UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Contribución a las redes de sensores corporales. Diseño y desarrollo de un dispositivo empotrado inalámbrico para registrar los movimientos realizados

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA INDUSTRIAL

Autor: **Elías Hernández Gómez**
Director: Juan Antonio López Riquelme
Codirectora: Nieves Pavón Pulido

Índice

Capítulo 1

Introducción

1.1 Contexto

El avance tecnológico no se detiene, y los procesos industriales nos otorgan dispositivos de un tamaño y un consumo energético cada vez más reducido. Esto abre la puerta a la llegada de nuevos dispositivos como los sensores corporales, que debido a su reducido coste, pueden ser combinados con dispositivos ampliamente implantados en la sociedad, como son los smartphones, permitiendo así que las nuevas tecnologías se apliquen a campos en los que hasta ahora su uso estaba limitado a determinados nichos de mercado.

Además, el uso de estándares aumenta la compatibilidad entre fabricantes, lo que aumenta las posibilidades de que cualquier consumidor pueda acceder a los nuevos dispositivos.

Uno de los sectores que más se ha beneficiado de la miniaturización de componentes es el de los sensores corporales. Estos sensores monitorizan y recopilan una serie de datos en tiempo real, que posteriormente pueden ser interpretados para inferir datos sobre el portador de dichos sensores. Por ello, su ámbito de aplicación abarca desde el sector deportivo, hasta la monitorización de la salud, disciplina conocida como e-health, en la cual se pueden analizar los datos obtenidos por el sensor para comprobar la evolución de un paciente que esté afectado por diversas patologías que requieran de una monitorización que hasta la fecha resultaba tosca, cara, y que además no se producía de

manera transparente para el paciente, mientras que con los sensores actuales el paciente muchas veces olvida que está siendo monitorizado constantemente.

De entre las muchas variables a monitorizar, para este trabajo se ha optado por registrar la aceleración producida por el brazo del paciente, ya que más allá del ámbito deportivo, no es un dato al que los sensores corporales presentes en el mercado le suelan prestar atención, y el desarrollo de una arquitectura que sea capaz de registrar la aceleración puede abrir la puerta a estudios relacionados con trastornos del movimiento como pueden ser el Parkinson.

En definitiva, las redes de sensores corporales conforman una rama de estudio bastante atractiva, en la que merece la pena la comprensión del hardware y del software que la hacen posible, desde el propio sensor que recopila los datos, pasando por la comunicación del sensor con el dispositivo inteligente que procesa los datos, hasta el envío de los datos a un servidor en internet que almacenará dicha información.

1.2 Objetivos

El objeto del trabajo es utilizar un sensor de tamaño reducido que permita obtener la aceleración del brazo del portador del mismo, transfiera dicha información a un microcontrolador ESP32, que será el encargado de obtener los valores del sensor y de enviarlos al dispositivo inteligente, en el cual se le permitirá al usuario la configuración de ciertos parámetros del sensor, y a través del cual los datos serán enviados a un servidor de Internet. En síntesis, los objetivos serían los siguientes:

- Seleccionar el sensor adecuado para registrar los movimientos de interés en el paciente.

- Seleccionar y programar el microcontrolador encargado de comunicarse con el sensor y con el dispositivo inteligente.

- Desarrollar una aplicación sobre arquitectura Android que se pueda comunicar bidireccionalmente con el microcontrolador, y que permita enviar los datos recopilados a un servidor de Internet.

- Comprobar el funcionamiento de la arquitectura desarrollada en un entorno real.

1.3 Desarrollo de la Memoria

1.3.1 Capítulo 1 - Introducción

En este capítulo se exponen los argumentos que han llevado a seleccionar el tema de este trabajo, se enumeran los objetivos a desarrollar y se presenta la memoria.

1.3.2 Capítulo 2. Estado del Arte

En este capítulo se exponen los diferentes sensores disponibles para medir la aceleración, así como varios microcontroladores comerciales que permiten la comunicación con el sensor y con el dispositivo inteligente. También se indican las diversas arquitecturas móviles sobre las que se puede implementar la aplicación encargada de comunicarse con el microcontrolador, y los diversos servidores encargados de recibir datos del conocido como Internet de las Cosas.

1.3.3 Capítulo 3. Descripción del Sistema

En este capítulo se exponen los elementos de hardware y de software que finalmente han sido seleccionados para realizar el trabajo.

1.3.4 Capítulo 4. Descripción del Hardware y Software Desarrollado

En este capítulo se explica la arquitectura de hardware y de software realizada para cumplir con el objetivo del trabajo.

1.3.5 Capítulo 5. Análisis de Resultados y Conclusiones

En este capítulo se analizan las conclusiones obtenidas tras implementar la arquitectura hardware y software y haberla probado en condiciones reales.

Capítulo 2

Estado del Arte

2.1 Introducción

Antes de comenzar a desarrollar la arquitecturas hardware y software, en las que se deben comprender cada uno de los elementos que componen las mismas, conviene conocer las redes de sensores corporales, entre las cuales se engloba el prototipo que se pretende desarrollar en este trabajo.

Una vez analizado el contexto, se explicará el interés en monitorizar la aceleración producida por el brazo, y por lo tanto se analizarán los diferentes sensores que podemos encontrar en el mercado para registrar esta aceleración.

Posteriormente, se estudiarán los diferentes microcontroladores que nos permitirán recibir los datos del sensor, así como el sistema de comunicación elegido para intercambiar datos entre el microcontrolador y el dispositivo inteligente.

Por último, se mostrarán las diversas alternativas disponibles sobre las que programar el dispositivo móvil, así como los servidores disponibles en el mercado para almacenar y procesar los datos recibidos por el dispositivo móvil.

2.2 Redes de sensores corporales

Las redes de sensores corporales, también conocidas como Body Area Network (BAN), o en su versión inalámbrica, Wireless Body Area Network (WBAN), consisten en el empleo de pequeños dispositivos que se pueden repartir a lo largo del cuerpo para monitorizar diferentes parámetros del usuario [1].

Estos dispositivos son ligeros, baratos y cuentan con una batería que les permite funcionar durante un tiempo razonable, por lo que son utilizados en varios sectores, como el de la salud, el deportivo o el militar [2].

La idea de utilizar diversos dispositivos alrededor del cuerpo fue introducida en 1996 por Zimmerman. El objetivo es que los dispositivos pudieran recopilar datos personales, por lo que se acuñó el término de Personal Area Network (PAN). Posteriormente, en 2001 Van Dam y otros autores se refirieron a este concepto como Body Area Network (BAN).

La arquitectura de estas redes se basa en disponer los sensores alrededor del cuerpo para que puedan recopilar los datos necesarios. Una vez que se ha recopilado la información, en función del número de sensores que se hayan utilizado, se pueden comunicar directamente con el dispositivo inteligente que enviará los datos a Internet, o será necesario que la información primero pase por un nodo intermedio, que será el encargado de hacer de puente entre los sensores y el dispositivo inteligente [3].

No hay un protocolo de comunicación establecido por defecto, por lo que en función de las necesidades específicas del ámbito en el que vayamos a aplicar los sensores, las comunicaciones entre los sensores y el dispositivo inteligente se pueden realizar por Bluetooth, ZigBee, WiFi, 3G, entre otros.

De la misma manera, una vez que el dispositivo inteligente ha recopilado la información de los sensores, puede comunicarse con Internet por WiFi, 3G, etc.

Ilustración 2-1. Red de sensores corporal

2.3 Acelerómetros comerciales

Una forma de detectar la presencia de trastornos del movimiento como puede ser el Parkinson, consiste en analizar el movimiento de las extremidades del individuo. Para ello, resulta de utilidad el empleo de acelerómetros, dado que detectan las variaciones de velocidad, y por lo tanto pueden utilizarse para comparar las aceleraciones de individuos sanos con respecto a individuos afectados por algún trastorno del movimiento.

2.3.1 Principio de funcionamiento del acelerómetro

Los acelerómetros son sensores que miden la variación de velocidad que experimenta un objeto. Para ello, cuentan con partes móviles que reaccionan a las fuerzas aplicadas al objeto [4].

Hay varios tipos de acelerómetros en función del principio de funcionamiento. Por un lado están los piezoeléctricos, en los que el material del sensor produce electricidad cuando se le somete a un esfuerzo físico. La aceleración producida en el objeto se relaciona con la intensidad producida.

También tenemos los piezoresistivos, en los que la aceleración incrementa la resistencia del material del sensor.

Por último, tenemos los capacitivos, en los que la aceleración provoca el movimiento entre las placas del sensor, lo que se refleja en un cambio en la capacidad del mismo, detectando así la aceleración producida.

De cara a desarrollar este trabajo, se va a utilizar un tipo concreto de acelerómetro, en el que el tamaño ha sido reducido al máximo. A estos dispositivos se los conoce como Micro Electro Mechanical Systems (MEMS), y son unos componentes de apenas unas décimas de milímetro, por lo que se pueden combinar varios de ellos en una placa como las que se mencionarán a continuación, para obtener varias funcionalidades en un tamaño reducido.

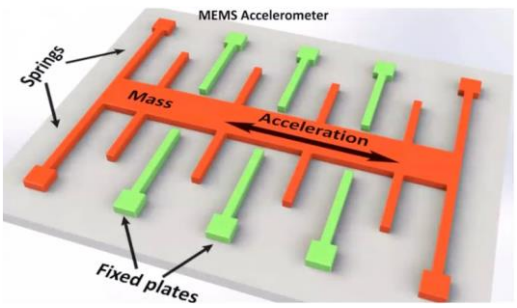

Ilustración 2-2. Acelerómetro capacitivo MEMS

2.3.2 LSM9DS1

Este dispositivo es un sensor con 9 grados de libertad. Entre sus características se encuentra un acelerómetro capaz de medir la aceleración en tres ejes, un giróscopo capaz de medir el giro en tres ejes, y un magnetómetro capaz de medir la intensidad del campo magnético en tres ejes. Por lo tanto, mediante la combinación de todos estos datos, podemos conocer en todo momento la posición y orientación de un objeto en el espacio [5].

Además, podemos ajustar el rango de medición de cada sensor, teniendo en cuenta que conforme aumentamos el rango disminuimos la sensibilidad. Para el acelerómetro podemos seleccionar un rango de ±2g, ±4g, ±8g y ±16g, para el giróscopo ±245º/s, ±500º/s y ±2000º/s, y para el magnetómetro de ±4G, ±8G, ±12G y ±16G.

En cuanto a la conectividad, el sensor es compatible con I^2C y con SPI, y se puede alimentar desde una fuente de entre 3 y 5 V de corriente continua. También cuenta con librerías para Arduino y para Adafruit. Su precio ronda los 20 €.

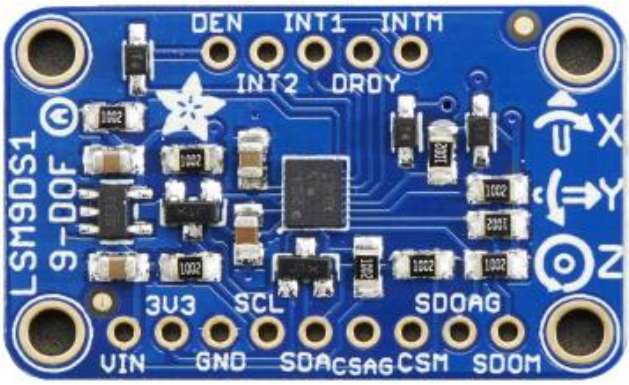

Ilustración 2-3. Parte frontal del LSM9DS1

Ilustración 2-4. Parte posterior del LSM9DS1

2.3.3 MPU6050

Este dispositivo cuenta con 6 grados de libertad, ya que dispone de un acelerómetro y de un giróscopo, los cuales pueden ser ajustados dentro de los siguientes rangos [6]. El acelerómetro ±2g, ±4g, ±8g y ±16g, y el giróscopo ±250º/s, ±500º/s, ±1000º y ±2000º/s.

En cuanto a las comunicaciones, cuenta con I^2C pero no con SPI, y soporte para Arduino y Adafruit. Respecto al precio ronda los 18 €.

Ilustración 2-5. MPU6050

2.3.4 ADXL345

Este dispositivo cuenta con un acelerómetro que se puede configurar con el rango ±2g, ±4g, ±8g y ±16g. Es bastante liviano, se puede comunicar mediante I^2C y SPI. La alimentación se realiza entre 3,3 y 5 V en corriente continua [7].

Tiene compatibilidad con Arduino, con Adafruit, y el precio ronda los 19 €.

Ilustración 2-6. ADXL345

2.4 Microcontroladores comerciales

Una vez obtenidos los datos de aceleración, hace falta un dispositivo capaz de comunicarse con el sensor para leer esos datos y para configurar los parámetros del sensor en caso de que sea necesario, así como de que tenga la capacidad de enviar los datos hacia un dispositivo inteligente que será con el que interactúe el usuario.

2.4.1 PSoC 6 BLE

Este dispositivo lo fabrica la compañía infineon, la cual adquirió a la antigua empresa que desarrollaba los PSoC, la empresa Cypress. El microcontrolador se puede adquirir mediante uno de los diversos Kit de desarrollo que ofrece la compañía, como por ejemplo con el CY8CKIT-062-BLE, que cuenta con la sexta generación de estos microcontroladores, que disponen de una CPU con arquitectura ARM M4 de 150 MHz, 1 MB de memoria flash, 288 KB de SRAM, 78 GPIO, 7 bloques programables analógicos, 56 bloques programables digitales, así como un sensor capacitivo [8].

El Kit de desarrollo incorpora una antena que permite realizar desarrollos que aprovechen el uso del Bluetooth de Baja Energía, también conocido como Bluetooth Low Energy (BLE), ya que este dispositivo es compatible con la versión 5.0 de Bluetooth. La antena emite a 2,4 GHz con una velocidad de transferencia de 2 Mbps.

El dispositivo es capaz de comunicarse mediante SPI, I^2C y UART, así como por USB. También cuenta con capacidad de emitir sonido, gracias a su modulador de pulsos.

El microcontrolador puede adquirirse como parte del Kit de desarrollo por unos 70€.

Ilustración 2-7. CY8CKIT-062-BLE

2.4.2 ESP32

Este dispositivo está fabricado por la compañía Espressif Systems. El Kit de desarrollo cuenta con un módulo ESP-WROOM-32, el cual utiliza una CPU con una frecuencia de 240 MHz. Dispone de un almacenamiento basado en 448 KB de ROM, 520 KB de SRAM y 4 MB de SPI flash [9].

Respecto a las conexiones, cuenta con soporte para tarjeta SD, UART, SPI, SDIO, I^2C, PWM, ADC, DAC, GPIO, y se puede alimentar desde un puerto USB de 5 V. El chip dispone de una antena que le permite comunicarse mediante WiFi, con la especificación 802.11b/g/n a una velocidad de 150 Mbps, o mediante Bluetooth, en su versión 4.2, por lo que admite el uso de Bluetooth de baja energía.

Las dimensiones del chip ascienden a 39 x 31 mm, y el precio se encuentra en unos 20 €.

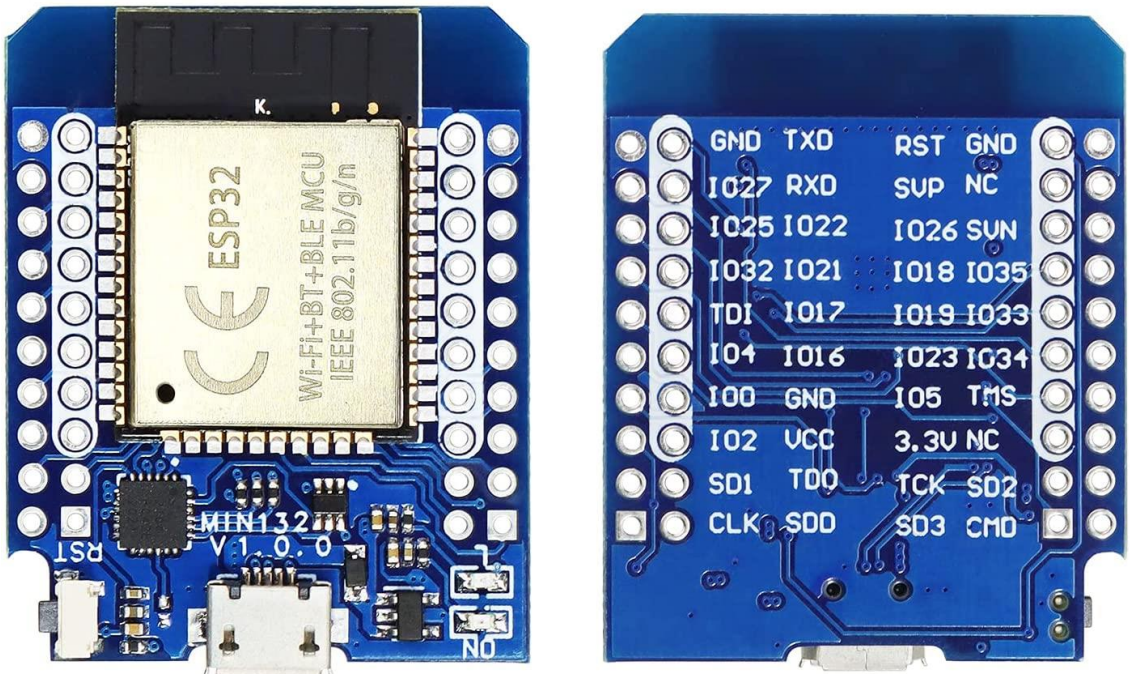

Ilustración 2-8. ESP32

2.4.3 nRF5340

Este dispositivo es un potente microcontrolador que incorpora dos procesadores ARM M33. También incorpora un microprocesador de 128 MHz. Para el almacenamiento cuenta con 1 MB flash, con 512 KB de RAM. El procesador encargado de las conexiones funciona a 64 MHz, tiene 256 KB de flash y 64 KB de RAM.

Respecto a las comunicaciones, soporta SPI, QSPI, USB, y sobre el Bluetooth, soporta la versión 5.3 con una velocidad de hasta 2 Mbps, junto con el Bluetooth de baja energía. También soporta NFC y ZigBee.

A su vez, el fabricante ha incluido medidas de seguridad como una unidad de almacenamiento de claves y almacenamiento seguro [10].

El microprocesador se ofrece como parte de un Kit de desarrollo por 40 €.

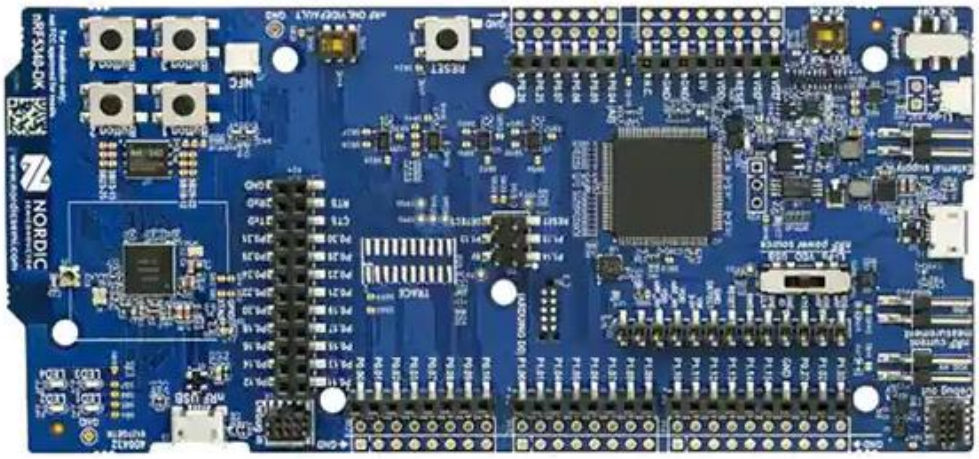

Ilustración 2-9. nRF5340

2.5 Sistemas de comunicación

Otro punto importante a la hora de seleccionar el microcontrolador, es la tecnología que utilizará para comunicarse con el dispositivo inteligente, dado que queremos que sea lo más transparente posible para el usuario.

2.5.1 Wi-Fi

Es una tecnología que utiliza ondas de radio para transmitir información de forma inalámbrica.

Esta tecnología empezó su andadura en 1985, cuando la Comisión Federal de Comunicaciones de Estados Unidos (FCC) liberó el espectro radioeléctrico de los 900 MHz, los 2,4 GHz y los 5,8 GHz para que cualquiera pudiera utilizarlo sin licencia.

Tras esto, las empresas empezaron a experimentar con el desarrollo de diferentes redes y dispositivos que utilizaban estas frecuencias, aunque al no existir un estándar, no había compatibilidad entre los distintos fabricantes [11].

Posteriormente, varias empresas punteras desarrollaron un estándar, conocido como 802.11, el cual fue aprobado en 1.997 por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE).

Tras esto, en 1.999 se creó la Wireless Ethernet Compatibility Alliance (WECA), actualmente conocida como la Wi-Fi Alliance, con el objetivo de dar a conocer el estándar.

Desde entonces, este estándar ha evolucionado ampliando sus capacidades. Actualmente, la última versión, Wi-Fi 7, también conocida como 802.11be, cuyos primeros dispositivos están previstos para 2024, permitirá una velocidad de transferencia de 40.000 Mbit/s, gracias al empleo de las bandas de 2,4 GHz, 5 GHz y 6 GHz, por lo que es una tecnología que se puede utilizar para enviar grandes cantidades de datos entre varios dispositivos [12].

Por otra parte, puede haber varias redes Wi-Fi que compartan canales del espectro radioeléctrico. No obstante, gracias a que cada paquete de datos enviado está identificado por el SSID del dispositivo emisor, esta tecnología está preparada para la coexistencia de varias redes Wi-Fi cercanas.

Por último, la tecnología cuenta con un alcance de hasta 100 metros, lo que unido a la posibilidad de encriptar los datos mediante WPA2, y a la madurez del mercado, en el que hay una amplia variedad de dispositivos que incorporan esta tecnología, la convierten en una forma de comunicarse muy potente y versátil.

2.5.2 Bluetooth

Al igual que Wi-Fi, Bluetooth es una tecnología que utiliza ondas de radio para transmitir información de forma inalámbrica, aunque al contrario que el primero, el alcance y el ancho de banda es menor.

Sus orígenes se remontan a 1.994, cuando la compañía Ericsson intentó sustituir las comunicaciones RS-232 existentes por un nuevo tipo de comunicación inalámbrica, que utilizaba una frecuencia similar a la de las Wi-Fi, pero enfocada en un alcance y consumo de energía inferiores [13].

Posteriormente, en 1.998 varias compañías fundaron el Bluetooth Special Interest Group (SIG) con vistas a desarrollar el estándar y licenciar los productos de los fabricantes.

En cuanto a sus características, esta tecnología utiliza la banda de entre 2,402 y 2,48 GHz. La última versión disponible es la 5.3, que admite una velocidad de hasta 50 Mb/s, y también permite encriptar la comunicación.

Por lo tanto, y aunque se puede utilizar para transferir archivos entre varios dispositivos de manera similar a las redes Wi-Fi, en los últimos años ha crecido su uso para comunicar a los dispositivos inteligentes con el denominado Internet de las Cosas (IoT), gracias al bajo consumo del conocido como Bluetooth de Baja Energía (BLE).

2.5.2.1 Bluetooth Low Energy

Esta tecnología se implementó a partir de la versión 4.0 de Bluetooth, con el objetivo de conseguir un menor consumo para realizar ciertas tareas que requieren una velocidad de transmisión de datos más reducida que utilizando Bluetooth tradicional [14].

Por lo tanto, y al contrario de lo que se suele transmitir en los medios de comunicación, no es una tecnología que permita utilizar el Bluetooth tradicional (para transferir archivos o escuchar música) utilizando menos energía, sino que es una tecnología que permite que Bluetooth consuma menos energía a la hora de realizar ciertas tareas, como puede ser la comunicación con pequeños dispositivos que envíen pequeños datos periódicamente, por lo tanto su uso está enfocado hacia Internet de las cosas (IoT), y por consiguiente su uso con los sensores utilizados en las redes de sensores corporales es muy acertado, dado que la mayoría de dispositivos móviles inteligentes incorporan Bluetooth.

Respecto a su arquitectura conviene conocer dos conceptos. El primero es el *Generic Access Profile* (GAP), que se encarga de determinar el rol con el que interactúan dos dispositivos (periférico o dispositivo central). El otro es el *Generic Attribute Profile* (GATT), que define la forma en la que se van a intercambiar los datos (quien realiza las peticiones de información y quién las responde).

Las transacciones GATT se realizan mediante objetos con la siguiente jerarquía: Perfil, Servicio, Característica. Un perfil es un conjunto de servicios, los cuales pueden ser personalizados por el fabricante, o pueden estar definidos por la asociación Bluetooth SIG. Los servicios a su vez contienen características. Las características contienen la información del sensor que queremos transmitir, y a su vez pueden tener un Descriptor, que describa por ejemplo la unidad en la que se envía la información, o que activen las notificaciones de la característica.

Por otra parte, los servicios y características se identifican mediante un UUID, los cuales tienen una longitud de 16 bits si han sido definidos por el Bluetooth SIG, y de 128 bits si son personalizados.

2.6 Dispositivos móviles inteligentes

El siguiente paso es seleccionar el dispositivo inteligente desde el que nos comunicaremos con el microcontrolador. Actualmente no hay por qué limitarse a un ordenador personal, sino que contamos con la posibilidad de utilizar un dispositivo móvil inteligente (smartphone) para ello, ya que cuenta con los estándares tecnológicos que hacen posible la comunicación con el resto de componentes de la arquitectura que se pretende implementar en este trabajo.

Conviene repasar brevemente el camino que han experimentado estos dispositivos, para ser conscientes de las capacidades de los dispositivos actuales. El primero se lanzó en 1992 de la mano de IBM, y se conocía como Simon Personal Communicator [15]. Tenía una pantalla en blanco y negro, un procesador que funcionaba a 16 MHz, 1 MB de memoria RAM y 1MB de almacenamiento y permitía llamar y enviar emails. Se vendía por aproximadamente 900 €.

Ilustración 2-10. IBM Simon Personal Communicator

En 2007, Apple lanza al mercado el primer iPhone, el cual contaba con una pantalla capacitiva, conectividad y funciones multimedia, y una tienda de aplicaciones que ampliaba las posibilidades del dispositivo.

Ilustración 2-11. iPhone de primera generación

A partir de aquí, los nuevos dispositivos han ido evolucionando en características, aunque manteniendo la esencia de este dispositivo.

En 2008, Google lanza Android, un sistema operativo basado en Linux, el cual, debido a su naturaleza abierta, ha sido adoptado por un gran número de fabricantes hasta nuestros días.

En la actualidad contamos con dispositivos que cuentan con una gran conectividad, como conexión con redes 3G, 4G, 5G, Wi-Fi, Bluetooth, NFC, UWB, así como unas especificaciones que alcanzan hasta procesadores de ocho núcleos a 2,8 GHz, pantallas de resolución 4K, numerosas cámaras fotográficas de hasta 100 MPx, 1 TB de almacenamiento y 8 GB de memoria RAM.

2.7 Sistemas Operativos de dispositivos móviles inteligentes

Una vez que ha finalizado la exposición del Hardware, es el momento de comenzar con el Software, para ello, el primer punto es seleccionar sobre qué sistema operativo se va a desarrollar la aplicación.

2.7.1 Android

Este sistema operativo fue creado en 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White, los cuales se lo vendieron a Google en 2005, momento a partir del cual se aceleró su desarrollo e implantación hasta convertirse en el sistema operativo para dispositivos móviles inteligentes más utilizado del mundo.

Es un sistema en el que la interfaz gráfica está escrita en Java, mientras que el núcleo del sistema se basa en el kernel Linux, el cual ha demostrado su versatilidad para funcionar sobre diversas arquitecturas de hardware. Esto permite que el sistema operativo se pueda ejecutar sobre un amplio portfolio de hardware, lo que permite que como desarrollador, tu producto pueda llegar a un gran número de usuarios, a cambio de no ser el sistema operativo más optimizado y fluido del mercado.

En cuanto a tecnología, y tras más de una década compitiendo contra iOS, actualmente es un sistema operativo maduro, compatible con la mayoría de tecnologías utilizadas en la actualidad, como pueden ser acelerómetros, giróscopos, GPS, Wi-Fi, Bluetooth, NFC, 5G, sensores biométricos, cifrado de datos, procesadores neuronales, realidad aumentada, etc.

De la misma manera, cuenta con una serie de funcionalidades que se consideran lo estándar para un dispositivo inteligente, como puede ser una pantalla de inicio, una barra de notificaciones, tecnología de reconocimiento de voz, entre otras, aunque una de sus diferencias con respecto a iOS, es que debido al uso de la licencia Apache 2.0, Google permite que los fabricantes introduzcan modificaciones en el sistema operativo, que permiten personalizarlo y mejorarlo con nuevas funcionalidades y apariencias.

Respecto a la evolución de este sistema, el primer dispositivo que se lanzó con Android fue el HTC Dream en 2007, el cual ejecutaba la versión 1.0 de Android. Actualmente, la última versión estable es la 12, la cual se lanzó en 2021, y Google continúa con su desarrollo dado que la versión 13 se encuentra en fase beta [16], lo que unido a su implantación en otro tipo de dispositivos como las tablets, los televisores y los automóviles, junto con la integración con los servicios de Google, lo convierten en un sistema operativo muy versátil.

Ilustración 2-12. Google Pixel 6 ejecutando Android 12

Como punto de mejora respecto a iOS, el sistema sigue sufriendo de la conocida como "fragmentación", debida a que los fabricantes suelen dar soporte durante dos años, lo cual asegura la actualización de los dispositivos a una o dos versiones mayores del sistema operativo. Google ha intentando luchar contra esto introduciendo tecnologías que permiten que las diferentes partes del sistema se puedan actualizar de manera independiente, así como introduciendo su propia línea de dispositivos móviles inteligentes, actualmente conocidos como los Google Pixel.

2.7.2 iOS

Este sistema operativo fue presentado en 2007, dado que al igual que Android, la tecnología de esa época empezaba a permitir el desarrollo de dispositivos con capacidades multimedia y de conectividad avanzada en un tamaño reducido [17].

En este caso, la empresa que lo desarrolla es Apple, y de inicio se concibió para funcionar en un único tipo de dispositivo, por lo que el sistema operativo está optimizado para un hardware determinado, y en líneas generales obtiene un rendimiento y fluidez superior a Android.

Por el contrario, los únicos dispositivos compatibles con este sistema operativo son los dispositivos fabricados por Apple, por lo que debido a sus altos precios, no es un sistema operativo que haya podido penetrar en todas las gamas de mercado, por lo que aunque su cuota de mercado es importante, no alcanza los niveles de popularidad de Android.

En cuanto a funcionalidades y conectividad, al igual que Android, es un sistema operativo que a estas alturas de desarrollo se considera bastante maduro, por lo que su principal diferencia es su integración en el ecosistema de Apple.

Ilustración 2-13. iPhone 12 ejecutando iOS 15

Respecto a su evolución, históricamente se ha considerado un sistema hermético, en el que Apple era la única que decidía las aplicaciones que se podían instalar en el dispositivo, así como la apariencia de la interfaz de usuario. Con el paso de los años, las posibilidades de personalización han ido aumentando, hasta llegar a la última versión disponible, que es la 15.6.

2.8 Entornos de desarrollo para el microcontrolador ESP32

Continuando con el software, uno de los puntos a seleccionar es el entorno de desarrollo que se va utilizar para programar el microcontrolador ESP32, dado que hay varias alternativas, y como se verá más adelante ha sido el dispositivo seleccionado.

2.8.1 Arduino IDE

Este entorno de desarrollo se creó para programar las placas Arduino, pero debido a su naturaleza de código libre, con el tiempo ha ido ampliando sus funcionalidades, hasta llegar al punto en el que mediante la instalación de librerías, se puede añadir soporte para programar placas desarrolladas por otras empresas, como es el caso del microcontrolador ESP32.

Por lo tanto, supone una buena alternativa en caso de que el usuario ya esté familiarizado con este entorno de desarrollo por trabajos previos, dado que la curva de aprendizaje sobre las herramientas y funcionalidades que otorga el entorno de desarrollo es inferior a la que puede ofrecer el uso de un entorno de desarrollo específico.

Ilustración 2-14. Arduino IDE

Por otra parte, su uso habitual para la programación de pequeñas protoboards, lo hacen una opción muy atractiva para la comunidad de desarrolladores, por lo que la documentación sobre este entorno de desarrollo es abundante, y favorece que los fabricantes de sensores y de microcontroladores añadan soporte para este entorno [18].

En cuanto a sus características, además del editor de texto, el compilador, el resaltador de sintaxis y el soporte para diferentes dispositivos de hardware, también cuenta con un monitor serie, que permite testear las aplicaciones en tiempo real.

2.8.2 ESP-IDF

Este entorno de desarrollo lo ha creado la compañía Espressif, la misma que ha diseñado el microcontrolador ESP32, por lo tanto es la herramienta oficial para programarlo, dado que contiene todas las librerías de código y scripts necesarios para poder compilar el código e instalarlo en el ESP32.

El nombre significa Espressif IoT Development Framework (ESP-IDF), y consta de tres herramientas de software. El Toolchain para compilar código, las Build tools para construir la aplicación para el ESP32, y el ESP-IDF que contiene la API para manejar el Toolchain [19].

La compañía recomienda instalar el ESP-IDF como un plugin que se ejecute sobre el IDE de nuestra preferencia, y también permite ejecutarlo directamente mediante una ventana de comandos.

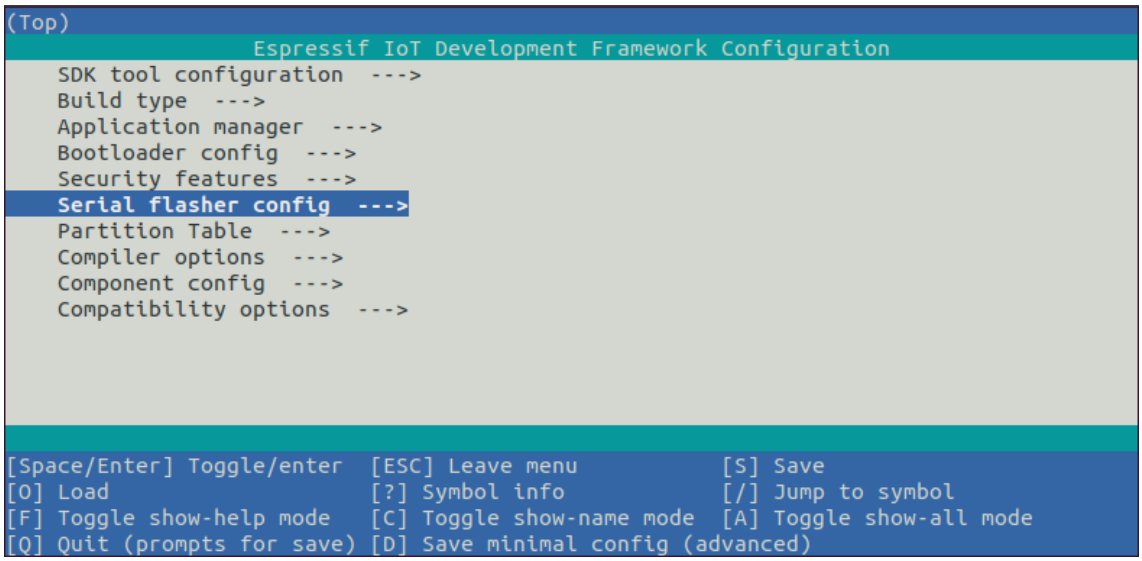

Ilustración 2-15. ESP-IDF mediante ventana de comandos

2.9 Plataformas IoT

El último punto a decidir respecto al software es la plataforma en la que se van a almacenar los datos obtenidos del sensor para su posterior visualización y análisis.

2.9.1 ThingSpeak

Esta plataforma permite añadir datos, para posteriormente visualizarlos y analizarlos mediante las numerosas herramientas que ofrece, dado que la empresa que ofrece el servicio es MathWorks, los cuales también cuentan en su portfolio con la herramienta MATLAB, por lo que una vez recopilados los datos, permiten escribir y ejecutar código de MATLAB directamente en la plataforma.

La plataforma cuenta con una API bien documentada y con numerosos ejemplos y posibilidades para actuar sobre los diferentes canales que creemos en la plataforma, por lo que es de las herramientas más potentes en cuanto a tratamiento de datos [20].

Para aprovechar todo el potencial hace falta una licencia comercial, dado que las licencias de uso gratuitas tienen limitaciones, como no cargar nuevos datos hasta que no hayan transcurrido 15 segundos desde la última carga, o limitaciones en cuanto al número de datos que se pueden almacenar y el tiempo que pueden ser almacenados, así como no crear más de 250 canales.

2.9.2 AWS IoT

Esta plataforma está desarrollada por Amazon, y al igual que ThingSpeak, cuenta con una API con la que podemos leer y escribir datos en la plataforma [21].

Su punto fuerte es la integración con el resto de servicios web de Amazon, como son el AWS IoT Device SDK, el AWS IoT Core for LoRaWAN y el AWS Command Line Interface. Aunque más que ofrecer una potente herramienta de análisis de datos, cuenta con la capacidad de interconectar los datos con herramientas externas al propio AWS, por lo que la integración no es tan transparente como en el caso de ThingSpeak.

A su vez, también cuenta con limitaciones dependiendo del tipo de suscripción, como pueden ser el límite al tamaño de los JSON, o el número de conexiones a la plataforma.

2.10 Conclusiones

Tras haber presentado las distintas alternativas que se pueden encontrar en el mercado para llevar a cabo este Trabajo, se ha hecho una selección con los elementos más adecuados para llevar a cabo este diseño:

- LSM9DS1: este es el sensor elegido para obtener la aceleración del cuerpo, debido a su versatilidad, a su bajo precio, y a su amplia documentación, lo que permite emplear librerías que facilitan la programación del sensor. Además del acelerómetro, también dispone de giróscopo y magnetómetro, en caso de que durante el desarrollo del trabajo se observe la necesidad de utilizar dichas funciones.

- ESP32: este es el microcontrolador seleccionado para leer los datos enviados por el acelerómetro, y de transmitirlos por Bluetooth de baja energía a un dispositivo móvil inteligente, debido a que cumple con los requisitos de conectividad necesarios para leer los datos del sensor y para enviarlos a un dispositivo móvil inteligente, además de poseer un coste razonable.

- Bluetooth Low Energy: esta es la tecnología de comunicación seleccionada para interconectar el microcontrolador con el dispositivo móvil inteligente, ya que funciona en una distancia reducida, lo cual permite por un lado que el microcontrolador consuma poca energía, y por otra parte evita problemas de privacidad, dado que la información no se emite a una distancia tan grande que permita ataques Man in the middle.

- Smartphone con Android: este es el dispositivo y el sistema operativo utilizados para desarrollar la aplicación con la que el dispositivo móvil inteligente podrá comunicarse con el microcontrolador, dado que es un sistema que cuenta con bastante documentación, y está muy extendido en el mercado, por lo que se garantiza que la aplicación sea compatible con más dispositivos.

- Entorno de desarrollo Arduino IDE: se ha seleccionado este entorno de desarrollo porque cuenta con abundante documentación, y al ser un entorno utilizado para programar una amplia variedad de dispositivos, resulta familiar y no requiere la curva de aprendizaje que supone el entorno de desarrollo oficial de la empresa que fabrica el microcontrolador.

- ThingSpeak: es la plataforma seleccionada para almacenar los datos recopilados por el sensor, ya que debido a su integración con MATLAB, permite que posteriormente los datos puedan ser sometidos a números análisis matemáticos, big data, así como inteligencia artificial para detectar patrones.

Capítulo 3

Descripción del Sistema

3.1 Introducción

Tras repasar las diferentes opciones disponibles en el mercado para construir la arquitectura del trabajo, es el momento de profundizar sobre las alternativas seleccionadas. Para ello, en primer lugar se explicará en qué consistirá la arquitectura definitiva para este proyecto. Posteriormente, se mostrará el proceso de puesta a punto de los entornos de desarrollo para programar el microcontrolador y el dispositivo móvil inteligente, así como de la plataforma de recogida y análisis de datos.

