }
```

Figura 4.8. Método *saveUser*

Este método crea un objeto de escritura en la base de datos y le inserta el usuario con los datos introducidos. Esto se realiza mediante el método *toContentValues*, que es donde se encuentra la

[6] Iniciar otra actividad: https://developer.android.com/training/basics/firstapp/starting-activity?hl=es-419
[7] Spinner: https://developer.android.com/guide/topics/ui/controls/spinner

información del nombre de columna y tipo de atributo que utiliza. Si se intenta introducir un tipo que no concuerde, este método da error y se puede identificar mejor.

```
31      public ContentValues toContentValues() {
32          ContentValues values = new ContentValues();
33          values.put(Definitions.UsuarioEntry.FIELD_NAME, name);
34          values.put(Definitions.UsuarioEntry.FIELD_AGE, age);
35          values.put(Definitions.UsuarioEntry.FIELD_BRAZO, brazo);
36          return values;
37      }
```

Figura 4.9. Método *toContentValues*

4.4.2 Bloque Menú

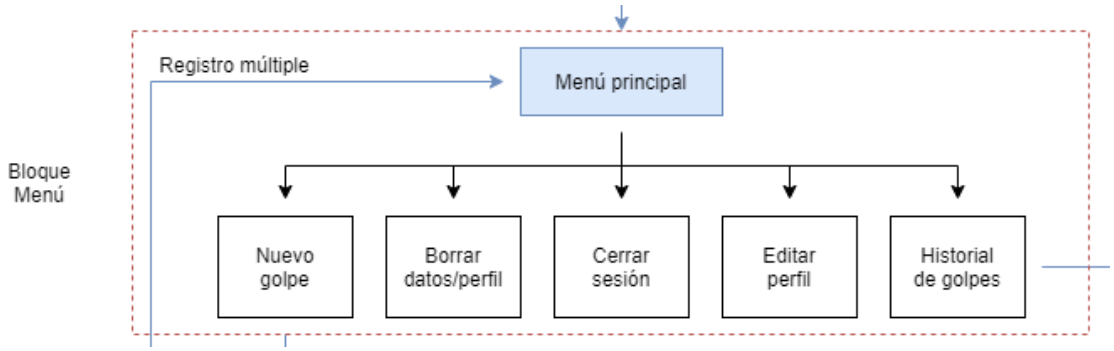

Figura 4.10. Estructura del módulo de menú principal de la app

Este es el módulo en el que nos encontraremos al iniciar sesión con el usuario deseado. Consiste básicamente en un menú en el cual tendremos distintas opciones entre las que elegir. La opción principal que se mostrará es la del botón de registrar un nuevo golpe. La otra opción que se podrá observar es el historial de golpes registrados. Para cambiar entre estos dos, se ha incluido un doble botón, en el cual se activa o una parte del botón u otra. Como ambas opciones se implementan en la actividad principal, que actúa como contenedor, al pulsar un botón se activa la visibilidad de los elementos correspondientes y se desactiva la otra opción.

Cada una de esas dos partes se implementa en su Fragment correspondiente, en ambos casos incluidos en la Actividad *MainActivity.java*.

- *MainActivity.java:* Esta actividad, como se ha comentado, funciona como contenedor de los Fragments, pero también se utiliza para implementar el resto de opciones de este Bloque, y que son más simples, ya que se basan en cambios en el perfil registrado en la base de datos.

  · Cerrar sesión: Simplemente finaliza esta actividad, con lo que se vuelve a la actividad que la había iniciado.

  · Borrar perfil/datos de golpes registrados: Es algo más complejo, ya que se crea un diálogo para evitar la pulsación involuntaria. Una vez se asegura la voluntariedad, se llama al método *delete*, de esta misma actividad, cuyos parámetros de entrada son el ID de usuario y un booleano que indica si se quiere borrar el usuario o los datos. Este método accede a otro método de la base de datos que borra el usuario (*deleteUser)* o los golpes registrados (*deleteMovement).*

· Editar perfil: En este caso, se simplifica bastante ya que se accede a una clase ya creada anteriormente, la de creación de perfil. Esta actividad se ha modificado teniendo en cuenta este uso, y por tanto, al enviarle como parámetro un ID de perfil, el código SQL a ejecutar no trata de crear un nuevo registro sino de editar uno ya existente. Este código no se implementa en el método *saveUser* como en el caso anterior (ver apartado 4.4.1), sino en el método *updateUser*, cuya diferencia es que ahora hay que buscar el usuario a modificar (operación de lectura) y realizar los cambios (escritura).

Los Fragments que se incluyen en esta actividad son:

- *RegisterActivityFragment.java:* implementa el botón de registrar una nueva actividad y el interruptor con el cual se seleccionará el modo de registro que se desee, individual o múltiple. El funcionamiento del botón es simple, al pulsarlo se lanza la actividad del siguiente bloque, el de conexión BLE.
  Para el funcionamiento del interruptor se ha utilizado el elemento *Switch* del layout, con lo que tenemos un tipo de botón que estará a ON o OFF, de manera similar al botón doble incluido en *MainActivity.* Se ha tomado como valor por defecto OFF, con lo que si no se pulsa se hará un registro de un solo golpe, y si se pulsa se realiza un registro múltiple.
  Al pulsar el botón, además del lanzamiento del siguiente módulo, se ha implementado el almacenamiento del resultado de este interruptor. Para ello, se ha utilizado *SharedPreferences.* Mediante esta clase proporcionada por Android[8], se puede guardar pequeños datos, como preferencias, en la propia aplicación, lo que lo hace ideal para esta tarea, ya que se puede acceder desde cualquier lugar de la propia app. Se almacena en forma de clave-valor.

- *RecordsFragment.java:* funciona de manera similar a la lista de selección de usuarios (apartado 4.4.1). Se accede a la base de datos para obtener la información almacenada en cada registro de golpe asociado al usuario correspondiente. Con esa información se crean las diferentes entradas y se incluyen en botones, que al pulsarlos nos llevan a ver detalles más específicos de los golpes, que ya sería el módulo de Análisis, que veremos más adelante.
  Básicamente, el método *loadActivities* accede a la BD y toma los datos deseados. Finalmente, los carga en los botones mediante el uso de un *Adapter*. Concretamente, los datos usados son la fecha del golpe, la duración, el tipo de golpe que se ha detectado y si es parte de una sesión de registro individual o múltiple.

4.4.3 Bloque Conexión BLE

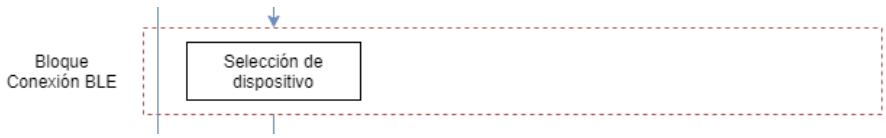

Figura 4.11. Estructura del módulo de conexión BLE

Este bloque es donde se lleva a cabo todo el proceso de conexión con el dispositivo wearable. Consiste en varios pasos, pero de cara al usuario solamente se mostrará uno de ellos, el de seleccionar el dispositivo al cual se quiere conectar.