3.2 Descripción de los elementos del dispositivo registrador de movimiento

El propósito del trabajo es construir un dispositivo que registre el movimiento y que envíe esa información de manera inalámbrica a un dispositivo móvil inteligente, que será el encargado de mostrarle la información al usuario, de almacenarla en una plataforma online, y de permitir que el usuario configure la sensibilidad del sensor.

Por lo tanto, las etapas y componentes del proceso se muestran en la siguiente figura:

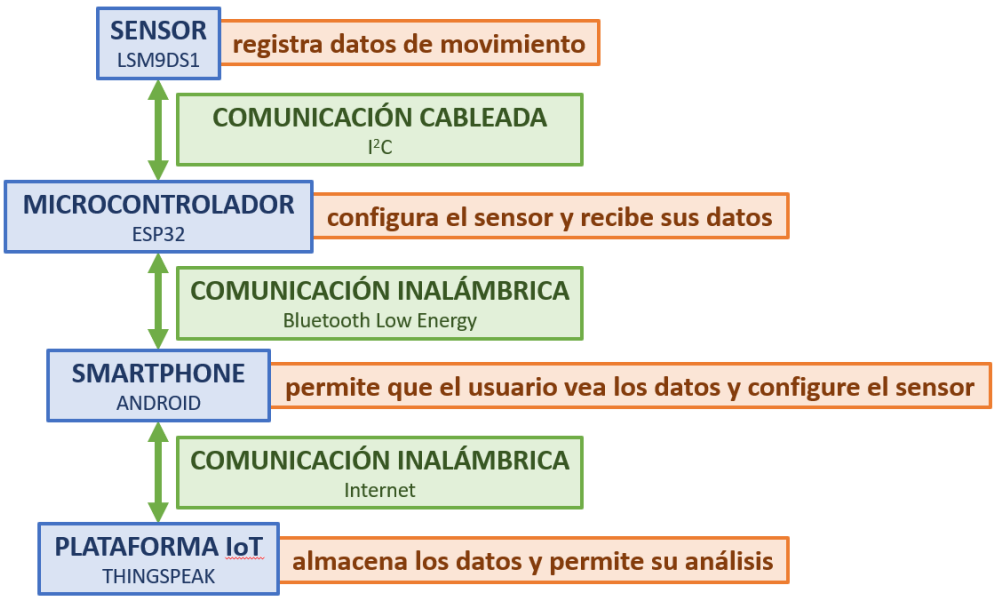

Ilustración 3-1. Flujograma del dispositivo registrador de movimiento

3.3 Desarrollo de aplicaciones para ESP32

Una vez que se dispone del ESP32, para programarlo hay que conectarlo al ordenador mediante un cable USB, pero antes hay que disponer del entorno de desarrollo correctamente configurado.

3.3.1 Instalación de Arduino IDE

En primer lugar hay que instalar el entorno de desarrollo Arduino IDE, el cual se puede descargar desde: https://www.arduino.cc/en/software

En este caso se ha seleccionado la versión para Windows [22].

Ilustración 3-2. Página web de descarga de Arduino IDE

Tras descargar el instalador, se instala el programa, aceptando la instalación del controlador de Adafruit para los puertos y del driver de Arduino para la interfaz USB.

3.3.2 Instalación en Arduino IDE de la tarjeta para comunicarse con ESP32

Tras instalar el Arduino IDE, lo iniciamos, y el siguiente paso es instalar la placa que nos permitirá programar el ESP32.

Para ello, abrimos las preferencias como se muestra en la siguiente figura, desde el menú Archivo del IDE:

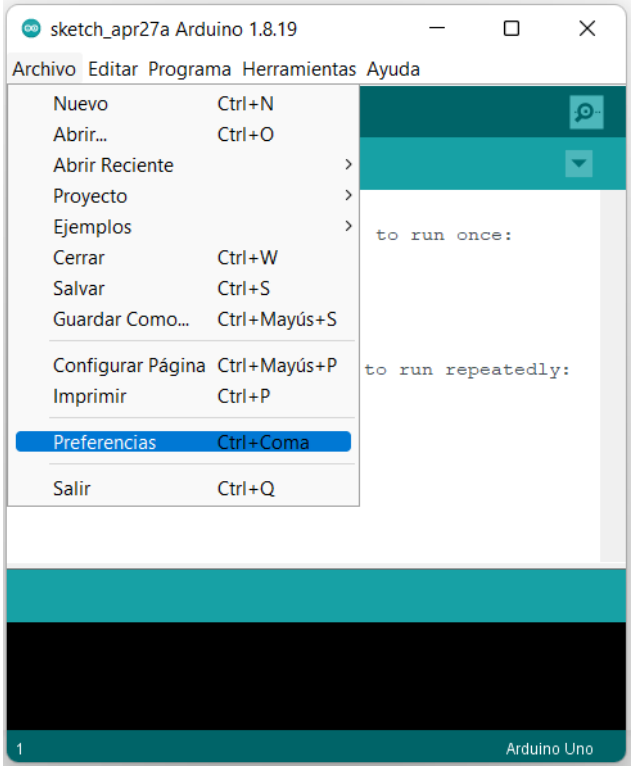

Ilustración 3-3. Preferencias de Arduino IDE

Nos dirigimos al Gestor de URLs Adicionales de Tarjetas e introducimos la siguiente dirección: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

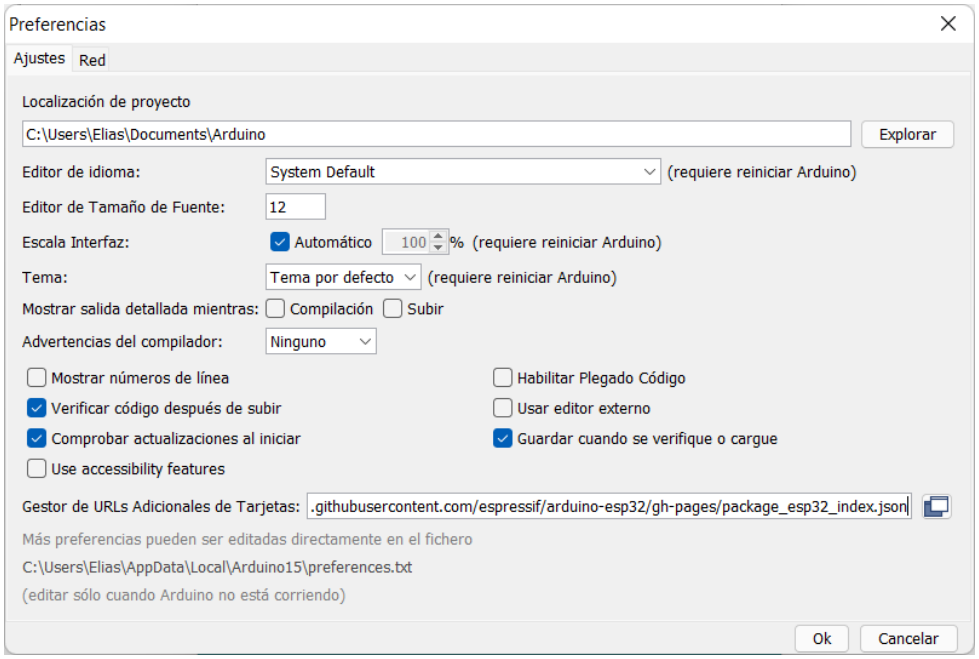

Ilustración 3-4. Gestor de URLs Adicionales de Tarjetas

Esto permitirá que el Board Manager de Arduino tenga acceso al repositorio de la empresa Espressif, desde el que podrá instalarse la Board que necesitamos para programar el ESP32.

Por lo tanto, ahora procedemos a abrir el Gestor de tarjetas desde las Herramientas del IDE.

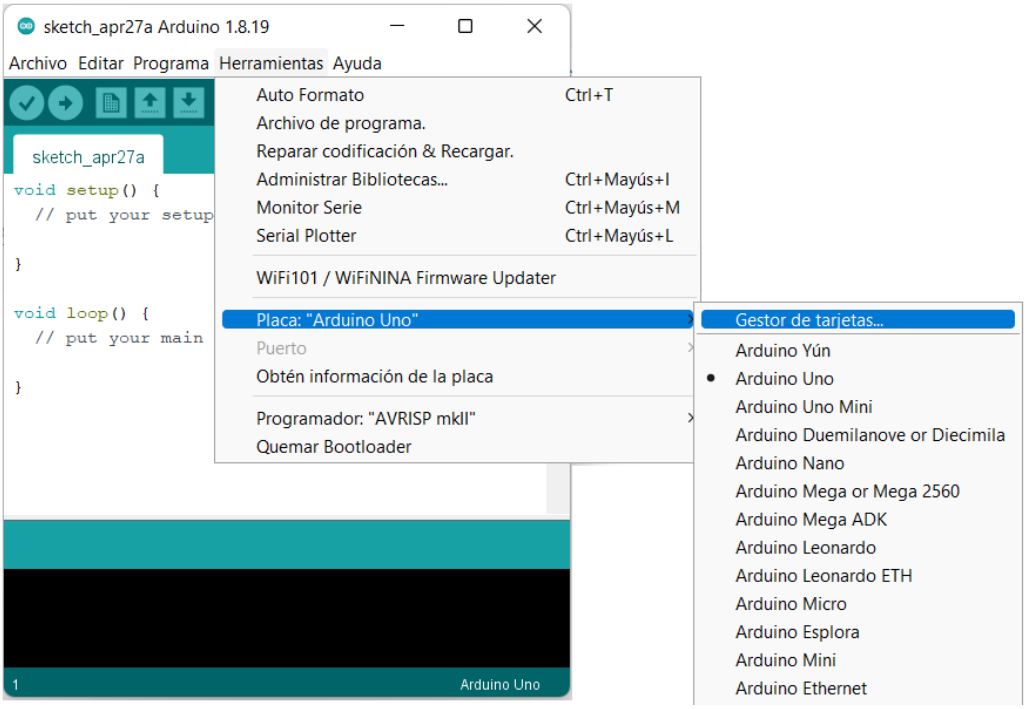

Ilustración 3-5. Acceso al Gestor de tarjetas

Buscamos "esp32" e instalamos la tarjeta, como se muestra en la siguiente figura.

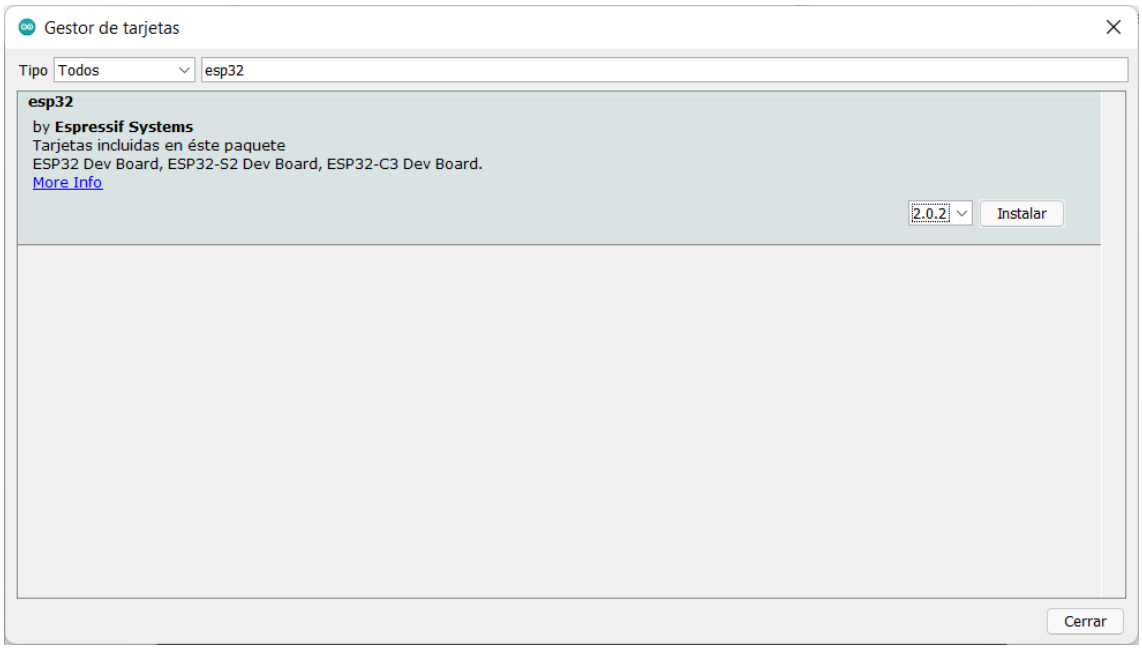

Ilustración 3-6. Gestor de tarjetas de Arduino IDE

Por último, reiniciamos Arduino IDE, y procedemos a seleccionar la Placa "ESP32 Dev Module" y el Puerto "COM5" en las Herramientas del IDE.

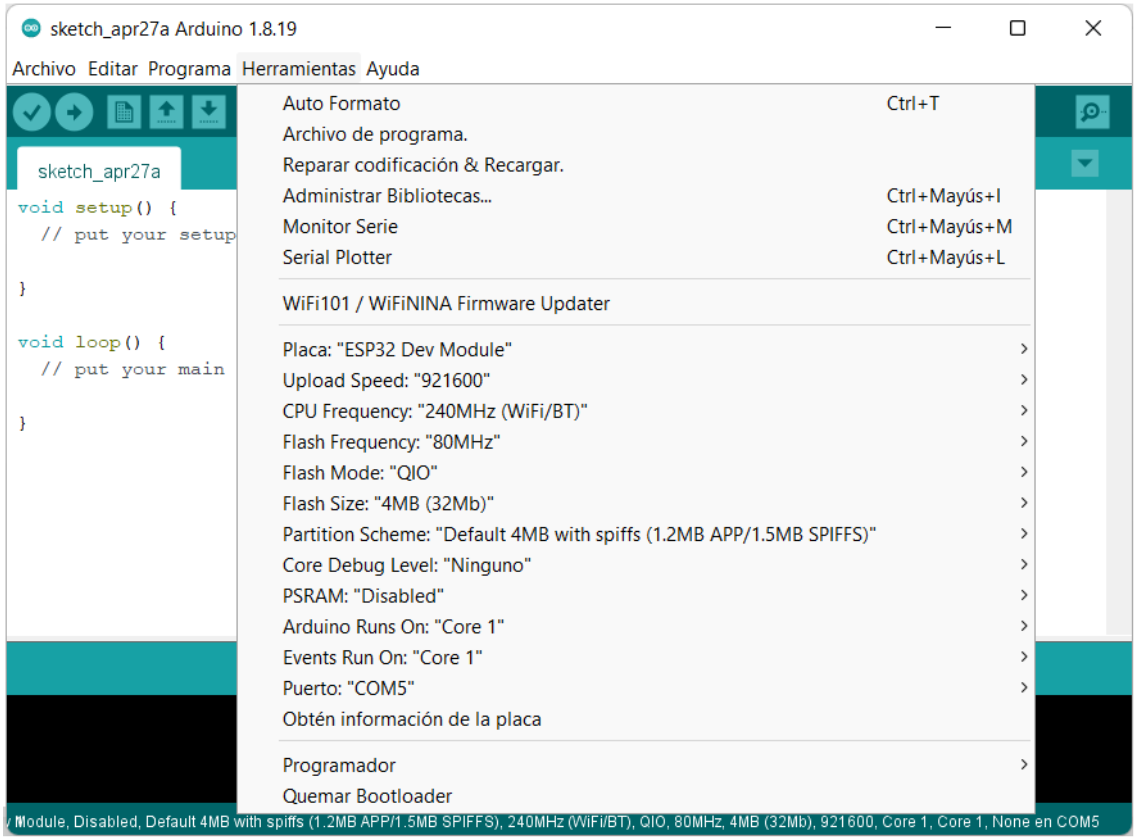

Ilustración 3-7. Selección de la placa y del puerto en Arduino IDE

3.3.3 Instalación en Arduino IDE de la librería para comunicarse con el sensor LSM9DS1

A continuación hay que instalar la librería que permita que el ESP32 se pueda comunicar con el sensor LSM9DS1. Para ello, desde las Herramientas del IDE abrimos el Administrador de Bibliotecas como se muestra en la siguiente figura.

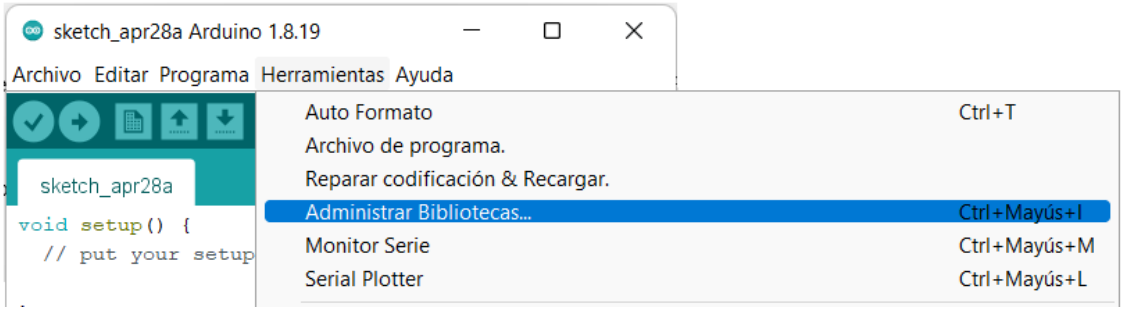

Ilustración 3-8. Acceso a Administrar Bibliotecas del Arduino IDE

Una vez dentro del Gestor de Librerías, introducimos "Adafruit LSM9DS1" e instalamos la librería, la cual solicitará que se instalen una serie de dependencias.

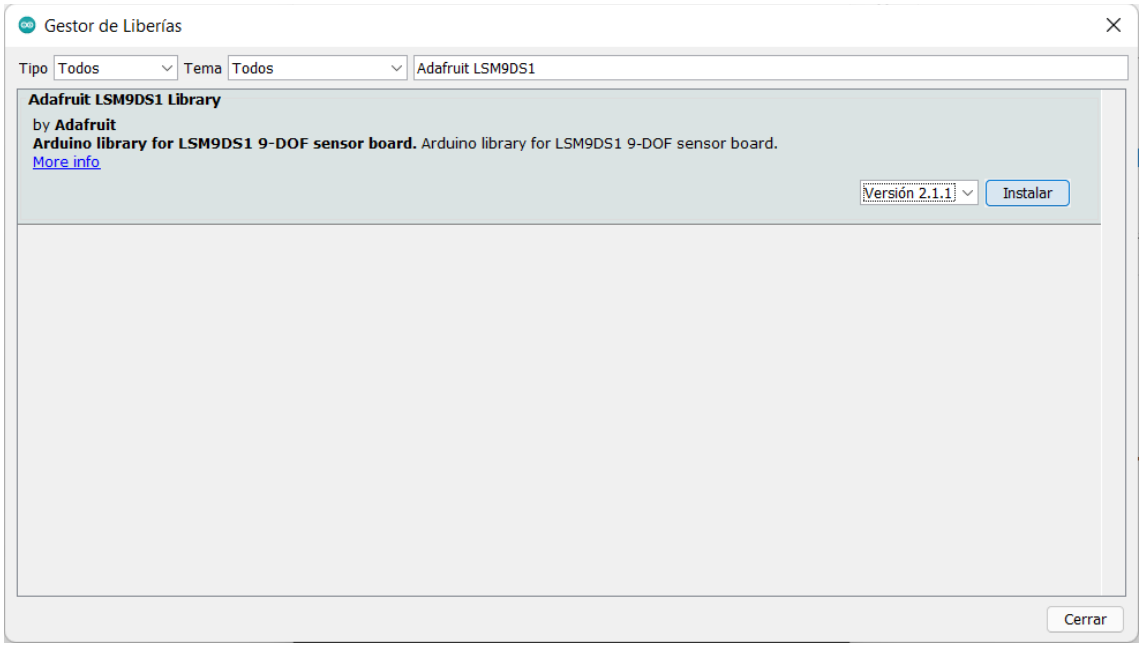

Ilustración 3-9. Gestor de Librerías de Arduino IDE

3.3.4 Creación de un proyecto en Arduino IDE

Una vez configurado el Arduino IDE, hay que crear un nuevo proyecto desde Archivo > Nuevo.

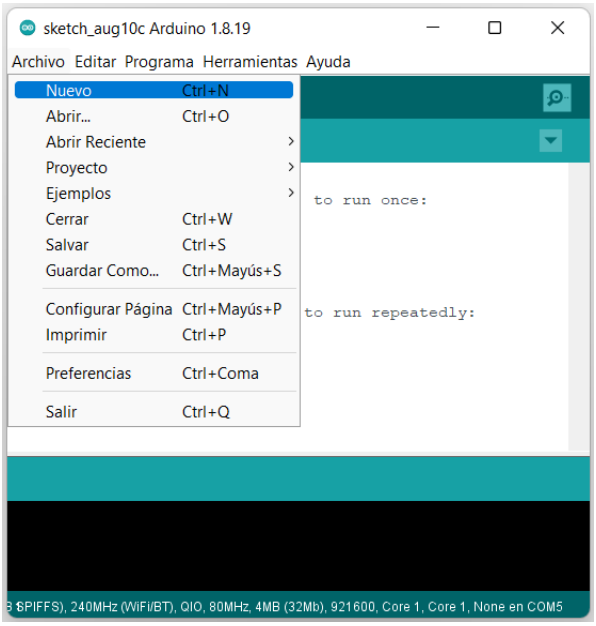

Ilustración 3-10. Creación de un proyecto en Arduino IDE

En la vista central está el editor de texto en el que hay que programar. Dentro hay una función setup() y una función loop(). En la función setup se introduce la programación inicial del microcontrolador, mientras que en la función loop se introduce la parte del programa que se estará ejecutando en bucle.

3.3.5 Programación del ESP32 desde Arduino IDE

Una vez que se ha realizado el programa, hay que pulsar en Programa > "Verificar" para comprobar que no hay errores de compilación.

Si todo ha salido correctamente, procedemos a conectar el ESP32 al ordenador mediante el cable USB, y pulsamos en Programa > "Subir", tras lo cual el programa se compilará y cargará en el ESP32.

En caso de que aparezca algún error, el panel inferior de Arduino IDE nos indicará la línea de código errónea o el error que se haya producido al intentar cargarlo en el ESP32.

3.4 Desarrollo de aplicaciones para Android

El siguiente elemento de software que hay que preparar es el entorno de desarrollo de aplicaciones para dispositivos Android.

3.4.1 Instalación de Java SDK

Hay que comenzar instalando el Java Software Development Kit (Java SDK), el cual proporciona las herramientas necesarias para compilar aplicaciones que se ejecutan en dispositivos compatibles con java, como es el caso de Android, dado que hay que transformar los archivos .java en bytecode [23].

En la web https://www.oracle.com/java/technologies/downloads/#jdk18-windows se puede descargar la versión correspondiente al sistema operativo utilizado, en este caso Windows.

Tras descargar el archivo ejecutable, se instala aceptando los términos de uso, y tras eso se puede continuar con la instalación del resto de componentes necesarios para programar aplicaciones para Android.

3.4.2 Instalación de Android Studio

A continuación hay que instalar la herramienta oficial que proporciona Google para programar aplicaciones para Android, a la cual se la denomina Android Studio [24].

Para ello, hay que dirigirse a la página de los desarrolladores de Android https://developer.android.com/studio y descargarse la versión correspondiente, en este caso la de Windows.

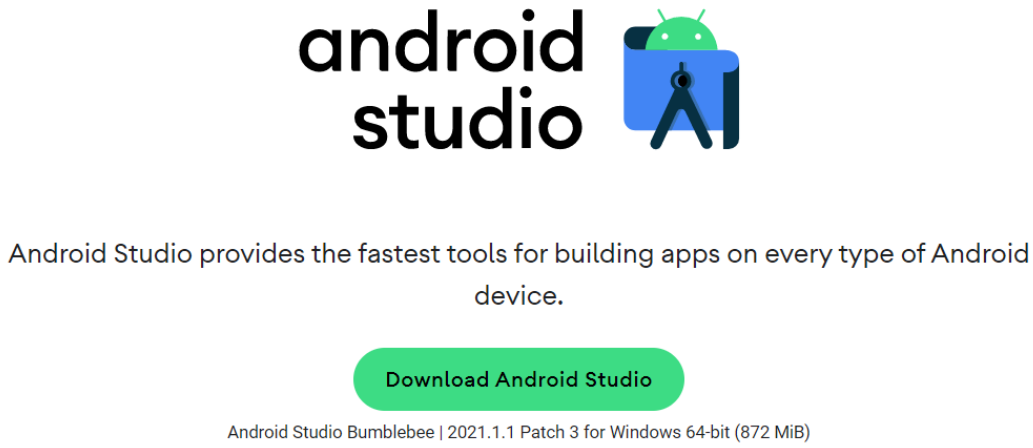

Ilustración 3-11. Página de descarga de Android Studio

Se aceptan los términos y condiciones y se procede a descargar el archivo ejecutable, el cual se abre una vez descargado, para comenzar la instalación guiada por el asistente.

En la instalación el asistente preguntará sobre los componentes a instalar, viniendo seleccionado por defecto Android Studio. A continuación se selecciona la ubicación en la que instalar el programa, y tras eso el programa se habrá instalado y se podrá iniciar Android Studio.

3.4.3 Configuración de Android Studio

Una vez instalado Android Studio, el propio asistente nos preguntará si queremos importar los ajustes desde alguna versión anterior de Android Studio.

En caso de ser una instalación limpia tendremos que elegir estos parámetros por primera vez. Para ello, en la siguiente ventana elegimos la realización de una instalación personalizada.

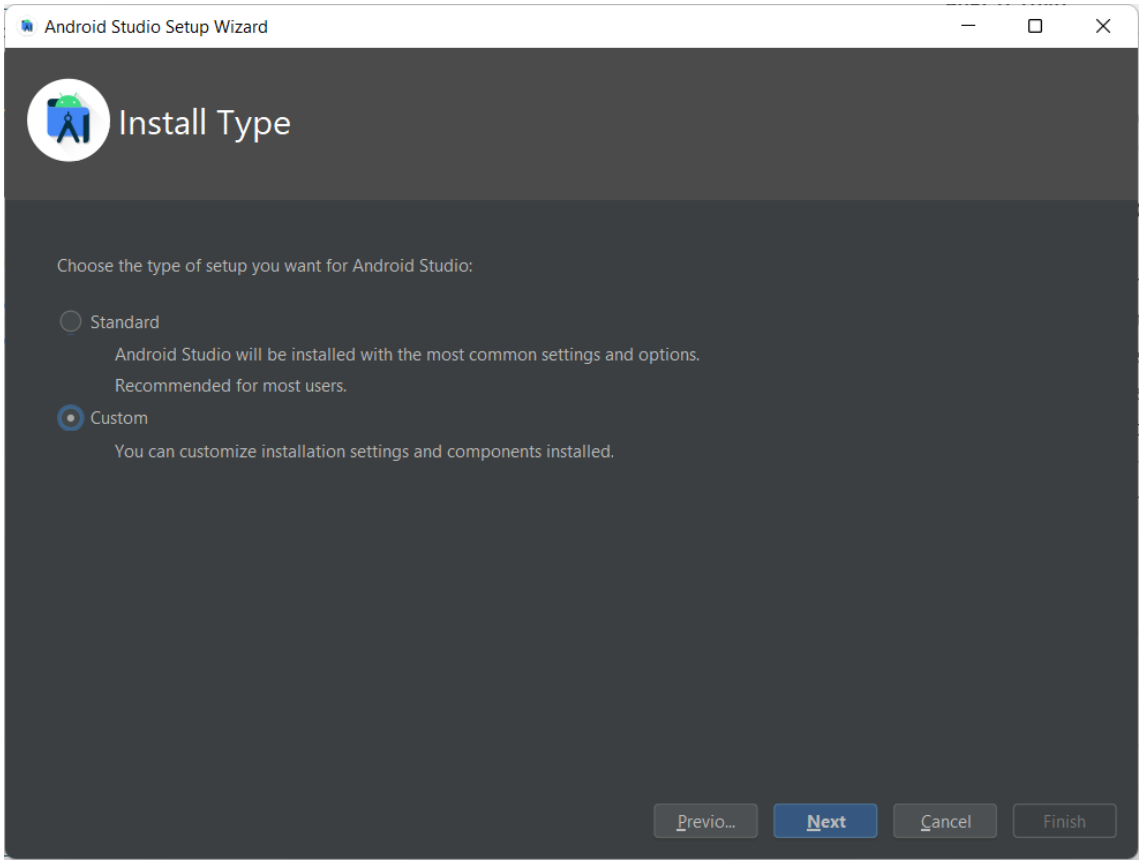

Ilustración 3-12. Selección del tipo de ajustes de Android Studio

Después seleccionamos la carpeta en la que se encuentra JDK

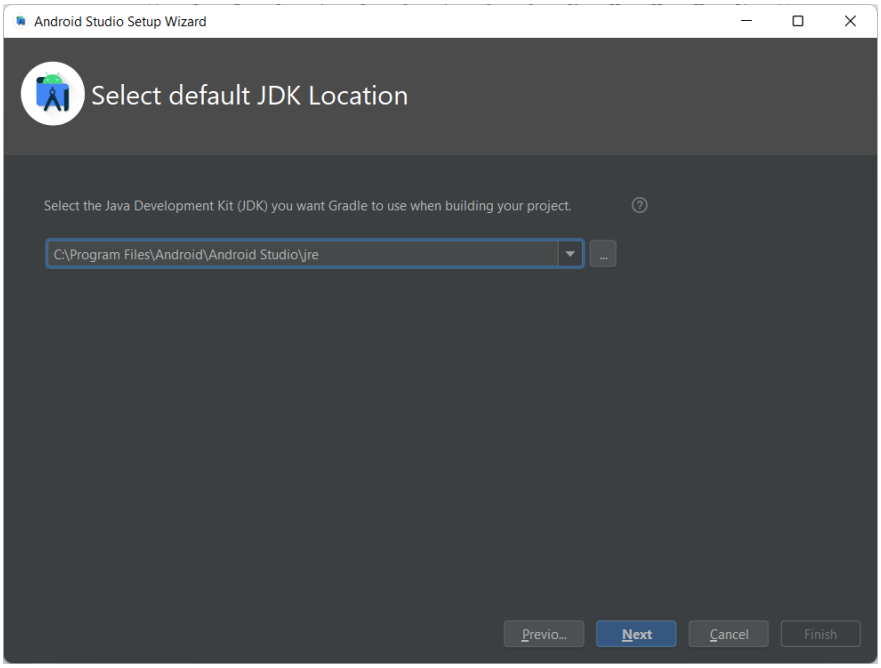

Ilustración 3-13. Selección de la carpeta donde se ubica JDK

Tras esto seleccionamos algunos ajustes estéticos como son el tema claro o el oscuro:

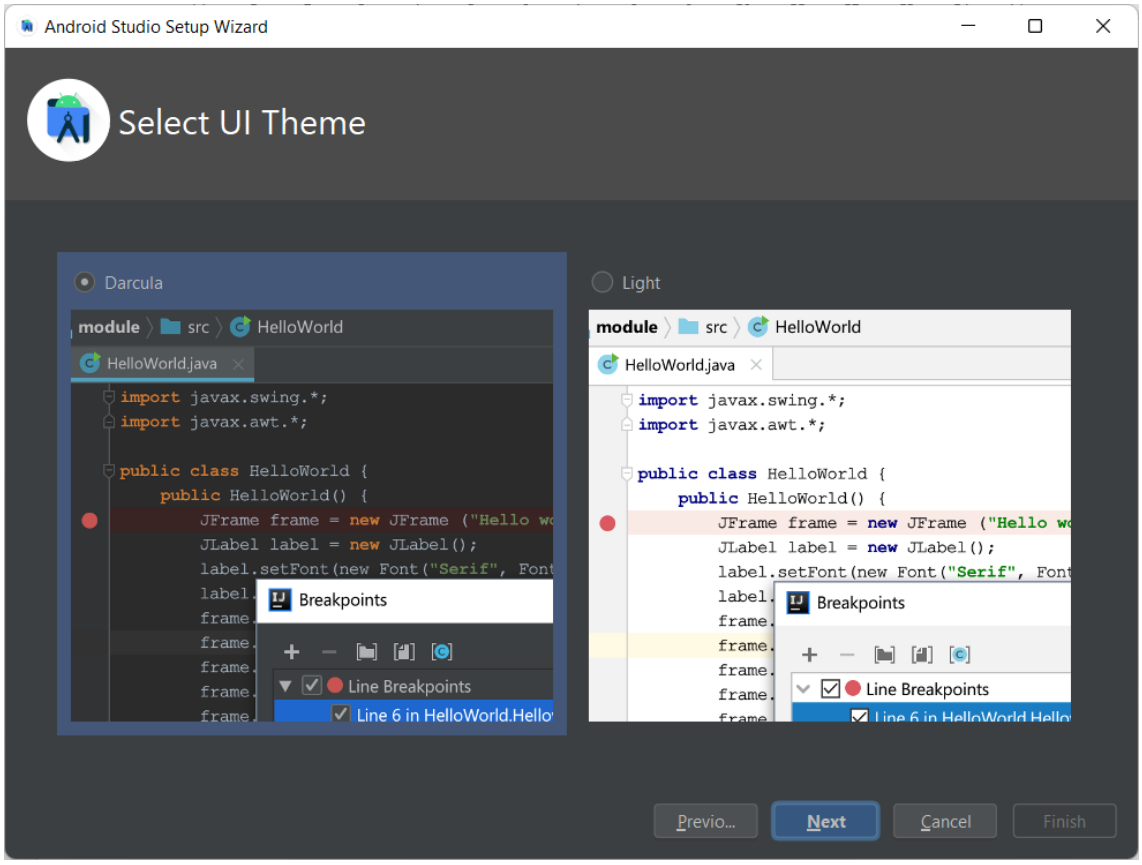

Ilustración 3-14. Selección del tema de Android Studio

El siguiente paso es seleccionar los componentes del SDK que queremos instalar. El SDK es el Software Development Kit, el cual consiste en una serie de herramientas, APIs y utilidades que permiten compilar y testear una aplicación para una versión determinada de Android, dado que cada versión de Android tiene su propio nivel de API.

Actualmente la última versión de Android es la 12, por lo que el asistente por defecto nos instala el SDK correspondiente a la versión 32 de la API.

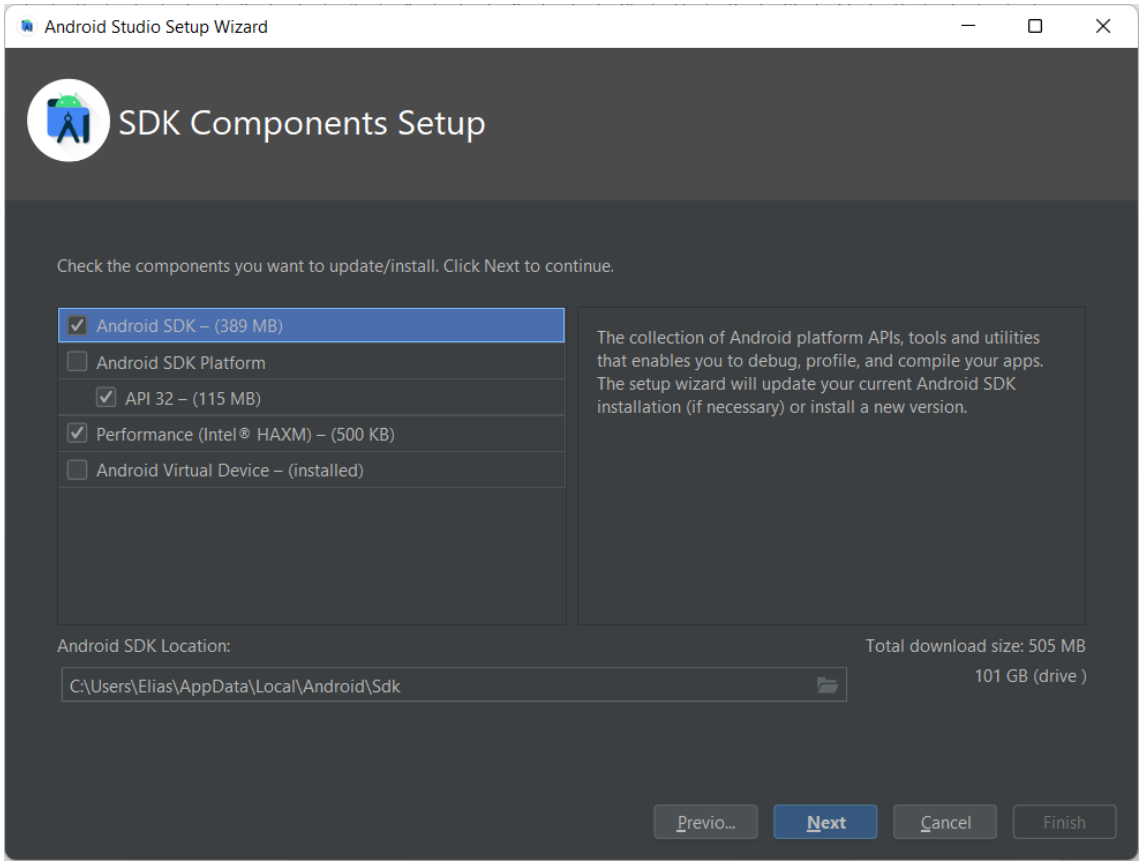

Ilustración 3-15. Instalación del SDK en Android Studio

En caso de que el procesador de nuestro ordenador sea Intel y cuente con la tecnología Hardware Accelerated Execution Manager (HAXM), el asistente nos preguntará sobre la memoria RAM que queremos asignar al emulador de Android, el cual nos permitirá testear nuestra aplicación sin necesidad de disponer de un dispositivo físico.

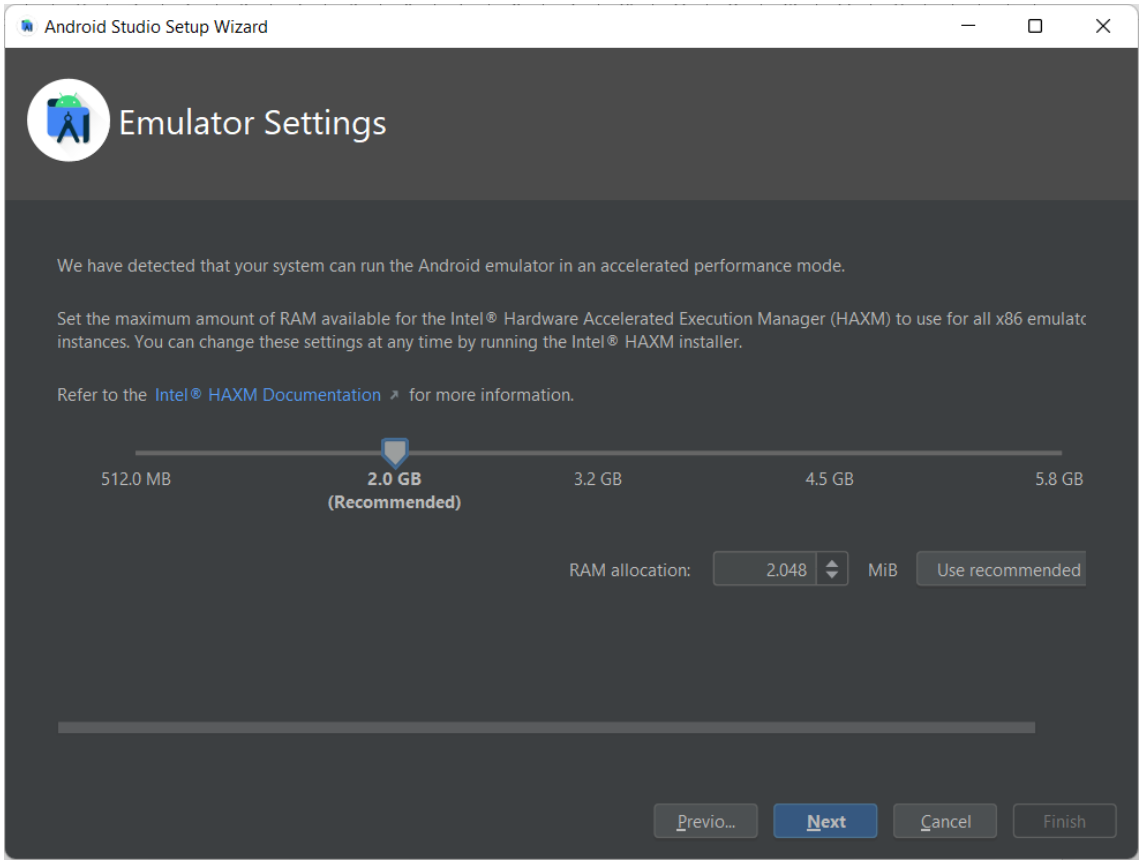

Ilustración 3-16. Instalación del SDK en Android Studio

Por último, el asistente nos muestra el resumen de las opciones y componentes que hemos seleccionado antes de que aceptemos el acuerdo de licencia y los componentes comiencen a descargarse y a instalarse.