[8] Clase Shared Preferences: https://developer.android.com/reference/android/content/SharedPreferences

Todo esto se realiza mediante los métodos de la Actividad *ActivityScan.java*, solamente con la inclusión de un Fragment, *DevicesListFragment.java*, en el cual se implementa la lista de dispositivos encontrados y a los que se pueda conectar. Adicionalmente, se ha tenido que realizar cambios en el manifiesto de la aplicación, *AndroidManifest.xml*, ya que es necesario pedirle los permisos al usuario para poder utilizar el Bluetooth y la localización del smartphone, sin los cuales la búsqueda BLE no funciona.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.project.MiTenisApp"
    android:installLocation="auto">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Figura 4.12. Petición de permisos en el Manifiesto

Además de definir estos permisos, han de ser pedidos al usuario de manera explícita en la propia actividad. Por tanto, el inicio de la actividad es la comprobación de que estos servicios están activados y la petición del permiso para poder utilizarlos.

Lo primero a realizar, y siguiendo las directrices de la documentación de Android Developers[9], será definir un *BluetoothAdapter* (representa el propio adaptador del dispositivo, la "radio Bluetooth") y un *BluetoothManager* (que controlará el proceso). El *Manager* se obtiene a través del *Adapter*. También necesitamos un Scanner para gestionar la búsqueda de dispositivos.

Todo esto ocurrirá al pulsar el botón "Registrar Golpe" en el menú principal, y sin un layout de fondo.

Después de esto, el proceso a llevar a cabo será el propio de Bluetooth Low Energy. Esto es, búsqueda de dispositivos, elección de un dispositivo y conexión con el dispositivo elegido.

Búsqueda de dispositivos

Este paso se basa en el uso de un método de la clase *leScanCallback*. Es lo que obtendremos al ejecutar el método *startScan*, que ejecutaremos durante 2,5 segundos, tiempo suficiente para mostrar todos los dispositivos cercanos.

Este tipo de clase almacena la información un dispositivo Bluetooth, incluyendo el nombre y la dirección, y es con lo que más adelante realizaremos la conexión. Al iniciarse esta actividad y comprobarse que está todo activado, se inicia el método *startScan*, con el parámetro de entrada *leScanCallback*, que es el método donde realizaremos las comprobaciones correspondientes acerca del dispositivo encontrado: si se ha recibido los datos de forma correcta y si su nombre se adecúa al que buscamos. En ese caso, se añade el dispositivo a la lista.

```
    private ScanCallback leScanCallback = onScanResult(callbackType, result) → {
            BluetoothDevice mBleDevice;
            mBleDevice = result.getDevice();
            if (mBleDevice != null
                    && mBleDevice.getName() != null
                    && !mBleDevices.containsValue(mBleDevice)) {
                //Añadir dispositivo a la lista
                mBleDevices.put(mBleDevice.getAddress(), mBleDevice);
                devicesArray = mBleDevices.keySet().toArray(new String[mBleDevices.keySet().size()]);
            }
    };
```

Figura 4.13. Método *leScanCallback*

[9] Documentación Android de BLE: https://developer.android.com/guide/topics/connectivity/bluetooth-le?hl=es-419

Elección de un dispositivo

Mostramos esa lista por pantalla de la misma forma que se hace en la lista de usuarios, es decir, cada entrada de la lista actúa como un botón. En este caso, el botón hace que se efectúe la conexión entre el *smartphone* y el dispositivo *wearable* elegido.

Para esto, se utiliza un Fragment, *DevicesListFragment*.java, cuyo funcionamiento es semejante al de la lista de usuarios y al de la lista de golpes, con la única diferencia de que en este caso no se realiza una lectura de la base de datos sino que se recibe un array de dispositivos y se debe crear la lista partiendo de esos datos. Esos datos luego se muestran en el layout *fragment_devices*.xml.

Se crea esta lista utilizando objetos del tipo *HashMap*, con lo que podemos añadir tanto la dirección MAC del dispositivo wearable como asociarle el propio dispositivo. Esto hace que sea más sencillo realizar la conexión simplemente pulsando en el dispositivo, ya que ese botón ya tiene el objeto BLEDevice que debe ser parámetro de la conexión.

Conexión

Se realiza mediante el método *connectBLEDevice*. Este método utiliza el *BluetoothGatt* y su método interno *connectGatt*. Recibe como parámetro de entrada el dispositivo al que hemos elegido conectarnos, en forma de objeto *BluetoothDevice*.

Una vez que la conexión ha sido exitosa, muestra un mensaje y nos direcciona hacia el siguiente módulo, donde se obtendrán y tratarán los datos de los sensores correspondientes y se almacenarán en la base de datos para su uso posterior.

4.4.4 Bloque de Obtención y Ajuste de Datos

Figura 4.14. Estructura del módulo de obtención y ajuste de los datos.

En este bloque se llevan a cabo los dos procesos mencionados en el título: la obtención de los datos captados por los sensores del wearable (mediante la conexión Bluetooth Low Energy) y el ajuste de dichos datos para poder ser tratados y analizados en adelante. También se realiza la detección del golpe. Finalmente se almacenan los datos ajustados en la base de datos.

La obtención de los datos incluye los métodos que controlan el acceso a los servicios y características en los que se van almacenando los datos que van tomando los sensores del reloj. El ajuste se realiza posteriormente, ya que estos datos vienen en un formato con el cual no se puede trabajar directamente y hay que adaptar a nuestras necesidades. Esto se divide a su vez en el uso de diferentes variables para incluir cada tipo de datos por separado y la conversión a las unidades de medida adecuadas para el análisis del golpe.

Todo esto se ejecuta en la misma actividad que el bloque anterior, *ActivityScan.java*, pero usando otros Fragments y el layout *fragment_start_capture.xml*, el cual se basa en un botón para iniciar la captura y detenerla cuando hayamos realizado el golpe.

Obtención de los datos

Para este proceso se ha tenido en cuenta el funcionamiento de Bluetooth Low Energy, concretamente GATT, explicado en el apartado 3.3.1 de la memoria. En este caso, el *wearable* trabajará como servidor en la conexión GATT, mientras que el *smartphone* lo hará como cliente.

En esta conexión, habrá que acceder a dos servicios: uno para configurar los sensores a la frecuencia deseada y otro para leer los propios sensores. Este último tiene la característica que almacena los valores de acelerómetro, giróscopo y magnetómetro en cada uno de los tres ejes. Para acceder a estos servicios necesitaremos conocer cada identificador UUID, que ya obtuvimos en el apartado 3.3.2.

Con los identificadores configurados, se avanza a la obtención de los datos, que se inicia al pulsar el botón de "Empezar captura". En ese momento, se ejecutan los métodos *enableNotifications* y *configureSensors*, que son los que permiten habilitar la lectura de los datos que vayan entrando a la característica de cada sensor y configurar dichos sensores para que tengan la frecuencia de toma de datos y los rangos adecuados de cada sensor.

```
668    public void configureSensors() {
669        setPeriod(mBleGatt, value: 50);      //Configurar frecuencia timer a 50 Hz (periodo 20ms)
670        setSensors(mBleGatt, sensor: 0, config: false, value: 4);     //Configurar ACC  FS:  4 g
671        setSensors(mBleGatt, sensor: 0, config: true, value: 50);     //Configurar ACC  ODR: 50 Hz
672        setSensors(mBleGatt, sensor: 1, config: false, value: 2000);  //Configurar GYR  FS:  2000 dps
673        setSensors(mBleGatt, sensor: 1, config: true, value: 50);     //Configurar GYR  ODR: 50 Hz
674        setSensors(mBleGatt, sensor: 2, config: false, value: 32);    //Configurar MAG  FS:  32  gauss
675        setSensors(mBleGatt, sensor: 2, config: true, value: 20);     //Configurar MAG  ODR: 20 Hz
676    }
```

Figura 4.15. Configuración de los sensores.

Una vez actividas las notificaciones de una característica GATT, el cliente de la conexión ya puede acceder a la información de la misma, recibir los datos. El método con el cual se obtienen es *onCharacteristicRead*, que se encuentra implementado en la llamada de la clase *BluetoothGattCallback*. Esta es una clase que se ejecuta de formas diferentes, según el estado de la conexión GATT. Tiene diferentes métodos como el anterior, que se ejecuta al realizar lecturas de características, pero también otros que se ejecutan al detectar cambios en la conexión (como un fallo o desconexión), o al detectar escrituras, como la configuración de los sensores.

En este método *onCharacteristicRead* se ha programado para que, al recibir la señal de lectura, envíe los datos de la característica a lo que se llama un manejador o *Handler*, que será el que realice el ajuste de los datos y el almacenamiento en la base de datos.

```
551    @Override
552    public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) {
553
554        if(characteristic.getUuid().equals(UUID_CHARACTERISTIC_SENSORS_DATA)){
555            mHandler.sendMessage(Message.obtain( h: null, MSG_MOVEMENT, characteristic));
556        }
557    }
```

Figura 4.16. Obtención de los datos de la característica.

Ajuste de los datos

Los datos que se obtienen de la característica tienen el siguiente formato:

Octets LSB	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Name	TS		Accelerometer						Gyroscope						Magnetometer					
Axis	-		X		Y		Z		X		Y		Z		X		Y		Z	
Value	0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX		0xXXXX	

Accelerometer payload: mg, signed int16
Gyroscope payload: tenth of dps, signed int16
Magnetometer payload: mGa, signed int16

Figura 4.17. Formato de los paquetes de movimiento. (8) (pág. 58 del manual)

Como se observa en la Figura 4.27, se trata de un paquete de 20 bytes en los que los primeros dos octetos son el *timestamp*, la marca de tiempo del paquete, y los siguientes 18 son los datos de los sensores, divididos en 6 bytes cada sensor, 2 bytes cada eje. Cada dato obtenido se representa con 4 dígitos hexadecimales, y las unidades utilizadas no son las adecuadas para nuestro trabajo.

Para acondicionar estos datos, se ha creado el método *split*. Este método llama, a su vez, a otros métodos de otra clase que hemos creado de forma externa, llamada *Conversion*.java. Esta clase consta de varios métodos:

- *convertTS:* coge el campo del *timestamp* y lo coloca en una variable. No necesita de conversión de unidades.
- *convertACC:* toma los datos de los campos de acelerómetro en cada uno de los ejes, los convierte a g (fuerza de la gravedad) y los coloca en un vector de tipo *Double*[].
- *convertGYR:* toma los datos de los ejes del giróscopo, los convierte a rad/s (radianes por segundo) y los coloca en un vector *Double*[].
- *convertMAG:* toma los datos de los ejes del magnetómetro, los convierte a Ga (gauss) y los coloca en un vector *Double*[].

Finalmente, el método *split* crea un objeto del tipo Movimiento (definido en la base de datos, apartado 4.3), utilizando para ello el identificador mov (se crea al iniciar el proceso), el *timestamp* y los datos de los sensores. Con el objeto de este tipo, el *Handler* va trabajando en la detección del golpe.

Detección del golpe

Como detección del golpe se entiende la detección del momento en el que empieza el golpe y en el que termina, esto es, que solamente se registre como Golpe en la base de datos el valor de los sensores dentro de ese periodo. Esto es necesario para el análisis posterior, ya que no todos los golpes tendrán la misma duración y no se puede analizar solamente utilizando los *timestamp*.

Por tanto, se ha utilizado el valor del acelerómetro en sus tres ejes para establecer estos límites. Se ha elegido porque su gráfica es la que mejor representa el movimiento, y se puede identificar de forma clara el movimiento que genera el golpe en ella.

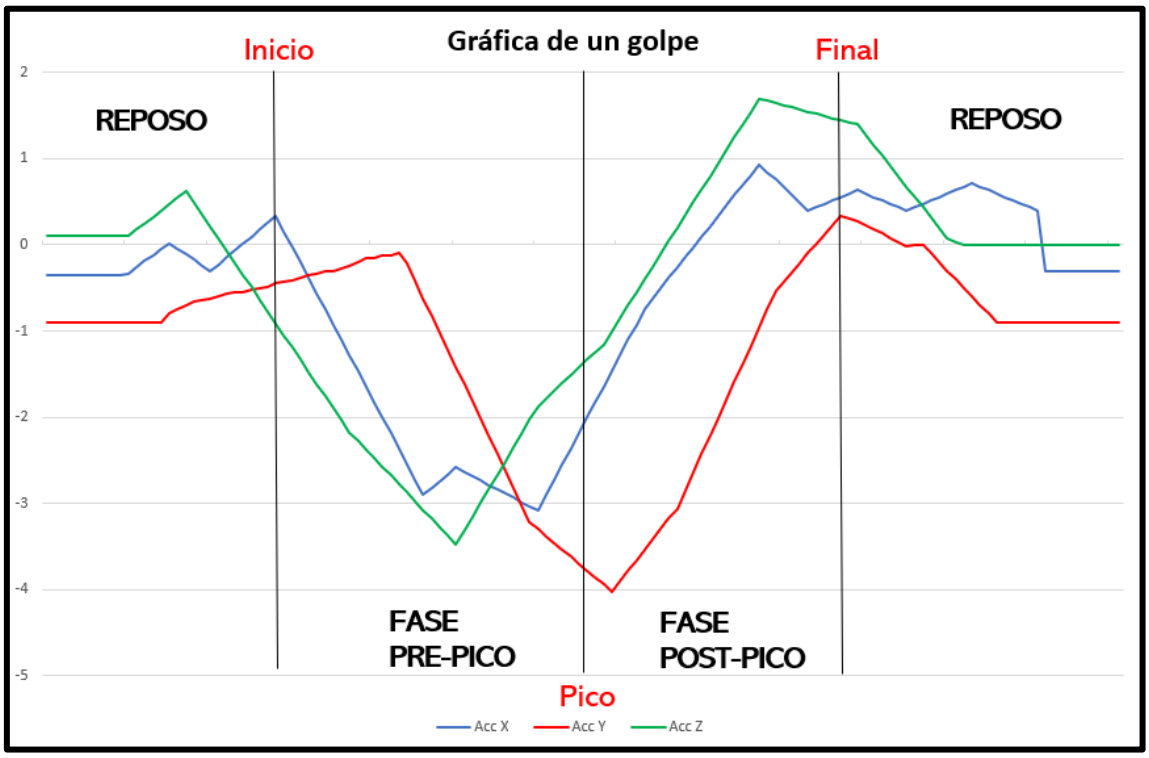

Figura 4.18. Gráfica de aceleración (3 ejes) de un golpe. Extraído de un golpe registrado con el *wearable*.