Una vez finalizada la instalación, hay que hacer un último paso antes de empezar a programar, ya que hay que seleccionar el SDK mínimo (la versión de Android mínima) con la que la aplicación de Android que se programe será compatible. Para eso, desde la ventana inicial de Android Studio, se selecciona el SDK Manager como se indica en la siguiente figura.

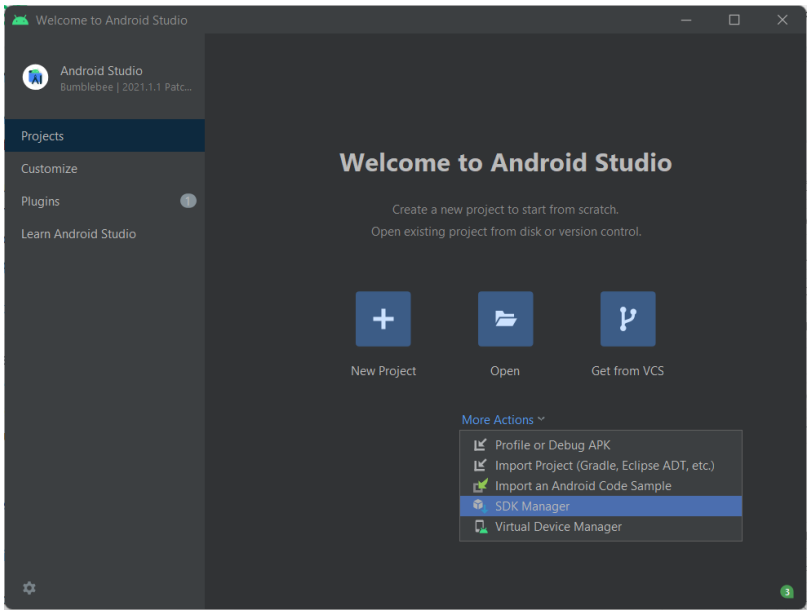

Ilustración 3-17. Apertura de SDK Manager en Android Studio

Una vez dentro, hay que irse a la pestaña de SDK Platforms y seleccionar el nivel de API mínimo para el que irá dirigida la aplicación, que en este caso será para Android 11, por lo que el SDK a instalar será el nivel 30, y habrá que seleccionar tanto el SDK como la imagen de Intel de dicho nivel de API, lo que ayudará a que el emulador se comporte de una manera más fluida.

Para que estos componentes se muestren hay que seleccionar la opción "Show Package Details". Por último se pulsa en "Aplicar", lo que hace que la descarga de los componentes de comienzo.

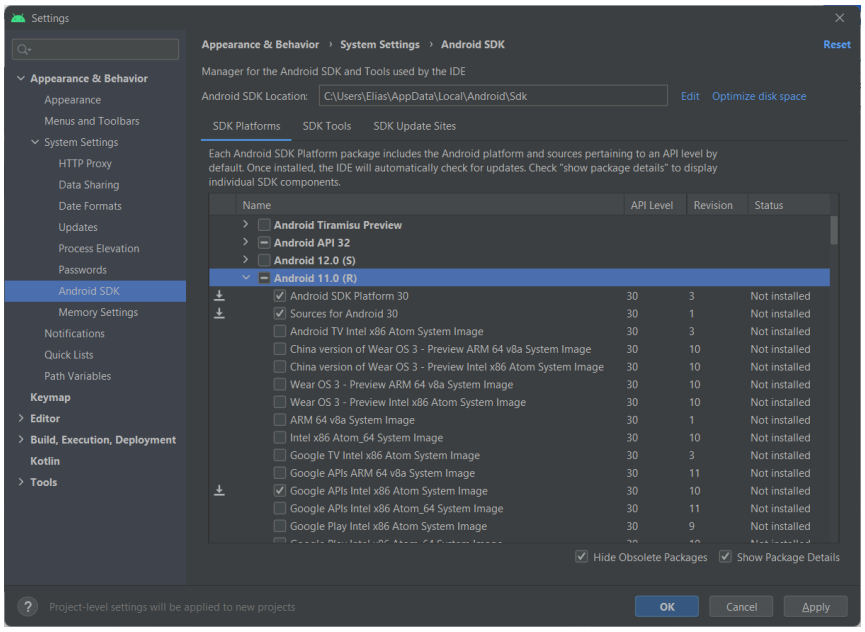

Ilustración 3-18. Selección de los componentes del SDK a instalar

3.4.4 Activar la depuración del dispositivo con Android

Con vistas a que la programación de la aplicación sea más sencilla, es altamente recomendable utilizar un dispositivo móvil inteligente (en adelante smartphone) que ejecute Android, para que Android Studio pueda ejecutar la aplicación de una manera más fluida y realista que con el emulador.

Para ello hay que activar las opciones de desarrollo, las cuales por defecto están ocultas para que los usuarios no modifiquen el sistema indebidamente. Para mostrarlas hay que ir a los ajustes del smartphone, bajar hasta el apartado de "Acerca del teléfono" y pulsar 7 veces en "Número de compilación". Esto habilitará las opciones de desarrollo.

Tras esto, hay que volver a los ajustes, entrar en el apartado "Sistema", y dentro habrá una opción llamada "Opciones de desarrollo", la cual habrá que activar. A su vez, dentro hay que desplazarse hasta el apartado "Depuración por USB" y activarla para que el smartphone pueda usarse para depurar la aplicación cuando se conecte al ordenador mediante el cable USB.

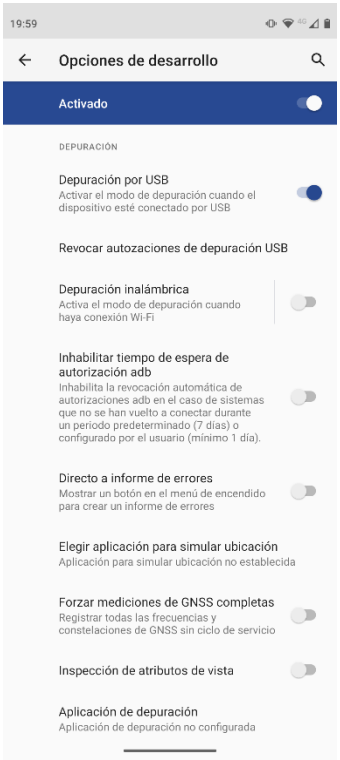

Ilustración 3-19. Activación de la depuración por USB en un smartphone

3.4.5 Creación de un proyecto en Android Studio

Una vez que se ha abierto Android Studio, hay que pulsar sobre el icono "New Project" que aparece en la ventana principal. Tras esto, el asistente nos sugiere una serie de actividades como base de nuestro proyecto, lo que puede ahorrarnos algo de tiempo, al incluirnos en el proyecto los archivos necesarios para el tipo de actividad que se haya seleccionado. Por defecto se puede seleccionar la "Empty Activity".

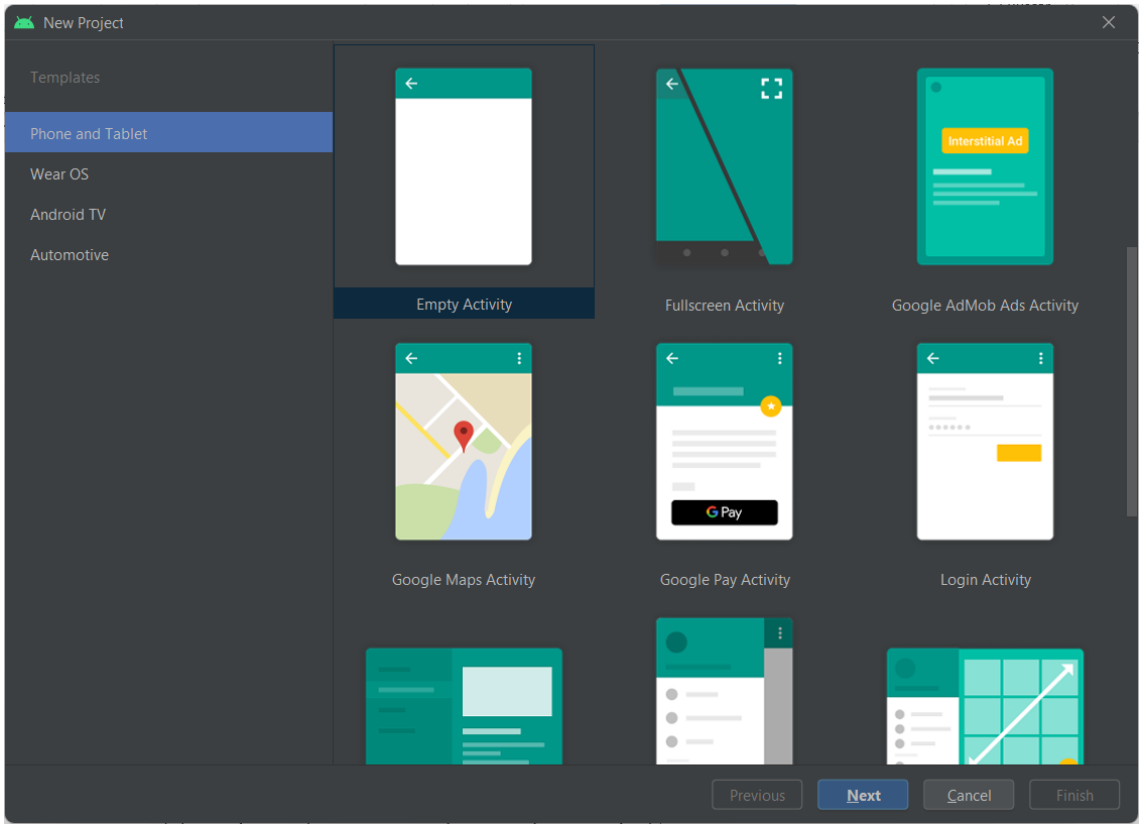

Ilustración 3-20. Selección de la Activity con la que se creará el proyecto

Tras esto, el asistente nos requiere una serie de datos que definirán las características de nuestra aplicación, como son el nombre de la misma, el nombre que tendrá el empaquetado, la carpeta del ordenador en la que se guardarán los archivos del proyecto, el lenguaje de programación a utilizar, el cual puede ser Java o Kotlin, y una de las más importantes, el SDK mínimo que utilizará la aplicación.

El SDK mínimo es importante porque establece el nivel mínimo de API que utilizará la aplicación, y por lo tanto determina dos cosas. En primer lugar, los dispositivos que ejecuten una versión de Android cuya API sea anterior a la que nosotros hayamos seleccionado, no podrán ejecutar la aplicación, por lo que si queremos que la aplicación llegue a un gran número de usuarios, hay que intentar que la API sea lo más antigua posible. Pero en segundo lugar, puede ser que la tecnología que necesita utilizar la

aplicación sólo se haya implementado en las versiones más recientes de la API, por lo que si se utiliza una API muy antigua, puede ser que la aplicación no sea capaz de realizar la función que se pretende.

Aquí entra en juego uno de los principales problemas de Android, la fragmentación, ya que programar para la versión más reciente, la 12, que fue lanzada a finales de 2021, cuya API corresponde a la versión 31, sólo permitiría que la aplicación se pueda ejecutar en el 13,5% de los dispositivos con Android existentes.

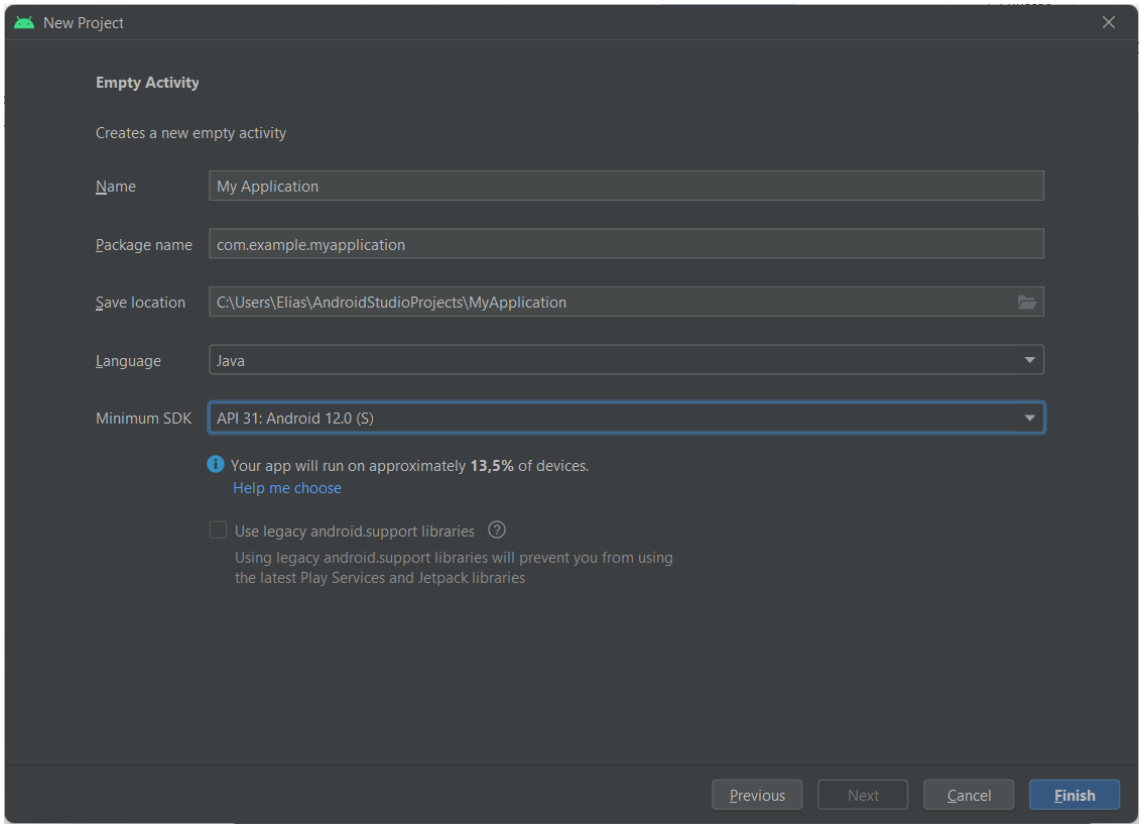

Ilustración 3-21. Selección de los parámetros de la aplicación

Solo queda pulsar en "Finish", y tras ese paso se creará la estructura del proyecto y se podrá comenzar a programar.

3.4.6 Programación del dispositivo inteligente desde Android Studio

Tras haber realizado la programación de la aplicación, la forma de probarla en el dispositivo móvil inteligente consiste en conectar el dispositivo mediante el cable USB al ordenador. Tras esto, Android Studio lo detectará y lo mostrará en la parte superior.

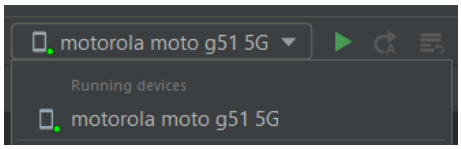

Ilustración 3-22. Smartphone conectado por USB al ordenador.

Una vez que el dispositivo ha sido detectado, hay que pulsar el icono "Run 'app'" dentro del menú "Run". Android Studio compilará el código y generará una aplicación que se instalará automáticamente en el dispositivo móvil inteligente, tras lo cual será ejecutada.

Durante la ejecución, gracias al uso del dispositivo, podemos depurar la aplicación en tiempo real, observando el panel "Logcat" y "App Inspection".

3.5 Utilización de la plataforma de IoT ThingSpeak

El uso de esta plataforma reside en crear los canales necesarios, y dentro de esos canales crear los campos necesarios que almacenarán los datos que ha registrado el sensor.

Por lo tanto, una vez que se ha creado una cuenta de usuario y que nos encontramos en la página principal de la plataforma, hay que dirigirse a los Channels del usuario como se muestra en la siguiente ilustración.

Ilustración 3-23. Página principal de ThingSpeak.

Una vez dentro de los canales del usuario, hay que pulsar en "New Channel", y tras eso aparecerá la página que permite especificar los datos y los campos que tendrá el canal, los cuales se explicarán más adelante durante la explicación del proceso seguido para crear el canal que se ha utilizado en el trabajo.

Tras crear el canal y haber accedido a él desde la propia página web, aparecerán una serie de pestañas que nos darán acceso a los diferentes aspectos que se pueden configurar del canal.

Las tres pestañas más importantes son por un lado la "Private View", la cual proporciona acceso a los datos que se han subido al canal, y permite configurar la apariencia de los mismos. Por otro lado, desde la pestaña "Channel Settings" se pueden modificar los datos del canal, por si hace falta modificar algún campo de los definidos durante la creación del canal. Por último, desde API Keys se tiene acceso a las llaves necesarias para poder leer y escribir en el canal, por lo que estas llaves se utilizarán más adelante durante la programación del trabajo.

3.6 Conclusiones

Una vez que se han comentado las diferentes aplicaciones que se van a utilizar para realizar el trabajo, el proceso se puede sintetizar de la siguiente manera.

En primer lugar, el sensor LSM9DS1 se situará en el cuerpo del usuario, y tras registrar la aceleración, la enviará mediante I^2C al microcontrolador ESP32, el cual se habrá programado con el entorno de desarrollo Arduino IDE, y podrá recibirla y enviarla mediante Bluetooth Low Energy al smartphone con Android, donde podrá ser mostrada al usuario mediante la aplicación desarrollada en Android Studio. Esta aplicación también permitirá que el usuario pueda configurar la sensibilidad del sensor, y a su vez enviará mediante internet los datos registrados a la plataforma de IoT ThingSpeak, en la que posteriormente podrá realizarse un análisis de los datos.

Capítulo 4

Descripción del Hardware y Software desarrollado

4.1 Introducción

Una vez que se han explicado los diferentes elementos que integran el sistema que se pretende desarrollar, es el momento de comenzar a construirlo y a programarlo.

Primero se explicará cómo realizar la conexión física entre el sensor y el microcontrolador.

A continuación se comentará el programa realizado para el microcontrolador, que hace posible que el microcontrolador lea los datos del sensor, configure su sensibilidad, y a su vez se pueda comunicar con el dispositivo móvil inteligente mediante el uso de Bluetooth Low Energy.

Lo siguiente será describir el programa realizado para el dispositivo móvil inteligente Android, que debe conectarse al microcontrolador mediante Bluetooth Low Energy, mostrar al usuario los datos que reciba del microcontrolador, enviarle la sensibilidad requerida por el usuario al microcontrolador, y enviar los datos a la plataforma ThingSpeak.

Finalmente, se mostrará el canal que se ha creado en ThingSpeak para que la aplicación de Android pueda enviarle los datos recopilados por el sensor.

4.2 Descripción de la arquitectura Hardware

4.2.1 Conexión entre el sensor LSM9DS1 y el ESP32

A nivel de hardware, una vez que se dispone del sensor LSM9DS1 y del microcontrolador ESP32, hay que conectarlos para que puedan comunicarse entre sí.

Para ello, la alimentación del sensor puede realizarse conectando su pin VIN al pin VCC del ESP32, y el circuito se cierra mediante la conexión del pin GND del sensor con el pin GND del ESP32.

Con esto el sensor ya estaría alimentado, por lo que ahora falta realizar la conexión que permita transferir datos entre ambos. Para ello, se ha optado por el protocolo I^2C, por lo que conectando el pin SCL del sensor con el pin IO22 del ESP32 se consigue fijar la señal de reloj entre ambos, y ya sólo quedaría conectar el pin SDA del sensor con el pin IO21 del ESP32, el cual se usará para transferir los datos.

Por simplicidad, el esquema de conexión se muestra en la siguiente figura.

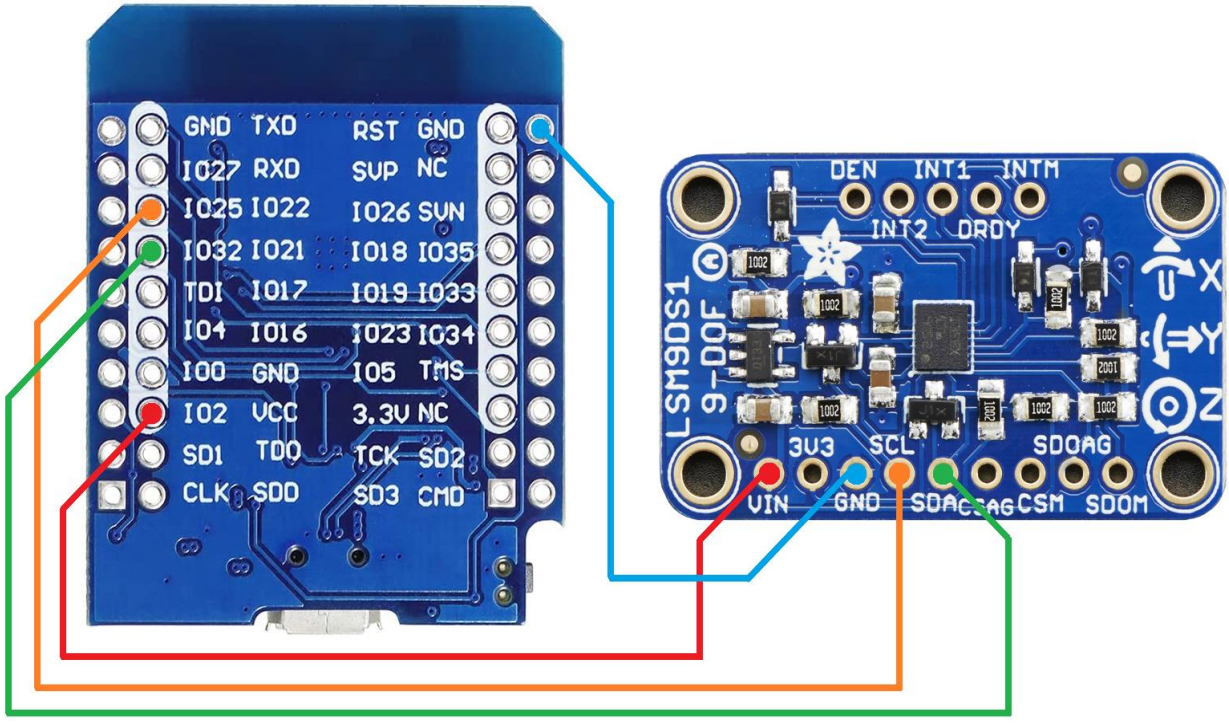

Ilustración 4-1. Esquema de conexión entre LSM9DS1 y ESP32

Por lo que una vez que se realiza la conexión, el resultado se puede observar en la siguiente figura.

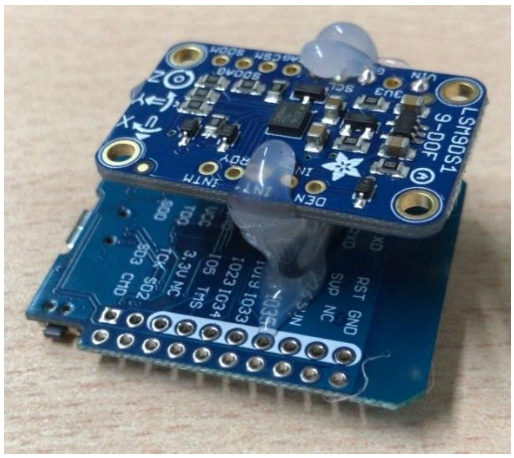

Ilustración 4-2. Conexión entre LSM9DS1 y ESP32

4.2.2 Integración en un guante

Dado que una posible aplicación de este trabajo es utilizarlo para monitorizar trastornos del movimiento, conviene que el dispositivo sea lo más transparente posible para el usuario. Para ello, se ha planteado su colocación en la muñeca mediante un guante. A su vez, el ESP32 ha tenido que alimentarse desde una batería externa basada en un Power Bank.

El montaje definitivo quedaría como se refleja en la siguiente figura.

Ilustración 4-3. ESP32 y LSM9DS1 instalados en un guante

4.3 Descripción del programa implementado en el ESP32

Para facilitar la comprensión del código desarrollado, en la siguiente figura se observa un flujograma simplificado del funcionamiento del programa.

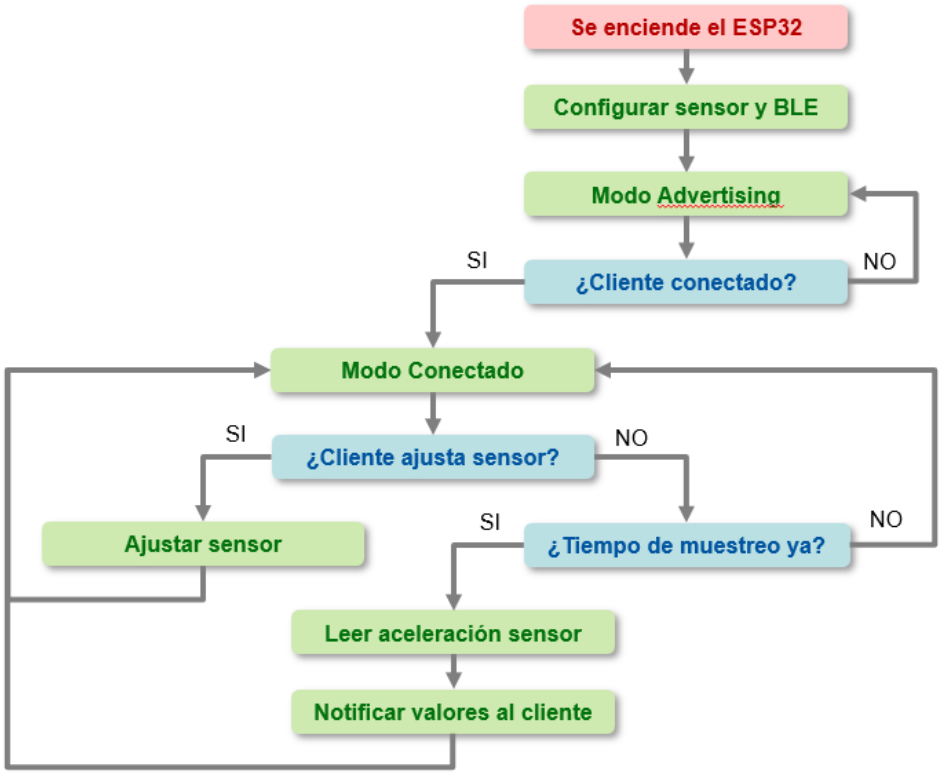

Ilustración 4-4. Flujograma del programa implementado en ESP32

Una vez que se enciende el dispositivo ESP32, el programa inicializa la configuración por defecto del sensor y del servidor Bluetooth, y pone al servidor en modo advertising a la espera de que se conecte un cliente. Una vez que el dispositivo Android se ha conectado, el ESP32 realiza un muestreo de los valores del acelerómetro y envía la información al cliente Android. A su vez, el servidor es capaz de recibir la configuración que envíe el cliente para ajustar la sensibilidad del sensor y el tiempo de muestreo.

Por lo tanto, para comenzar con la programación del ESP32, el primer paso es crear un nuevo proyecto en Arduino IDE, como se ha explicado en el Capítulo 3, tras lo cual se puede empezar a programar.

Tras esto, se pueden utilizar una serie de librerías que simplifican y facilitan el uso de determinados componentes, como en este caso ocurre al querer utilizar el sensor LSM9DS1 y Bluetooth Low Energy.

Para programar el sensor LSM9DS1 se ha utilizado la librería:

- `Adafruit_LSM9DS1.h`

Para utilizar el Bluetooth Low Energy se han utilizado las siguientes librerías:

- `BLEDevice.h`
- `BLEServer.h`
- `BLEUtils.h`
- `BLE2902.h`

Una vez añadidas (ver línea 4 del Anexo 7.1), para poder utilizar el sensor, hay que crear una variable del tipo `Adafruit_LSM9DS1`, lo cual es posible gracias al uso de la librería del mismo nombre (ver línea 18 del Anexo 7.1).

Una vez que tenemos una variable que nos permite realizar operaciones sobre el sensor, arrancamos el sensor utilizando la función `begin()` de la librería del sensor (ver línea 339 del Anexo 7.1), y establecemos la sensibilidad que tendrá el acelerómetro por defecto con la función `setupAccel()`, la cual en el caso del acelerómetro nos permite elegir entre cuatro posibilidades: una sensibilidad de 2G, 4G, 8G ó 16G. Por defecto estableceremos la sensibilidad en el mínimo, dado que los movimientos que se pretenden monitorizar no superan el doble de la aceleración de la gravedad, y por el contrario se quiere obtener la máxima precisión posible, por lo que se introduce el parámetro `LSM9DS1_ACCELRANGE_2G` en la función `setupAccel()` para hacer esto posible (ver línea 27 del Anexo 7.1).

Tras inicializar el sensor, el siguiente paso es inicializar Bluetooth Low Energy. Para ello, y de forma similar al sensor, hay que crear una variable que nos permita trabajar sobre este sistema de comunicación, por lo que se crea una variable del tipo `BLEDevice`, lo cual es posible gracias a la utilización de la librería correspondiente (ver línea 347 del Anexo 7.1).

El siguiente paso es crear un servidor Bluetooth, lo cual se realiza creando una variable de tipo `BLEServer`, en la que almacenaremos el resultado de utilizar la función `createServer()` sobre la variable `BLEDevice` que hemos creado anteriormente (ver línea 350 del Anexo 7.1).

Una vez que se ha creado el servidor, hay que registrarle una función callback del tipo `BLEServerCallbacks` que sea invocada cuando un cliente se conecte o se

desconecte del servidor (ver línea 354 del Anexo 7.1). Dentro de esta función existe una comprobación, en la que si el servidor ha ejecutado la función `onConnect()` es porque un cliente se ha conectado (ver línea 167 del Anexo 7.1), y por lo tanto se actualiza una variable que nos servirá para conocer si hay algún cliente conectado. Además, si un cliente se desconecta, dentro de este callback se llamará a la función `onDisconnect()`, por lo que también se podrá actualizar la variable que indique que no hay ningún cliente conectado (ver línea 174 del Anexo 7.1).

A continuación, siguiendo con la jerarquía del Bluetooth Low Energy, el servidor contiene servicios, por lo que hay que crear el servicio del acelerómetro, para ello se crea una variable de tipo `BLEService`, y en ella se almacena el resultado de llamar a la función `createService(UUID)` de la variable `BLEServer`, donde el UUID puede venir definido por el consorcio SIG o ser personalizado, como es este caso (ver línea 359 del Anexo 7.1).

De la misma manera, el servicio contiene características, por lo que hay que crear una variable de tipo `BLECharacteristic`, y en ella se almacena el resultado de llamar a la función `createCharacteristic(parámetros)` de la variable `BLEService`, donde "parámetros" indica las propiedades que tendrá la característica (ver línea 365 del Anexo 7.1), como son el `UUID`, y las diferentes opciones que se podrán ejecutar sobre ella, como leer (`READ`), escribir (`WRITE`), notificar (`NOTIFY`) e indicar (`INDICATE`).

A su vez, hay que dejar registrada en la característica una función callback del tipo `BLECharacteristicCallbacks` para cuando el cliente haya escrito en ella, dado que queremos que el cliente pueda configurar tanto la sensibilidad del acelerómetro como el tiempo que debe transcurrir entre muestra y muestra (ver línea 377 del Anexo 7.1). En esta función por lo tanto se analizará el nuevo valor que el cliente ha escrito en alguna de las dos características (sensibilidad o tiempo de muestreo), y en función de eso se establecerá la sensibilidad y el tiempo de muestro adecuados.

Continuando con la jerarquía de Bluetooth Low Energy, las características contienen descriptores, por lo que hay que crear una variable del tipo `BLEDescriptor`, y establecer el descriptor en la variable de tipo `BLECharacteristic` utilizando la función `addDescriptor()`. Por último, con `setValue()` se establece el valor que tendrá el descriptor, que puede ser una variable de tipo string (ver línea 383 del Anexo 7.1).

Una vez que se ha definido la jerarquía del servicio, se arranca (ver línea 468 del Anexo 7.1) utilizando la llamada a la función `start()`.

Cuando se hayan definido y arrancado todos los servicios, hay que poner al dispositivo en modo advertising. Para ello hay que crear una variable de tipo

`BLEAdvertising`, en la que se añadirán los UUID de los servicios que hayamos arrancado, y mediante la función `startAdvertising()` aplicada sobre la variable de tipo `BLEDevice`, el EPS32 comenzará a emitir balizas para que un cliente pueda encontrarlo y conectarse a él (ver línea 791 del Anexo 7.1).

Hasta aquí la parte de configuración inicial, por lo que antes de comentar la parte que se va a estar ejecutando en bucle, conviene definir la jerarquía de Bluetooth Low Energy que se pretende implementar en la placa ESP32.

- Device: ESP32
    - Servicio: Acelerómetro

      UUID: 00000001-0001-0001-0001-000000000001

        - Característica: Aceleración eje x

          UUID: 00000001-0001-0001-0001-000000000002

        - Característica: Aceleración eje y

          UUID: 00000001-0001-0001-0001-000000000003

        - Característica: Aceleración eje z

          UUID: 00000001-0001-0001-0001-000000000004

        - Característica: Ajustar sensibilidad

          UUID: 00000001-0001-0001-0001-000000000005

    - Servicio: Configuración

      UUID: 00000001-0001-0001-0004-000000000001

        - Característica: Ajustar muestreo

          UUID: 00000001-0001-0001-0004-000000000003

        - Característica: Conocer sensibilidad

          UUID: 00000001-0001-0001-0004-000000000004

Dentro del servicio "Acelerómetro", las características correspondientes a los ejes x y z, contienen los valores del sensor que serán notificados al cliente. La característica "Ajustar sensibilidad" es una característica que puede ser modificada por el cliente para ajustar la sensibilidad del sensor.

Dentro del servicio "Configuración", la característica "Ajustar muestreo" puede ser modificada por el cliente para ajustar el tiempo entre las tomas de datos. A su vez, la característica "Conocer sensibilidad" puede ser leída por el cliente para conocer la sensibilidad establecida en el sensor.

Respecto a la parte que se va a estar ejecutando de manera continua. En primer lugar, gracias al callback que indicaba si había clientes conectados, podemos comprobar si ya hay clientes conectados, o si por el contrario hay que poner al servidor de nuevo en modo advertising (ver línea 995 del Anexo 7.1) utilizando `startAdvertising()`.

En caso de que haya algún cliente conectado, procedemos a realizar una toma de datos del acelerómetro (ver línea 813 del Anexo 7.1) mediante la función `read()` de la variable `Adafruit_LSM9DS1`.

Tras haber tomado los datos, se almacenan en variables de tipo `sensors_event_t` mediante la función `getEvent()` de la variable `Adafruit_LSM9DS1`. Aunque sólo se necesitan los datos del acelerómetro, al estar utilizando una función de una librería, nos obliga a leer y almacenar los datos de cuatro sensores diferentes, que son el acelerómetro, el magnetómetro, el giroscopio y la temperatura, aunque el LSM9DS1 no incorpora sensor de temperatura (ver línea 819 del Anexo 7.1).

Por último, se coge el valor obtenido del acelerómetro, el cual contiene decimales, y se almacena en un string (ver línea 836 del Anexo 7.1) para posteriormente ser enviado al cliente (ver línea 848 del Anexo 7.1) mediante la función `notify()` de la variable de tipo `BLECharacteristic`.

Este proceso se repite para cada una de las características que queremos notificar al cliente. El motivo de hacer esto, es que si un cliente se conecta al servidor y lee las características, obtiene el valor en ese momento, pero si quiere obtener el nuevo valor que se haya obtenido tras realizar otra lectura del sensor, la forma óptima consiste en que sea el propio servidor el que notifique al cliente de que hay nuevos valores disponibles.

Por otra parte, durante el desarrollo del trabajo se estuvieron probando los tres sensores, por lo que la estructura y el propio código están preparados para notificar al cliente los valores del acelerómetro, del giroscopio y del magnetómetro.

4.4 Descripción de la aplicación implementada en Android

Una vez que el ESP32 es capaz de comunicarse con el sensor, y de actuar como un servidor de Bluetooth Low Energy que puede recibir la conexión de un cliente, hay que implementar la aplicación del dispositivo Android que se conectará al ESP32 y que permitirá que el usuario interactúe con el sensor. En la siguiente figura se puede observar el comportamiento de la aplicación.

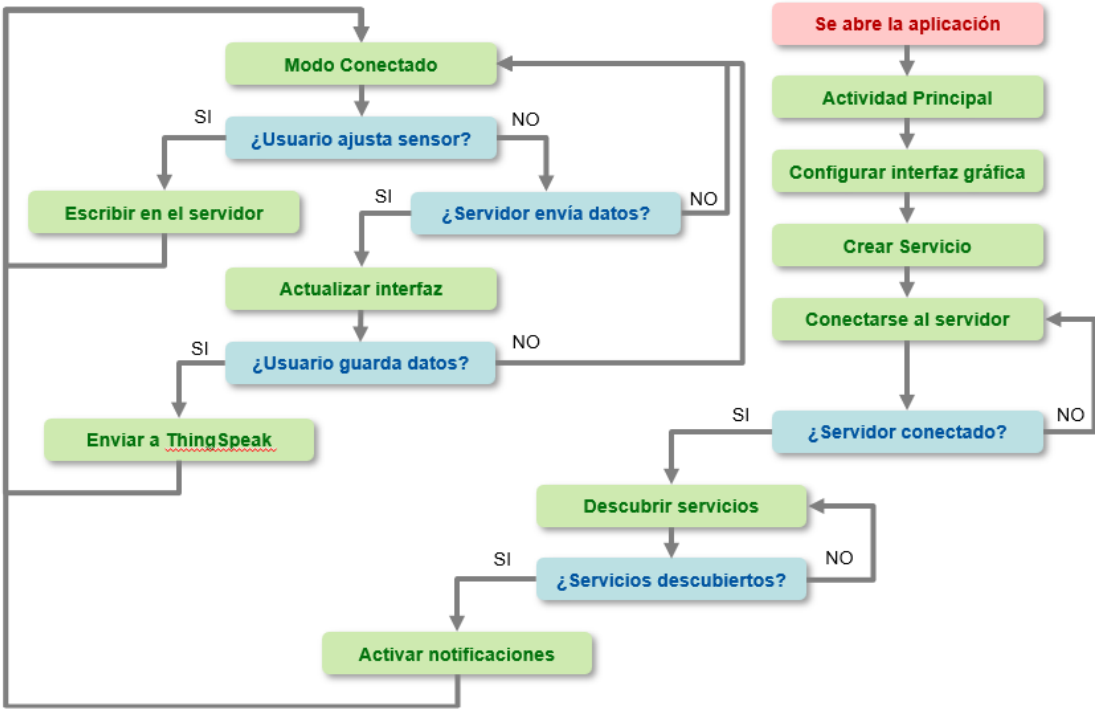

Ilustración 4-5. Flujograma del programa implementado en Android

Tras abrir la aplicación, se ejecuta la actividad principal, que se encarga de crear la interfaz gráfica, obtener la referencia a los controles de la interfaz, e inicializar el servicio que se va a ocupar de gestionar la conexión mediante Bluetooth. Cuando el usuario interactúe con la aplicación, la actividad principal le indicará al servicio las acciones que tiene que realizar mediante Bluetooth para comunicarse con el ESP32, y a su vez, cuando el ESP32 envíe información mediante Bluetooth, el servicio recibirá la información y la transferirá a la actividad principal para que el usuario pueda visualizarla.

Por consiguiente, siguiendo con el protocolo establecido para efectuar una conexión cliente-servidor mediante Bluetooth Low Energy, la aplicación intentará conectarse al ESP32, descubrirá sus servicios, leerá sus características, activará las notificaciones y le mostrará la información al usuario cada vez que el ESP32 envíe una nueva ronda de datos, concluyendo el proceso con el envío de la información a la

plataforma ThingSpeak siempre que el usuario lo desee. Además, el usuario podrá configurar la sensibilidad y el tiempo de muestreo del sensor escribiendo en las características del ESP32.

4.4.1 Creación del proyecto en Android Studio

Para ello, en primer lugar hay que crear un nuevo proyecto en Android Studio, utilizando los pasos descritos en el Capítulo 3. La única variable que se deberá decidir es el SDK mínimo, que en este caso se ha establecido en la versión de API 30, que corresponde a Android 11, por lo que la aplicación no utiliza la última versión del sistema operativo, que es la 12, para aumentar la compatibilidad con los dispositivos existentes en el mercado, dado que a pesar del tiempo transcurrido desde su lanzamiento, su cuota de mercado no supera el 13,5% según los datos proporcionados por el propio asistente de creación de proyectos de Android Studio, por lo que la versión 11 supone un buen equilibrio entre compatibilidad con los dispositivos existentes, y el uso de una API moderna que permita aprovechar las últimas novedades de seguridad introducidas en Android.

4.4.2 Estructura y funcionamiento de la aplicación de Android

Al contrario de lo que ocurre con la programación del ESP32 mediante Arduino IDE, un proyecto de Android consta de numerosas carpetas y archivos que le dan forma a la aplicación.

Durante este apartado se irán comentando cuales han sido los archivos que ha habido que modificar conforme se avance en la explicación sobre la programación de la aplicación, aunque como adelanto, y con vistas a ofrecer una pequeña visión general de la aplicación, se ofrece el siguiente resumen:

- Actividad Principal: cuando el usuario ejecuta la aplicación, la actividad principal es la primera parte de código que ejecuta Android, por lo que la funcionalidad de la aplicación debe comenzar a programarse por aquí. Consta de dos partes. Por un lado tiene un fichero .java en el cual se programa la lógica de la aplicación, y por otra parte tiene un fichero .xml en el que se define la apariencia de la interfaz de usuario.

- Servicio: se invoca desde la actividad principal, y su función es gestionar la comunicación Bluetooth Low Energy en segundo plano, por lo que no necesita interfaz gráfica. Por lo tanto, sólo consta de un fichero .java que se debe añadir al proyecto, dado que por defecto el asistente no lo crea.

- AndroidManifest.xml: este fichero recoge las actividades, servicios y content providers necesarios para que la aplicación funcione.

- Colors.xml: en este fichero se recogen los colores que se van a utilizar en la interfaz gráfica, para que sea más sencillo modificarlos de manera centralizada.

- Carpeta drawable: en esta carpeta se introducen las imágenes e iconos utilizados en la aplicación.

La aplicación comienza a ejecutarse desde la Actividad Principal, por lo que el primer paso dentro del fichero .java de la misma, es crear una clase (ver línea 86 del Anexo 7.2) que extiende de la clase `Activity`.

Una vez situados dentro de la clase, Android ejecuta varios métodos en función del estado en el que se encuentre la actividad (abierta, suspendida, terminada…), por lo que si acabamos de abrir la aplicación, el primer método que se ejecuta dentro de la clase es el método `onCreate()`, por lo que hay que continuar programando dentro de este método (ver línea 1751 del Anexo 7.2).

Lo primero que hay que hacer cuando se crea una actividad que va a mostrarle una interfaz gráfica al usuario, es generar la interfaz gráfica, por lo que el siguiente paso es utilizar el método `setContentView()` haciendo referencia al fichero .xml de la Actividad Principal (ver línea 1753 del Anexo 7.2), el cual contiene el código con las instrucciones sobre qué elementos hay que mostrar en la interfaz, qué aspecto tienen, y dónde están situados.

A la hora de construir la interfaz gráfica, tenemos la opción de usar el asistente gráfico de Android Studio, el cual permite situar los elementos a lo largo de la interfaz, mientras que por debajo va generando el código fuente. El problema de esto es que no controlamos el código que se genera, y por lo tanto no controlamos la jerarquía de los elementos de la interfaz gráfica, por lo que la aproximación que realice Android Studio puede desembocar en que cuando posteriormente deseemos controlar los elementos de la interfaz desde el fichero .java de la Actividad Principal, el comportamiento no sea el deseado.

Por lo tanto se ha optado por programar la interfaz directamente escribiendo el código necesario. Para ello, los "ladrillos" utilizados han consistido en `LinearLayout`, los cuales se han ido combinando de manera horizontal y vertical para conseguir la jerarquía necesaria de los datos mostrados en pantalla (ver línea 26 del Anexo 7.3).

Dentro de los `LinearLayout` se han colocado los elementos que son visibles para el usuario, como son los:

- `Button`: botones con texto que el usuario puede pulsar.
- `ImageButton`: botones con imágenes que el usuario puede pulsar.
- `TextView`: cuadro que contiene un texto que se muestra al usuario.
- `EditText`: cuadro que contiene un texto que el usuario puede editar.

El lenguaje es XML, por lo que dentro de cada uno de estos elementos sólo hay que definir las propiedades del elemento, como puede ser el tamaño, el color, el contenido y su posición respecto al resto de elementos. Además, uno de los campos que se rellenan es el `id` del elemento, lo que ayuda a que posteriormente desde el fichero .java se pueda llamar a los elementos de la interfaz que se han definido en el fichero .xml para configurar sus valores en tiempo de ejecución (ver línea 29 del Anexo 7.3).

A su vez, para conseguir que la interfaz sea más sencilla de comprender, los elementos se han agrupado en tres tarjetas (ver línea 17 del Anexo 7.3), mediante el uso de `CardView`, por lo que antes de continuar explicando la lógica de la aplicación, y de cara a hacerse una composición mental de los elementos de la interfaz, en la siguiente figura se observa el aspecto que tiene la aplicación cuando el usuario la ejecuta sin que haya ningún dispositivo ESP32 conectado.

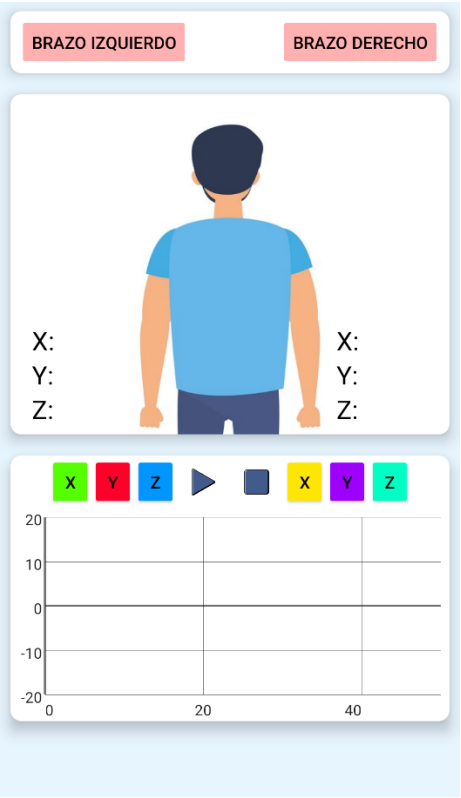

Ilustración 4-6. Aplicación de Android con ningún ESP32 conectado

La aplicación está diseñada para tomar datos de hasta dos sensores, por lo tanto, si cada sensor se conecta a un dispositivo ESP32, la aplicación es capaz de comunicarse a la vez con dos dispositivos ESP32 mediante el uso de Bluetooth Low Energy.

Por ello, la tarjeta superior se encarga de mostrar el estado de la conexión con los dispositivos ESP32, los cuales se han diseñado para ser instalados en la muñeca del usuario, y por lo tanto registran las aceleraciones producidas en el brazo izquierdo y en el brazo derecho del usuario. Así, mientras no haya ningún dispositivo conectado, ambos botones permanecerán en color rojo (ver línea 47 del Anexo 7.3).

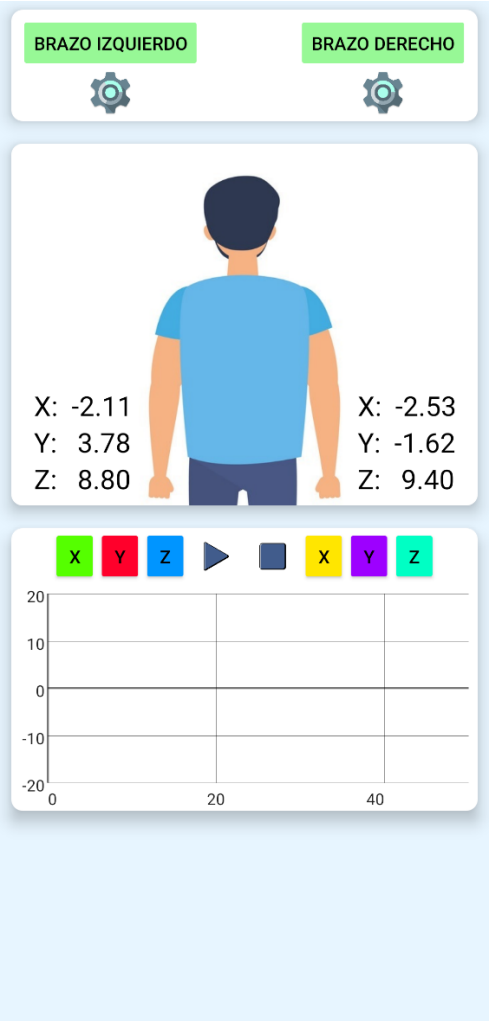

Ilustración 4-7. Aplicación de Android con dos ESP32 conectados

Cuando uno de los ESP32 se conecta al dispositivo Android, su botón cambiará al color verde, indicando que se ha realizado la conexión, y aparecerá un botón que permitirá configurar la sensibilidad y el tiempo de muestreo del sensor (ver línea 60 del Anexo 7.3). Además, la aplicación mostrará en tiempo real en la tarjeta del medio los valores que está enviando el sensor (ver línea 713 del Anexo 7.3).

Ilustración 4-8. Aplicación de Android configurando uno de los ESP32

Por último, en la tarjeta inferior se muestra una gráfica con los valores que el usuario quiera almacenar del sensor (ver línea 819 del Anexo 7.3). Para ello, cuando el usuario pulsa el botón "Play", la aplicación comienza a almacenar los valores del sensor (ver línea 887 del Anexo 7.3), dando la posibilidad mediante el botón "Pause" de que el usuario detenga temporalmente la captura de datos, y una vez que el usuario pulsa el botón "Stop", la aplicación finaliza la toma de datos, mostrándoselos al usuario y enviándolos a la plataforma ThingSpeak. La gráfica cuenta con la posibilidad de mostrar u ocultar los valores requeridos, así como de hacer zoom y desplazarse por los datos capturados (ver línea 848 del Anexo 7.3).

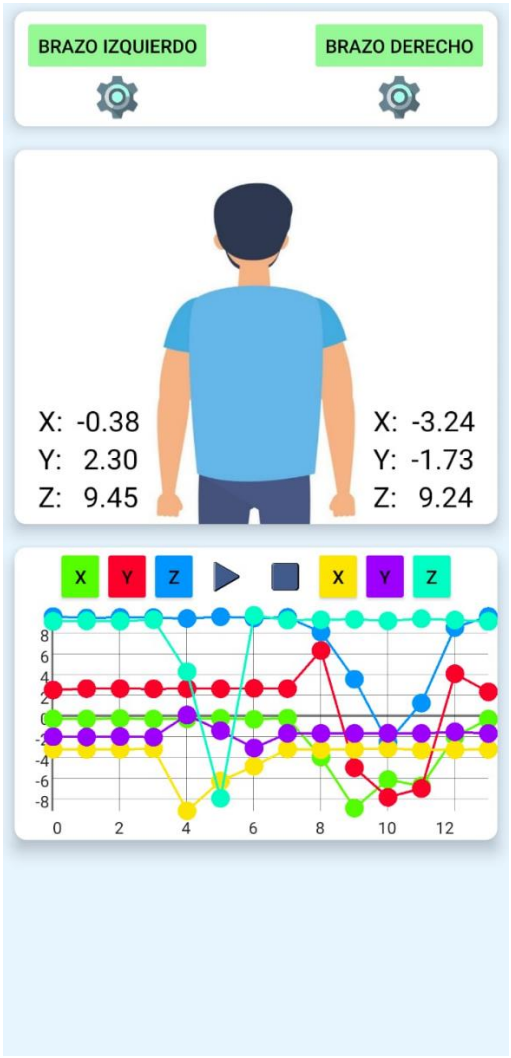

Ilustración 4-9. Aplicación de Android capturando datos de los dos ESP32

Una vez que se ha comentado la interfaz gráfica definida por el fichero .xml, hay que volver al fichero .java de la Actividad Principal para continuar explicando la lógica implementada en la aplicación.

Nos encontrábamos dentro del método `onCreate()` de la clase `Activity` de la Actividad Principal, y como queremos que la aplicación pueda utilizar la conectividad que proporciona Bluetooth Low Energy, hay que solicitarle permiso al usuario en tiempo de ejecución. En las últimas versiones de Android se hace necesario pedir permiso para `BLUETOOTH`, `BLUETOOTH_ADMIN`, `BLUETOOTH_CONNECT`, y `BLUETOOTH_SCAN`, incluyendo la solicitud de permiso tanto en el fichero AndroidManifest.xml (ver línea 4 del Anexo 7.5) como dentro de cada método de la clase que quiera hacer uso de estas funciones (ver línea 1757 del Anexo 7.2).

En caso de que el Bluetooth no esté activado cuando el usuario abre la aplicación, el método `requestPermissions()` de la actividad se encarga de solicitar al usuario que lo active (ver línea 1773 del Anexo 7.2).

El siguiente paso es obtener una referencia a los diferentes elementos de los que está constituida la interfaz, dado que algunos de los elementos queremos que se puedan modificar durante la ejecución de la aplicación, ya sea porque el sensor ha enviado nuevos datos, o porque el usuario interactúa con la aplicación.

Para ello, hay que definir un objeto para cada uno de los elementos de la interfaz que queremos controlar. El objeto será del mismo tipo que el elemento, y la manera de vincular el elemento de la interfaz con esta variable es mediante el uso del método `findViewById()`, en el cual introduciremos el `id` que se ha definido en el fichero .xml (ver línea 1862 del Anexo 7.2).

A su vez, aprovechando que se ha obtenido la referencia a los elementos de la interfaz, es un buen momento para configurar el comportamiento de los elementos cuando el usuario interactúa con ellos. Para ello, dado que la forma de interactuar consiste en pulsar dichos elementos en la pantalla, hay que utilizar el método `setOnClickListener()` sobre los objetos con los que se puede interactuar. Y dentro a su vez utilizar el método `onClick()`, el cual se ejecuta cuando el usuario pulsa sobre el elemento (ver línea 1870 del Anexo 7.2).

El detalle sobre qué elementos de la interfaz se modifican se comentará más adelante conforme la lógica implementada para gestionar el Bluetooth vaya necesitándolos.

Por último, hay que mencionar que para generar el gráfico se ha creado un objeto del tipo GraphView (ver línea 2079 del Anexo 7.2), el cual se ha obtenido de importar una librería externa creada por Jonas Gehring [25]. Para ello, dentro del fichero .java y fuera de la clase de la actividad principal, hay que utilizar el comando `import` junto con la URL en la que se ubica la librería (ver línea 40 del Anexo 7.2).

Además de esta librería, hay que incluir cualquier otra librería necesaria para ejecutar la aplicación, en las que normalmente, al formar parte de la API, el propio Android Studio realiza la sugerencia de realizar su importación al proyecto conforme las vamos necesitando al crear instancias de clases incluidas en esas librerías.

Una vez que la interfaz se ha creado y que hemos conseguido la referencia a los controles para poder modificar su apariencia y contenido, y que la aplicación tiene permiso para utilizar el Bluetooth, hay que comenzar con la lógica que permite realizar la conexión mediante Bluetooth Low Energy.

Lo primero es crear un objeto del tipo `BluetoothManager`, el cual permite acceder al Bluetooth del dispositivo (ver línea 1829 del Anexo 7.2), y sobre este, invocar a su método `getAdapter()`, el cual devolverá un objeto del tipo `BluetoothAdapter`, el cual es necesario para solicitarle al usuario que active el Bluetooth del dispositivo en caso de no haberlo activado antes de entrar a la aplicación (ver línea 1831 del Anexo 7.2).

Lo siguiente es crear el `Intent` que lanzará el servicio que estará en segundo plano gestionando la conexión Bluetooth (ver línea 1843 del Anexo 7.2). Mediante `bindService()` se enlaza la actividad principal con el servicio para que ambos puedan intercambiar datos (ver línea 1850 del Anexo 7.2). Además, creando un objeto de tipo `ServiceConnection()`, conseguimos que la actividad principal se entere del estado en el que se encuentra el servicio, para saber si el servicio se ha creado o se ha finalizado.

En caso de que el servicio se haya creado correctamente, dentro del `ServiceConnection()` se llama al método `onServiceConnected()`, y por lo tanto ya estamos en disposición de intentar conectarnos al ESP32 del brazo izquierdo y del brazo derecho (ver línea 292 del Anexo 7.2).

Para ello, hay que enviarle una orden al servicio, y la forma de hacerlo es creando un objeto de tipo `Service`, sobre el cual se invocará a un método que se ha definido dentro del fichero .java del servicio (ver línea 302 del Anexo 7.2). A este método se le pasa como parámetro la dirección MAC del dispositivo al que queremos conectarnos, la cual conocemos tanto para el ESP32 del brazo izquierdo como para el del derecho. En este caso, la dirección MAC del brazo izquierdo es `78:21:84:7A:F5:D2`, mientras que la del brazo derecho es `78:21:84:7C:1E:6E`.

Llegados a este punto, la actividad principal se quedaría escuchando a la información que le envíe el servicio gracias al uso de un `BroadcastReceiver` (ver línea 325 del Anexo 7.2), cuyo método `onReceive()` se invoca cuando se recibe una de las acciones definidas en el `IntentFilter` (ver línea 1695 del Anexo 7.2), las cuales se han asociado al `BroadcastReceiver` mediante el uso de un `registerReceiver` (ver línea 1731 del Anexo 7.2).

Las acciones que activan al `BroadcastReceiver` son la conexión de un dispositivo, la desconexión, el descubrimiento de servicios y la disponibilidad de un nuevo dato.

A su vez, para que esto funcione, en el fichero AndroidManifest.xml es necesario incluir el servicio y el intent-filter (ver línea 26 del Anexo 7.5).

Por lo tanto, antes de entrar en el detalle de qué ocurre en la actividad principal cuando se recibe un dato, es el momento de explicar el funcionamiento del servicio, el cual sólo consta de un fichero .java

Una vez dentro del fichero .java del servicio, lo primero (ver línea 26 del Anexo 7.4) es crear una clase del tipo `Service`, y dentro de ella se creará un objeto del tipo `IBinder` para enlazar el servicio con la Actividad Principal (ver línea 850 del Anexo 7.4). A partir de aquí comenzará la lógica del servicio.

Por supuesto y al igual que en la Actividad Principal, cada vez que dentro de un método haya que realizar alguna operación que requiera utilizar el Bluetooth Low Energy, será necesario comprobar si la aplicación dispone de permiso para utilizarlo (ver línea 185 del Anexo 7.4).

Lo siguiente, una vez que el servicio se ha enlazado con la Actividad Principal, es ejecutar la primera orden que le hemos dado desde la Actividad Principal, que no es otra que intentar conectarse al ESP32 (tanto el ubicado en el brazo izquierdo como en el derecho). Para ello, lo primero es disponer de un objeto que pueda realizar acciones sobre el Bluetooth del dispositivo Android, por lo que hay que crear un objeto de tipo `BluetoothDevice`, al que se le pasará como parámetro la dirección MAC enviada por la Actividad Principal (ver línea 922 del Anexo 7.4).

Una vez que se ha creado el `BluetoothDevice`, utilizando el método `connectGatt()` sobre él, le damos la orden a Android para que intente conectarse al ESP32 mediante Bluetooth Low Energy (ver línea 926 del Anexo 7.4).

El resultado de esta conexión se almacena en un objeto de tipo `BluetoothGatt`, que será el que se utilizará posteriormente para ejecutar acciones sobre el ESP32 al que nos hayamos conectado.

A su vez, sobre el objeto `BluetoothGatt` se dejará registrado un objeto de tipo `BluetoothGattCallback`, que será el encargado de detectar si nos hemos conectado o desconectado del ESP32, de saber si los servicios del ESP32 se han descubierto, de saber si se ha leído una característica del ESP32, o de saber si una característica del ESP32 ha modificado su valor, lo que a efectos prácticos permite recibir notificaciones del ESP32 (ver línea 129 del Anexo 7.4).

Por lo tanto, si el intento de conexión con el ESP32 ha sido exitoso, el `BluetoothGattCallback` llamará (ver línea 135 del Anexo 7.4) a su método `onConnectionStateChange()`, dentro del cual podremos mandar un `broadcastUpdate` a la Actividad principal para indicarle que nos acabamos de conectar

al ESP32, y así modificar elementos de la interfaz, como puede ser pasar de rojo a verde el color del botón que indica si estamos conectados o desconectados del ESP32 (ver línea 149 del Anexo 7.4). Además, dentro del propio servicio lanzamos la siguiente tarea (ver línea 161 del Anexo 7.4), que es descubrir los servicios del ESP32 mediante el método `discoverServices()` del objeto `BluetoothGatt`.

Una vez que los servicios del ESP32 se han descubierto, el `BluetoothGattCallback` llamará a su método `onServicesDiscovered()`, dentro del cual podremos mandar un `broadcastUpdate` a la Actividad principal para indicarle que acabamos de descubrir los servicios del ESP32 (ver línea 182 del Anexo 7.4), y así desde la Actividad Principal ejecutar la siguiente tarea.

Pero antes de explicar cuál es la siguiente tarea y de mostrar cómo hace la Actividad Principal para recibir estos mensajes del servicio, hay que finalizar un último paso que se ejecuta dentro del servicio una vez que los servicios del ESP32 se han descubierto, que no es otro que obtener una referencia a las características contenidas dentro de los servicios del ESP32. Para ello, hay que crear objetos de tipo `BluetoothGattCharacteristic` en los que se almacena el resultado de llamar al método `getService(UUID)` sobre el objeto `BluetoothGatt`, y a su vez al método `getCharacteristic(UUID)` sobre éste, debiendo introducir los UUID de los servicios y de las características que se han descrito en el apartado sobre la programación del ESP32 (ver línea 243 del Anexo 7.4).

Estos objetos nos proporcionan una referencia permanente a estas características, por lo que facilitarán el posterior uso de los mismos en el código. Por consiguiente, una vez que se ha obtenido la referencia a las características, hay que habilitar las notificaciones para que el ESP32 pueda avisarnos cada vez que obtenga un nuevo valor del sensor. Para ello, basta con utilizar el método `setCharacteristicNotification`() sobre el la característica del `BluetoothGatt` en la que queramos activar las notificaciones (ver línea 270 del Anexo 7.4).

Volviendo a la Actividad Principal y como se ha comentado antes, el `BroadcastReceiver` se activa y llama al método `onReceive()` cuando el servicio envía un `broadcastUpdate` que contiene alguna de las acciones definidas en el `IntentFilter` (ver línea 325 del Anexo 7.2).

Tras esto, y continuando en el punto en el que se acababan de descubrir los servicios y habíamos activado las notificaciones, estamos en disposición de empezar a recibir datos del ESP32 cada vez que el sensor registre un nuevo valor de la aceleración en el eje x, y, z. Pero antes, desde la Actividad Principal vamos a dar la orden de que el servicio consulte cual es el muestreo y la sensibilidad del sensor del ESP32, para que el usuario pueda conocer sus valores antes de configurarlos y establecer un nuevo valor.

Para ello, en primer lugar (ver línea 409 del Anexo 7.2) le decimos al servicio que invoque al método `readCharacteristic()` del objeto `BluetoothGatt` sobre la característica del ESP32 que contiene el dato (ver línea 1055 del Anexo 7.4).

En cuanto Android lea la característica del EPS32, se llamará al método `onCharacteristicRead()` del objeto `BluetoothGattCallback` (ver línea 288 del Anexo 7.4), el cual emitirá un `broadcastUpdate` (ver línea 291 del Anexo 7.4) que será leído por el método `onReceive()` del `BroadcastReceiver` (ver línea 325 del Anexo 7.2), y mostrará el dato del muestreo actual en la interfaz (ver línea 912 del Anexo 7.2), y tras esto solicitará al servicio que averigüe cual es la sensibilidad actual del sensor (ver línea 925 del Anexo 7.2).

El servicio nuevamente invocará al método `readCharacteristic()` del objeto `BluetoothGatt` sobre la característica del ESP32 que contiene el dato (ver línea 1177 del Anexo 7.4).

En cuanto Android lea la característica del ESP32, se llamará al método `onCharacteristicRead()` del objeto `BluetoothGattCallback` (ver línea 288 del Anexo 7.4), el cual emitirá un `broadcastUpdate` que será leído por el método `onReceive()` del `BroadcastReceiver` (ver línea 325 del Anexo 7.2), y mostrará el dato de la sensibilidad al usuario (ver línea 935 del Anexo 7.2).

A continuación, antes de comentar lo que ocurre cuando se recibe un dato del sensor, hay que explicar lo que hace el servicio cuando el usuario quiere modificar la sensibilidad o el tiempo de muestreo del sensor del ESP32.

En caso de que el usuario quiera configurar la sensibilidad, se le plantean las cuatro opciones disponibles para este sensor, que son 2G, 4G, 8G y 16G. Una vez que el usuario selecciona la sensibilidad deseada (ver línea 1937 del Anexo 7.2), le decimos al servicio que invoque al método `writeCharacteristic()` del objeto `BluetoothGatt` sobre la característica del ESP32 que contiene el dato que queremos modificar (ver línea 1955 del Anexo 7.2), indicándole cuál será su nuevo valor, y pudiendo decidir si queremos recibir una confirmación de escritura por parte del ESP32 (ver línea 659 del Anexo 7.4).

Utilizando el mismo código se ejecuta la modificación del tiempo de muestreo del sensor, sólo que esta vez el usuario es libre de introducir el valor en milisegundos, en lugar de elegir entre varias opciones predeterminadas (ver línea 671 del Anexo 7.4).

En definitiva, desde la Actividad Principal se pueden enviar datos al servicio llamando a métodos contenidos en el servicio. Y desde el servicio se pueden enviar datos a la Actividad Principal enviando `broadcastUpdate`, los cuales envían una pareja de datos,

consistente en un identificador para saber qué característica ha enviado el dato y en el propio dato.

Por último, cuando el sensor envía un nuevo dato del acelerómetro, como no lo realiza mediante una petición de lectura del dispositivo Android sino como una notificación, el servicio llama al método `onCharacteristicChanged()` del objeto `BluetoothGattCallback` (ver línea 299 del Anexo 7.4), el cual emitirá un `broadcastUpdate` (ver línea 301 del Anexo 7.4) que será leído por el método `onReceive()` del `BroadcastReceiver`, y mostrará el dato de la aceleración al usuario (ver línea 436 del Anexo 7.2).

Volviendo a la Actividad Principal, tras haber recibido el dato de la aceleración y habérselo mostrado al usuario, queda la última funcionalidad de la aplicación, que consiste en mostrar un histórico de valores registrados por la aplicación, y en enviar esos datos a la plataforma ThingSpeak.

Para ello, hay que crear un objeto de tipo `GraphView` (ver línea 121 del Anexo 7.2), el cual se ha obtenido de importar la librería mencionada durante la explicación de la interfaz gráfica. Esto generará un "lienzo" sobre el que se podrán mostrar los valores obtenidos, así que lo siguiente es elegir la manera de mostrar los valores.

Se ha optado por mostrar los valores mediante la representación de puntos, para lo cual hace falta crear un objeto de tipo `PointsGraphSeries` (ver línea 125 del Anexo 7.2) y mediante la representación de líneas que conecten los puntos, por lo que también se ha creado un objeto de tipo `LineGraphSeries` (ver línea 123 del Anexo 7.2).

Estos dos objetos hay que añadirlos al gráfico que se ha creado mediante el uso del método `addSeries()` sobre el objeto de tipo `GraphView` (ver línea 2353 del Anexo 7.2). Por último, mediante el método `getViewport()` del objeto `GraphView` se puede personalizar la apariencia del gráfico, como la leyenda y el tamaño del mismo (ver línea 2633 del Anexo 7.2).

Por supuesto, es posible modificar los colores de este gráfico, así como los del resto de elementos utilizados en la interfaz. Para ello, se ha optado por hacer referencias al color almacenado en el fichero colors.xml, lo que permite acceder de manera rápida y centralizada a la lista de colores utilizados en la aplicación, por si posteriormente se decide realizar alguna modificación en los mismos (ver línea 20 del Anexo 7.6).

Finalizando con el gráfico, en la interfaz se han añadido una serie de botones (ver línea 2081 del Anexo 7.2) que permiten que el usuario comience la toma de datos, la pause, la reanude y la detenga, o incluso que pueda mostrar u ocultar los valores de alguno de los

ejes del acelerómetro, lo cual se realiza incluyendo condiciones "if" que comprueben si cuando se ha recibido un dato del acelerómetro en la Actividad Principal el usuario ha dado permiso o no para almacenar el dato en el gráfico (ver línea 449 del Anexo 7.2).

El último paso es subir los datos a la plataforma ThingSpeak cuando el usuario finalice la toma de datos. Esto requiere que la aplicación acceda a Internet, por lo que en el fichero AndroidManifest.xml hay que conceder permiso a `INTERNET` y a `ACCESS_NETWORK_STATE` (ver línea 11 del Anexo 7.5).

De acuerdo a la documentación proporcionada por ThingSpeak, se va a realizar el envío de datos en formato JSON desde el dispositivo Android hasta el servidor realizando una petición HTTP POST. Los datos en JSON deben tener el siguiente formato:

```
{
   "write_api_key": "148VVPL6CFF3LG0H",
   "updates": [{
                "delta_t": 1,
                "field1": 1.0,
                "field2": 2.0,
                "field3": 3.0
          },
          {
                "delta_t": 2,
                "field1": 1.1,
                "field2": 2.2
                "field3": 3.3
          }
   ]
}
```

Donde `write_api_key` se corresponde a la clave de la API que nos ha otorgado ThingSpeak para escribir datos en la plataforma desde un dispositivo externo a ella, y donde los diferentes `field1`, `field2`… se corresponde a los valores de las aceleraciones enviados por el ESP32, que en el caso de contar con un único dispositivo ESP32 conectado al dispositivo Android rellenará tres campos, y en caso de tener los dos conectados rellenará seis campos (ver línea 502 del Anexo 7.2).

Para lograr la estructura de datos requerida por el JSON que se enviará a ThingSpeak, hay que crear una serie de objetos, que deberán situarse con la siguiente jerarquía. En primer lugar un `JSONOBject` que contendrá el campo `write_api_key`, seguido de un `JSONArray` que contendrá a los `JSONOBject` con los valores enviados por el acelerómetro (ver línea 262 del Anexo 7.2). Los datos se añaden mediante el uso del método `put()` sobre el objeto JSON (ver línea 514 del Anexo 7.2).

La aplicación continuará almacenando datos en el JSON hasta que el usuario detenga la captura de datos (ver línea 2160 del Anexo 7.2). En ese momento la aplicación enviará los datos a ThingSpeak. Para conseguir esto, dado que las conexiones a Internet deben realizarse en un hilo independiente al de la Actividad Principal (ver línea 2212 del Anexo 7.2), hay que crear un nuevo hilo usando la instrucción `new Thread()`, y sobrescribir su método `run()`.

Dentro hay que crear un objeto de tipo `URL` e introducir la siguiente dirección: https://api.thingspeak.com/channels/1789950/bulk_update.json, que se corresponde al canal que se ha creado en ThingSpeak y a la manera correcta de cargar un JSON con los datos que hemos recopilado del sensor (ver línea 2220 del Anexo 7.2).

También hará falta un objeto del tipo `HttpsURLConnection` para realizar la conexión al servidor mediante el método `openConnection()` (ver línea 2225 del Anexo 7.2).

A su vez, en el objeto `HttpsURLConnection` hay que permitir que la aplicación de Android pueda escribir en el servidor (ver línea 2221 del Anexo 7.2) mediante el método `setDoOutput(true)`, hay que establecer que el método de conexión utilizado por HTTP sea POST mediante el método `setRequestMethod("POST")` (ver línea 2236 del Anexo 7.2), y hay que indicarle al servidor que los datos que le vamos a enviar están en formato JSON mediante el método `setRequestProperty("Content-Type", "application/json")` (ver línea 2244 del Anexo 7.2).

Finalmente, para poder enviar el JSON, hay que crear un objeto de tipo `DataOutputStream`, el cual escribirá (ver línea 2262 del Anexo 7.2) los datos en el servidor mediante su método `writeBytes()` (ver línea 2267 del Anexo 7.2).

Cabe remarcar que para poder ejecutar tareas sobre objetos de tipo `JSONObject`, objetos `HttpsURLConnection` u objetos de tipo `URL`, hay que realizar las tareas dentro de estructuras de tipo `try {} catch {}`, (ver línea 2215 del Anexo 7.2) para poder manejar las excepciones que ocurran (ver línea 2288 del Anexo 7.2).

Una vez que se ha finalizado con la programación del EPS32 y de la aplicación de Android, cabe destacar, que aunque la jerarquía de servicios y características del Bluetooth Low Energy que se ha implementado se podría haber simplificado para que sólo hubiera un servicio y cinco características, se ha optado por utilizar más de un servicio y más de una característica, incluso habiendo cierta redundancia en algunas partes, para dejar una estructura de código más modular y escalable, en la que ya se haya estudiado el código necesario para manejar más de un servicio y más de una característica, para que a futuro se puedan expandir las posibilidades del ESP32, ya sea utilizando más sensores, o definiendo una serie de servicios y características estandarizados por el consorcio SIG, con vistas a una posible implementación comercial.

4.5 Descripción de la arquitectura implementada en ThingSpeak

Después de programar la aplicación de Android, el último elemento de la arquitectura es la plataforma ThingSpeak.

4.5.1 Creación del Channel en ThingSpeak

Una vez que hemos iniciado sesión en la plataforma, y como se ha explicado en el Capítulo 3, hay que pulsar en el botón "New Channel" para crear un nuevo canal.

A continuación hay que indicar los datos del canal:

- Name: Datos
- Description: datos recopilados del acelerómetro del brazo izquierdo y del brazo derecho
- Field 1: Brazo Izquierdo - Aceleración X
- Field 2: Brazo Izquierdo - Aceleración Y
- Field 3: Brazo Izquierdo - Aceleración Z
- Field 4: Brazo Derecho - Aceleración X
- Field 5: Brazo Derecho - Aceleración Y
- Field 6: Brazo Derecho - Aceleración Z

El resto de datos son opcionales, por lo que al no ser necesarios para que la aplicación de Android pueda enviar los datos, no se han rellenado.

Tras esto, pulsando en "Private View" podemos acceder a los datos que nos haya enviado el dispositivo con Android, los cuales aparecerán debajo del panel "Channel Stats". Los paneles con la información son personalizables, por lo que se puede elegir la apariencia del gráfico pulsando sobre el botón de opciones de cada gráfico.

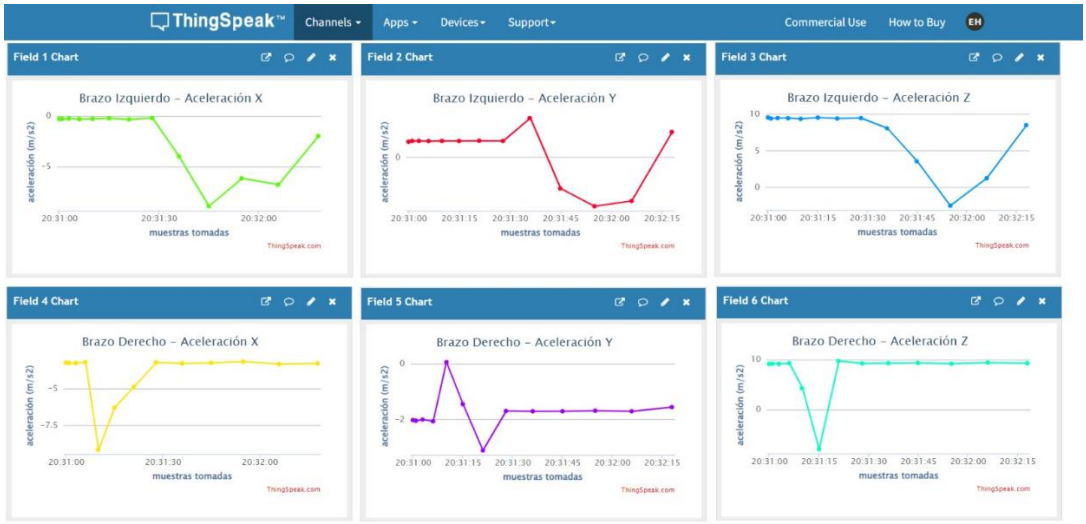

Ilustración 4-10. ThingSpeak mostrando los datos enviados por Android

A su vez, en este panel aparece un botón llamado "MATLAB Analysis" que permite realizar análisis sobre los datos obtenidos.

4.5.2 Obtención de los datos de la API de Thingspeak para escribir en el Channel

Un apartado importante de ThingSpeak es el que indica los datos que hay que introducir en la aplicación de Android para que la aplicación pueda comunicarse con ThingSpeak para enviarle datos.

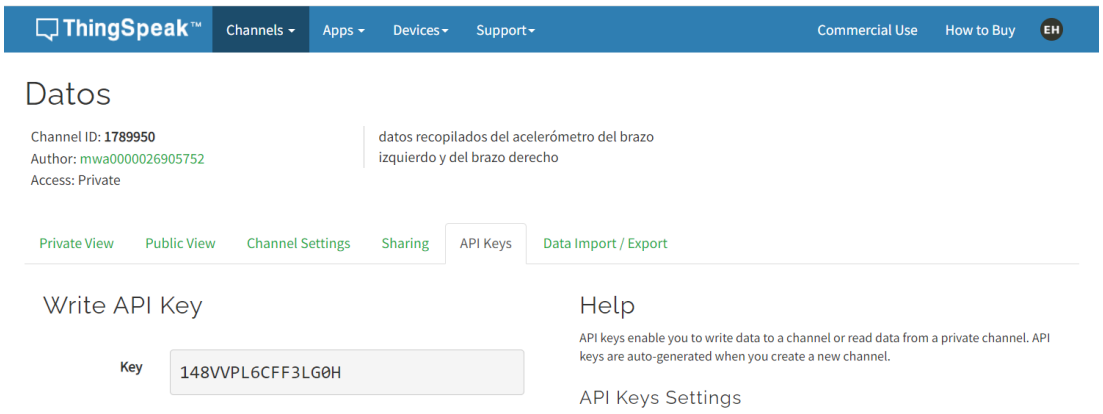

Ilustración 4-11. ThingSpeak mostrando la clave de la API para escribir datos

Desde el dashboard de ThingSpeak, accediendo a nuestro canal, y dentro de él al apartado "API Keys", se accede al lugar donde aparece la "Write API Key", que en este caso corresponde al valor 148VVPL6CFF3LG0H. Este valor permitirá que la aplicación de Android pueda escribir los datos del JSON en el servidor.

4.6 Conclusiones

Se han repasado los elementos de la arquitectura hardware y software desarrollados durante el trabajo, comenzando por el cableado del sensor LSM9DS1 al ESP32, así como su integración en un dispositivo de pulsera, lo que ha requerido que la alimentación del sensor se realice desde una pila en lugar de mediante una conexión por el puerto USB, y también ha requerido de realizar un encapsulado para poder ubicar el conjunto en el brazo del usuario.

Después se ha explicado la programación que ha habido que realizar en Arduino IDE para programar el EPS32 y su integración con el sensor, así como la programación necesaria en Android Studio para programar la aplicación de Android que ha permitido que el ESP32 y el dispositivo con Android se comuniquen mediante Bluetooth Low Energy, y que el usuario pueda consultar los datos y configurar el sensor desde la aplicación.

Finalmente, se ha explicado la configuración que ha habido que realizar en la plataforma ThingSpeak para que estuviera preparada para recibir datos por internet desde el dispositivo Android.