Como se observa en la figura 4.18, un golpe se divide en cuatro fases. En las partes de reposo existe una pequeña variación, que aparece porque el brazo no está estático al 100%. El punto de inicio se alcanza cuando al menos dos de los tres ejes ya han comenzado el movimiento, con lo que el valor absoluto del movimiento, es decir, la amplitud, empieza a crecer. Entonces se entra en la fase inicial o pre-pico. Mediante la observación, se ha llegado a la conclusión de que el eje Y de la aceleración es el que alcanza siempre el pico en último lugar, con lo que cuando se llegue a dicho momento se acabará esta fase y comenzara la fase post-pico. Esta fase terminará cuando los tres ejes hayan superado cierto nivel, con lo que se habrá vuelto a la fase de reposo (que no es del todo igual que la inicial).

Todo este proceso se ha implementado en la aplicación mediante el uso de flags o banderas. Los flags son variables de tipo *boolean*, es decir, un bit que está a *true* o *false*, y se activan o desactivan al gusto del programador cuando se pase cierto hito o situación.

En nuestro caso, se inicializarán las tres flags a *false*. Una vez se detecte el inicio, se activará (pasará a *true*) el primer flag. En ese momento, comenzará el proceso de tomar los datos de los sensores. Una vez se llegue al pico, se activará el segundo flag. Esto no cambia el proceso de toma de datos. A la llegada al final del golpe, se activará el tercer flag, lo que hará que se detenga la toma de datos.

Este proceso se controla mediante el *Handler*. A estos tres flags, se añade un cuarto que se activa cuando se detecta la pulsación del botón que pedirá el registro de un nuevo golpe (ver apartado siguiente). La activación de este flag hará que el proceso termine, se resetee las variables que lo controlan y se vuelva a iniciar.

Registro en Base de Datos

Al pulsar el botón de "Stop" o el de "Nuevo Golpe", se procede a crear un nuevo registro de la tabla Golpes. El identificador que se utiliza es el parámetro Mov que se ha definido al iniciar el golpe, y que corresponde con la fecha y hora en formato AAAA-mm-dd HH:mm:ss, por lo que no se repetirá, ya que no es posible registrar dos golpes dentro del mismo segundo. Este mismo identificador se coloca en cada uno de los movimientos que se han ido obteniendo de los sensores y que se van guardando como registros en la tabla Movimientos conforme van llegando, una vez que nos encontramos dentro de la zona de registro, es decir, en la fase pre-pico o post-pico.

El número de registros de la tabla Movimientos que se crean en el registro de un golpe será la duración de ese golpe dividida entre los 20ms de periodo de muestreo. Este número no será demasiado elevado, ya que las fases válidas de un golpe (los periodos donde se guardan los datos) no tienen una duración elevada, sino de menos de un segundo.

En este momento, como aún no se ha llevado a cabo el análisis, se coloca en la columna correspondiente del registro del golpe como "Sin analizar".

Una vez realizado todo este proceso, funciona de dos maneras diferentes: si el registro realizado es de un solo golpe, se lanza la actividad de análisis del golpe, donde se podrá ver los datos y gráficas del mismo, así como el tipo de golpe que el sistema ha detectado. En el caso de que el registro sea de múltiples golpes, se vuelve al menú principal de la aplicación, donde se habrá añadido los golpes realizados al historial y estarán sin analizar. Al entrar en los detalles se realizará el análisis de cada golpe de forma individual.

4.4.5 Bloque Análisis

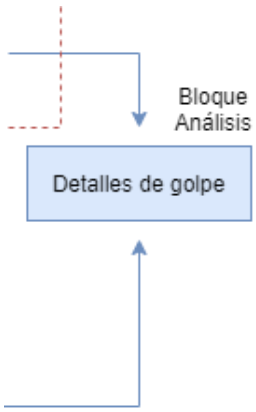

Figura 4.19. Estructura del módulo de análisis.

El bloque de Análisis se basa en una Actividad, *DetallesActivity.java*, y dos Fragments, *DetallesFragment.java* y *GraficasFragment.java.* Se puede acceder tanto desde el menú principal como desde el registro del golpe, y no varía si se realiza un registro múltiple, ya que los detalles siempre tratan el golpe como unidad individual.

La Actividad es la que actúa como base para el conjunto, obteniendo los datos del golpe, realizando el análisis y mostrando lo que el Fragment que esté activo necesite. Se ha creado un botón semejante al del menú principal, en el que se marca una opción u otra, la de Detalles o la de Gráficas.

Para el análisis se ha implementado el Algoritmo de Madgwick[10]. El objetivo era el de, utilizando los datos de los sensores, obtener la representación del golpe en cuaterniones, con lo que resulta más sencillo realizar un análisis y diferenciar entre los diferentes tipos de golpes de raqueta. Con ello, se ha conseguido un análisis sencillo, en el que la aplicación interpreta los datos del cuaternión obtenido y define qué tipo de golpe (derecha, revés o saque) ha realizado el usuario en el registro.

El procedimiento consiste en obtener los datos de movimiento de los tres sensores almacenados en la base de datos, y a partir de ellos, ir obteniendo el cuaternión correspondiente, que se guarda en una variable. También se obtiene el dato del cuaternión final, con el que se realizará la estimación del tipo de golpe. Mediante la observación, se ha podido identificar ciertas diferencias en este valor entre los diferentes tipos de golpes: cada golpe tiene un valor de cada componente que sirve para identificarlo. Para ello, se ha utilizado umbrales ya que no suele ser un valor exacto. De esta manera, el sistema ha de realizar un número de operaciones reducido, la comprobación de que cada componente esté dentro de un umbral para certificar si es ese tipo de golpe o no.