Capítulo 5

Análisis de Resultados y Conclusiones

5.1 Pruebas y análisis de resultados

Una vez que se han desarrollado los elementos de hardware y de software del trabajo, hay que verificar que el sistema funciona de la manera esperada.

Los elementos de la arquitectura son el sensor LSM9DS1, el microcontrolador ESP32, el dispositivo Android y la plataforma ThingSpeak.

Comenzando por el sensor LSM9DS1, hay que verificar que el cableado se encuentra correctamente realizado con el ESP32, y que los valores de aceleración detectados son coherentes y se encuentran dentro del rango de sensibilidad seleccionado, para descartar posibles errores de calibración del sensor.

Respecto al ESP32, el dispositivo con Android debe ser capaz de detectarlo mediante el uso de Bluetooth Low Energy en un periodo de tiempo de apenas unos segundos desde que ambos dispositivos están encendidos, por lo que si la aplicación de Android tarda más de 5 segundos en conectarse al ESP32 puede ser un indicio de que el ESP32 se encuentra en mal estado o que la batería que lo alimenta no está entregando la potencia o tensión suficiente para que los elementos del ESP32 funcionen correctamente.

A su vez, cuando el ESP32 se coloque en el brazo del usuario mediante un guante, hay que asegurarse de que está sujeto correctamente, para evitar que el ESP32 se desplace

respecto al brazo, y que por lo tanto los valores de aceleración no reflejen el movimiento real del brazo, sino que en su lugar su valor sea la suma de la aceleración experimentada por el brazo junto a la aceleración experimentada por el ESP32 respecto al brazo.

Sobre el dispositivo con Android, conviene asegurarse de que el dispositivo ejecuta una versión de Android igual o superior a la versión 11, y de que cuenta con soporte para Bluetooth Low Energy. También se recomienda que Bluetooth se encuentre encendido antes de acceder a la aplicación, para agilizar la conexión al ESP32, aunque no es obligatorio, ya que en caso de acceder a la aplicación con el Bluetooth apagado, la propia aplicación le solicitará al usuario que lo active.

Por último, la plataforma ThingSpeak no requiere que se realice ninguna configuración previa al uso de la aplicación de Android, por lo que la única verificación que se puede realizar antes de comenzar a utilizar la aplicación, es acceder al canal de ThingSpeak desde la web de la plataforma, para asegurarse de que el servidor se encuentra activo y de que por tanto no hay ninguna incidencia en los servidores que impida que cuando la aplicación de Android intente enviar los datos, estos no puedan alcanzar al servidor y por lo tanto no se almacenen.

Tras analizar las recomendaciones que hay que tener en cuenta sobre cada elemento del trabajo, lo siguiente que hay que comentar es el procedimiento que hay que seguir para probar que el sistema funciona.

Lo primero que hay que hacer es ubicar el sistema ESP32-LSM9DS1 en el brazo del usuario, y habiendo dotado al sistema previamente de alimentación mediante una batería externa de tipo Power Bank, lo que hará que el ESP32 ejecute el programa que se le ha cargado y que por lo tanto a los pocos segundos de estar alimentándose pueda ser detectado por un dispositivo compatible con Bluetooth.

A continuación, hay que encender el Bluetooth en el dispositivo Android y abrir la aplicación, la cual se conectará automáticamente al ESP32 y comenzará a mostrar el valor de las aceleraciones enviadas por el sensor sin necesidad de que el usuario intervenga. A su vez, en la pantalla del dispositivo se indicará mediante el color verde que la aplicación se ha podido conectar por Bluetooth al ESP32, y mostrará el botón de ajustes por si el usuario quiere configurar la sensibilidad o el periodo de muestreo del sensor.

En caso de que el dispositivo se aleje o la conexión Bluetooth se interrumpa, el dispositivo con Android lo indicará mostrando el color rojo y dejando de mostrar valores de aceleración en tiempo real. A su vez, volverá a conectarse automáticamente al ESP32 en cuanto se encuentre dentro del alcance sin que el usuario tenga que intervenir.

La última prueba consiste en darle al botón "Play" del gráfico para que la aplicación empiece a almacenar los valores de aceleración registrados, y tras darle al botón "Stop", la aplicación dejará de registrar los valores de aceleración y los enviará a la plataforma ThingSpeak, en la que entrando desde la página web y accediendo a la vista privada del Canal, se pueden consultar los datos que ha enviado el dispositivo Android.

Analizando los datos del acelerómetro en tiempo real, así como los registrados en el gráfico, se puede comprobar que el sensor funciona correctamente, ya que si se orienta cualquiera de los ejes del acelerómetro hacia el suelo, se obtiene con un margen de error de varias centésimas el valor de la aceleración producida por la atracción gravitatoria que ejerce la tierra sobre todos los cuerpos situados sobre ella, de aproximadamente 9,81 m/s^2.

También cabe destacar que durante la ejecución de las pruebas hubo algún problema a la hora de enviar los datos a ThingSpeak, ya que la cuenta utilizada para realizar las pruebas contaba con una licencia gratuita, la cual tiene una limitación en cuanto a la cantidad de datos que se pueden enviar y en cuanto al tiempo mínimo permitido entre envío y envío, por lo que de cara a aprovechar todo el potencial de la plataforma y evitar errores durante el uso de la aplicación, se recomienda utilizar una licencia de pago.

5.2 Conclusiones y trabajo futuro

Por último, hay que analizar el grado de consecución del trabajo respecto a los objetivos que se habían marcado inicialmente, comenzando por el proceso de selección de los elementos del trabajo.

El primer paso fue seleccionar el sensor adecuado para registrar la aceleración, por lo que hubo que valorar entre las diferentes alternativas presentes en el mercado, y elegir un sensor como el LSM9DS1 con probada experiencia en proyectos de este tipo, cuya placa electrónica además contaba con otro tipo de sensores y ajustes de sensibilidad que permitían ampliar las posibilidades del trabajo en caso de ser necesario.

Tras esto, hubo que elegir un microcontrolador que fuera capaz de comunicarse con el sensor, y que contara con la tecnología de comunicación inalámbrica necesaria para intercambiar datos con el dispositivo móvil, por lo que el ESP32 contaba con la potencia suficiente para realizar esta tarea, y con la posibilidad de realizar una conexión mediante Bluetooth Low Energy. Además, su compatibilidad con el entorno de desarrollo Arduino IDE y con el lenguaje de programación utilizado para los proyectos de este suponía un aliciente para que fuera el candidato elegido, dado que evitaba la necesidad de aprender un nuevo lenguaje y entorno de programación propietario.

Para la comunicación entre el ESP32 y el dispositivo móvil, y tras estudiar las alternativas presentes en el mercado, se optó por el uso de Bluetooth Low Energy, debido a su madurez en el mercado y su bajo consumo de energía, vital para el EPS32 una vez que se coloca en el brazo del usuario y por lo tanto se aleja de una fuente de alimentación cableada.

Respecto al dispositivo que se iba a encargar de comunicarse con el ESP32 para obtener los datos, configurar el sensor y enviarlos a la plataforma de Internet, se optó por un dispositivo móvil inteligente que ejecute Android, debido a la amplia documentación disponible para programar, así como a la gran cuota de mercado de ese sistema operativo.

Una vez que la aplicación de Android disponía de los datos, había que enviarlos a una plataforma en Internet, por lo que tras valorar varias alternativas se optó por una plataforma que aunque no es la punta de lanza en cuanto a capacidades Web, lo compensa con la parte posterior a la adquisición de los datos, es decir, el análisis de los mismos, dado que ThingSpeak cuenta con el soporte de los creadores de MATLAB, lo que le hace ganar enteros entre la comunidad académica.

Por consiguiente, habiendo comprobado que los componentes del trabajo cumplen su cometido, se valora de manera positiva la consecución de los objetivos fijados al comienzo del mismo, aunque no se cierra la puerta a optimizar el proceso mediante el uso de nuevo hardware y nuevas APIs propietarias que reduzcan el consumo y mejoren la velocidad de los componentes.

De cara al futuro, se puede trabajar sobre los siguientes aspectos:

- Elegir sensores específicos para el tipo de datos que se necesitan monitorizar, eliminando así el resto de componentes que incluye el sensor que no se utilizan y por lo tanto abriendo la posibilidad de que las dimensiones del encapsulado y el peso disminuyan, lo que permite que el dispositivo se sienta más liviano y natural para el usuario.

- Buscar microcontroladores cuya única funcionalidad se ciña a la utilizada en el trabajo, para conseguir que las dimensiones del encapsulado se reduzcan al mínimo, y a su vez buscar dispositivos en los que el fabricante haya implementado modos de bajo consumo bastante agresivos que permitan aumentar la autonomía del dispositivo al máximo que otorgue la batería utilizada.

- Buscar un sistema de sujeción del encapsulado al brazo del usuario que luzca más estético, cuyas dimensiones se reduzcan al mínimo necesario para alojar al

microcontrolador y al sensor, y cuya sujeción permita que el sensor no se mueva respecto al brazo del usuario, para evitar errores de interpretación en los datos.

- Estudiar un sistema de alimentación para el microcontrolador basado en una batería Li-Po de dimensiones acordes al encapsulado propuesto en el punto anterior.

- Estudiar el impacto en el consumo de batería entre utilizar una jerarquía de servicios y características de Bluetooth Low Energy que siga las directrices marcadas por el Bluetooth SIG, respecto a una jerarquía completamente personalizada y simplificada, libre de duplicidades o de datos que requiere el Bluetooth SIG, pero que no son necesarios para que el sistema cumpla su función, a costa de destruir cualquier tipo de compatibilidad con aplicaciones que respeten las directrices del Bluetooth SIG.

- Simplificar la aplicación de Android y los algoritmos y funciones implementados en la misma, así como añadir nuevas funcionalidades que permitan almacenar varios perfiles de usuario en la aplicación, o mejorar la interconexión entre la aplicación y ThingSpeak, añadiendo la posibilidad de eliminar los datos almacenados en ThingSpeak, o de consultarlos y mostrarlos en el gráfico.

- Partiendo de los datos almacenados en ThingSpeak, estudiar la implementación de algoritmos de Big Data que analicen los datos y obtengan conclusiones sobre los mismos de manera automática, de cara a poder detectar patrones o trastornos del movimiento en el usuario analizado.

Referencias

[1] Md. Taslim Arefin, Mohammad Hanif Ali, A. K. M. Fazlul Haque, "Wireless Body Area Network: An Overview and Various Applications" (2017)
https://www.scirp.org/journal/paperinformation.aspx?paperid=76200

[2] "History Of The Body Area Networks" (2017)
https://www.ukessays.com/essays/information-technology/history-of-the-body-area-networks-information-technology-essay.php

[3] Rahat Ali Khan, Al-Sakib Khan Pathan, "The state-of-the-art wireless body area sensor networks: A survey" (2018)
https://journals.sagepub.com/doi/10.1177/1550147718768994

[4] Daniel Jost, "What is an accelerometer?" (2019)
https://www.fierceelectronics.com/sensors/what-accelerometer

[5] "Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout" (2017)
https://learn.adafruit.com/assets/38874

[6] "MÓDULO MPU6050: ACELERÓMETRO, GIROSCOPIO I2C"
https://naylampmechatronics.com/sensores-posicion-inerciales-gps/33-modulo-mpu6050-acelerometro-giroscopio-i2c.html

[7] "Acelerómetro 3 ejes ADXL345"
https://tienda.bricogeek.com/acelerometros/1158-acelerometro-3-ejes-adxl345-2g4g8g16g.html

[8] "CY8CKIT-062-BLE"
https://www.infineon.com/cms/en/product/evaluation-boards/cy8ckit-062-ble/

[9] "ESP32-WROOM-32E DATASHEET"
https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf

[10] "nRF5340 DK"
https://www.nordicsemi.com/Products/Development-hardware/nRF5340-DK

[11] "Wi-Fi"
https://www.britannica.com/technology/Wi-Fi

[12] "Wi-Fi"
https://en.wikipedia.org/wiki/Wi-Fi

[13] Robert Triggs, Calvin Wankhede, "A little history of Bluetooth" (2022)
https://www.androidauthority.com/history-bluetooth-explained-846345/

[14] "Bluetooth"
https://en.wikipedia.org/wiki/Bluetooth

[15] Kyle Anderson, "The Evolution of The Smartphone" (2021)
https://storymaps.arcgis.com/stories/43449bc48bbc4937b440e9ed3e2ea11c

[16] "Android (operating system)"
https://en.wikipedia.org/wiki/Android_(operating_system)

[17] "iOS"
https://en.wikipedia.org/wiki/IOS

[18] "Arduino Integrated Development Environment (IDE) v1"
https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics

[19] "ESP-IDF Programming Guide"
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html

[20] "ThingSpeak"
https://es.mathworks.com/help/thingspeak/

[21] "AWS IoT Analytics"
https://aws.amazon.com/es/iot-analytics/

[22] "Arduino IDE 1.8.19"
https://www.arduino.cc/en/software

[23] "Java Downloads"
https://www.oracle.com/java/technologies/downloads/#jdk18-windows

[24] "android studio"
https://developer.android.com/studio

[25] Jonas Gehring "GraphView" (2013)
https://github.com/jjoe64/GraphView

ANEXOS

7.1 Código Fuente ESP32

```
1   // LIBRERIAS
2
3     // SENSOR
4       #include <Adafruit_LSM9DS1.h>
5
6     // BLE
7       #include <BLEDevice.h>
8       #include <BLEServer.h>
9       #include <BLEUtils.h>
10      #include <BLE2902.h>
11
12
13  // VARIABLES
14
15
16    // SENSOR
17      // creamos el sensor
18      Adafruit_LSM9DS1 sensor = Adafruit_LSM9DS1();
19
20      // variables para calcular la posicion del norte geografico con el
21  magnetometro
22      //float declinacion=0.46;
23      //float angulo=0;
24
25      // funcion para configurar la sensibilidad del sensor
26      void setupSensor(){
27        // configuramos la sensibilidad del acelerometro (2G, 4G, 8G,
28  16G)
29        sensor.setupAccel(sensor.LSM9DS1_ACCELRANGE_2G);
30
31        // configuramos la sensibilidad del magnetometro (4GAUSS,
32  8GAUSS, 12GAUSS, 16GAUSS)
33        sensor.setupMag(sensor.LSM9DS1_MAGGAIN_4GAUSS);
34
35        // configuramos la sensibilidad del giroscopio (245DPS, 500DPS,
36  2000DPS)
37        sensor.setupGyro(sensor.LSM9DS1_GYROSCALE_245DPS);
38      }
39
40    // BLE
41      // creamos el servidor BLE
42      BLEServer* servidor = NULL;
```

```
43
44      // creamos las caracteristicas BLE
45
46        BLECharacteristic* caracteristica_acelerometro_x = NULL;
47        BLECharacteristic* caracteristica_acelerometro_y = NULL;
48        BLECharacteristic* caracteristica_acelerometro_z = NULL;
49        BLECharacteristic* caracteristica_acelerometro_sensibilidad =
50  NULL;
51
52        BLECharacteristic* caracteristica_magnetometro_x = NULL;
53        BLECharacteristic* caracteristica_magnetometro_y = NULL;
54        BLECharacteristic* caracteristica_magnetometro_z = NULL;
55        BLECharacteristic* caracteristica_magnetometro_sensibilidad =
56  NULL;
57
58        BLECharacteristic* caracteristica_giroscopio_x = NULL;
59        BLECharacteristic* caracteristica_giroscopio_y = NULL;
60        BLECharacteristic* caracteristica_giroscopio_z = NULL;
61        BLECharacteristic* caracteristica_giroscopio_sensibilidad =
62  NULL;
63
64        BLECharacteristic* caracteristica_configuracion_nombre = NULL;
65        BLECharacteristic* caracteristica_configuracion_muestreo = NULL;
66        BLECharacteristic* caracteristica_configuracion_sensibilidad =
67  NULL;
68
69
70
71
72      // creamos los descriptores BLE
73
74        BLEDescriptor* descriptor_acelerometro_x = new BLE2902();
75        BLEDescriptor* descriptor_acelerometro_y = new BLE2902();
76        BLEDescriptor* descriptor_acelerometro_z = new BLE2902();
77        BLEDescriptor* descriptor_acelerometro_sensibilidad = new
78  BLE2902();
79
80        BLEDescriptor* descriptor_magnetometro_x = new BLE2902();
81        BLEDescriptor* descriptor_magnetometro_y = new BLE2902();
82        BLEDescriptor* descriptor_magnetometro_z = new BLE2902();
83        BLEDescriptor* descriptor_magnetometro_sensibilidad = new
84  BLE2902();
85
86        BLEDescriptor* descriptor_giroscopio_x = new BLE2902();
87        BLEDescriptor* descriptor_giroscopio_y = new BLE2902();
88        BLEDescriptor* descriptor_giroscopio_z = new BLE2902();
89        BLEDescriptor* descriptor_giroscopio_sensibilidad = new
90  BLE2902();
91
92        BLEDescriptor* descriptor_configuracion_nombre = new BLE2902();
93        BLEDescriptor* descriptor_configuracion_muestreo = new
94  BLE2902();
95        BLEDescriptor* descriptor_configuracion_sensibilidad = new
96  BLE2902();
```

```
 97
 98
 99
100      // creamos una variable para saber si hay clientes conectados al
101  servidor
102      bool cliente_conectado_al_servidor = false;
103
104      // creamos una variable para configurar el tiempo de muestreo del
105  sensor en milisegundos
106      int muestreo = 200;
107
108      // creamos una variable para conocer la sensibilidad del sensor
109      int sensibilidad = 1;
110
111      // creamos los UUID de los servicios y de las caracteristicas
112        // acelerómetro
113        #define uuid_servicio_acelerometro
114  "00000001-0001-0001-0001-000000000001"
115        #define uuid_caracteristica_acelerometro_x
116  "00000001-0001-0001-0001-000000000002"
117        #define uuid_caracteristica_acelerometro_y
118  "00000001-0001-0001-0001-000000000003"
119        #define uuid_caracteristica_acelerometro_z
120  "00000001-0001-0001-0001-000000000004"
121        #define uuid_caracteristica_acelerometro_sensibilidad
122  "00000001-0001-0001-0001-000000000005"
123
124        // magnetómetro
125        #define uuid_servicio_magnetometro
126  "00000001-0001-0001-0002-000000000001"
127        #define uuid_caracteristica_magnetometro_x
128  "00000001-0001-0001-0002-000000000002"
129        #define uuid_caracteristica_magnetometro_y
130  "00000001-0001-0001-0002-000000000003"
131        #define uuid_caracteristica_magnetometro_z
132  "00000001-0001-0001-0002-000000000004"
133        #define uuid_caracteristica_magnetometro_sensibilidad
134  "00000001-0001-0001-0002-000000000005"
135
136        // giroscopio
137        #define uuid_servicio_giroscopio
138  "00000001-0001-0001-0003-000000000001"
139        #define uuid_caracteristica_giroscopio_x
140  "00000001-0001-0001-0003-000000000002"
141        #define uuid_caracteristica_giroscopio_y
142  "00000001-0001-0001-0003-000000000003"
143        #define uuid_caracteristica_giroscopio_z
144  "00000001-0001-0001-0003-000000000004"
145        #define uuid_caracteristica_giroscopio_sensibilidad
146  "00000001-0001-0001-0003-000000000005"
147
148        // configuracion
149        #define uuid_servicio_configuracion
150  "00000001-0001-0001-0004-000000000001"
```

```
151       #define uuid_caracteristica_configuracion_nombre
152   "00000001-0001-0001-0004-000000000002"
153       #define uuid_caracteristica_configuracion_muestreo
154   "00000001-0001-0001-0004-000000000003"
155       #define uuid_caracteristica_configuracion_sensibilidad
156   "00000001-0001-0001-0004-000000000004"
157
158
159
160
161       // funcion para cuando un cliente se conecta o se desconecta del
162   servidor
163       class el_servidor_ha_detectado_un_cliente: public
164   BLEServerCallbacks {
165
166         // cliente conectado al servidor
167         void onConnect(BLEServer* servidor) {
168
169           // actualizamos la variable
170           cliente_conectado_al_servidor = true;
171           };
172
173         // cliente desconectado del servidor
174         void onDisconnect(BLEServer* servidor) {
175
176           // actualizamos la variable
177           cliente_conectado_al_servidor = false;
178           }
179       };
180
181
182       // funcion para cuando un cliente selecciona (escribe) la
183   sensibilidad del sensor o cambia el nombre
184       class el_cliente_ha_escrito_en_la_caracteristica: public
185   BLECharacteristicCallbacks {
186           void onWrite(BLECharacteristic *caracteristica_1) {
187
188             // ACELEROMETRO
189               // guardamos la sensibilidad configurada por el cliente
190   para el acelerometro
191               std::string acelerometro_sensibilidad =
192   caracteristica_acelerometro_sensibilidad->getValue();
193
194               // configuramos la sensibilidad del acelerometro en
195   funcion de la opcion elegida por el cliente
196               if(acelerometro_sensibilidad=="1"){
197                 sensor.setupAccel(sensor.LSM9DS1_ACCELRANGE_2G);
198
199                   // actualizamos el valor de la caracteristica con el
200   nuevo valor de la variable sensibilidad, convirtiendolo a un string
201                   sensibilidad=1;
202                   char valor_configuracion_sensibilidad[8];
203                   dtostrf(sensibilidad, 1, 0,
204   valor_configuracion_sensibilidad);
```

```
205                 caracteristica_configuracion_sensibilidad-
206   >setValue(valor_configuracion_sensibilidad);
207
208               } else if(acelerometro_sensibilidad=="2"){
209                 sensor.setupAccel(sensor.LSM9DS1_ACCELRANGE_4G);
210
211                 // actualizamos el valor de la caracteristica con el
212   nuevo valor de la variable sensibilidad, convirtiendolo a un string
213                 sensibilidad=2;
214                 char valor_configuracion_sensibilidad[8];
215                 dtostrf(sensibilidad, 1, 0,
216   valor_configuracion_sensibilidad);
217                 caracteristica_configuracion_sensibilidad-
218   >setValue(valor_configuracion_sensibilidad);
219
220               } else if(acelerometro_sensibilidad=="3"){
221                 sensor.setupAccel(sensor.LSM9DS1_ACCELRANGE_8G);
222
223                 // actualizamos el valor de la caracteristica con el
224   nuevo valor de la variable sensibilidad, convirtiendolo a un string
225                 sensibilidad=3;
226                 char valor_configuracion_sensibilidad[8];
227                 dtostrf(sensibilidad, 1, 0,
228   valor_configuracion_sensibilidad);
229                 caracteristica_configuracion_sensibilidad-
230   >setValue(valor_configuracion_sensibilidad);
231
232               } else if(acelerometro_sensibilidad=="4"){
233                 sensor.setupAccel(sensor.LSM9DS1_ACCELRANGE_16G);
234
235                 // actualizamos el valor de la caracteristica con el
236   nuevo valor de la variable sensibilidad, convirtiendolo a un string
237                 sensibilidad=4;
238                 char valor_configuracion_sensibilidad[8];
239                 dtostrf(sensibilidad, 1, 0,
240   valor_configuracion_sensibilidad);
241                 caracteristica_configuracion_sensibilidad-
242   >setValue(valor_configuracion_sensibilidad);
243
244               } else {
245                 // si el cliente no ha cambiado la sensibilidad, no toco
246   nada
247               }
248
249
250             // MAGNETOMETRO
251               // guardamos la sensibilidad configurada por el cliente
252   para el magnetometro
253               std::string magnetometro_sensibilidad =
254   caracteristica_magnetometro_sensibilidad->getValue();
255
256               // configuramos la sensibilidad del magnetometro en
257   funcion de la opcion elegida por el cliente
258               if(magnetometro_sensibilidad=="1"){
```

```
259               sensor.setupMag(sensor.LSM9DS1_MAGGAIN_4GAUSS);
260
261             } else if(magnetometro_sensibilidad=="2"){
262               sensor.setupMag(sensor.LSM9DS1_MAGGAIN_8GAUSS);
263
264             } else if(magnetometro_sensibilidad=="3"){
265               sensor.setupMag(sensor.LSM9DS1_MAGGAIN_12GAUSS);
266
267             } else if(magnetometro_sensibilidad=="4"){
268               sensor.setupMag(sensor.LSM9DS1_MAGGAIN_16GAUSS);
269
270             } else {
271               // si el cliente no ha cambiado la sensibilidad, no toco
272   nada
273             }
274
275
276           // GIROSCOPIO
277             // guardamos la sensibilidad configurada por el cliente
278   para el giroscopio
279             std::string giroscopio_sensibilidad =
280   caracteristica_giroscopio_sensibilidad->getValue();
281
282             // configuramos la sensibilidad del giroscopio en funcion
283   de la opcion elegida por el cliente
284             if(giroscopio_sensibilidad=="1"){
285               sensor.setupGyro(sensor.LSM9DS1_GYROSCALE_245DPS);
286
287             } else if(giroscopio_sensibilidad=="2"){
288               sensor.setupGyro(sensor.LSM9DS1_GYROSCALE_500DPS);
289
290             } else if(giroscopio_sensibilidad=="3"){
291               sensor.setupGyro(sensor.LSM9DS1_GYROSCALE_2000DPS);
292
293             } else {
294               // si el cliente no ha cambiado la sensibilidad, no toco
295   nada
296             }
297
298
299           // NOMBRE
300             // guardamos el nuevo nombre configurado por el cliente
301             std::string configuracion_nombre =
302   caracteristica_configuracion_nombre->getValue();
303
304             // creamos el array de caracteres
305             char nombre[configuracion_nombre.length()+1];
306
307             // convertimos el string en un array de caracteres
308             strcpy(nombre, configuracion_nombre.c_str());
309
310             // configuramos el nuevo nombre
311             esp_ble_gap_set_device_name(nombre);
312
```

```
313
314           // MUESTREO
315             // guardamos el muestreo configurado por el cliente
316             std::string configuracion_muestreo =
317   caracteristica_configuracion_muestreo->getValue();
318
319             // convertimos el string en un int
320             muestreo = std::stoi(configuracion_muestreo);
321
322             caracteristica_configuracion_muestreo-
323   >setValue(configuracion_muestreo);
324
325
326         }
327     };
328
329
330   // SETUP
331   void setup(){
332
333     // SENSOR
334       // iniciamos la comunicacion serie con el ESP32 para poder leer
335   los datos en el monitor serie
336       Serial.begin(115200);
337
338       // iniciamos el sensor
339       sensor.begin();
340
341       // configuramos la escala y la sensibilidad del sensor
342       setupSensor();
343
344
345     // BLE
346       // creamos el dispositivo BLE
347       BLEDevice::init("ESP32");
348
349       // creamos el servidor BLE
350       servidor = BLEDevice::createServer();
351
352       // le decimos al servidor que nos avise cuando se conecte o se
353   desconecte un cliente
354       servidor->setCallbacks(new el_servidor_ha_detectado_un_cliente());
355
356
357       // ACELEROMETRO
358         // creamos el servicio del acelerometro
359         BLEService *servicio_acelerometro = servidor-
360   >createService(uuid_servicio_acelerometro);
361
362         // EJE X
363           // creamos la caracteristica en la que guardaremos el valor
364   del eje x del acelerometro
365           caracteristica_acelerometro_x = servicio_acelerometro-
366   >createCharacteristic(
```

```
                              uuid_caracteristica_acelerometro_x,
                              BLECharacteristic::PROPERTY_READ   |
                              BLECharacteristic::PROPERTY_WRITE  |
                              BLECharacteristic::PROPERTY_NOTIFY |
                              BLECharacteristic::PROPERTY_INDICATE
                            );

        // le decimos a la caracteristica en la que guardaremos el
valor del eje x del acelerometro que nos avise cuando el cliente
escriba
        caracteristica_acelerometro_x->setCallbacks(new
el_cliente_ha_escrito_en_la_caracteristica());

        // creamos el descriptor para poder identificar a la
caracteristica en la que guardaremos el valor del eje x del
acelerometro
        caracteristica_acelerometro_x-
>addDescriptor(descriptor_acelerometro_x);
        descriptor_acelerometro_x-
>setValue("caracteristica_acelerometro_x");

      // EJE Y
        // creamos la caracteristica en la que guardaremos el valor
del eje y del acelerometro
        caracteristica_acelerometro_y = servicio_acelerometro-
>createCharacteristic(
                              uuid_caracteristica_acelerometro_y,
                              BLECharacteristic::PROPERTY_READ   |
                              BLECharacteristic::PROPERTY_WRITE  |
                              BLECharacteristic::PROPERTY_NOTIFY |
                              BLECharacteristic::PROPERTY_INDICATE
                            );

        // le decimos a la caracteristica en la que guardaremos el
valor del eje y del acelerometro que nos avise cuando el cliente
escriba
        caracteristica_acelerometro_y->setCallbacks(new
el_cliente_ha_escrito_en_la_caracteristica());

        // creamos el descriptor para poder identificar a la
caracteristica en la que guardaremos el valor del eje y del
acelerometro
        caracteristica_acelerometro_y-
>addDescriptor(descriptor_acelerometro_y);
        descriptor_acelerometro_y-
>setValue("caracteristica_acelerometro_y");

      // EJE Z
        // creamos la caracteristica en la que guardaremos el valor
del eje z del acelerometro
        caracteristica_acelerometro_z = servicio_acelerometro-
>createCharacteristic(
                              uuid_caracteristica_acelerometro_z,
                              BLECharacteristic::PROPERTY_READ   |
```

```
421                             BLECharacteristic::PROPERTY_WRITE  |
422                             BLECharacteristic::PROPERTY_NOTIFY |
423                             BLECharacteristic::PROPERTY_INDICATE
424                           );
425
426         // le decimos a la caracteristica en la que guardaremos el
427   valor del eje z del acelerometro que nos avise cuando el cliente
428   escriba
429         caracteristica_acelerometro_z->setCallbacks(new
430   el_cliente_ha_escrito_en_la_caracteristica());
431
432         // creamos el descriptor para poder identificar a la
433   caracteristica en la que guardaremos el valor del eje z del
434   acelerometro
435         caracteristica_acelerometro_z-
436   >addDescriptor(descriptor_acelerometro_z);
437         descriptor_acelerometro_z-
438   >setValue("caracteristica_acelerometro_z");
439
440       // SENSIBILIDAD
441         // creamos la caracteristica en la que guardaremos el valor de
442   la sensibilidad del acelerometro
443         caracteristica_acelerometro_sensibilidad =
444   servicio_acelerometro->createCharacteristic(
445
446   uuid_caracteristica_acelerometro_sensibilidad,
447                             BLECharacteristic::PROPERTY_READ   |
448                             BLECharacteristic::PROPERTY_WRITE  |
449                             BLECharacteristic::PROPERTY_NOTIFY |
450                             BLECharacteristic::PROPERTY_INDICATE
451                           );
452
453         // le decimos a la caracteristica en la que guardaremos el
454   valor de la sensibilidad del acelerometro que nos avise cuando el
455   cliente escriba
456         caracteristica_acelerometro_sensibilidad->setCallbacks(new
457   el_cliente_ha_escrito_en_la_caracteristica());
458
459         // creamos el descriptor para poder identificar a la
460   caracteristica en la que guardaremos el valor de la sensibilidad del
461   acelerometro
462         caracteristica_acelerometro_sensibilidad-
463   >addDescriptor(descriptor_acelerometro_sensibilidad);
464         descriptor_acelerometro_sensibilidad-
465   >setValue("caracteristica_acelerometro_sensibilidad");
466
467       // arrancamos el servicio
468       servicio_acelerometro->start();
469
470
471     // MAGNETOMETRO
472       // creamos el servicio del acelerometro
473       BLEService *servicio_magnetometro = servidor-
474   >createService(uuid_servicio_magnetometro);
```

```
475
476       // EJE X
477         // creamos la caracteristica en la que guardaremos el valor
478   del eje x del magnetometro
479         caracteristica_magnetometro_x = servicio_magnetometro-
480   >createCharacteristic(
481                             uuid_caracteristica_magnetometro_x,
482                             BLECharacteristic::PROPERTY_READ   |
483                             BLECharacteristic::PROPERTY_WRITE  |
484                             BLECharacteristic::PROPERTY_NOTIFY |
485                             BLECharacteristic::PROPERTY_INDICATE
486                           );
487
488         // le decimos a la caracteristica en la que guardaremos el
489   valor del eje x del magnetometro que nos avise cuando el cliente
490   escriba
491         caracteristica_magnetometro_x->setCallbacks(new
492   el_cliente_ha_escrito_en_la_caracteristica());
493
494         // creamos el descriptor para poder identificar a la
495   caracteristica en la que guardaremos el valor del eje x del
496   magnetometro
497         caracteristica_magnetometro_x-
498   >addDescriptor(descriptor_magnetometro_x);
499         descriptor_magnetometro_x-
500   >setValue("caracteristica_magnetometro_x");
501
502       // EJE Y
503         // creamos la caracteristica en la que guardaremos el valor
504   del eje y del magnetometro
505         caracteristica_magnetometro_y = servicio_magnetometro-
506   >createCharacteristic(
507                             uuid_caracteristica_magnetometro_y,
508                             BLECharacteristic::PROPERTY_READ   |
509                             BLECharacteristic::PROPERTY_WRITE  |
510                             BLECharacteristic::PROPERTY_NOTIFY |
511                             BLECharacteristic::PROPERTY_INDICATE
512                           );
513
514         // le decimos a la caracteristica en la que guardaremos el
515   valor del eje y del magnetometro que nos avise cuando el cliente
516   escriba
517         caracteristica_magnetometro_y->setCallbacks(new
518   el_cliente_ha_escrito_en_la_caracteristica());
519
520         // creamos el descriptor para poder identificar a la
521   caracteristica en la que guardaremos el valor del eje y del
522   magnetometro
523         caracteristica_magnetometro_y-
524   >addDescriptor(descriptor_magnetometro_y);
525         descriptor_magnetometro_y-
526   >setValue("caracteristica_magnetometro_y");
527
528       // EJE Z
```

```
529         // creamos la caracteristica en la que guardaremos el valor
530   del eje z del magnetometro
531         caracteristica_magnetometro_z = servicio_magnetometro-
532   >createCharacteristic(
533                               uuid_caracteristica_magnetometro_z,
534                               BLECharacteristic::PROPERTY_READ   |
535                               BLECharacteristic::PROPERTY_WRITE  |
536                               BLECharacteristic::PROPERTY_NOTIFY |
537                               BLECharacteristic::PROPERTY_INDICATE
538                             );
539
540         // le decimos a la caracteristica en la que guardaremos el
541   valor del eje z del magnetometro que nos avise cuando el cliente
542   escriba
543         caracteristica_magnetometro_z->setCallbacks(new
544   el_cliente_ha_escrito_en_la_caracteristica());
545
546         // creamos el descriptor para poder identificar a la
547   caracteristica en la que guardaremos el valor del eje z del
548   magnetometro
549         caracteristica_magnetometro_z-
550   >addDescriptor(descriptor_magnetometro_z);
551         descriptor_magnetometro_z-
552   >setValue("caracteristica_magnetometro_z");
553
554       // SENSIBILIDAD
555         // creamos la caracteristica en la que guardaremos el valor de
556   la sensibilidad del magnetometro
557         caracteristica_magnetometro_sensibilidad =
558   servicio_magnetometro->createCharacteristic(
559
560   uuid_caracteristica_magnetometro_sensibilidad,
561                               BLECharacteristic::PROPERTY_READ   |
562                               BLECharacteristic::PROPERTY_WRITE  |
563                               BLECharacteristic::PROPERTY_NOTIFY |
564                               BLECharacteristic::PROPERTY_INDICATE
565                             );
566
567         // le decimos a la caracteristica en la que guardaremos el
568   valor de la sensibilidad del magnetometro que nos avise cuando el
569   cliente escriba
570         caracteristica_magnetometro_sensibilidad->setCallbacks(new
571   el_cliente_ha_escrito_en_la_caracteristica());
572
573         // creamos el descriptor para poder identificar a la
574   caracteristica en la que guardaremos el valor de la sensibilidad del
575   magnetometro
576         caracteristica_magnetometro_sensibilidad-
577   >addDescriptor(descriptor_magnetometro_sensibilidad);
578         descriptor_magnetometro_sensibilidad-
579   >setValue("caracteristica_magnetometro_sensibilidad");
580
581       // arrancamos el servicio
582       servicio_magnetometro->start();
```

```
583
584
585       // GIROSCOPIO
586         // creamos el servicio del giroscopio
587         BLEService *servicio_giroscopio = servidor-
588   >createService(uuid_servicio_giroscopio);
589
590         // EJE X
591           // creamos la caracteristica en la que guardaremos el valor
592   del eje x del giroscopio
593           caracteristica_giroscopio_x = servicio_giroscopio-
594   >createCharacteristic(
595                               uuid_caracteristica_giroscopio_x,
596                               BLECharacteristic::PROPERTY_READ   |
597                               BLECharacteristic::PROPERTY_WRITE  |
598                               BLECharacteristic::PROPERTY_NOTIFY |
599                               BLECharacteristic::PROPERTY_INDICATE
600                             );
601
602           // le decimos a la caracteristica en la que guardaremos el
603   valor del eje x del giroscopio que nos avise cuando el cliente escriba
604           caracteristica_giroscopio_x->setCallbacks(new
605   el_cliente_ha_escrito_en_la_caracteristica());
606
607           // creamos el descriptor para poder identificar a la
608   caracteristica en la que guardaremos el valor del eje x del giroscopio
609           caracteristica_giroscopio_x-
610   >addDescriptor(descriptor_giroscopio_x);
611           descriptor_giroscopio_x-
612   >setValue("caracteristica_giroscopio_x");
613
614         // EJE Y
615           // creamos la caracteristica en la que guardaremos el valor
616   del eje y del giroscopio
617           caracteristica_giroscopio_y = servicio_giroscopio-
618   >createCharacteristic(
619                               uuid_caracteristica_giroscopio_y,
620                               BLECharacteristic::PROPERTY_READ   |
621                               BLECharacteristic::PROPERTY_WRITE  |
622                               BLECharacteristic::PROPERTY_NOTIFY |
623                               BLECharacteristic::PROPERTY_INDICATE
624                             );
625
626           // le decimos a la caracteristica en la que guardaremos el
627   valor del eje y del giroscopio que nos avise cuando el cliente escriba
628           caracteristica_giroscopio_y->setCallbacks(new
629   el_cliente_ha_escrito_en_la_caracteristica());
630
631           // creamos el descriptor para poder identificar a la
632   caracteristica en la que guardaremos el valor del eje y del giroscopio
633           caracteristica_giroscopio_y-
634   >addDescriptor(descriptor_giroscopio_y);
635           descriptor_giroscopio_y-
636   >setValue("caracteristica_giroscopio_y");
```

```
637
638         // EJE Z
639           // creamos la caracteristica en la que guardaremos el valor
640   del eje z del giroscopio
641           caracteristica_giroscopio_z = servicio_giroscopio-
642   >createCharacteristic(
643                               uuid_caracteristica_giroscopio_z,
644                               BLECharacteristic::PROPERTY_READ   |
645                               BLECharacteristic::PROPERTY_WRITE  |
646                               BLECharacteristic::PROPERTY_NOTIFY |
647                               BLECharacteristic::PROPERTY_INDICATE
648                             );
649
650           // le decimos a la caracteristica en la que guardaremos el
651   valor del eje z del giroscopio que nos avise cuando el cliente escriba
652           caracteristica_giroscopio_z->setCallbacks(new
653   el_cliente_ha_escrito_en_la_caracteristica());
654
655           // creamos el descriptor para poder identificar a la
656   caracteristica en la que guardaremos el valor del eje z del giroscopio
657           caracteristica_giroscopio_z-
658   >addDescriptor(descriptor_giroscopio_z);
659           descriptor_giroscopio_z-
660   >setValue("caracteristica_giroscopio_z");
661
662         // SENSIBILIDAD
663           // creamos la caracteristica en la que guardaremos el valor de
664   la sensibilidad del giroscopio
665           caracteristica_giroscopio_sensibilidad = servicio_giroscopio-
666   >createCharacteristic(
667
668   uuid_caracteristica_giroscopio_sensibilidad,
669                               BLECharacteristic::PROPERTY_READ   |
670                               BLECharacteristic::PROPERTY_WRITE  |
671                               BLECharacteristic::PROPERTY_NOTIFY |
672                               BLECharacteristic::PROPERTY_INDICATE
673                             );
674
675           // le decimos a la caracteristica en la que guardaremos el
676   valor de la sensibilidad del giroscopio que nos avise cuando el
677   cliente escriba
678           caracteristica_giroscopio_sensibilidad->setCallbacks(new
679   el_cliente_ha_escrito_en_la_caracteristica());
680
681           // creamos el descriptor para poder identificar a la
682   caracteristica en la que guardaremos el valor de la sensibilidad del
683   giroscopio
684           caracteristica_giroscopio_sensibilidad-
685   >addDescriptor(descriptor_giroscopio_sensibilidad);
686           descriptor_giroscopio_sensibilidad-
687   >setValue("caracteristica_giroscopio_sensibilidad");
688
689         // arrancamos el servicio
690         servicio_giroscopio->start();
```

```
691
692
693       // CONFIGURACION
694
695         // creamos el servicio de la configuracion
696         BLEService *servicio_configuracion = servidor-
697   >createService(uuid_servicio_configuracion);
698
699
700         // NOMBRE
701           // creamos la caracteristica en la que guardaremos el valor
702   del nuevo nombre
703           caracteristica_configuracion_nombre = servicio_configuracion-
704   >createCharacteristic(
705                               uuid_caracteristica_configuracion_nombre,
706                               BLECharacteristic::PROPERTY_READ   |
707                               BLECharacteristic::PROPERTY_WRITE  |
708                               BLECharacteristic::PROPERTY_NOTIFY |
709                               BLECharacteristic::PROPERTY_INDICATE
710                             );
711
712           // le decimos a la caracteristica en la que guardaremos el
713   valor del nombre que nos avise cuando el cliente escriba
714           caracteristica_configuracion_nombre->setCallbacks(new
715   el_cliente_ha_escrito_en_la_caracteristica());
716
717           // creamos el descriptor para poder identificar a la
718   caracteristica en la que guardaremos el valor del nombre
719           caracteristica_configuracion_nombre-
720   >addDescriptor(descriptor_configuracion_nombre);
721           descriptor_configuracion_nombre-
722   >setValue("caracteristica_configuracion_nombre");
723
724
725         // MUESTREO
726           // creamos la caracteristica en la que guardaremos el valor
727   del nuevo nombre
728           caracteristica_configuracion_muestreo =
729   servicio_configuracion->createCharacteristic(
730
731   uuid_caracteristica_configuracion_muestreo,
732                               BLECharacteristic::PROPERTY_READ   |
733                               BLECharacteristic::PROPERTY_WRITE  |
734                               BLECharacteristic::PROPERTY_NOTIFY |
735                               BLECharacteristic::PROPERTY_INDICATE
736                             );
737
738           // le decimos a la caracteristica en la que guardaremos el
739   valor del muestreo que nos avise cuando el cliente escriba
740           caracteristica_configuracion_muestreo->setCallbacks(new
741   el_cliente_ha_escrito_en_la_caracteristica());
742
743           // creamos el descriptor para poder identificar a la
744   caracteristica en la que guardaremos el valor del muestreo
```

```
745         caracteristica_configuracion_muestreo-
746 >addDescriptor(descriptor_configuracion_muestreo);
747         descriptor_configuracion_muestreo-
748 >setValue("caracteristica_configuracion_muestreo");
749 
750         // inicializamos el valor de la caracteristica con el valor
751 actual de la variable muestreo, convirtiendolo a un string
752         char valor_configuracion_muestreo[8];
753         dtostrf(muestreo, 6, 0, valor_configuracion_muestreo);
754         caracteristica_configuracion_muestreo-
755 >setValue(valor_configuracion_muestreo);
756 
757 
758       // SENSIBILIDAD
759         // creamos la caracteristica en la que guardaremos el valor de
760 la sensibilidad actual
761         caracteristica_configuracion_sensibilidad =
762 servicio_configuracion->createCharacteristic(
763 
764 uuid_caracteristica_configuracion_sensibilidad,
765                             BLECharacteristic::PROPERTY_READ   |
766                             BLECharacteristic::PROPERTY_WRITE  |
767                             BLECharacteristic::PROPERTY_NOTIFY |
768                             BLECharacteristic::PROPERTY_INDICATE
769                           );
770 
771         // creamos el descriptor para poder identificar a la
772 caracteristica en la que guardaremos el valor del muestreo
773         caracteristica_configuracion_sensibilidad-
774 >addDescriptor(descriptor_configuracion_sensibilidad);
775         descriptor_configuracion_sensibilidad-
776 >setValue("caracteristica_configuracion_sensibilidad");
777 
778         // inicializamos el valor de la caracteristica con el valor
779 actual de la variable sensibilidad, convirtiendolo a un string
780         char valor_configuracion_sensibilidad[8];
781         dtostrf(sensibilidad, 1, 0, valor_configuracion_sensibilidad);
782         caracteristica_configuracion_sensibilidad-
783 >setValue(valor_configuracion_sensibilidad);
784 
785 
786       // arrancamos el servicio
787       servicio_configuracion->start();
788 
789 
790     // ponemos al dispositivo en modo advertising
791     BLEAdvertising *modo_advertising = BLEDevice::getAdvertising();
792     modo_advertising->addServiceUUID(uuid_servicio_acelerometro);
793     modo_advertising->addServiceUUID(uuid_servicio_magnetometro);
794     modo_advertising->addServiceUUID(uuid_servicio_giroscopio);
795     modo_advertising->addServiceUUID(uuid_servicio_configuracion);
796     modo_advertising->setScanResponse(false);
797     modo_advertising->setMinPreferred(0x0);
798     BLEDevice::startAdvertising();
```

```
799   }
800
801
802   // LOOP
803   void loop(){
804
805     // BLE
806
807       // clientes conectados al servidor
808       if (cliente_conectado_al_servidor) {
809
810           // SENSOR
811             // le decimos al sensor que realice una nueva toma de datos
812   del acelerometro, del magnetometro y del giroscopio
813             sensor.read();
814
815             // almacenamos los datos del sensor (la funcion incluida en
816   la libreria requiere que almacenemos tambien la temperatura, aunque el
817   sensor no tenga la capacidad de tomarla)
818             sensors_event_t a, m, g, temp;
819             sensor.getEvent(&a, &m, &g, &temp);
820
821
822             // ACELEROMETRO
823
824               // mostramos el valor del acelerometro
825               //Serial.print("Acelerometro X: ");
826   Serial.print(a.acceleration.x); Serial.print(" m/s^2");
827               //Serial.print("\tY: "); Serial.print(a.acceleration.y);
828   Serial.print(" m/s^2 ");
829               //Serial.print("\tZ: "); Serial.print(a.acceleration.z);
830   Serial.println(" m/s^2 ");
831
832               // EJE X
833                 // guardamos los datos del eje x del acelerometro en la
834   caracteristica, convirtiendolos a un string para poder guardar los
835   decimales y el signo
836                 char lectura_acelerometro_x[8];
837                 dtostrf(a.acceleration.x, 6, 2, lectura_acelerometro_x);
838                 caracteristica_acelerometro_x-
839   >setValue(lectura_acelerometro_x);
840
841                   // en caso de querer guardar un numero entero sin
842   signo
843                   //caracteristica_acelerometro_x-
844   >setValue((uint8_t*)&a.acceleration.x, 4);
845
846                 // notificamos al cliente el nuevo valor del eje x del
847   acelerometro
848                 caracteristica_acelerometro_x->notify();
849
850               // EJE Y
851                 // guardamos los datos del eje y del acelerometro en la
852   caracteristica
```

```
853                 char lectura_acelerometro_y[8];
854                 dtostrf(a.acceleration.y, 6, 2, lectura_acelerometro_y);
855                 caracteristica_acelerometro_y-
856   >setValue(lectura_acelerometro_y);
857
858                 // notificamos al cliente el nuevo valor del eje y del
859   acelerometro
860                 caracteristica_acelerometro_y->notify();
861
862               // EJE Z
863                 // guardamos los datos del eje z del acelerometro en la
864   caracteristica
865                 char lectura_acelerometro_z[8];
866                 dtostrf(a.acceleration.z, 6, 2, lectura_acelerometro_z);
867                 caracteristica_acelerometro_z-
868   >setValue(lectura_acelerometro_z);
869
870                 // notificamos al cliente el nuevo valor del eje z del
871   acelerometro
872                 caracteristica_acelerometro_z->notify();
873
874
875             // MAGNETOMETRO
876
877               // mostramos el valor del magnetometro
878               //Serial.print("Magnetometro X: ");
879   Serial.print(m.magnetic.x);   Serial.print(" uT");
880               //Serial.print("\tY: "); Serial.print(m.magnetic.y);
881   Serial.print(" uT");
882               //Serial.print("\tZ: "); Serial.print(m.magnetic.z);
883   Serial.println(" uT");
884
885               // CALCULOS POR SI QUEREMOS MOSTRAR EL ANGULO ENTRE EL
886   NORTE GEOGRAFICO Y EL EJE X DEL SENSOR
887                 // calculamos el ángulo entre el eje X del sensor y el
888   norte magnetico
889                 //angulo = atan2(m.magnetic.y, m.magnetic.x);
890
891                 // convertimos el angulo de radianes a grados
892                 //angulo=angulo*(180/M_PI);
893
894                 // tenemos en cuenta la declinacion magnetica para
895   calcular el angulo entre el eje x del sensor y el norte geografico
896                 //angulo=angulo-declinacion;
897
898                 // mostramos el angulo entre el eje x del sensor y el
899   norte geografico
900                 //Serial.print("Angulo entre el eje X y el norte
901   geografico: ");
902                 //Serial.print(angulo,0);
903
904               // EJE X
905                 // guardamos los datos del eje x del magnetometro en la
906   caracteristica
```

```
907                 char lectura_magnetometro_x[8];
908                 dtostrf(m.magnetic.x, 6, 2, lectura_magnetometro_x);
909                 caracteristica_magnetometro_x-
910   >setValue(lectura_magnetometro_x);
911
912                 // notificamos al cliente el nuevo valor del eje x del
913   magnetometro
914                 caracteristica_magnetometro_x->notify();
915
916               // EJE Y
917                 // guardamos los datos del eje y del magnetometro en la
918   caracteristica
919                 char lectura_magnetometro_y[8];
920                 dtostrf(m.magnetic.y, 6, 2, lectura_magnetometro_y);
921                 caracteristica_magnetometro_y-
922   >setValue(lectura_magnetometro_y);
923
924                 // notificamos al cliente el nuevo valor del eje y del
925   magnetometro
926                 caracteristica_magnetometro_y->notify();
927
928               // EJE Z
929                 // guardamos los datos del eje z del magnetometro en la
930   caracteristica
931                 char lectura_magnetometro_z[8];
932                 dtostrf(m.magnetic.z, 6, 2, lectura_magnetometro_z);
933                 caracteristica_magnetometro_z-
934   >setValue(lectura_magnetometro_z);
935
936                 // notificamos al cliente el nuevo valor del eje z del
937   magnetometro
938                 caracteristica_magnetometro_z->notify();
939
940
941             // GIROSCOPIO
942
943               // mostramos el valor del giroscopio
944               //Serial.print("Giroscopio   X: ");
945   Serial.print(g.gyro.x);   Serial.print(" rad/s");
946               //Serial.print("\tY: "); Serial.print(g.gyro.y);
947   Serial.print(" rad/s");
948               //Serial.print("\tZ: "); Serial.print(g.gyro.z);
949   Serial.println(" rad/s");
950
951               // EJE X
952                 // guardamos los datos del eje x del giroscopio en la
953   caracteristica
954                 char lectura_giroscopio_x[8];
955                 dtostrf(g.gyro.x, 6, 2, lectura_giroscopio_x);
956                 caracteristica_giroscopio_x-
957   >setValue(lectura_giroscopio_x);
958
959                 // notificamos al cliente el nuevo valor del eje x del
960   giroscopio
```

```
961               caracteristica_giroscopio_x->notify();
962
963             // EJE Y
964               // guardamos los datos del eje y del giroscopio en la
965   caracteristica
966               char lectura_giroscopio_y[8];
967               dtostrf(g.gyro.y, 6, 2, lectura_giroscopio_y);
968               caracteristica_giroscopio_y-
969   >setValue(lectura_giroscopio_y);
970
971               // notificamos al cliente el nuevo valor del eje y del
972   giroscopio
973               caracteristica_giroscopio_y->notify();
974
975             // EJE Z
976               // guardamos los datos del eje z del giroscopio en la
977   caracteristica
978               char lectura_giroscopio_z[8];
979               dtostrf(g.gyro.z, 6, 2, lectura_giroscopio_z);
980               caracteristica_giroscopio_z-
981   >setValue(lectura_giroscopio_z);
982
983               // notificamos al cliente el nuevo valor del eje z del
984   giroscopio
985               caracteristica_giroscopio_z->notify();
986
987
988
989         // esperamos antes de que el servidor le envie otra ronda de
990   datos al cliente
991         delay(muestreo);
992       }
993
994       // no hay clientes conectados al servidor
995       if (!cliente_conectado_al_servidor) {
996
997         // esperamos y ponemos al servidor en modo advertising
998         delay(200);
999
1000        servidor->startAdvertising();
1001      }
1002
1003  }
```