Otra alternativa es recorrer todo el movimiento, ya que la gráfica de cada componente del golpe es característica del movimiento realizado, pero se descartó ya que, si bien sería más exacto a la hora de identificar el golpe sin errores, no existía tanta diferencia con el procedimiento elegido como para justificar el gran aumento en el coste computacional (ya que habría que recorrer cuatro vectores de cientos de valores y compararlos con otro que actuara de modelo).

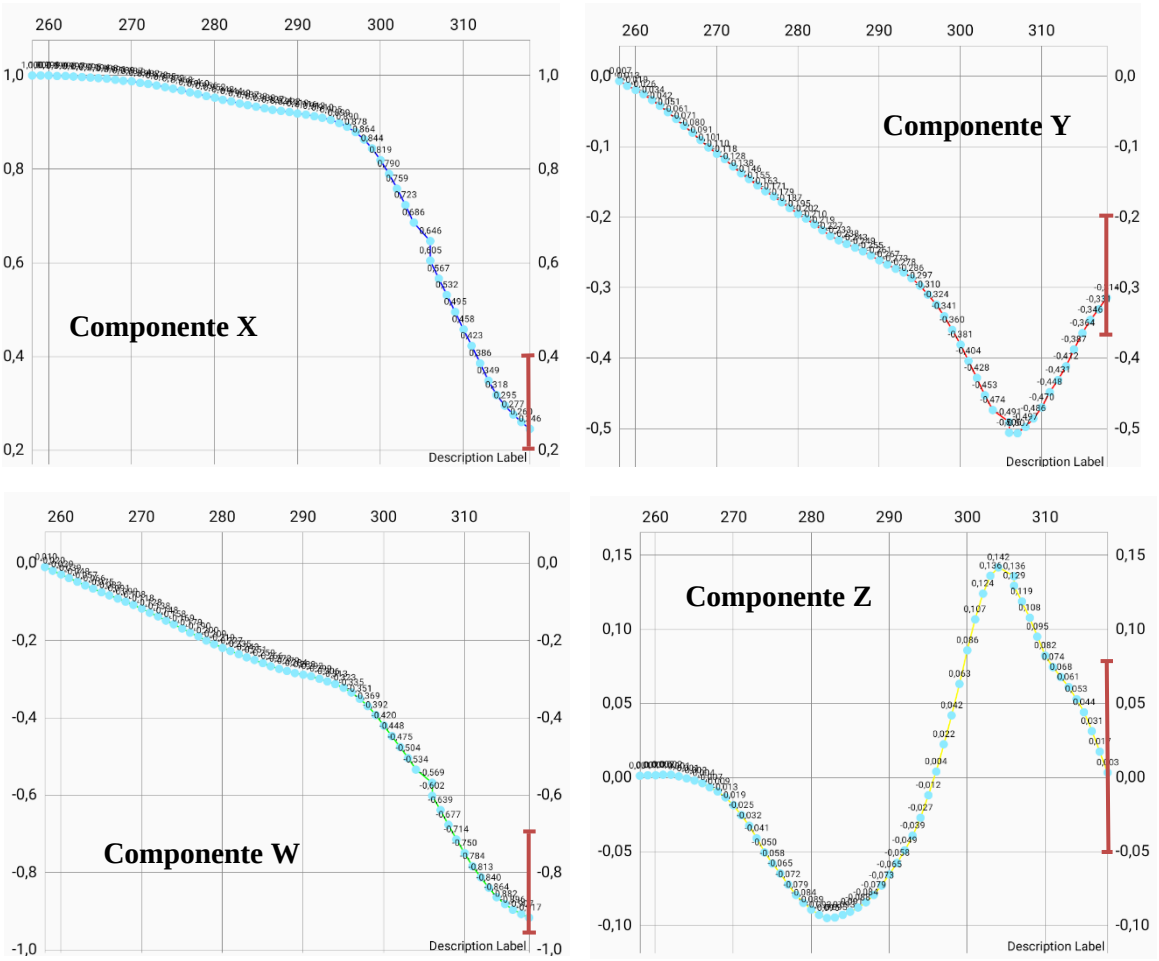

Figura 4.20. Gráficas resultantes de un golpe de revés. En rojo, marcados los umbrales elegidos para la detección.

[10] Código del algoritmo para Android en GitHub: https://github.com/IdeoG/android-madgwick

Como se observa en la figura 4.20, se toma el valor final del cuaternión, obtenido a partir de los valores almacenados en la base de datos en última posición, y se comprueba si está dentro de ciertos umbrales, que son diferentes según el tipo de golpe. Tienen que cumplir las cuatro componentes, lo que hace que la probabilidad de confusión en la detección (que elija un tipo de golpe que no es el que se ha realizado) es baja, ya que es difícil que, al realizar un golpe, las cuatro componentes estén dentro de los umbrales de otro tipo de golpe. Sí que hay más probabilidades de que no se ajuste en ninguno de los golpes preestablecidos, lo que conlleva que de vez en cuando dé error y muestre "Mala detección".

DetallesFragment.java

En este Fragment se presentan los datos del Golpe (fecha, duración, tipo de golpe). Aparece por defecto al lanzarse la actividad. Consiste en un layout, *fragment_activity_detail.xml*, con campos de texto en la parte superior, donde se introducen estos datos obtenidos de base de datos, y una imagen en la parte inferior, donde se muestra el tipo de golpe que el sistema ha detectado, o si la detección ha resultado errónea.

GraficasFragment.java

En este Fragment se muestran los datos gráficos del recorrido del cuaternión en el registro del golpe. Aparece al pulsarse el botón inferior correspondiente, y muestra cuatro botones en la parte superior para cambiar entre las cuatro componentes del cuaternión. Se ha implementado para dar la posibilidad de comparar entre golpes de forma visual, ya que si la detección da error, puede ser tanto por un fallo en los umbrales elegidos como por un fallo en el registro del golpe. Observando la gráfica podremos entender si es por un motivo u otro, además de conocer al detalle el movimiento que se realiza al golpear la bola.

Para representar las gráficas, se ha utilizado la clase *LineDataSet* de la librería *MPAndroidChart*.[11] Esta clase nos permite representar un objeto de tipo *Entry*, con los datos de cuaternión (valor numérico) y su *timestamp* asociado.

[11] Librería MPAndroidChart: https://weeklycoding.com/mpandroidchart-documentation/

4.5 Resultados obtenidos: Aplicación móvil

En este apartado se procede a mostrar el resultado final que se ha obtenido con el desarrollo de la aplicación.