7.2 Código Fuente Android: Actividad_Principal.java

```
1   package elias.prueba;
2
3   import static java.lang.Integer.parseInt;
4   import static java.net.Proxy.Type.HTTP;
5
6   import android.Manifest;
7   import android.bluetooth.BluetoothAdapter;
8   import android.bluetooth.BluetoothGatt;
9   import android.bluetooth.BluetoothGattCharacteristic;
10  import android.bluetooth.BluetoothManager;
11  import android.content.BroadcastReceiver;
12  import android.content.ComponentName;
13  import android.content.Context;
14  import android.content.Intent;
15  import android.content.IntentFilter;
16  import android.content.ServiceConnection;
17  import android.content.pm.PackageManager;
18  import android.content.res.ColorStateList;
19  import android.graphics.Color;
20  import android.graphics.PorterDuff;
21  import android.icu.util.Output;
22  import android.os.AsyncTask;
23  import android.os.Bundle;
24  import android.app.Activity;
25  import android.os.IBinder;
26  import android.util.Log;
27  import android.view.MotionEvent;
28  import android.view.inputmethod.InputMethodManager;
29  import android.widget.Button;
30  import android.widget.EditText;
31  import android.view.View;
32  import android.widget.ImageButton;
33  import android.widget.LinearLayout;
34  import android.widget.TextView;
35
36  import androidx.annotation.NonNull;
37  import androidx.core.app.ActivityCompat;
38  import androidx.core.content.ContextCompat;
39
40  import com.jjoe64.graphview.DefaultLabelFormatter;
41  import com.jjoe64.graphview.GraphView;
42  import com.jjoe64.graphview.Viewport;
43  import com.jjoe64.graphview.series.DataPoint;
44  import com.jjoe64.graphview.series.LineGraphSeries;
45  import com.jjoe64.graphview.series.PointsGraphSeries;
46
47  import org.apache.http.params.HttpParams;
48  import org.json.JSONArray;
49  import org.json.JSONException;
50  import org.json.JSONObject;
51
```

```
52  import java.io.BufferedInputStream;
53  import java.io.BufferedOutputStream;
54  import java.io.BufferedReader;
55  import java.io.DataOutputStream;
56  import java.io.File;
57  import java.io.IOException;
58  import java.io.InputStream;
59  import java.io.InputStreamReader;
60  import java.io.OutputStream;
61  import java.io.OutputStreamWriter;
62  import java.net.HttpURLConnection;
63  import java.net.MalformedURLException;
64  import java.net.URL;
65  import java.nio.charset.Charset;
66  import java.nio.charset.StandardCharsets;
67  import java.security.KeyManagementException;
68  import java.security.KeyStore;
69  import java.security.NoSuchAlgorithmException;
70  import java.security.cert.CertificateException;
71  import java.security.cert.X509Certificate;
72  import java.text.NumberFormat;
73  import java.util.ArrayList;
74  import java.util.Iterator;
75  import java.util.List;
76  import java.util.Map;
77  import java.util.UUID;
78
79  import javax.net.ssl.HttpsURLConnection;
80  import javax.net.ssl.SSLContext;
81  import javax.net.ssl.SSLException;
82  import javax.net.ssl.SSLSocketFactory;
83  import javax.net.ssl.TrustManager;
84  import javax.net.ssl.X509TrustManager;
85
86  public class principal extends Activity {
87
88
89      // BLE
90
91
92          // BRAZO IZQUIERDO
93
94              // sensor al que me conecto
95                  final String direccion_MAC_brazo_izquierdo
96  ="78:21:84:7A:F5:D2";
97
98              // SECCIÓN donde se muestra el estado y el botón de
99  configuración del sensor
100                 private Button boton_brazo_izquierdo_estado;
101                 private ImageButton
102 boton_brazo_izquierdo_configuracion;
103
104             // SECCIÓN donde se muestran los ajustes del sensor
105                 private LinearLayout tarjeta_ajustes_brazo_izquierdo;
```

```
106                 private EditText
107   texto_configuracion_muestreo_brazo_izquierdo;
108                 private Boolean
109   mostrando_tarjeta_ajustes_brazo_izquierdo=false;
110                 private Button boton_2G_brazo_izquierdo;
111                 private Button boton_4G_brazo_izquierdo;
112                 private Button boton_8G_brazo_izquierdo;
113                 private Button boton_16G_brazo_izquierdo;
114
115             // SECCIÓN donde se muestran los datos del acelerometro
116                 private TextView texto_brazo_izquierdo_acelerometro_x;
117                 private TextView texto_brazo_izquierdo_acelerometro_y;
118                 private TextView texto_brazo_izquierdo_acelerometro_z;
119
120             // SECCIÓN donde se muestra el gráfico con los valores
121                 private GraphView grafico_brazo_izquierdo;
122
123                 private LineGraphSeries<DataPoint>
124   serie_datos_linea_brazo_izquierdo_aceleracion_x;
125                 private PointsGraphSeries<DataPoint>
126   serie_datos_punto_brazo_izquierdo_aceleracion_x;
127
128                 private LineGraphSeries<DataPoint>
129   serie_datos_linea_brazo_izquierdo_aceleracion_y;
130                 private PointsGraphSeries<DataPoint>
131   serie_datos_punto_brazo_izquierdo_aceleracion_y;
132
133                 private LineGraphSeries<DataPoint>
134   serie_datos_linea_brazo_izquierdo_aceleracion_z;
135                 private PointsGraphSeries<DataPoint>
136   serie_datos_punto_brazo_izquierdo_aceleracion_z;
137
138                 private boolean
139   primer_valor_grafica_brazo_izquierdo_aceleracion_x = true;
140                 private boolean
141   primer_valor_grafica_brazo_izquierdo_aceleracion_y = true;
142                 private boolean
143   primer_valor_grafica_brazo_izquierdo_aceleracion_z = true;
144
145                 private boolean
146   puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_x=false;
147                 private boolean
148   puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_y=false;
149                 private boolean
150   puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_z=false;
151
152                 private int
153   numero_valor_grafica_brazo_izquierdo_aceleracion_x = 0;
154                 private int
155   numero_valor_grafica_brazo_izquierdo_aceleracion_y = 0;
156                 private int
157   numero_valor_grafica_brazo_izquierdo_aceleracion_z = 0;
158
159
```

```
160                 private ImageButton boton_brazo_izquierdo_play;
161                 private ImageButton boton_brazo_izquierdo_pause;
162                 private ImageButton boton_brazo_izquierdo_stop;
163
164                 private Button
165   boton_brazo_izquierdo_mostrar_aceleracion_x;
166                 private Button
167   boton_brazo_izquierdo_mostrar_aceleracion_y;
168                 private Button
169   boton_brazo_izquierdo_mostrar_aceleracion_z;
170
171                 private boolean
172   boton_brazo_izquierdo_mostrar_aceleracion_x_pulsado=true;
173                 private boolean
174   boton_brazo_izquierdo_mostrar_aceleracion_y_pulsado=true;
175                 private boolean
176   boton_brazo_izquierdo_mostrar_aceleracion_z_pulsado=true;
177
178
179
180
181       // BRAZO DERECHO
182
183               // sensor al que me conecto
184                   final String direccion_MAC_brazo_derecho
185   ="78:21:84:7C:1E:6E";
186
187               // SECCIÓN donde se muestra el estado y el botón de
188   configuración del sensor
189                   private Button boton_brazo_derecho_estado;
190                   private ImageButton boton_brazo_derecho_configuracion;
191
192               // SECCIÓN donde se muestran los ajustes del sensor
193                   private LinearLayout tarjeta_ajustes_brazo_derecho;
194                   private EditText
195   texto_configuracion_muestreo_brazo_derecho;
196                   private Boolean
197   mostrando_tarjeta_ajustes_brazo_derecho=false;
198                   private Button boton_2G_brazo_derecho;
199                   private Button boton_4G_brazo_derecho;
200                   private Button boton_8G_brazo_derecho;
201                   private Button boton_16G_brazo_derecho;
202
203               // SECCIÓN donde se muestran los datos del acelerometro
204                   private TextView texto_brazo_derecho_acelerometro_x;
205                   private TextView texto_brazo_derecho_acelerometro_y;
206                   private TextView texto_brazo_derecho_acelerometro_z;
207
208               // SECCIÓN donde se muestra el gráfico con los valores
209                   private LineGraphSeries<DataPoint>
210   serie_datos_linea_brazo_derecho_aceleracion_x;
211                   private PointsGraphSeries<DataPoint>
212   serie_datos_punto_brazo_derecho_aceleracion_x;
213
```

```
214                 private LineGraphSeries<DataPoint>
215   serie_datos_linea_brazo_derecho_aceleracion_y;
216                 private PointsGraphSeries<DataPoint>
217   serie_datos_punto_brazo_derecho_aceleracion_y;
218
219                 private LineGraphSeries<DataPoint>
220   serie_datos_linea_brazo_derecho_aceleracion_z;
221                 private PointsGraphSeries<DataPoint>
222   serie_datos_punto_brazo_derecho_aceleracion_z;
223
224                 private boolean
225   primer_valor_grafica_brazo_derecho_aceleracion_x = true;
226                 private boolean
227   primer_valor_grafica_brazo_derecho_aceleracion_y = true;
228                 private boolean
229   primer_valor_grafica_brazo_derecho_aceleracion_z = true;
230
231                 private boolean
232   puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_x=false;
233                 private boolean
234   puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_y=false;
235                 private boolean
236   puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_z=false;
237
238                 private int
239   numero_valor_grafica_brazo_derecho_aceleracion_x = 0;
240                 private int
241   numero_valor_grafica_brazo_derecho_aceleracion_y = 0;
242                 private int
243   numero_valor_grafica_brazo_derecho_aceleracion_z = 0;
244
245                 private Button
246   boton_brazo_derecho_mostrar_aceleracion_x;
247                 private Button
248   boton_brazo_derecho_mostrar_aceleracion_y;
249                 private Button
250   boton_brazo_derecho_mostrar_aceleracion_z;
251
252                 private boolean
253   boton_brazo_derecho_mostrar_aceleracion_x_pulsado=true;
254                 private boolean
255   boton_brazo_derecho_mostrar_aceleracion_y_pulsado=true;
256                 private boolean
257   boton_brazo_derecho_mostrar_aceleracion_z_pulsado=true;
258
259
260       // INTERNET
261
262           // version JSON Object
263           private JSONObject servidor_JSON = new JSONObject();
264           private JSONArray servidor_JSON_updates = new JSONArray();
265           private JSONObject servidor_JSON_update = new JSONObject();
266           private DataOutputStream servidor_envio = null;
267
```

```
268         private boolean puedo_añadir_valores_JSON = false;
269         private int fields_rellenados_JSON = 0;
270         private int delta_t = 0;
271         private boolean brazo_izquierdo_conectado = false;
272         private boolean brazo_derecho_conectado = false;
273 
274         private URL servidor_direccion = null;
275         private HttpsURLConnection servidor_conexion = null;
276 
277 
278 
279 
280 
281     // Variables Bluetooth
282         public static BluetoothAdapter mBluetoothAdapter;
283         private BluetoothLeService mBluetoothLeService;
284 
285     // con mServiceConnection manejo los eventos generados por el
286 servicio BluetoothLeService
287         private final ServiceConnection mServiceConnection = new
288 ServiceConnection() {
289 
290             // cuando el servicio se haya creado
291             @Override
292             public void onServiceConnected(ComponentName
293 componentName, IBinder service) {
294                 // obtengo una referencia al servicio
295                 mBluetoothLeService =
296 ((BluetoothLeService.LocalBinder) service).getService();
297 
298                 // si el servicio se ha inicializado correctamente
299 llamo al método connect, para conectarme al dispositivo bluetooth
300                 Log.i("problema", "principal: connect izquierdo");
301 
302 mBluetoothLeService.connect_brazo_izquierdo(direccion_MAC_brazo_izquie
303 rdo);
304                 Log.i("problema", "principal: connect derecho");
305 
306 mBluetoothLeService.connect_brazo_derecho(direccion_MAC_brazo_derecho)
307 ;
308 
309             }
310 
311             // cuando el servicio se haya terminado
312             @Override
313             public void onServiceDisconnected(ComponentName
314 componentName) {
315                 // elimino mi referencia al (ya muerto) servicio
316                 mBluetoothLeService = null;
317             }
318         };
319 
320         // en este método manejo los broadcasts enviados desde el
321 BluetoothLeService
```

```
322         private final BroadcastReceiver mGattUpdateReceiver = new
323   BroadcastReceiver() {
324             @Override
325             public void onReceive(Context context, Intent intent) {
326
327                 // almaceno el dato que he recibido del intent que me
328   han enviado
329                 final String action = intent.getAction();
330
331
332
333                 // BRAZO IZQUIERDO
334
335                     // me he conectado a un dispositivo
336                     if
337   (BluetoothLeService.ACTION_GATT_CONNECTED_brazo_izquierdo.equals(actio
338   n)) {
339
340                         brazo_izquierdo_conectado=true;
341
342                         runOnUiThread(new Runnable() {
343                             @Override
344                             public void run() {
345
346   boton_brazo_izquierdo_estado.setBackgroundTintList(ColorStateList.valu
347   eOf(getColor(R.color.sensor_conectado)));
348
349   boton_brazo_izquierdo_configuracion.setVisibility(View.VISIBLE);
350                                     Log.i("problema", "principal:
351   conectado a izquierdo, estado: verde, configuracion: visible");
352
353                             }
354                         });
355
356
357                     // me he desconectado de un dispositivo
358                     } else if
359   (BluetoothLeService.ACTION_GATT_DISCONNECTED_brazo_izquierdo.equals(ac
360   tion)) {
361
362                         brazo_izquierdo_conectado=false;
363
364                         // actualizo la interfaz para que el usuario
365   vea que ya no estamos recibiendo valores, en lugar de mostrar el
366   ultimo valor recibido
367                         runOnUiThread(new Runnable() {
368                             @Override
369                             public void run() {
370
371
372   texto_brazo_izquierdo_acelerometro_x.setText(String.valueOf("X:" ));
373
374   texto_brazo_izquierdo_acelerometro_y.setText(String.valueOf("Y:" ));
```

```
375
376   texto_brazo_izquierdo_acelerometro_z.setText(String.valueOf("Z:" ));
377
378
379   boton_brazo_izquierdo_estado.setBackgroundTintList(ColorStateList.valu
380   eOf(getColor(R.color.sensor_no_conectado)));
381
382   boton_brazo_izquierdo_configuracion.setVisibility(View.GONE);
383
384   tarjeta_ajustes_brazo_izquierdo.setVisibility(LinearLayout.GONE);
385
386                                   Log.i("problema", "principal:
387   desconectado a izquierdo, estado: rojo, configuracion: oculta");
388
389
390                               }
391                           });
392
393                           // intento reconectarme al dispositivo
394
395   mBluetoothLeService.connect_brazo_izquierdo(direccion_MAC_brazo_izquie
396   rdo);
397
398
399                       // se ha terminado correctamente la busqueda de
400   servicios
401                       } else if
402   (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED_brazo_izquierdo.eq
403   uals(action)) {
404                           Log.i("problema", "principal: servicios
405   descubiertos brazo izquierdo");
406
407                           // obtengo el muestreo actual del dispositivo
408
409   mBluetoothLeService.conocer_muestreo_brazo_izquierdo();
410
411
412                       // acabo de recibir datos del dispositivo, ya sea
413   porque se ha terminado una operacion de lectura que yo he iniciado o
414   porque he recibido una notificacion desde el dispositivo
415                       } else if
416   (BluetoothLeService.ACTION_DATA_AVAILABLE_brazo_izquierdo.equals(actio
417   n)) {
418
419
420                           // ACELEROMETRO
421
422                               // comprobamos si la caracteristica de la
423   que hemos recibido la notificacion es la del acelerometro x
424
425   if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
426   lerometro_x")) {
427
```

```
                                    final String recibido_acelerometro_x =
intent.getExtras().getString("recibido_acelerometro_x");

                                                runOnUiThread(new
Runnable() {
                                                    @Override
                                                    public void run() {

texto_brazo_izquierdo_acelerometro_x.setText(String.valueOf("X: " +
recibido_acelerometro_x));

                                                    }
                                                });

                                                // convertir string a
float
                                                //Float
dato_a_actualizar_2=Float.parseFloat(dato_a_actualizar);



if(puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_x){

if(primer_valor_grafica_brazo_izquierdo_aceleracion_x){


primer_valor_grafica_brazo_izquierdo_aceleracion_x=false;

numero_valor_grafica_brazo_izquierdo_aceleracion_x=0;

                                                // reseteamos la serie de
datos de puntos

serie_datos_punto_brazo_izquierdo_aceleracion_x.resetData(new
DataPoint[] {});

                                                // reseteamos la serie de
datos de lineas

serie_datos_linea_brazo_izquierdo_aceleracion_x.resetData(new
DataPoint[] {});

                                            }

                                                // añadimos un valor a la
serie de datos de puntos

serie_datos_punto_brazo_izquierdo_aceleracion_x.appendData(new
DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_x,
Float.parseFloat(recibido_acelerometro_x)), true, 10000);

                                                // añadimos un valor a la
serie de datos de lineas
```

```
481
482   serie_datos_linea_brazo_izquierdo_aceleracion_x.appendData(new
483   DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_x,
484   Float.parseFloat(recibido_acelerometro_x)), true, 10000);
485
486
487
488   numero_valor_grafica_brazo_izquierdo_aceleracion_x++;
489
490                                       }
491
492
493
494
495                                   // THINGSPEAK
496
497                                   if(puedo_añadir_valores_JSON){
498
499                                       try{
500
501                                           // compruebo si el brazo
502   izquierdo y el brazo derecho están conectados
503
504   if(brazo_izquierdo_conectado==brazo_derecho_conectado){
505
506                                               // ya he rellenado todos
507   los valores de esta ronda de toma de datos
508
509   if(fields_rellenados_JSON==6){
510
511                                                   // asi que voy a meter
512   los valores en el array
513
514   servidor_JSON_updates.put(servidor_JSON_update);
515
516                                                   // reseteo las
517   variables para empezar una nueva ronda de toma de datos
518
519   fields_rellenados_JSON=0;
520
521   servidor_JSON_update=null;
522
523   servidor_JSON_update=new JSONObject();
524
525                                               }
526
527                                           // sólo está conectado el
528   brazo izquierdo
529                                           }else{
530
531                                               // ya he rellenado todos
532   los valores de esta ronda de toma de datos
533
534   if(fields_rellenados_JSON==3){
```

```
535
536                                                 // asi que voy a meter
537   los valores en el array
538
539   servidor_JSON_updates.put(servidor_JSON_update);
540
541                                                 // reseteo las
542   variables para empezar una nueva ronda de toma de datos
543
544   fields_rellenados_JSON=0;
545
546   servidor_JSON_update=null;
547
548   servidor_JSON_update=new JSONObject();
549
550   Log.i("json","servidor_JSON_updates = " +
551   servidor_JSON_updates.toString(4));
552
553                                             }
554                                         }
555
556                                         // compruebo si acabo de
557   empezar la toma de datos y por lo tanto tengo que incluir la marca de
558   tiempo
559                                         if(fields_rellenados_JSON==0){
560
561   servidor_JSON_update.put("delta_t", delta_t);
562                                             delta_t++;
563
564                                         }
565
566                                         // incluyo el valor del
567   acelerómetro en el JSON
568
569   servidor_JSON_update.put("field1",
570   Float.parseFloat(recibido_acelerometro_x));
571                                         fields_rellenados_JSON++;
572                                         Log.i("json","field1 = " +
573   Float.parseFloat(recibido_acelerometro_x));
574
575                                     } catch (JSONException e){
576                                         // excepcion JSON
577                                     }
578
579                                 }
580
581
582
583                             }
584
585
586                             // comprobamos si la caracteristica de la
587   que hemos recibido la notificacion es la del acelerometro y
```

```
588
589 if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
590 lerometro_y")) {
591
592                                 final String recibido_acelerometro_y =
593 intent.getExtras().getString("recibido_acelerometro_y");
594
595                                 runOnUiThread(new Runnable() {
596                                     @Override
597                                     public void run() {
598
599 texto_brazo_izquierdo_acelerometro_y.setText(String.valueOf("Y: " +
600 recibido_acelerometro_y));
601
602                                     }
603                                 });
604
605
606 if(puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_y){
607
608 if(primer_valor_grafica_brazo_izquierdo_aceleracion_y){
609
610
611 primer_valor_grafica_brazo_izquierdo_aceleracion_y=false;
612
613 numero_valor_grafica_brazo_izquierdo_aceleracion_y=0;
614
615                                         // reseteamos la serie de
616 datos de puntos
617
618 serie_datos_punto_brazo_izquierdo_aceleracion_y.resetData(new
619 DataPoint[] {});
620
621                                         // reseteamos la serie de
622 datos de lineas
623
624 serie_datos_linea_brazo_izquierdo_aceleracion_y.resetData(new
625 DataPoint[] {});
626
627                                     }
628
629                                     // añadimos un valor a la serie de
630 datos de puntos
631
632 serie_datos_punto_brazo_izquierdo_aceleracion_y.appendData(new
633 DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_y,
634 Float.parseFloat(recibido_acelerometro_y)), true, 10000);
635
636                                     // añadimos un valor a la serie de
637 datos de lineas
638
639 serie_datos_linea_brazo_izquierdo_aceleracion_y.appendData(new
640 DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_y,
641 Float.parseFloat(recibido_acelerometro_y)), true, 10000);
```

```
642
643
644
645   numero_valor_grafica_brazo_izquierdo_aceleracion_y++;
646
647                                   }
648
649
650
651                                   // THINGSPEAK
652
653                                   if(puedo_añadir_valores_JSON){
654
655                                       try{
656
657                                           // compruebo si el brazo
658   izquierdo y el brazo derecho están conectados
659
660   if(brazo_izquierdo_conectado==brazo_derecho_conectado){
661
662                                               // ya he rellenado todos
663   los valores de esta ronda de toma de datos
664
665   if(fields_rellenados_JSON==6){
666
667                                                   // asi que voy a meter
668   los valores en el array
669
670   servidor_JSON_updates.put(servidor_JSON_update);
671
672                                                   // reseteo las
673   variables para empezar una nueva ronda de toma de datos
674
675   fields_rellenados_JSON=0;
676
677   servidor_JSON_update=null;
678
679   servidor_JSON_update=new JSONObject();
680
681                                               }
682
683                                           // sólo está conectado el
684   brazo izquierdo
685                                           }else{
686
687                                               // ya he rellenado todos
688   los valores de esta ronda de toma de datos
689
690   if(fields_rellenados_JSON==3){
691
692                                                   // asi que voy a meter
693   los valores en el array
694
695   servidor_JSON_updates.put(servidor_JSON_update);
```

```
696
697                                                 // reseteo las
698 variables para empezar una nueva ronda de toma de datos
699
700 fields_rellenados_JSON=0;
701
702 servidor_JSON_update=null;
703
704 servidor_JSON_update=new JSONObject();
705
706 Log.i("json","servidor_JSON_updates = " +
707 servidor_JSON_updates.toString(4));
708
709
710                                             }
711                                         }
712
713                                         // compruebo si acabo de
714 empezar la toma de datos y por lo tanto tengo que incluir la marca de
715 tiempo
716                                         if(fields_rellenados_JSON==0){
717
718 servidor_JSON_update.put("delta_t", delta_t);
719                                             delta_t++;
720
721                                         }
722
723                                         // incluyo el valor del
724 acelerómetro en el JSON
725
726 servidor_JSON_update.put("field2",
727 Float.parseFloat(recibido_acelerometro_y));
728                                         fields_rellenados_JSON++;
729                                         Log.i("json","field2 = " +
730 Float.parseFloat(recibido_acelerometro_y));
731
732                                     } catch (JSONException e){
733                                         // excepcion JSON
734                                     }
735
736                                 }
737
738
739
740                             }
741
742                             // comprobamos si la caracteristica de la
743 que hemos recibido la notificacion es la del acelerometro z
744
745 if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
746 lerometro_z")) {
747
748                                 final String recibido_acelerometro_z =
749 intent.getExtras().getString("recibido_acelerometro_z");
```

```
750
751                                 runOnUiThread(new Runnable() {
752                                     @Override
753                                     public void run() {
754
755   texto_brazo_izquierdo_acelerometro_z.setText(String.valueOf("Z: " +
756   recibido_acelerometro_z));
757
758                                     }
759                                 });
760
761
762   if(puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_z){
763
764   if(primer_valor_grafica_brazo_izquierdo_aceleracion_z){
765
766
767   primer_valor_grafica_brazo_izquierdo_aceleracion_z=false;
768
769   numero_valor_grafica_brazo_izquierdo_aceleracion_z=0;
770
771                                         // reseteamos la serie de
772   datos de puntos
773
774   serie_datos_punto_brazo_izquierdo_aceleracion_z.resetData(new
775   DataPoint[] {});
776
777                                         // reseteamos la serie de
778   datos de lineas
779
780   serie_datos_linea_brazo_izquierdo_aceleracion_z.resetData(new
781   DataPoint[] {});
782
783                                     }
784
785                                     // añadimos un valor a la serie de
786   datos de puntos
787
788   serie_datos_punto_brazo_izquierdo_aceleracion_z.appendData(new
789   DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_z,
790   Float.parseFloat(recibido_acelerometro_z)), true, 10000);
791
792                                     // añadimos un valor a la serie de
793   datos de lineas
794
795   serie_datos_linea_brazo_izquierdo_aceleracion_z.appendData(new
796   DataPoint(numero_valor_grafica_brazo_izquierdo_aceleracion_z,
797   Float.parseFloat(recibido_acelerometro_z)), true, 10000);
798
799
800
801   numero_valor_grafica_brazo_izquierdo_aceleracion_z++;
802
803                                 }
```

```
804
805
806
807                                 // THINGSPEAK
808
809                                 if(puedo_añadir_valores_JSON){
810
811                                     try{
812
813                                         // compruebo si el brazo
814   izquierdo y el brazo derecho están conectados
815
816   if(brazo_izquierdo_conectado==brazo_derecho_conectado){
817
818                                             // ya he rellenado todos
819   los valores de esta ronda de toma de datos
820
821   if(fields_rellenados_JSON==6){
822
823                                                 // asi que voy a meter
824   los valores en el array
825
826   servidor_JSON_updates.put(servidor_JSON_update);
827
828                                                 // reseteo las
829   variables para empezar una nueva ronda de toma de datos
830
831   fields_rellenados_JSON=0;
832
833   servidor_JSON_update=null;
834
835   servidor_JSON_update=new JSONObject();
836
837                                             }
838
839                                         // sólo está conectado el
840   brazo izquierdo
841                                         }else{
842
843                                             // ya he rellenado todos
844   los valores de esta ronda de toma de datos
845
846   if(fields_rellenados_JSON==3){
847
848                                                 // asi que voy a meter
849   los valores en el array
850
851   servidor_JSON_updates.put(servidor_JSON_update);
852
853                                                 // reseteo las
854   variables para empezar una nueva ronda de toma de datos
855
856   fields_rellenados_JSON=0;
```

```
857
858   servidor_JSON_update=null;
859
860   servidor_JSON_update=new JSONObject();
861
862   Log.i("json","servidor_JSON_updates = " +
863   servidor_JSON_updates.toString(4));
864
865
866                                             }
867                                         }
868
869                                         // compruebo si acabo de
870   empezar la toma de datos y por lo tanto tengo que incluir la marca de
871   tiempo
872                                         if(fields_rellenados_JSON==0){
873
874   servidor_JSON_update.put("delta_t", delta_t);
875                                             delta_t++;
876
877                                         }
878
879                                         // incluyo el valor del
880   acelerómetro en el JSON
881
882   servidor_JSON_update.put("field3",
883   Float.parseFloat(recibido_acelerometro_z));
884                                         fields_rellenados_JSON++;
885                                         Log.i("json","field3 = " +
886   Float.parseFloat(recibido_acelerometro_z));
887
888
889                                     } catch (JSONException e){
890                                         // excepcion JSON
891                                     }
892
893                                 }
894
895
896
897                             }
898
899
900                         // CONFIGURACION
901
902                             // comprobamos si la caracteristica de la
903   que hemos recibido la notificacion es la del muestreo
904
905   if(intent.getExtras().getString("recibido_quien").equals("recibido_con
906   figuracion_muestreo")) {
907
908                                 final String
909   recibido_configuracion_muestreo =
910   intent.getExtras().getString("recibido_configuracion_muestreo");
```

```
911
912                                 runOnUiThread(new Runnable() {
913                                     @Override
914                                     public void run() {
915
916
917   texto_configuracion_muestreo_brazo_izquierdo.setText(String.valueOf(re
918   cibido_configuracion_muestreo));
919                                     }
920                                 });
921
922                                 // tras conocer el muestreo, ahora
923   intento conocer la sensibilidad
924
925   mBluetoothLeService.conocer_sensibilidad_brazo_izquierdo();
926
927                             }
928
929                             // comprobamos si la caracteristica de la
930   que hemos recibido la notificacion es la de la sensibilidad
931
932   if(intent.getExtras().getString("recibido_quien").equals("recibido_con
933   figuracion_sensibilidad")) {
934
935                                 final String
936   recibido_configuracion_sensibilidad =
937   intent.getExtras().getString("recibido_configuracion_sensibilidad");
938
939
940                                 if
941   (recibido_configuracion_sensibilidad.equals("1")) {
942
943
944   boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
945   getColor(R.color.boton_pulsado)));
946
947   boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
948   getColor(R.color.boton_no_pulsado)));
949
950   boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
951   getColor(R.color.boton_no_pulsado)));
952
953   boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
954   (getColor(R.color.boton_no_pulsado)));
955
956                                 }
957
958                                 if
959   (recibido_configuracion_sensibilidad.equals("2")) {
960
961
962   boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
963   getColor(R.color.boton_no_pulsado)));
```

```
964
965   boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
966   getColor(R.color.boton_pulsado)));
967
968   boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
969   getColor(R.color.boton_no_pulsado)));
970
971   boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
972   (getColor(R.color.boton_no_pulsado)));
973                                   }
974
975                                   if
976   (recibido_configuracion_sensibilidad.equals("3")) {
977
978
979   boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
980   getColor(R.color.boton_no_pulsado)));
981
982   boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
983   getColor(R.color.boton_no_pulsado)));
984
985   boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
986   getColor(R.color.boton_pulsado)));
987
988   boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
989   (getColor(R.color.boton_no_pulsado)));
990                                   }
991
992                                   if
993   (recibido_configuracion_sensibilidad.equals("4")) {
994
995
996   boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
997   getColor(R.color.boton_no_pulsado)));
998
999   boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
1000  getColor(R.color.boton_no_pulsado)));
1001
1002  boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
1003  getColor(R.color.boton_no_pulsado)));
1004
1005  boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
1006  (getColor(R.color.boton_pulsado)));
1007                                  }
1008
1009                              }
1010
1011
1012                    }
1013
1014
1015
1016
1017
```

```
1018                // BRAZO DERECHO
1019
1020                    // me he conectado a un dispositivo
1021                    if
1022 (BluetoothLeService.ACTION_GATT_CONNECTED_brazo_derecho.equals(action)
1023 ) {
1024
1025                        brazo_derecho_conectado=true;
1026
1027                        runOnUiThread(new Runnable() {
1028                            @Override
1029                            public void run() {
1030
1031 boton_brazo_derecho_estado.setBackgroundTintList(ColorStateList.valueO
1032 f(getColor(R.color.sensor_conectado)));
1033
1034 boton_brazo_derecho_configuracion.setVisibility(View.VISIBLE);
1035                                Log.i("problema", "principal:
1036 conectado a derecho, estado: verde, configuracion: visible");
1037
1038                            }
1039                        });
1040
1041
1042                    // me he desconectado de un dispositivo
1043                    } else if
1044 (BluetoothLeService.ACTION_GATT_DISCONNECTED_brazo_derecho.equals(acti
1045 on)) {
1046
1047                        brazo_derecho_conectado=false;
1048
1049                        // actualizo la interfaz para que el usuario
1050 vea que ya no estamos recibiendo valores, en lugar de mostrar el
1051 ultimo valor recibido
1052                        runOnUiThread(new Runnable() {
1053                            @Override
1054                            public void run() {
1055
1056
1057 texto_brazo_derecho_acelerometro_x.setText(String.valueOf("X:" ));
1058
1059 texto_brazo_derecho_acelerometro_y.setText(String.valueOf("Y:" ));
1060
1061 texto_brazo_derecho_acelerometro_z.setText(String.valueOf("Z:" ));
1062
1063
1064 boton_brazo_derecho_estado.setBackgroundTintList(ColorStateList.valueO
1065 f(getColor(R.color.sensor_no_conectado)));
1066
1067 boton_brazo_derecho_configuracion.setVisibility(View.GONE);
1068
1069 tarjeta_ajustes_brazo_derecho.setVisibility(LinearLayout.GONE);
1070                                Log.i("problema", "principal:
1071 desconectado a derecho, estado: rojo, configuracion: oculta");
```

```
                            }
                        });

                        // intento reconectarme al dispositivo

mBluetoothLeService.connect_brazo_derecho(direccion_MAC_brazo_derecho)
;


                    // se ha terminado correctamente la busqueda de
servicios
                    } else if
(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED_brazo_derecho.equa
ls(action)) {
                        Log.i("problema", "principal: servicios
descubiertos brazo derecho");

                        // obtengo el muestreo actual del dispositivo

mBluetoothLeService.conocer_muestreo_brazo_derecho();


                    // acabo de recibir datos del dispositivo, ya sea
porque se ha terminado una operacion de lectura que yo he iniciado o
porque he recibido una notificacion desde el dispositivo
                    } else if
(BluetoothLeService.ACTION_DATA_AVAILABLE_brazo_derecho.equals(action)
) {


                        // ACELEROMETRO

                        // comprobamos si la caracteristica de la que
hemos recibido la notificacion es la del acelerometro x

if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
lerometro_x")) {

                            final String recibido_acelerometro_x =
intent.getExtras().getString("recibido_acelerometro_x");

                            runOnUiThread(new Runnable() {
                                @Override
                                public void run() {

texto_brazo_derecho_acelerometro_x.setText(String.valueOf("X: " +
recibido_acelerometro_x));

                                }
                            });

                            // convertir string a float
```

```
                                //Float
dato_a_actualizar_2=Float.parseFloat(dato_a_actualizar);


                                // GRAFICO


if(puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_x){

if(primer_valor_grafica_brazo_derecho_aceleracion_x){


primer_valor_grafica_brazo_derecho_aceleracion_x=false;

numero_valor_grafica_brazo_derecho_aceleracion_x=0;

                                    // reseteamos la serie de datos de
puntos

serie_datos_punto_brazo_derecho_aceleracion_x.resetData(new
DataPoint[] {});

                                    // reseteamos la serie de datos de
lineas

serie_datos_linea_brazo_derecho_aceleracion_x.resetData(new
DataPoint[] {});

                                    // la serie que permite que todo
se actualice es serie_datos_punto_brazo_izquierdo_aceleracion_x

//serie_datos_punto_brazo_izquierdo_aceleracion_x.appendData(new
DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_x,
Float.parseFloat(recibido_acelerometro_x)), true, 10000);

                                }


                                Log.i("problema", "datos que voy a
añadir a la serie: " +
numero_valor_grafica_brazo_derecho_aceleracion_x + " " +
Float.parseFloat(recibido_acelerometro_x));

                                // añadimos un valor a la serie de
datos de puntos

serie_datos_punto_brazo_derecho_aceleracion_x.appendData(new
DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_x,
Float.parseFloat(recibido_acelerometro_x)), true, 10000);

                                Log.i("problema", "X maxima extraida
de la serie: " +
serie_datos_punto_brazo_derecho_aceleracion_x.getHighestValueX());
```

```
                                Log.i("problema", "Y maxima extraida
de la serie: " +
serie_datos_punto_brazo_derecho_aceleracion_x.getHighestValueY());


                                // añadimos un valor a la serie de
datos de lineas

serie_datos_linea_brazo_derecho_aceleracion_x.appendData(new
DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_x,
Float.parseFloat(recibido_acelerometro_x)), true, 10000);


numero_valor_grafica_brazo_derecho_aceleracion_x++;

                            }



                            // THINGSPEAK

                            if(puedo_añadir_valores_JSON){

                                try{

                                    // compruebo si el brazo izquierdo
y el brazo derecho están conectados

if(brazo_izquierdo_conectado==brazo_derecho_conectado){

                                        // ya he rellenado todos los
valores de esta ronda de toma de datos
                                        if(fields_rellenados_JSON==6){

                                            // asi que voy a meter los
valores en el array

servidor_JSON_updates.put(servidor_JSON_update);

                                            // reseteo las variables
para empezar una nueva ronda de toma de datos
                                            fields_rellenados_JSON=0;
                                            servidor_JSON_update=null;
                                            servidor_JSON_update=new
JSONObject();

                                        }

                                    // sólo está conectado el brazo
derecho
                                    }else{

                                        // ya he rellenado todos los
valores de esta ronda de toma de datos
```

```
1232                                        if(fields_rellenados_JSON==3){
1233
1234                                            // asi que voy a meter los
1235   valores en el array
1236
1237   servidor_JSON_updates.put(servidor_JSON_update);
1238
1239                                            // reseteo las variables
1240   para empezar una nueva ronda de toma de datos
1241                                            fields_rellenados_JSON=0;
1242                                            servidor_JSON_update=null;
1243                                            servidor_JSON_update=new
1244   JSONObject();
1245
1246                                        }
1247                                    }
1248
1249                                    // compruebo si acabo de empezar
1250   la toma de datos y por lo tanto tengo que incluir la marca de tiempo
1251                                    if(fields_rellenados_JSON==0){
1252
1253   servidor_JSON_update.put("delta_t", delta_t);
1254                                        delta_t++;
1255
1256                                    }
1257
1258                                    // incluyo el valor del
1259   acelerómetro en el JSON
1260                                    servidor_JSON_update.put("field4",
1261   Float.parseFloat(recibido_acelerometro_x));
1262                                    fields_rellenados_JSON++;
1263
1264                                } catch (JSONException e){
1265                                    // excepcion JSON
1266                                }
1267
1268                            }
1269
1270
1271
1272
1273
1274                        }
1275
1276
1277                        // comprobamos si la caracteristica de la que
1278   hemos recibido la notificacion es la del acelerometro y
1279
1280   if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
1281   lerometro_y")) {
1282
1283                            final String recibido_acelerometro_y =
1284   intent.getExtras().getString("recibido_acelerometro_y");
1285
```

```
                                runOnUiThread(new Runnable() {
                                    @Override
                                    public void run() {

texto_brazo_derecho_acelerometro_y.setText(String.valueOf("Y: " +
recibido_acelerometro_y));

                                    }
                                });


if(puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_y){

if(primer_valor_grafica_brazo_derecho_aceleracion_y){


primer_valor_grafica_brazo_derecho_aceleracion_y=false;

numero_valor_grafica_brazo_derecho_aceleracion_y=0;

                                    // reseteamos la serie de datos de
puntos

serie_datos_punto_brazo_derecho_aceleracion_y.resetData(new
DataPoint[] {});

                                    // reseteamos la serie de datos de
lineas

serie_datos_linea_brazo_derecho_aceleracion_y.resetData(new
DataPoint[] {});

                                }

                                // añadimos un valor a la serie de
datos de puntos

serie_datos_punto_brazo_derecho_aceleracion_y.appendData(new
DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_y,
Float.parseFloat(recibido_acelerometro_y)), true, 10000);

                                // añadimos un valor a la serie de
datos de lineas

serie_datos_linea_brazo_derecho_aceleracion_y.appendData(new
DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_y,
Float.parseFloat(recibido_acelerometro_y)), true, 10000);



numero_valor_grafica_brazo_derecho_aceleracion_y++;

                                }