4.5.1 Inicio

Al abrir la aplicación, lo primero que aparece en pantalla es una imagen de bienvenida a modo de "*splash screen*", que dura unos segundos antes de cargar la primera actividad de la aplicación.

Figura 4.21. Pantalla de inicio.

Es la primera imagen que ve el usuario y su utilidad es la de presentar la aplicación.

4.5.2 Bloque de usuarios

La siguiente pantalla que se muestra es la de usuarios, donde se elige el usuario con el que se quiere iniciar la sesión y se da la opción de registrar un nuevo usuario.

Los usuarios se colocan en una lista en la que cada entrada actúa como un botón que inicia la sesión con el usuario elegido. Para registrar un usuario nuevo, se pulsa el botón inferior con el que se accede al menú correspondiente.

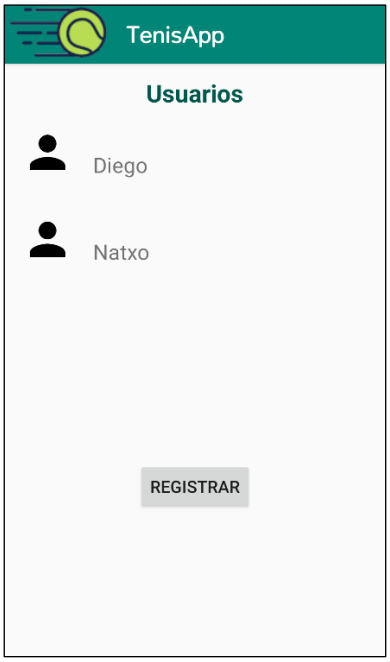

Figura 4.22. Menú de inicio de sesión.

El menú de registro de usuario incluye tres campos de introducción de datos, uno de cada tipo: el primero sirve para introducir el nombre y por tanto es un campo de texto, el segundo es para la edad y es un campo numérico, y el último es para introducir el brazo dominante del usuario a la hora de realizar golpes, y, pese a que también el dato que se introduce es de texto, se ha utilizado un formato de lista desplegable para que el usuario no introduzca una respuesta no esperada con la que podría haber fallos.

Una vez que se pulsa el botón aceptar, se vuelve al menú de inicio de sesión.

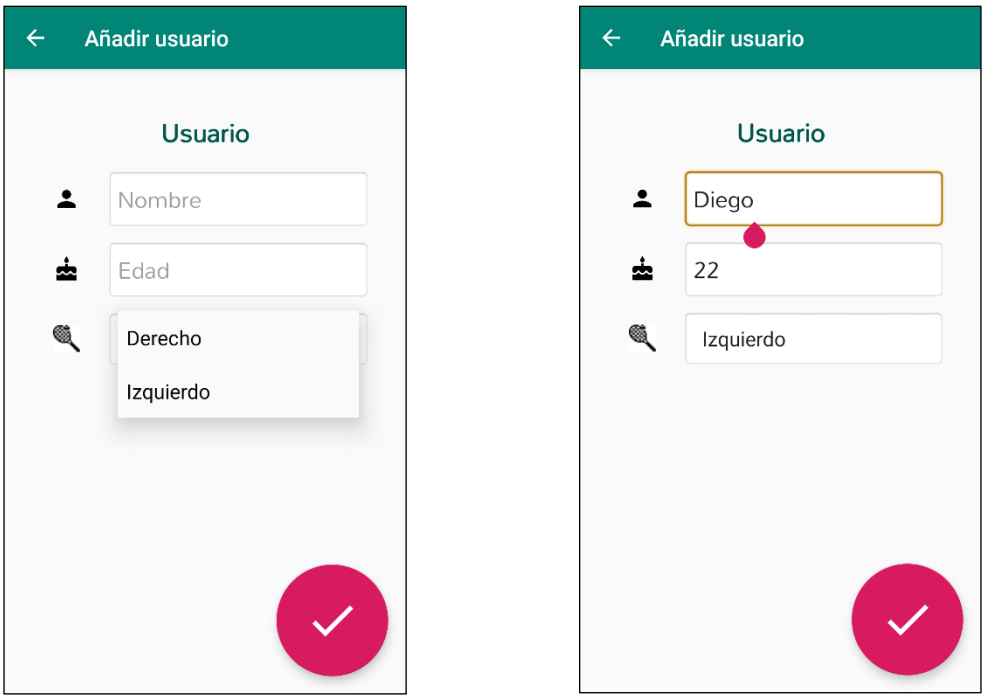

Figura 4.23. Menú de registro de usuario.

4.5.3 Menú Principal de la aplicación

Es el punto al que se accede tras iniciar sesión. Es el menú donde se puede realizar todas las acciones una vez se ha definido el usuario que se quiere utilizar.

Figura 4.24. Menú principal.

Consiste en varias partes, siendo la principal en cuanto a tamaño y visibilidad el botón central de registro de nuevo golpe, que incluye un interruptor con el que se elige si se quiere realizar un registro de un solo golpe o de múltiples golpes. En la parte superior encontramos un menú desplegable donde tendremos varias opciones, y finalmente en la parte inferior un botón doble, para cambiar entre el menú y el historial de golpes registrados.

Figura 4.25. Desplegable superior.

Cuando se pulsa una opción que comporte el borrado de algún registro, aparece un diálogo para evitar errores que no se puedan arreglar.

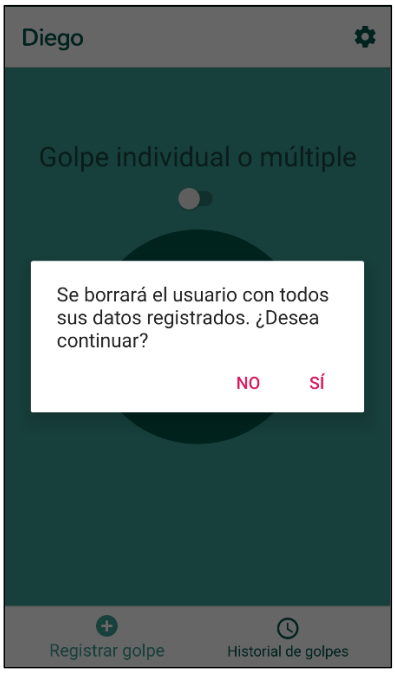

Figura 4.26. Diálogo de confirmación.

En la parte inferior se puede observar el botón doble, en el que por defecto está activo el menú principal con el registro de golpe, pero que al pulsar la otra parte del botón se cambia al historial de golpes. Al pulsar en un golpe, se muestran los detalles del mismo.

Figura 4.27. Historial de los golpes realizados.

4.5.4 Registro de un nuevo golpe

Al pulsar el botón de nuevo golpe en el menú principal, se accede a la lista de los dispositivos cercanos para elegir a cuál se desea conectar.

Figura 4.28. Elección de dispositivo BLE.