```

```
1340
1341
1342
1343                                // THINGSPEAK
1344
1345                                if(puedo_añadir_valores_JSON){
1346
1347                                    try{
1348
1349                                        // compruebo si el brazo izquierdo
1350  y el brazo derecho están conectados
1351
1352  if(brazo_izquierdo_conectado==brazo_derecho_conectado){
1353
1354                                            // ya he rellenado todos los
1355  valores de esta ronda de toma de datos
1356                                            if(fields_rellenados_JSON==6){
1357
1358                                                // asi que voy a meter los
1359  valores en el array
1360
1361  servidor_JSON_updates.put(servidor_JSON_update);
1362
1363                                                // reseteo las variables
1364  para empezar una nueva ronda de toma de datos
1365                                                fields_rellenados_JSON=0;
1366                                                servidor_JSON_update=null;
1367                                                servidor_JSON_update=new
1368  JSONObject();
1369
1370                                            }
1371
1372                                        // sólo está conectado el brazo
1373  derecho
1374                                        }else{
1375
1376                                            // ya he rellenado todos los
1377  valores de esta ronda de toma de datos
1378                                            if(fields_rellenados_JSON==3){
1379
1380                                                // asi que voy a meter los
1381  valores en el array
1382
1383  servidor_JSON_updates.put(servidor_JSON_update);
1384
1385                                                // reseteo las variables
1386  para empezar una nueva ronda de toma de datos
1387                                                fields_rellenados_JSON=0;
1388                                                servidor_JSON_update=null;
1389                                                servidor_JSON_update=new
1390  JSONObject();
1391
1392                                            }
1393                                    }
```

```
1394
1395                                    // compruebo si acabo de empezar
1396  la toma de datos y por lo tanto tengo que incluir la marca de tiempo
1397                                    if(fields_rellenados_JSON==0){
1398
1399  servidor_JSON_update.put("delta_t", delta_t);
1400                                        delta_t++;
1401
1402                                    }
1403
1404                                    // incluyo el valor del
1405  acelerómetro en el JSON
1406                                    servidor_JSON_update.put("field5",
1407  Float.parseFloat(recibido_acelerometro_y));
1408                                    fields_rellenados_JSON++;
1409
1410                                } catch (JSONException e){
1411                                    // excepcion JSON
1412                                }
1413
1414                            }
1415
1416
1417
1418
1419
1420                        }
1421
1422                        // comprobamos si la caracteristica de la que
1423  hemos recibido la notificacion es la del acelerometro z
1424
1425  if(intent.getExtras().getString("recibido_quien").equals("recibido_ace
1426  lerometro_z")) {
1427
1428                            final String recibido_acelerometro_z =
1429  intent.getExtras().getString("recibido_acelerometro_z");
1430
1431                            runOnUiThread(new Runnable() {
1432                                @Override
1433                                public void run() {
1434
1435  texto_brazo_derecho_acelerometro_z.setText(String.valueOf("Z: " +
1436  recibido_acelerometro_z));
1437
1438                                }
1439                            });
1440
1441
1442  if(puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_z){
1443
1444  if(primer_valor_grafica_brazo_derecho_aceleracion_z){
1445
1446
1447  primer_valor_grafica_brazo_derecho_aceleracion_z=false;
```

```
1448
1449    numero_valor_grafica_brazo_derecho_aceleracion_z=0;
1450
1451                                        // reseteamos la serie de datos de
1452    puntos
1453
1454    serie_datos_punto_brazo_derecho_aceleracion_z.resetData(new
1455    DataPoint[] {});
1456
1457                                        // reseteamos la serie de datos de
1458    lineas
1459
1460    serie_datos_linea_brazo_derecho_aceleracion_z.resetData(new
1461    DataPoint[] {});
1462
1463                                    }
1464
1465                                    // añadimos un valor a la serie de
1466    datos de puntos
1467
1468    serie_datos_punto_brazo_derecho_aceleracion_z.appendData(new
1469    DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_z,
1470    Float.parseFloat(recibido_acelerometro_z)), true, 10000);
1471
1472                                    // añadimos un valor a la serie de
1473    datos de lineas
1474
1475    serie_datos_linea_brazo_derecho_aceleracion_z.appendData(new
1476    DataPoint(numero_valor_grafica_brazo_derecho_aceleracion_z,
1477    Float.parseFloat(recibido_acelerometro_z)), true, 10000);
1478
1479
1480
1481    numero_valor_grafica_brazo_derecho_aceleracion_z++;
1482
1483                                }
1484
1485
1486
1487
1488                                // THINGSPEAK
1489
1490                                if(puedo_añadir_valores_JSON){
1491
1492                                    try{
1493
1494                                        // compruebo si el brazo izquierdo
1495    y el brazo derecho están conectados
1496
1497    if(brazo_izquierdo_conectado==brazo_derecho_conectado){
1498
1499                                            // ya he rellenado todos los
1500    valores de esta ronda de toma de datos
1501                                            if(fields_rellenados_JSON==6){
```

```
1502
1503                                            // asi que voy a meter los
1504   valores en el array
1505
1506   servidor_JSON_updates.put(servidor_JSON_update);
1507
1508                                            // reseteo las variables
1509   para empezar una nueva ronda de toma de datos
1510                                            fields_rellenados_JSON=0;
1511                                            servidor_JSON_update=null;
1512                                            servidor_JSON_update=new
1513   JSONObject();
1514
1515                                        }
1516
1517                                    // sólo está conectado el brazo
1518   derecho
1519                                    }else{
1520
1521                                        // ya he rellenado todos los
1522   valores de esta ronda de toma de datos
1523                                        if(fields_rellenados_JSON==3){
1524
1525                                            // asi que voy a meter los
1526   valores en el array
1527
1528   servidor_JSON_updates.put(servidor_JSON_update);
1529
1530                                            // reseteo las variables
1531   para empezar una nueva ronda de toma de datos
1532                                            fields_rellenados_JSON=0;
1533                                            servidor_JSON_update=null;
1534                                            servidor_JSON_update=new
1535   JSONObject();
1536
1537                                        }
1538                                    }
1539
1540                                    // compruebo si acabo de empezar
1541   la toma de datos y por lo tanto tengo que incluir la marca de tiempo
1542                                    if(fields_rellenados_JSON==0){
1543
1544   servidor_JSON_update.put("delta_t", delta_t);
1545                                        delta_t++;
1546
1547                                    }
1548
1549                                    // incluyo el valor del
1550   acelerómetro en el JSON
1551                                    servidor_JSON_update.put("field6",
1552   Float.parseFloat(recibido_acelerometro_z));
1553                                    fields_rellenados_JSON++;
1554
1555                                } catch (JSONException e){
```

```
                                    // excepcion JSON
                                }

                            }




                        }


                        // CONFIGURACION

                        // comprobamos si la caracteristica de la que
hemos recibido la notificacion es la del muestreo

if(intent.getExtras().getString("recibido_quien").equals("recibido_con
figuracion_muestreo")) {

                            final String
recibido_configuracion_muestreo =
intent.getExtras().getString("recibido_configuracion_muestreo");

                            runOnUiThread(new Runnable() {
                                @Override
                                public void run() {


texto_configuracion_muestreo_brazo_derecho.setText(String.valueOf(reci
bido_configuracion_muestreo));
                                }
                            });

                            // tras conocer el muestreo, ahora intento
conocer la sensibilidad

mBluetoothLeService.conocer_sensibilidad_brazo_derecho();

                        }

                        // comprobamos si la caracteristica de la que
hemos recibido la notificacion es la de la sensibilidad

if(intent.getExtras().getString("recibido_quien").equals("recibido_con
figuracion_sensibilidad")) {

                            final String
recibido_configuracion_sensibilidad =
intent.getExtras().getString("recibido_configuracion_sensibilidad");


                            if
(recibido_configuracion_sensibilidad.equals("1")) {
```

```
1610
1611
1612    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1613    tColor(R.color.boton_pulsado)));
1614
1615    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1616    tColor(R.color.boton_no_pulsado)));
1617
1618    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1619    tColor(R.color.boton_no_pulsado)));
1620
1621    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
1622    etColor(R.color.boton_no_pulsado)));
1623
1624                                }
1625
1626                                if
1627    (recibido_configuracion_sensibilidad.equals("2")) {
1628
1629
1630    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1631    tColor(R.color.boton_no_pulsado)));
1632
1633    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1634    tColor(R.color.boton_pulsado)));
1635
1636    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1637    tColor(R.color.boton_no_pulsado)));
1638
1639    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
1640    etColor(R.color.boton_no_pulsado)));
1641                                }
1642
1643                                if
1644    (recibido_configuracion_sensibilidad.equals("3")) {
1645
1646
1647    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1648    tColor(R.color.boton_no_pulsado)));
1649
1650    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1651    tColor(R.color.boton_no_pulsado)));
1652
1653    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1654    tColor(R.color.boton_pulsado)));
1655
1656    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
1657    etColor(R.color.boton_no_pulsado)));
1658                                }
1659
1660                                if
1661    (recibido_configuracion_sensibilidad.equals("4")) {
1662
```

```
1663
1664    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1665    tColor(R.color.boton_no_pulsado)));
1666
1667    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1668    tColor(R.color.boton_no_pulsado)));
1669
1670    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
1671    tColor(R.color.boton_no_pulsado)));
1672
1673    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
1674    etColor(R.color.boton_pulsado)));
1675                                }
1676
1677                            }
1678
1679
1680                        }
1681
1682
1683
1684                }
1685            };
1686
1687            // el IntentFilter permite que el BroadcastReceiver sea
1688    llamado cuando el Intent enviado por el servicio contenga una de las
1689    siguientes acciones
1690            private static IntentFilter makeGattUpdateIntentFilter() {
1691                final IntentFilter intentFilter = new IntentFilter();
1692
1693                // BRAZO IZQUIERDO
1694
1695    intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED_brazo_
1696    izquierdo);
1697
1698    intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED_bra
1699    zo_izquierdo);
1700
1701    intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVE
1702    RED_brazo_izquierdo);
1703
1704    intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE_brazo_
1705    izquierdo);
1706
1707                // BRAZO DERECHO
1708
1709    intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED_brazo_
1710    derecho);
1711
1712    intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED_bra
1713    zo_derecho);
1714
1715    intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVE
1716    RED_brazo_derecho);
```

```
1717
1718    intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE_brazo_
1719    derecho);
1720
1721                return intentFilter;
1722            }
1723
1724            @Override
1725            protected void onResume() {
1726                super.onResume();
1727
1728                // con registerReceiver se asocia al BroadcastReceiver el
1729    filtro que necesita para activarse cuando se reciba alguna de las
1730    acciones contenidas en el filtro
1731                registerReceiver(mGattUpdateReceiver,
1732    makeGattUpdateIntentFilter());
1733            }
1734
1735            @Override
1736            protected void onPause() {
1737                super.onPause();
1738
1739                unregisterReceiver(mGattUpdateReceiver);
1740            }
1741
1742            @Override
1743            protected void onDestroy() {
1744                super.onDestroy();
1745                unbindService(mServiceConnection);
1746                mBluetoothLeService = null;
1747            }
1748
1749
1750        @Override
1751        public void onCreate(Bundle savedInstanceState) {
1752            super.onCreate(savedInstanceState);
1753            setContentView(R.layout.principal);
1754
1755            // BLE
1756
1757            // Comprobamos los permisos
1758                final int CODIGO_PERMISOS_BLUETOOTH = 1;
1759                final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1760                final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1761                final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1762
1763                // BLUETOOTH
1764                    int estadoDePermiso_1 =
1765    ContextCompat.checkSelfPermission(principal.this,
1766    Manifest.permission.BLUETOOTH);
1767                    if(estadoDePermiso_1
1768    ==PackageManager.PERMISSION_GRANTED) {
1769                        // Aquí el usuario dio permisos para acceder al
1770    bluetooth
```

```
1771                } else {
1772                    // Si no, entonces pedimos permisos...
1773                    ActivityCompat.requestPermissions(principal.this,
1774                            new
1775  String[]{Manifest.permission.BLUETOOTH},
1776                            CODIGO_PERMISOS_BLUETOOTH);
1777                }
1778
1779            // BLUETOOTH_ADMIN
1780            int estadoDePermiso_2 =
1781  ContextCompat.checkSelfPermission(principal.this,
1782  Manifest.permission.BLUETOOTH_ADMIN);
1783            if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
1784  {
1785                // Aquí el usuario dio permisos para acceder al
1786  bluetooth
1787            } else {
1788                // Si no, entonces pedimos permisos...
1789                ActivityCompat.requestPermissions(principal.this,
1790                        new
1791  String[]{Manifest.permission.BLUETOOTH_ADMIN},
1792                        CODIGO_PERMISOS_BLUETOOTH_ADMIN);
1793            }
1794
1795            // BLUETOOTH_CONNECT
1796            int estadoDePermiso_3 =
1797  ContextCompat.checkSelfPermission(principal.this,
1798  Manifest.permission.BLUETOOTH_CONNECT);
1799            if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
1800  {
1801                // Aquí el usuario dio permisos para acceder al
1802  bluetooth
1803            } else {
1804                // Si no, entonces pedimos permisos...
1805                ActivityCompat.requestPermissions(principal.this,
1806                        new
1807  String[]{Manifest.permission.BLUETOOTH_CONNECT},
1808                        CODIGO_PERMISOS_BLUETOOTH_CONNECT);
1809            }
1810
1811            // BLUETOOTH_SCAN
1812            int estadoDePermiso_4 =
1813  ContextCompat.checkSelfPermission(principal.this,
1814  Manifest.permission.BLUETOOTH_SCAN);
1815            if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
1816  {
1817                // Aquí el usuario dio permisos para acceder al
1818  bluetooth
1819            } else {
1820                // Si no, entonces pedimos permisos...
1821                ActivityCompat.requestPermissions(principal.this,
1822                        new
1823  String[]{Manifest.permission.BLUETOOTH_SCAN},
1824                        CODIGO_PERMISOS_BLUETOOTH_SCAN);
```

```
1825            }
1826
1827
1828            // inicializo el adaptador bluetooth
1829            final BluetoothManager bluetoothManager =
1830  (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
1831            mBluetoothAdapter = bluetoothManager.getAdapter();
1832
1833            // compruebo si el bluetooth esta encendido
1834            if (mBluetoothAdapter == null ||
1835  !mBluetoothAdapter.isEnabled()) {
1836                Intent enableBtIntent = new
1837  Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
1838                startActivityForResult(enableBtIntent, 1);
1839            }
1840
1841            // Creamos el Intent con la actividad en la que estoy y la
1842  actividad a la que voy
1843            Intent gattServiceIntent = new Intent(this,
1844  BluetoothLeService.class);
1845            // creamos el servicio (usando bindService para que la
1846  actividad aprincipal se enlace con el servicio para poder intercambiar
1847  datos)
1848            // con mServiceConnection se pueden manejar los eventos
1849  que genere el servicio
1850            bindService(gattServiceIntent, mServiceConnection,
1851  BIND_AUTO_CREATE);
1852
1853
1854
1855
1856        // Obtenemos una referencia a los controles de la interfaz
1857
1858            // BRAZO IZQUIERDO
1859
1860                // SECCIÓN donde se muestra el estado y el botón de
1861  configuración del sensor
1862                    boton_brazo_izquierdo_estado =
1863  (Button)findViewById(R.id.boton_brazo_izquierdo_estado);
1864                    boton_brazo_izquierdo_configuracion =
1865  (ImageButton)findViewById(R.id.boton_brazo_izquierdo_configuracion);
1866
1867                    // configuramos el boton de configuración, para
1868  que muestre u oculte la Sección de los ajustes del sensor
1869
1870  boton_brazo_izquierdo_configuracion.setOnClickListener(new
1871  View.OnClickListener() {
1872                            @Override
1873                            public void onClick(View v) {
1874
1875                                // la Sección de los ajustes ya se
1876  está mostrando y el usuario pulsa el botón de configuración
1877
1878  if(mostrando_tarjeta_ajustes_brazo_izquierdo==true){
```

```
1879
1880                                    // oculto la sección de los
1881    ajustes del brazo izquierdo
1882
1883    tarjeta_ajustes_brazo_izquierdo.setVisibility(LinearLayout.GONE);
1884
1885    mostrando_tarjeta_ajustes_brazo_izquierdo=false;
1886
1887                                  // la Sección de los ajustes está
1888    oculta y el usuario pulsa el botón de configuración
1889                                  } else{
1890
1891                                      // oculto la sección de los
1892    ajustes del brazo derecho
1893
1894    tarjeta_ajustes_brazo_derecho.setVisibility(LinearLayout.GONE);
1895
1896    mostrando_tarjeta_ajustes_brazo_derecho=false;
1897
1898                                      // muestro la sección de los
1899    ajustes del brazo izquierdo
1900
1901    tarjeta_ajustes_brazo_izquierdo.setVisibility(LinearLayout.VISIBLE);
1902
1903    mostrando_tarjeta_ajustes_brazo_izquierdo=true;
1904
1905                                  }
1906
1907                              }
1908                          });
1909
1910
1911                  // SECCIÓN donde se muestran los ajustes del sensor
1912                      tarjeta_ajustes_brazo_izquierdo =
1913    (LinearLayout)findViewById(R.id.tarjeta_ajustes_brazo_izquierdo);
1914                      texto_configuracion_muestreo_brazo_izquierdo =
1915    (EditText)findViewById(R.id.texto_configuracion_muestreo_brazo_izquier
1916    do);
1917                      Button
1918    boton_configuracion_muestreo_brazo_izquierdo =
1919    (Button)findViewById(R.id.boton_configuracion_muestreo_brazo_izquierdo
1920    );
1921
1922                      // botones de sensibilidad
1923                          boton_2G_brazo_izquierdo =
1924    (Button)findViewById(R.id.boton_2G_brazo_izquierdo);
1925                          boton_4G_brazo_izquierdo =
1926    (Button)findViewById(R.id.boton_4G_brazo_izquierdo);
1927                          boton_8G_brazo_izquierdo =
1928    (Button)findViewById(R.id.boton_8G_brazo_izquierdo);
1929                          boton_16G_brazo_izquierdo =
1930    (Button)findViewById(R.id.boton_16G_brazo_izquierdo);
1931
1932
```

```
                    // ajustamos la sensibilidad

                        // BOTON 2G

boton_2G_brazo_izquierdo.setOnClickListener(new View.OnClickListener()
{
                                @Override
                                public void onClick(View v) {

boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_pulsado)));

boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_no_pulsado)));

boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_no_pulsado)));

boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
(getColor(R.color.boton_no_pulsado)));


mBluetoothLeService.writeCharacteristic_brazo_izquierdo("boton_2G",
"boton_2G");

                                }
                            });

                        // BOTON 4G

boton_4G_brazo_izquierdo.setOnClickListener(new View.OnClickListener()
{
                                @Override
                                public void onClick(View v) {

boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_no_pulsado)));

boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_pulsado)));

boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
getColor(R.color.boton_no_pulsado)));

boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
(getColor(R.color.boton_no_pulsado)));


mBluetoothLeService.writeCharacteristic_brazo_izquierdo("boton_4G",
"boton_4G");

                                }
                            });