Al pulsarlo, se realiza la conexión y se avanza al registro del golpe. Esta pantalla será diferente según si se ha elegido registro de un golpe o múltiple. En este último caso, añadimos el golpe realizado con el botón que aparece, de manera que se pasa a registrar uno nuevo.

Figura 4.29. Pantallas de registro de golpe individual (izq.) y múltiple (der.).

Al terminar el registro, si es un registro individual se pasa a los detalles del golpe que se acaba de realizar, y si es un golpe múltiple se redirige al menú principal.

4.5.5 Detalles del golpe

En este bloque tenemos dos pantallas: una con detalles generales del golpe, donde podremos ver la duración, fecha, tipo de golpe, dispositivo con el que se ha registrado y brazo dominante, además de una imagen que corresponde con el golpe detectado; y otra con las gráficas del recorrido de los cuaterniones en el golpe.

También contiene un botón para eliminar el golpe en caso de que no se quiera mantener en el historial y otro para guardar los datos de los sensores almacenados en un documento y exportarlo a la memoria interna del teléfono, útil de cara a posibles análisis externos.

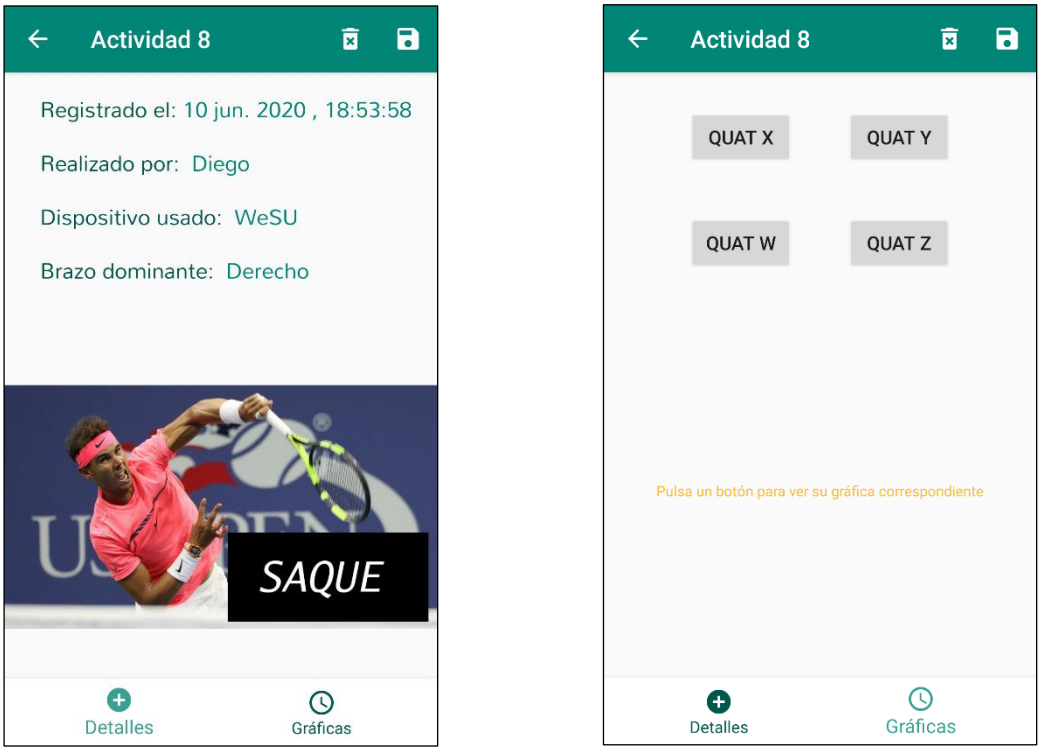

Figura 4.30. Detalles del golpe.

Capítulo 5. Conclusiones y propuesta de trabajo futuro

5.1 Conclusiones

Se propuso como objetivo principal el de realizar una aplicación que sirviera como herramienta para deportistas de nivel aficionado a la hora de medir diversos aspectos de la práctica del deporte y así poder dirigirse hacia una mejora técnica. Se puede decir que este objetivo se ha cumplido, ya que, mediante el uso de la aplicación creada y el dispositivo wearable utilizado, se permite monitorizar el movimiento realizado a la hora de dar un golpe de tenis, lo que da una gama de posibles lecturas: desde la técnica de golpeo, pudiendo comparar el golpe registrado en el momento con cualquier otro golpe (lo que puede llevar a entender posibles fallos o defectos a corregir), como observar la consistencia dentro de un grupo de golpes consecutivos, e incluso medir el progreso del usuario hacia el objetivo, teniendo disponible el historial de golpes con su fecha de registro.

Una aplicación así constituye una buena base para este tipo de usuarios que no tienen acceso a sistemas ya existentes de este tipo pero normalmente restringidos para usuarios avanzados, ya sea por su complejidad o por su precio, dirigido a deportistas de alto nivel. En cambio, con esta app, que utiliza un sistema de medida de bajo precio y se implementa en un smartphone como el que tiene la mayoría de la población, se acerca este tipo de herramientas a usuarios de nivel bajo, que son la mayoría de los deportistas que existen.

También se ha cumplido con el objetivo de entender mejor los sistemas utilizados, tanto el de comunicación como el entorno de desarrollo. El entorno Android Studio se utiliza en el grado, pero de manera introductoria. En este proyecto se ha profundizado en su uso y se ha aprendido a implementar nuevos tipos de clases, como los Fragments, que resultan muy útiles. En cuanto al sistema Bluetooth Low Energy, se ha pasado de un nivel de conocimiento muy básico y obtenido en forma de cultura general a entenderlo perfectamente, sabiendo cómo se realiza la conexión entre dispositivos y cómo se accede a la información que transporta.

Otro objetivo cumplido es el de no haber realizado cambios en el firmware del wearable. Mediante un estudio del mercado se ha acabado eligiendo el STEVAL-WESU, con el que se ha conseguido implementar la aplicación utilizando el dispositivo con la configuración de fábrica, solamente cambiando algunos parámetros como la frecuencia de muestreo pero sin mayor problema, cosa que sí surgía si se utilizaba otro dispositivo. Esto ha hecho que se simplifique una parte del proyecto, ahorrando tiempo que se ha podido utilizar en otros aspectos.

No obstante, se puede mencionar algún problema que ha surgido en el desarrollo del proyecto. El mayor inconveniente que ha tenido lugar ha sido la dificultad de uso de los sensores del wearable en relación con el algoritmo de Madgwick. Cada vez que se realizaba un pequeño cambio en algún paso, aparecía algún fallo en la monitorización o en la representación del cuaternión, por lo que se ha tenido que ir con cuidado en ese aspecto.

5.2 Propuesta de trabajo futuro

Como se ha mencionado, esta aplicación puede funcionar como base para la monitorización del rendimiento en un deporte. Esto quiere decir que se trata de una implementación sencilla, que requiere de poca potencia computacional y que se realiza en un smartphone normal. Por tanto, el análisis que realiza es bastante simple, y adaptable a otro tipo de deporte de manera que pueda monitorizar casi cualquier tipo de movimiento.

Como trabajo futuro, la propuesta sería la de trabajar con el algoritmo de detección de golpes. Se puede aumentar su precisión si se realiza una comparación del golpe entero y no solamente del final, lo que puede dar lugar a errores en la detección. Para esto, se propone el registro de un golpe que se utilice como ejemplo a partir del cual el resto se comparen y, mediante el uso de un

algoritmo de correlación, se obtenga un porcentaje de similitud entre el golpe realizado y el ejemplo. Mediante este método podría mejorarse el sistema hasta el punto de poder detectarse todo tipo de golpes y no solamente los tres básicos que se detectan en esta aplicación, con lo que podría adaptarse mejor al pádel, donde las voleas son básicas y en este proyecto no se estudian de forma específica.

Como alternativa, también se propone un análisis que implemente *machine learning*. Esto es, que mediante la introducción de datos, en este caso el registro de golpes, la propia aplicación aprenda y pueda identificar los golpes variando los requisitos por sí sola, sin la necesidad de un programador, ajustando los umbrales de comparación a medida que se observan distintos golpes.

Un análisis de este tipo necesitaría de mayor potencia computacional, por lo tanto otra propuesta sería la de integrar esta aplicación con el uso de lo que se conoce como *cloud computing* o computación en la nube. Mediante esta integración, se podría realizar el análisis de los datos obtenidos de forma remota en la nube, lo que aceleraría el proceso y permitiría aumentar la complejidad.

Por último, se puede hacer lo mismo con la base de datos. Para el proyecto se ha elegido una base de datos SQLite local por su facilidad de uso, pero es posible implementar una base de datos MySQL y alojarla en un servidor web o en la nube, de forma que almacene no solamente a los usuarios locales sino a todos los que utilicen la app. Esto haría que se pudiera implementar un sistema de comparación entre perfiles y obtención de datos de otros usuarios con los que comparar, y por tanto más posibilidades de mejora del rendimiento.

En definitiva, se puede, utilizando esta app como base, aumentar la complejidad y las funcionalidades hasta cubrir un espectro más amplio de deportistas, tanto en número de deportes como en nivel.

Capítulo 6. Bibliografía

[1] RFET (Real Federación Española de Tenis) y Padel Federación. Número de Licencias. 2019.

http://www.rfet.es/es_licencias_introduccion.html

https://www.padelfederacion.es/Datos_Federacion.asp?Id=0

[2] Wavemaker. Estudio Live Panel, noviembre de 2019.

https://www.marketingnews.es/investigacion/noticia/1156101031605/wearables-crece-numero-de-espanoles-tienen-uno.1.html

[3] GlobalData. Estudio de mercado wearable, febrero de 2020.

https://www.globaldata.com/wearable-tech-is-maturing-but-is-still-out-of-the-mainstream-says-globaldata/

[4] Hexiwear, MikroElektronica, Serbia. Todos los datos extraídos del apartado 'Hardware'

https://www.mikroe.com/hexiwear

Acelerómetro y Magnetómetro: https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf

Giróscopo: https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf

[5] NXP Semiconductors, Países Bajos.

Microprocesador:

https://www.nxp.com/products/processors-and-microcontrollers/armmicrocontrollers/general-purpose-mcus/k-series-cortex-m4/k6x-ethernet/kinetis-k64-120-mhz-256-kb-sram-microcontrollers-mcus-based-on-arm-cortex-m4-core:K64_120

Procesador BLE:

https://www.nxp.com/products/wireless/bluetooth-low-energy/kinetis-kw40z-2-4-ghz-dual-mode-bluetooth-low-energy-and-802-15-4-wireless-radio-microcontroller-mcu-based-on-arm-cortex-m0-plus-core:KW40Z

Sensores:

https://www.nxp.com/products/sensors/motion-sensors/6-axis/digital-motion-sensor-3daccelerometer-2g-4g-8g-plus-3d-magnetometer:FXOS8700CQ

https://www.nxp.com/products/no-longer-manufactured/3-axis-digital-gyroscope:FXAS21002C

[6] SensorTag CC13xx – CC3200, Texas Instruments, EEUU.

http://www.ti.com/product/CC1350

Datasheet: http://www.ti.com/lit/ds/symlink/cc1350.pdf

http://www.ti.com/product/CC3200

Datasheet: https://www.ti.com/lit/ds/symlink/cc3200.pdf

[7] MPU-9250. Invensense, EEUU.

https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/

Datasheet: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

[8] STEVAL-WESU1. ST Microelectronics, Suiza.

https://www.st.com/en/evaluation-tools/steval-wesu1.html

Datasheet: https://www.st.com/resource/en/data_brief/steval-wesu1.pdf

Manual de usuario: https://www.st.com/resource/en/user_manual/dm00279614-how-to-use-the-stevalwesu1-stmicroelectronics.pdf

Acelerómetro y Giróscopo: https://www.st.com/resource/en/datasheet/lsm6ds3.pdf

Magnetómetro: https://www.st.com/resource/en/datasheet/lis3mdl.pdf

[9] ¿Qué es Android? https://www.android.com/intl/es_es/what-is-android/

Google Inc, EEUU.

[10] Datos y características de iOS. https://www.apple.com/es/ios/ios-13/ Apple Inc, EEUU.

[11] Gráfico propio, datos de StatCounter. Diciembre de 2019.

https://gs.statcounter.com/os-market-share/mobile/worldwide

[12] Kai OS. https://www.kaiostech.com KaiOS Technologies Inc, EEUU.

[13] Counterpoint Research, cuota de mercado de *smartphones* por marca, 2019.
https://www.counterpointresearch.com/global-smartphone-share/

[14] CCN-CERT, Dispositivos y comunicaciones móviles. Informe Anual 2019.

https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/4625-ccn-cert-ia-03-20-informe-anual-2019-dispositivos-y-comunicaciones-moviles-1/file.html

[15] Android Studio. https://developer.android.com/studio/

[16] Greg Welch, Gary Bishop. *An Introduction to the Kalman Filter, 2001.*

http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf

[17] Sebastian O.H. Madgwick. *Madgwick Algorithm orientation filter.* University of Bristol, 2009

https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf

https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/

[18] Wikipedia, Ángulos de Euler

https://es.wikipedia.org/wiki/Ángulos_de_Euler

[19] Especificaciones BLE

https://web.archive.org/web/20170310111443/https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy

[20] Introducción a GATT.

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

[21] UUID.

https://www.bluetooth.com/specifications/assigned-numbers/service-discovery/