```

```
1987                            // BOTON 8G
1988
1989    boton_8G_brazo_izquierdo.setOnClickListener(new View.OnClickListener()
1990    {
1991                                    @Override
1992                                    public void onClick(View v) {
1993
1994    boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
1995    getColor(R.color.boton_no_pulsado)));
1996
1997    boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
1998    getColor(R.color.boton_no_pulsado)));
1999
2000    boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
2001    getColor(R.color.boton_pulsado)));
2002
2003    boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
2004    (getColor(R.color.boton_no_pulsado)));
2005
2006
2007    mBluetoothLeService.writeCharacteristic_brazo_izquierdo("boton_8G",
2008    "boton_8G");
2009                                    }
2010                                });
2011
2012                            // BOTON 16G
2013
2014    boton_16G_brazo_izquierdo.setOnClickListener(new
2015    View.OnClickListener() {
2016                                    @Override
2017                                    public void onClick(View v) {
2018
2019    boton_2G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
2020    getColor(R.color.boton_no_pulsado)));
2021
2022    boton_4G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
2023    getColor(R.color.boton_no_pulsado)));
2024
2025    boton_8G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf(
2026    getColor(R.color.boton_no_pulsado)));
2027
2028    boton_16G_brazo_izquierdo.setBackgroundTintList(ColorStateList.valueOf
2029    (getColor(R.color.boton_pulsado)));
2030
2031
2032    mBluetoothLeService.writeCharacteristic_brazo_izquierdo("boton_16G",
2033    "boton_16G");
2034
2035                                    }
2036                                });
2037
2038
2039                        // ajustamos el tiempo de muestreo
2040
```

```
                        // MUESTREO

boton_configuracion_muestreo_brazo_izquierdo.setOnClickListener(new
View.OnClickListener() {
                                @Override
                                public void onClick(View v) {


mBluetoothLeService.writeCharacteristic_brazo_izquierdo("boton_configu
racion_muestreo",
texto_configuracion_muestreo_brazo_izquierdo.getText().toString());

                                    // cerramos el teclado en pantalla
cuando el usuario pulsa el botón de enviar el muestreo
                                    InputMethodManager
inputMethodManager =
(InputMethodManager)getSystemService(INPUT_METHOD_SERVICE);

inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken
(),0);

                                }
                            });


                // SECCIÓN donde se muestran los datos del
acelerometro
                    texto_brazo_izquierdo_acelerometro_x =
(TextView)findViewById(R.id.texto_brazo_izquierdo_acelerometro_x);
                    texto_brazo_izquierdo_acelerometro_y =
(TextView)findViewById(R.id.texto_brazo_izquierdo_acelerometro_y);
                    texto_brazo_izquierdo_acelerometro_z =
(TextView)findViewById(R.id.texto_brazo_izquierdo_acelerometro_z);


                // SECCIÓN donde se muestra el gráfico con los valores

                    // obtenemos la referencia al gráfico
                    grafico_brazo_izquierdo =
(GraphView)findViewById(R.id.grafico_brazo_izquierdo);
                    boton_brazo_izquierdo_play =
(ImageButton)findViewById(R.id.boton_brazo_izquierdo_play);

                        // configuramos el comportamiento del boton
play al pulsarlo

boton_brazo_izquierdo_play.setOnClickListener(new
View.OnClickListener() {
                            @Override
                            public void onClick(View v) {


boton_brazo_izquierdo_play.setVisibility(LinearLayout.GONE);
```

```
2094
2095    boton_brazo_izquierdo_pause.setVisibility(LinearLayout.VISIBLE);
2096
2097
2098    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_x=true;
2099
2100    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_y=true;
2101
2102    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_z=true;
2103
2104
2105    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_x=true;
2106
2107    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_y=true;
2108
2109    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_z=true;
2110
2111
2112                                    // INTERNET PARA ENVIAR DATOS CON
2113    THINGSPEAK
2114                                    puedo_añadir_valores_JSON=true;
2115
2116                                }
2117                            });
2118
2119                        boton_brazo_izquierdo_pause =
2120    (ImageButton)findViewById(R.id.boton_brazo_izquierdo_pause);
2121
2122                            // configuramos el comportamiento del boton
2123    pause al pulsarlo
2124
2125    boton_brazo_izquierdo_pause.setVisibility(LinearLayout.GONE);
2126
2127
2128    boton_brazo_izquierdo_pause.setOnClickListener(new
2129    View.OnClickListener() {
2130                                @Override
2131                                public void onClick(View v) {
2132
2133
2134    boton_brazo_izquierdo_pause.setVisibility(LinearLayout.GONE);
2135
2136    boton_brazo_izquierdo_play.setVisibility(LinearLayout.VISIBLE);
2137
2138
2139    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_x=false;
2140
2141    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_y=false;
2142
2143    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_z=false;
2144
2145
2146    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_x=false;
```

```
2147
2148    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_y=false;
2149
2150    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_z=false;
2151
2152
2153                                    // INTERNET PARA ENVIAR DATOS CON
2154    THINGSPEAK
2155                                    puedo_añadir_valores_JSON=false;
2156
2157                                }
2158                            });
2159
2160                        boton_brazo_izquierdo_stop =
2161    (ImageButton)findViewById(R.id.boton_brazo_izquierdo_stop);
2162
2163                            // configuramos el comportamiento del boton
2164    stop al pulsarlo
2165
2166    boton_brazo_izquierdo_stop.setOnClickListener(new
2167    View.OnClickListener() {
2168                                @Override
2169                                public void onClick(View v) {
2170
2171
2172    boton_brazo_izquierdo_pause.setVisibility(LinearLayout.GONE);
2173
2174    boton_brazo_izquierdo_play.setVisibility(LinearLayout.VISIBLE);
2175
2176
2177    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_x=false;
2178
2179    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_y=false;
2180
2181    puedo_mostrar_valores_grafica_brazo_izquierdo_aceleracion_z=false;
2182
2183
2184    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_x=false;
2185
2186    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_y=false;
2187
2188    puedo_mostrar_valores_grafica_brazo_derecho_aceleracion_z=false;
2189
2190
2191    primer_valor_grafica_brazo_izquierdo_aceleracion_x=true;
2192
2193    primer_valor_grafica_brazo_izquierdo_aceleracion_y=true;
2194
2195    primer_valor_grafica_brazo_izquierdo_aceleracion_z=true;
2196
2197
2198    primer_valor_grafica_brazo_derecho_aceleracion_x=true;
2199
2200    primer_valor_grafica_brazo_derecho_aceleracion_y=true;
```

```
2201
2202    primer_valor_grafica_brazo_derecho_aceleracion_z=true;
2203
2204
2205                                    // INTERNET PARA ENVIAR DATOS CON
2206    THINGSPEAK
2207                                    puedo_añadir_valores_JSON=false;
2208
2209
2210                                    // para conectarme a internet tengo
2211    que ejecutar otro hilo
2212                                    new Thread(new Runnable() {
2213                                        @Override
2214                                        public void run() {
2215                                            try{
2216
2217
2218                                                // Creamos la URL del
2219    servidor al que nos vamos a conectar
2220                                                servidor_direccion = new
2221    URL("https://api.thingspeak.com/channels/1789950/bulk_update.json");
2222
2223                                                // Nos conectamos al
2224    servidor
2225                                                servidor_conexion =
2226    (HttpsURLConnection) servidor_direccion.openConnection();
2227
2228                                                // permitimos que la
2229    aplicación pueda escribir en el servidor
2230
2231    servidor_conexion.setDoOutput(true);
2232
2233                                                // elegimos el método de
2234    conexión usado por HTTP (POST para subir JSON)
2235
2236    servidor_conexion.setRequestMethod("POST");
2237
2238                                                // Añadimos el encabezado
2239    (Header) a la petición que le vamos a hacer al servidor
2240
2241    //servidor_conexion.setRequestProperty("User-Agent",
2242    "aplicacion_prueba");
2243
2244    servidor_conexion.setRequestProperty("Content-Type",
2245    "application/json");
2246
2247    servidor_conexion.setRequestProperty("Accept", "application/json");
2248
2249                                                // añadimos los datos del
2250    acelerómetro al JSON
2251
2252    servidor_JSON.put("write_api_key", "148VVPL6CFF3LG0H");
2253
2254    servidor_JSON.put("updates", servidor_JSON_updates);
```

```
2255
2256                                            Log.i("json",
2257    servidor_JSON.toString(4));
2258
2259                                            // enviamos los datos al
2260    servidor
2261                                            servidor_envio = new
2262    DataOutputStream(servidor_conexion.getOutputStream());
2263
2264                                            // la cuenta gratuita de
2265    thingspeak solo permite realizar un envio cada 15 segundos
2266
2267    servidor_envio.writeBytes(servidor_JSON.toString(4));
2268
2269                                            // Comprobamos si el
2270    servidor ha respondido favorablemente
2271                                            Log.i("internet",
2272    "respuesta servidor:" + servidor_conexion.getResponseCode());
2273
2274                                            // borramos los datos del
2275    envio
2276                                            servidor_envio.flush();
2277                                            servidor_envio.close();
2278                                            servidor_JSON=null;
2279                                            servidor_JSON=new
2280    JSONObject();
2281
2282                                            // cerramos la conexión
2283
2284    //servidor_conexion.disconnect();
2285
2286
2287
2288                                        } catch (MalformedURLException
2289    e){
2290                                            // URL equivocada
2291
2292                                        } catch (IOException e){
2293                                            // error de red
2294
2295                                            } catch (JSONException e)
2296    {
2297                                            // error de JSON
2298
2299                                        } finally {
2300
2301    //servidor_conexion.disconnect();
2302                                        }
2303                                    }
2304                                }).start();
2305
2306
2307
2308
```

```
2309
2310                            }
2311                        });
2312
2313                    boton_brazo_izquierdo_mostrar_aceleracion_x =
2314   (Button)findViewById(R.id.boton_brazo_izquierdo_mostrar_aceleracion_x)
2315   ;
2316
2317                        // configuramos el comportamiento del boton al
2318   pulsarlo
2319
2320   boton_brazo_izquierdo_mostrar_aceleracion_x.setOnClickListener(new
2321   View.OnClickListener() {
2322
2323                            @Override
2324                            public void onClick(View v) {
2325
2326   if(boton_brazo_izquierdo_mostrar_aceleracion_x_pulsado){
2327
2328   boton_brazo_izquierdo_mostrar_aceleracion_x.setBackgroundTintList(Colo
2329   rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_x_no_p
2330   ulsado)));
2331
2332                                    // quitamos la serie de datos de
2333   puntos del gráfico
2334
2335   grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_izquierdo
2336   _aceleracion_x);
2337
2338                                    // quitamos la serie de datos de
2339   lineas del gráfico
2340
2341   grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_izquierdo
2342   _aceleracion_x);
2343
2344                                } else{
2345
2346   boton_brazo_izquierdo_mostrar_aceleracion_x.setBackgroundTintList(Colo
2347   rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_x_puls
2348   ado)));
2349
2350                                    // añadimos la serie de datos de
2351   puntos al gráfico
2352
2353   grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
2354   eleracion_x);
2355
2356                                    // añadimos la serie de datos de
2357   lineas al gráfico
2358
2359   grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
2360   eleracion_x);
2361
2362                                }
```

```
2363
2364
2365    boton_brazo_izquierdo_mostrar_aceleracion_x_pulsado=!boton_brazo_izqui
2366    erdo_mostrar_aceleracion_x_pulsado;
2367
2368                                }
2369                            });
2370
2371
2372                        boton_brazo_izquierdo_mostrar_aceleracion_y =
2373    (Button)findViewById(R.id.boton_brazo_izquierdo_mostrar_aceleracion_y)
2374    ;
2375
2376                            // configuramos el comportamiento del boton al
2377    pulsarlo
2378
2379    boton_brazo_izquierdo_mostrar_aceleracion_y.setOnClickListener(new
2380    View.OnClickListener() {
2381
2382                                @Override
2383                                public void onClick(View v) {
2384
2385    if(boton_brazo_izquierdo_mostrar_aceleracion_y_pulsado){
2386
2387    boton_brazo_izquierdo_mostrar_aceleracion_y.setBackgroundTintList(Colo
2388    rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_y_no_p
2389    ulsado)));
2390
2391                                        // quitamos la serie de datos de
2392    puntos del gráfico
2393
2394    grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_izquierdo
2395    _aceleracion_y);
2396
2397                                        // quitamos la serie de datos de
2398    lineas del gráfico
2399
2400    grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_izquierdo
2401    _aceleracion_y);
2402
2403                                    } else{
2404
2405    boton_brazo_izquierdo_mostrar_aceleracion_y.setBackgroundTintList(Colo
2406    rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_y_puls
2407    ado)));
2408
2409                                        // añadimos la serie de datos de
2410    puntos al gráfico
2411
2412    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
2413    eleracion_y);
2414
2415                                        // añadimos la serie de datos de
2416    lineas al gráfico
```

```
2417
2418    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
2419    eleracion_y);
2420
2421                                    }
2422
2423
2424    boton_brazo_izquierdo_mostrar_aceleracion_y_pulsado=!boton_brazo_izqui
2425    erdo_mostrar_aceleracion_y_pulsado;
2426
2427                                }
2428                            });
2429
2430
2431                        boton_brazo_izquierdo_mostrar_aceleracion_z =
2432    (Button)findViewById(R.id.boton_brazo_izquierdo_mostrar_aceleracion_z)
2433    ;
2434
2435                            // configuramos el comportamiento del boton al
2436    pulsarlo
2437
2438    boton_brazo_izquierdo_mostrar_aceleracion_z.setOnClickListener(new
2439    View.OnClickListener() {
2440
2441                                @Override
2442                                public void onClick(View v) {
2443
2444    if(boton_brazo_izquierdo_mostrar_aceleracion_z_pulsado){
2445
2446    boton_brazo_izquierdo_mostrar_aceleracion_z.setBackgroundTintList(Colo
2447    rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_z_no_p
2448    ulsado)));
2449
2450                                        // quitamos la serie de datos de
2451    puntos del gráfico
2452
2453    grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_izquierdo
2454    _aceleracion_z);
2455
2456                                        // quitamos la serie de datos de
2457    lineas del gráfico
2458
2459    grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_izquierdo
2460    _aceleracion_z);
2461
2462                                    } else{
2463
2464    boton_brazo_izquierdo_mostrar_aceleracion_z.setBackgroundTintList(Colo
2465    rStateList.valueOf(getColor(R.color.brazo_izquierdo_aceleracion_z_puls
2466    ado)));
2467
2468                                        // añadimos la serie de datos de
2469    puntos al gráfico
```

```
2470
2471    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
2472    eleracion_z);
2473
2474                                        // añadimos la serie de datos de
2475    lineas al gráfico
2476
2477    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
2478    eleracion_z);
2479
2480                                    }
2481
2482
2483    boton_brazo_izquierdo_mostrar_aceleracion_z_pulsado=!boton_brazo_izqui
2484    erdo_mostrar_aceleracion_z_pulsado;
2485
2486                                }
2487                            });
2488
2489
2490
2491
2492                        // SERIES DE DATOS
2493                            // BRAZO IZQUIERDO
2494                                // ACELERACIÓN X
2495                                    // creamos una serie de datos que
2496    mostrará puntos
2497
2498    serie_datos_punto_brazo_izquierdo_aceleracion_x = new
2499    PointsGraphSeries<>(new DataPoint[] {});
2500
2501    serie_datos_punto_brazo_izquierdo_aceleracion_x.setColor(getColor(R.co
2502    lor.brazo_izquierdo_aceleracion_x_pulsado));
2503
2504                                    // creamos una serie de datos que
2505    mostrará lineas
2506
2507    serie_datos_linea_brazo_izquierdo_aceleracion_x = new
2508    LineGraphSeries<>(new DataPoint[] {});
2509
2510    serie_datos_linea_brazo_izquierdo_aceleracion_x.setColor(getColor(R.co
2511    lor.brazo_izquierdo_aceleracion_x_pulsado));
2512
2513
2514                                // ACELERACIÓN Y
2515                                    // creamos una serie de datos que
2516    mostrará puntos
2517
2518    serie_datos_punto_brazo_izquierdo_aceleracion_y = new
2519    PointsGraphSeries<>(new DataPoint[] {});
2520
2521    serie_datos_punto_brazo_izquierdo_aceleracion_y.setColor(getColor(R.co
2522    lor.brazo_izquierdo_aceleracion_y_pulsado));
2523
```

```
2524                                // creamos una serie de datos que
2525   mostrará lineas
2526
2527   serie_datos_linea_brazo_izquierdo_aceleracion_y = new
2528   LineGraphSeries<>(new DataPoint[] {});
2529
2530   serie_datos_linea_brazo_izquierdo_aceleracion_y.setColor(getColor(R.co
2531   lor.brazo_izquierdo_aceleracion_y_pulsado));
2532
2533
2534                               // ACELERACIÓN Z
2535                                   // creamos una serie de datos que
2536   mostrará puntos
2537
2538   serie_datos_punto_brazo_izquierdo_aceleracion_z = new
2539   PointsGraphSeries<>(new DataPoint[] {});
2540
2541   serie_datos_punto_brazo_izquierdo_aceleracion_z.setColor(getColor(R.co
2542   lor.brazo_izquierdo_aceleracion_z_pulsado));
2543
2544                                   // creamos una serie de datos que
2545   mostrará lineas
2546
2547   serie_datos_linea_brazo_izquierdo_aceleracion_z = new
2548   LineGraphSeries<>(new DataPoint[] {});
2549
2550   serie_datos_linea_brazo_izquierdo_aceleracion_z.setColor(getColor(R.co
2551   lor.brazo_izquierdo_aceleracion_z_pulsado));
2552
2553
2554
2555
2556           // AÑADIR LA SERIE DE DATOS AL GRÁFICO
2557                           // BRAZO IZQUIERDO
2558                               // ACELERACION X
2559                                   // añadimos la serie de datos de
2560   puntos al gráfico
2561
2562   grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
2563   eleracion_x);
2564
2565                                   // añadimos la serie de datos de
2566   lineas al gráfico
2567
2568   grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
2569   eleracion_x);
2570
2571                               // ACELERACION Y
2572                                   // añadimos la serie de datos de
2573   puntos al gráfico
2574
2575   grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
2576   eleracion_y);
2577
```

```
                                    // añadimos la serie de datos de
lineas al gráfico

grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
eleracion_y);

                                // ACELERACION Z
                                    // añadimos la serie de datos de
puntos al gráfico

grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_izquierdo_ac
eleracion_z);

                                    // añadimos la serie de datos de
lineas al gráfico

grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_izquierdo_ac
eleracion_z);


                        // MODIFICAR LA APARIENCIA DEL GRÁFICO
                /*
                            // texto en el eje horizontal y vertical junto
a cada valor

grafico_brazo_izquierdo.getGridLabelRenderer().setLabelFormatter(new
DefaultLabelFormatter() {
                                @Override
                                public String formatLabel(double value,
boolean isValueX) {
                                    if (isValueX) {
                                        // eje x
                                        return super.formatLabel(value,
isValueX) + " s";
                                    } else {
                                        // eje y
                                        return super.formatLabel(value,
isValueX) + " m/s2";
                                    }
                                }
                            });


                            // título del eje horizontal y del eje
vertical

//grafico_brazo_izquierdo.getGridLabelRenderer().setHorizontalAxisTitl
e("tiempo (segundos)");

//grafico_brazo_izquierdo.getGridLabelRenderer().setVerticalAxisTitle(
"aceleración (m/s2)");

                 */

```

```
2632                        // span del eje horizontal y vertical
2633                        Viewport grafico_brazo_izquierdo_ejes =
2634  grafico_brazo_izquierdo.getViewport();
2635
2636                        // rango de valores
2637
2638  grafico_brazo_izquierdo_ejes.setXAxisBoundsManual(true);
2639                            grafico_brazo_izquierdo_ejes.setMinX(0);
2640                            grafico_brazo_izquierdo_ejes.setMaxX(50);
2641
2642
2643  grafico_brazo_izquierdo_ejes.setYAxisBoundsManual(true);
2644                            grafico_brazo_izquierdo_ejes.setMinY(-20);
2645                            grafico_brazo_izquierdo_ejes.setMaxY(20);
2646
2647                        // desplazar con el dedo
2648
2649  grafico_brazo_izquierdo_ejes.setScrollable(true);
2650
2651  grafico_brazo_izquierdo_ejes.setScrollableY(true);
2652
2653                        // hacer zoom
2654
2655  grafico_brazo_izquierdo_ejes.setScalable(true);
2656
2657  grafico_brazo_izquierdo_ejes.setScalableY(true);
2658
2659
2660
2661
2662
2663
2664
2665          // BRAZO DERECHO
2666
2667                  // SECCIÓN donde se muestra el estado y el botón de
2668  configuración del sensor
2669                      boton_brazo_derecho_estado =
2670  (Button)findViewById(R.id.boton_brazo_derecho_estado);
2671                      boton_brazo_derecho_configuracion =
2672  (ImageButton)findViewById(R.id.boton_brazo_derecho_configuracion);
2673
2674                      // configuramos el boton de configuración, para
2675  que muestre u oculte la Sección de los ajustes del sensor
2676
2677  boton_brazo_derecho_configuracion.setOnClickListener(new
2678  View.OnClickListener() {
2679                          @Override
2680                          public void onClick(View v) {
2681
2682                              // la Sección de los ajustes ya se está
2683  mostrando y el usuario pulsa el botón de configuración
2684
2685  if(mostrando_tarjeta_ajustes_brazo_derecho==true){
```

```
2686
2687                                    // oculto la sección de los ajustes
2688    del brazo derecho
2689
2690    tarjeta_ajustes_brazo_derecho.setVisibility(LinearLayout.GONE);
2691
2692    mostrando_tarjeta_ajustes_brazo_derecho=false;
2693
2694                                    // la Sección de los ajustes está
2695    oculta y el usuario pulsa el botón de configuración
2696                                } else{
2697
2698                                    // oculto la sección de los ajustes
2699    del brazo izquierdo
2700
2701    tarjeta_ajustes_brazo_izquierdo.setVisibility(LinearLayout.GONE);
2702
2703    mostrando_tarjeta_ajustes_brazo_izquierdo=false;
2704
2705                                    // muestro la sección de los ajustes
2706    del brazo derecho
2707
2708    tarjeta_ajustes_brazo_derecho.setVisibility(LinearLayout.VISIBLE);
2709
2710    mostrando_tarjeta_ajustes_brazo_derecho=true;
2711
2712                                }
2713
2714                            }
2715                        });
2716
2717
2718            // SECCIÓN donde se muestran los ajustes del sensor
2719                        tarjeta_ajustes_brazo_derecho =
2720    (LinearLayout)findViewById(R.id.tarjeta_ajustes_brazo_derecho);
2721                        texto_configuracion_muestreo_brazo_derecho =
2722    (EditText)findViewById(R.id.texto_configuracion_muestreo_brazo_derecho
2723    );
2724                        Button boton_configuracion_muestreo_brazo_derecho
2725    =
2726    (Button)findViewById(R.id.boton_configuracion_muestreo_brazo_derecho);
2727
2728                        // botones de sensibilidad
2729                        boton_2G_brazo_derecho =
2730    (Button)findViewById(R.id.boton_2G_brazo_derecho);
2731                        boton_4G_brazo_derecho =
2732    (Button)findViewById(R.id.boton_4G_brazo_derecho);
2733                        boton_8G_brazo_derecho =
2734    (Button)findViewById(R.id.boton_8G_brazo_derecho);
2735                        boton_16G_brazo_derecho =
2736    (Button)findViewById(R.id.boton_16G_brazo_derecho);
2737
2738
2739                        // ajustamos la sensibilidad
```

```
2740
2741                            // BOTON 2G
2742                            boton_2G_brazo_derecho.setOnClickListener(new
2743    View.OnClickListener() {
2744                                @Override
2745                                public void onClick(View v) {
2746
2747    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2748    tColor(R.color.boton_pulsado)));
2749
2750    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2751    tColor(R.color.boton_no_pulsado)));
2752
2753    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2754    tColor(R.color.boton_no_pulsado)));
2755
2756    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
2757    etColor(R.color.boton_no_pulsado)));
2758
2759
2760    mBluetoothLeService.writeCharacteristic_brazo_derecho("boton_2G",
2761    "boton_2G");
2762
2763                                }
2764                            });
2765
2766                            // BOTON 4G
2767                            boton_4G_brazo_derecho.setOnClickListener(new
2768    View.OnClickListener() {
2769                                @Override
2770                                public void onClick(View v) {
2771
2772    boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2773    tColor(R.color.boton_no_pulsado)));
2774
2775    boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2776    tColor(R.color.boton_pulsado)));
2777
2778    boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
2779    tColor(R.color.boton_no_pulsado)));
2780
2781    boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
2782    etColor(R.color.boton_no_pulsado)));
2783
2784
2785    mBluetoothLeService.writeCharacteristic_brazo_derecho("boton_4G",
2786    "boton_4G");
2787
2788                                }
2789                            });
2790
2791                            // BOTON 8G
2792                            boton_8G_brazo_derecho.setOnClickListener(new
2793    View.OnClickListener() {
```

```
                                @Override
                                public void onClick(View v) {

boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_no_pulsado)));

boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_no_pulsado)));

boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_pulsado)));

boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
etColor(R.color.boton_no_pulsado)));


mBluetoothLeService.writeCharacteristic_brazo_derecho("boton_8G",
"boton_8G");
                                }
                            });

                            // BOTON 16G
                            boton_16G_brazo_derecho.setOnClickListener(new
View.OnClickListener() {
                                @Override
                                public void onClick(View v) {

boton_2G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_no_pulsado)));

boton_4G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_no_pulsado)));

boton_8G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(ge
tColor(R.color.boton_no_pulsado)));

boton_16G_brazo_derecho.setBackgroundTintList(ColorStateList.valueOf(g
etColor(R.color.boton_pulsado)));


mBluetoothLeService.writeCharacteristic_brazo_derecho("boton_16G",
"boton_16G");

                                }
                            });


                        // ajustamos el tiempo de muestreo

                            // MUESTREO

boton_configuracion_muestreo_brazo_derecho.setOnClickListener(new
View.OnClickListener() {
                                @Override
```

```
2848                            public void onClick(View v) {
2849
2850
2851    mBluetoothLeService.writeCharacteristic_brazo_derecho("boton_configura
2852    cion_muestreo",
2853    texto_configuracion_muestreo_brazo_derecho.getText().toString());
2854
2855                                    // cerramos el teclado en pantalla
2856    cuando el usuario pulsa el botón de enviar el muestreo
2857                                    InputMethodManager inputMethodManager
2858    = (InputMethodManager)getSystemService(INPUT_METHOD_SERVICE);
2859
2860    inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken
2861    (),0);
2862
2863                                }
2864                            });
2865
2866
2867                    // SECCIÓN donde se muestran los datos del
2868    acelerometro
2869                        texto_brazo_derecho_acelerometro_x =
2870    (TextView)findViewById(R.id.texto_brazo_derecho_acelerometro_x);
2871                        texto_brazo_derecho_acelerometro_y =
2872    (TextView)findViewById(R.id.texto_brazo_derecho_acelerometro_y);
2873                        texto_brazo_derecho_acelerometro_z =
2874    (TextView)findViewById(R.id.texto_brazo_derecho_acelerometro_z);
2875
2876
2877            // SECCIÓN donde se muestra el gráfico con los valores
2878
2879                // obtenemos la referencia al gráfico
2880                boton_brazo_derecho_mostrar_aceleracion_x =
2881    (Button)findViewById(R.id.boton_brazo_derecho_mostrar_aceleracion_x);
2882
2883                // configuramos el comportamiento del boton al pulsarlo
2884
2885    boton_brazo_derecho_mostrar_aceleracion_x.setOnClickListener(new
2886    View.OnClickListener() {
2887
2888                    @Override
2889                    public void onClick(View v) {
2890
2891    if(boton_brazo_derecho_mostrar_aceleracion_x_pulsado){
2892
2893    boton_brazo_derecho_mostrar_aceleracion_x.setBackgroundTintList(ColorS
2894    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_x_no_pulsa
2895    do)));
2896
2897                            // quitamos la serie de datos de puntos del
2898    gráfico
2899
2900    grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_derecho_a
2901    celeracion_x);
```

```
2902
2903                            // quitamos la serie de datos de lineas del
2904    gráfico
2905
2906    grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_derecho_a
2907    celeracion_x);
2908
2909
2910                        } else{
2911
2912    boton_brazo_derecho_mostrar_aceleracion_x.setBackgroundTintList(ColorS
2913    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_x_pulsado)
2914    ));
2915
2916                            // añadimos la serie de datos de puntos al
2917    gráfico
2918
2919    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
2920    eracion_x);
2921
2922                            // añadimos la serie de datos de lineas al
2923    gráfico
2924
2925    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
2926    eracion_x);
2927
2928                        }
2929
2930
2931    boton_brazo_derecho_mostrar_aceleracion_x_pulsado=!boton_brazo_derecho
2932    _mostrar_aceleracion_x_pulsado;
2933
2934                    }
2935                });
2936
2937
2938                boton_brazo_derecho_mostrar_aceleracion_y =
2939    (Button)findViewById(R.id.boton_brazo_derecho_mostrar_aceleracion_y);
2940
2941                // configuramos el comportamiento del boton al pulsarlo
2942
2943    boton_brazo_derecho_mostrar_aceleracion_y.setOnClickListener(new
2944    View.OnClickListener() {
2945
2946                    @Override
2947                    public void onClick(View v) {
2948
2949    if(boton_brazo_derecho_mostrar_aceleracion_y_pulsado){
2950
2951    boton_brazo_derecho_mostrar_aceleracion_y.setBackgroundTintList(ColorS
2952    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_y_no_pulsa
2953    do)));
2954
```

```
2955                        // quitamos la serie de datos de puntos del
2956    gráfico
2957
2958    grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_derecho_a
2959    celeracion_y);
2960
2961                        // quitamos la serie de datos de lineas del
2962    gráfico
2963
2964    grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_derecho_a
2965    celeracion_y);
2966
2967                    } else{
2968
2969    boton_brazo_derecho_mostrar_aceleracion_y.setBackgroundTintList(ColorS
2970    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_y_pulsado)
2971    ));
2972
2973                        // añadimos la serie de datos de puntos al
2974    gráfico
2975
2976    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
2977    eracion_y);
2978
2979                        // añadimos la serie de datos de lineas al
2980    gráfico
2981
2982    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
2983    eracion_y);
2984
2985                    }
2986
2987
2988    boton_brazo_derecho_mostrar_aceleracion_y_pulsado=!boton_brazo_derecho
2989    _mostrar_aceleracion_y_pulsado;
2990
2991                }
2992            });
2993
2994
2995            boton_brazo_derecho_mostrar_aceleracion_z =
2996    (Button)findViewById(R.id.boton_brazo_derecho_mostrar_aceleracion_z);
2997
2998            // configuramos el comportamiento del boton al pulsarlo
2999
3000    boton_brazo_derecho_mostrar_aceleracion_z.setOnClickListener(new
3001    View.OnClickListener() {
3002
3003                @Override
3004                public void onClick(View v) {
3005
3006    if(boton_brazo_derecho_mostrar_aceleracion_z_pulsado){
3007
3008    boton_brazo_derecho_mostrar_aceleracion_z.setBackgroundTintList(ColorS
```

```
3009    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_z_no_pulsa
3010    do)));
3011
3012                            // quitamos la serie de datos de puntos del
3013    gráfico
3014
3015    grafico_brazo_izquierdo.removeSeries(serie_datos_punto_brazo_derecho_a
3016    celeracion_z);
3017
3018                            // quitamos la serie de datos de lineas del
3019    gráfico
3020
3021    grafico_brazo_izquierdo.removeSeries(serie_datos_linea_brazo_derecho_a
3022    celeracion_z);
3023
3024                        } else{
3025
3026    boton_brazo_derecho_mostrar_aceleracion_z.setBackgroundTintList(ColorS
3027    tateList.valueOf(getColor(R.color.brazo_derecho_aceleracion_z_pulsado)
3028    ));
3029
3030                            // añadimos la serie de datos de puntos al
3031    gráfico
3032
3033    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
3034    eracion_z);
3035
3036                            // añadimos la serie de datos de lineas al
3037    gráfico
3038
3039    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
3040    eracion_z);
3041
3042                        }
3043
3044
3045    boton_brazo_derecho_mostrar_aceleracion_z_pulsado=!boton_brazo_derecho
3046    _mostrar_aceleracion_z_pulsado;
3047
3048                    }
3049                });
3050
3051
3052
3053
3054                // SERIES DE DATOS
3055                    // BRAZO DERECHO
3056                        // ACELERACIÓN X
3057                            // creamos una serie de datos que mostrará
3058    puntos
3059                            serie_datos_punto_brazo_derecho_aceleracion_x
3060    = new PointsGraphSeries<>(new DataPoint[] {});
```

```
3061
3062    serie_datos_punto_brazo_derecho_aceleracion_x.setColor(getColor(R.colo
3063    r.brazo_derecho_aceleracion_x_pulsado));
3064
3065                            // creamos una serie de datos que mostrará
3066    lineas
3067                            serie_datos_linea_brazo_derecho_aceleracion_x
3068    = new LineGraphSeries<>(new DataPoint[] {});
3069
3070    serie_datos_linea_brazo_derecho_aceleracion_x.setColor(getColor(R.colo
3071    r.brazo_derecho_aceleracion_x_pulsado));
3072
3073
3074                        // ACELERACIÓN Y
3075                            // creamos una serie de datos que mostrará
3076    puntos
3077                            serie_datos_punto_brazo_derecho_aceleracion_y
3078    = new PointsGraphSeries<>(new DataPoint[] {});
3079
3080    serie_datos_punto_brazo_derecho_aceleracion_y.setColor(getColor(R.colo
3081    r.brazo_derecho_aceleracion_y_pulsado));
3082
3083                            // creamos una serie de datos que mostrará
3084    lineas
3085                            serie_datos_linea_brazo_derecho_aceleracion_y
3086    = new LineGraphSeries<>(new DataPoint[] {});
3087
3088    serie_datos_linea_brazo_derecho_aceleracion_y.setColor(getColor(R.colo
3089    r.brazo_derecho_aceleracion_y_pulsado));
3090
3091
3092                        // ACELERACIÓN Z
3093                            // creamos una serie de datos que mostrará
3094    puntos
3095                            serie_datos_punto_brazo_derecho_aceleracion_z
3096    = new PointsGraphSeries<>(new DataPoint[] {});
3097
3098    serie_datos_punto_brazo_derecho_aceleracion_z.setColor(getColor(R.colo
3099    r.brazo_derecho_aceleracion_z_pulsado));
3100
3101                            // creamos una serie de datos que mostrará
3102    lineas
3103                            serie_datos_linea_brazo_derecho_aceleracion_z
3104    = new LineGraphSeries<>(new DataPoint[] {});
3105
3106    serie_datos_linea_brazo_derecho_aceleracion_z.setColor(getColor(R.colo
3107    r.brazo_derecho_aceleracion_z_pulsado));
3108
3109
3110
3111
3112                // AÑADIR LA SERIE DE DATOS AL GRÁFICO
3113                    // BRAZO DERECHO
3114                        // ACELERACION X
```

```
3115                            // añadimos la serie de datos de puntos al
3116    gráfico
3117
3118    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
3119    eracion_x);
3120
3121                            // añadimos la serie de datos de lineas al
3122    gráfico
3123
3124    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
3125    eracion_x);
3126
3127                        // ACELERACION Y
3128                            // añadimos la serie de datos de puntos al
3129    gráfico
3130
3131    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
3132    eracion_y);
3133
3134                            // añadimos la serie de datos de lineas al
3135    gráfico
3136
3137    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
3138    eracion_y);
3139
3140                        // ACELERACION Z
3141                            // añadimos la serie de datos de puntos al
3142    gráfico
3143
3144    grafico_brazo_izquierdo.addSeries(serie_datos_punto_brazo_derecho_acel
3145    eracion_z);
3146
3147                            // añadimos la serie de datos de lineas al
3148    gráfico
3149
3150    grafico_brazo_izquierdo.addSeries(serie_datos_linea_brazo_derecho_acel
3151    eracion_z);
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161        }
3162
3163    }
```

7.3 Código Fuente Android: Actividad_Principal.xml

```
<!-- ACTIVIDAD PRINCIPAL -->
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    android:animateLayoutChanges="true"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/fondo">


    <!-- SECCION DONDE SE MUESTRA EL ESTADO Y EL BOTÓN DE
CONFIGURACION DE LOS SENSORES -->
    <androidx.cardview.widget.CardView
        android:id="@+id/tarjeta_estado_y_configuracion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        card_view:cardCornerRadius="10dp"
        card_view:cardElevation="10dp"
        android:layout_margin="10sp"
        >

        <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:tools="http://schemas.android.com/tools"
            android:id="@+id/LinearLayout2"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal"
            android:padding="5sp"
            android:gravity="center">


            <!-- BRAZO IZQUIERDO -->
            <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                xmlns:tools="http://schemas.android.com/tools"
                android:id="@+id/layout_ajustes_brazo_izquierdo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:gravity="top">

                <!-- BOTON QUE MUESTRA SI EL SENSOR ESTÁ CONECTADO -->
                <Button
                    android:id="@+id/boton_brazo_izquierdo_estado"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
```

```
52                      android:text="BRAZO IZQUIERDO"
53                      android:textColor="#000000"
54                      android:textSize="16sp"
55                      android:enabled="false"
56
57  android:backgroundTint="@color/sensor_no_conectado"
58                      />
59
60                  <!-- BOTÓN DE CONFIGURACIÓN -->
61                  <ImageButton
62
63  android:id="@+id/boton_brazo_izquierdo_configuracion"
64                      android:layout_width="40sp"
65                      android:layout_height="40sp"
66                      android:visibility="gone"
67                      android:layout_gravity="center"
68                      android:background="@drawable/ajustes"/>
69
70              </LinearLayout>
71
72              <View
73                  android:layout_width="85sp"
74                  android:layout_height="match_parent"
75                  />
76
77              <!-- BRAZO DERECHO -->
78              <LinearLayout
79  xmlns:android="http://schemas.android.com/apk/res/android"
80                  xmlns:tools="http://schemas.android.com/tools"
81                  android:id="@+id/layout_ajustes_brazo_derecho"
82                  android:layout_width="wrap_content"
83                  android:layout_height="wrap_content"
84                  android:orientation="vertical"
85                  android:gravity="top">
86
87                  <!-- BOTON QUE MUESTRA SI EL SENSOR ESTÁ CONECTADO -->
88                  <Button
89                      android:id="@+id/boton_brazo_derecho_estado"
90                      android:layout_width="wrap_content"
91                      android:layout_height="wrap_content"
92                      android:text="BRAZO DERECHO"
93                      android:textColor="#000000"
94                      android:textSize="16sp"
95                      android:enabled="false"
96
97  android:backgroundTint="@color/sensor_no_conectado"/>
98
99                  <!-- BOTÓN DE CONFIGURACIÓN -->
100                 <ImageButton
101
102 android:id="@+id/boton_brazo_derecho_configuracion"
103                     android:layout_width="40sp"
104                     android:layout_height="40sp"
105                     android:visibility="gone"
```

```
106                     android:layout_gravity="center"
107                     android:background="@drawable/ajustes"/>
108
109             </LinearLayout>
110
111
112         </LinearLayout>
113
114     </androidx.cardview.widget.CardView>
115
116
117
118     <!-- SECCIÓN DONDE SE AJUSTA EL BRAZO IZQUIERDO -->
119
120     <LinearLayout
121 xmlns:android="http://schemas.android.com/apk/res/android"
122         android:id="@+id/tarjeta_ajustes_brazo_izquierdo"
123         android:layout_width="match_parent"
124         android:layout_height="wrap_content"
125         xmlns:card_view="http://schemas.android.com/apk/res-auto"
126         android:orientation="vertical"
127         android:background="@color/fondo"
128         android:visibility="gone">
129
130
131         <!-- TITULO -->
132         <androidx.cardview.widget.CardView
133             android:layout_width="match_parent"
134             android:layout_height="wrap_content"
135             android:layout_gravity="center"
136             card_view:cardCornerRadius="10dp"
137             card_view:cardElevation="10dp"
138             android:layout_margin="10sp"
139             android:visibility="visible">
140
141             <LinearLayout
142 xmlns:android="http://schemas.android.com/apk/res/android"
143                 android:layout_width="match_parent"
144                 android:layout_height="match_parent"
145                 xmlns:card_view="http://schemas.android.com/apk/res-
146 auto"
147                 android:gravity="center"
148                 android:orientation="vertical">
149
150                 <View
151                     android:layout_width="match_parent"
152                     android:layout_height="10sp"
153                     />
154
155                 <TextView
156                     android:layout_width="wrap_content"
157                     android:layout_height="wrap_content"
158                     android:text="AJUSTES DEL BRAZO IZQUIERDO"
159                     android:textColor="#000000"
```

```
                    android:textSize="24sp"
                    android:textStyle="bold"
                    />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />



            </LinearLayout>


        </androidx.cardview.widget.CardView>



        <!-- AJUSTAR LA SENSIBILIDAD -->
        <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            card_view:cardCornerRadius="10dp"
            card_view:cardElevation="10dp"
            android:layout_margin="10sp">

            <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                xmlns:card_view="http://schemas.android.com/apk/res-
auto"
                android:gravity="center"
                android:orientation="vertical">

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="SENSIBILIDAD DEL ACELERÓMETRO"
                    android:textColor="#000000"
                    android:textSize="22sp"
                    />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

```

```
214                 <LinearLayout
215 xmlns:android="http://schemas.android.com/apk/res/android"
216                     android:layout_width="match_parent"
217                     android:layout_height="match_parent"
218                     android:gravity="center"
219
220 xmlns:card_view="http://schemas.android.com/apk/res-auto"
221                     android:orientation="horizontal">
222
223                     <Button
224                         android:id="@+id/boton_2G_brazo_izquierdo"
225                         android:layout_width="wrap_content"
226                         android:layout_height="wrap_content"
227                         android:text="2G"
228                         android:textColor="#000000"
229                         android:textSize="16sp"
230
231 android:backgroundTint="@color/boton_no_pulsado"/>
232
233                     <View
234                         android:layout_width="10sp"
235                         android:layout_height="match_parent"
236                         />
237
238                     <Button
239                         android:id="@+id/boton_4G_brazo_izquierdo"
240                         android:layout_width="wrap_content"
241                         android:layout_height="wrap_content"
242                         android:text="4G"
243                         android:textColor="#000000"
244                         android:textSize="16sp"
245
246 android:backgroundTint="@color/boton_no_pulsado"/>
247
248                     <View
249                         android:layout_width="10sp"
250                         android:layout_height="match_parent"
251                         />
252
253                     <Button
254                         android:id="@+id/boton_8G_brazo_izquierdo"
255                         android:layout_width="wrap_content"
256                         android:layout_height="wrap_content"
257                         android:text="8G"
258                         android:textColor="#000000"
259                         android:textSize="16sp"
260
261 android:backgroundTint="@color/boton_no_pulsado"/>
262
263                     <View
264                         android:layout_width="10sp"
265                         android:layout_height="match_parent"
266                         />
267
```

```
                    <Button
                        android:id="@+id/boton_16G_brazo_izquierdo"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="16G"
                        android:textColor="#000000"
                        android:textSize="16sp"

android:backgroundTint="@color/boton_no_pulsado"/>
                </LinearLayout>

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

            </LinearLayout>


        </androidx.cardview.widget.CardView>


        <!-- AJUSTAR EL TIEMPO DE MUESTREO -->
        <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            card_view:cardCornerRadius="10dp"
            card_view:cardElevation="10dp"
            android:layout_margin="10sp">

            <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                xmlns:card_view="http://schemas.android.com/apk/res-
auto"
                android:gravity="center"
                android:orientation="vertical">

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="TIEMPO ENTRE MUESTRA Y MUESTRA"
                    android:textColor="#000000"
                    android:textSize="22sp"
                    />

                <View
```

```
322                     android:layout_width="match_parent"
323                     android:layout_height="10sp"
324                     />
325
326                 <LinearLayout
327 xmlns:android="http://schemas.android.com/apk/res/android"
328                     android:layout_width="match_parent"
329                     android:layout_height="match_parent"
330                     android:gravity="center"
331
332 xmlns:card_view="http://schemas.android.com/apk/res-auto"
333                     android:orientation="horizontal">
334
335                     <EditText
336
337 android:id="@+id/texto_configuracion_muestreo_brazo_izquierdo"
338                         android:layout_width="wrap_content"
339                         android:layout_height="wrap_content"
340                         android:inputType="number"
341                         android:textColor="#000000"
342                         android:textSize="22sp" />
343
344                     <TextView
345                         android:layout_width="wrap_content"
346                         android:layout_height="wrap_content"
347                         android:text="ms"
348                         android:textColor="#000000"
349                         android:textSize="20sp"
350                         />
351
352                     <View
353                         android:layout_width="10sp"
354                         android:layout_height="match_parent"
355                         />
356
357                     <Button
358
359 android:id="@+id/boton_configuracion_muestreo_brazo_izquierdo"
360                         android:layout_width="wrap_content"
361                         android:layout_height="wrap_content"
362                         android:text="ENVIAR"
363                         android:textColor="#000000"
364                         android:textSize="16sp"
365
366 android:backgroundTint="@color/boton_pulsado"/>
367
368                     <View
369                         android:layout_width="30sp"
370                         android:layout_height="match_parent"
371                         />
372
373                 </LinearLayout>
374
375                 <View
```

```
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

            </LinearLayout>

        </androidx.cardview.widget.CardView>

    </LinearLayout>









    <!-- SECCIÓN DONDE SE AJUSTA EL BRAZO DERECHO -->

    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/tarjeta_ajustes_brazo_derecho"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:orientation="vertical"
        android:background="@color/fondo"
        android:visibility="gone">


        <!-- TITULO -->
        <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            card_view:cardCornerRadius="10dp"
            card_view:cardElevation="10dp"
            android:layout_margin="10sp"
            android:visibility="visible">

            <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                xmlns:card_view="http://schemas.android.com/apk/res-
auto"
                android:gravity="center"
                android:orientation="vertical">

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

                <TextView
```

```
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="AJUSTES DEL BRAZO DERECHO"
                    android:textColor="#000000"
                    android:textSize="24sp"
                    android:textStyle="bold"
                    />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />



            </LinearLayout>


        </androidx.cardview.widget.CardView>



        <!-- AJUSTAR LA SENSIBILIDAD -->
        <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            card_view:cardCornerRadius="10dp"
            card_view:cardElevation="10dp"
            android:layout_margin="10sp">

            <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                xmlns:card_view="http://schemas.android.com/apk/res-
auto"
                android:gravity="center"
                android:orientation="vertical">

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="SENSIBILIDAD DEL ACELERÓMETRO"
                    android:textColor="#000000"
                    android:textSize="22sp"
                    />

                <View
```

```
484                     android:layout_width="match_parent"
485                     android:layout_height="10sp"
486                     />
487
488                 <LinearLayout
489 xmlns:android="http://schemas.android.com/apk/res/android"
490                     android:layout_width="match_parent"
491                     android:layout_height="match_parent"
492                     android:gravity="center"
493
494 xmlns:card_view="http://schemas.android.com/apk/res-auto"
495                     android:orientation="horizontal">
496
497                     <Button
498                         android:id="@+id/boton_2G_brazo_derecho"
499                         android:layout_width="wrap_content"
500                         android:layout_height="wrap_content"
501                         android:text="2G"
502                         android:textColor="#000000"
503                         android:textSize="16sp"
504
505 android:backgroundTint="@color/boton_no_pulsado"/>
506
507                     <View
508                         android:layout_width="10sp"
509                         android:layout_height="match_parent"
510                         />
511
512                     <Button
513                         android:id="@+id/boton_4G_brazo_derecho"
514                         android:layout_width="wrap_content"
515                         android:layout_height="wrap_content"
516                         android:text="4G"
517                         android:textColor="#000000"
518                         android:textSize="16sp"
519
520 android:backgroundTint="@color/boton_no_pulsado"/>
521
522                     <View
523                         android:layout_width="10sp"
524                         android:layout_height="match_parent"
525                         />
526
527                     <Button
528                         android:id="@+id/boton_8G_brazo_derecho"
529                         android:layout_width="wrap_content"
530                         android:layout_height="wrap_content"
531                         android:text="8G"
532                         android:textColor="#000000"
533                         android:textSize="16sp"
534
535 android:backgroundTint="@color/boton_no_pulsado"/>
536
537                     <View
```

```
538                         android:layout_width="10sp"
539                         android:layout_height="match_parent"
540                         />
541
542                     <Button
543                         android:id="@+id/boton_16G_brazo_derecho"
544                         android:layout_width="wrap_content"
545                         android:layout_height="wrap_content"
546                         android:text="16G"
547                         android:textColor="#000000"
548                         android:textSize="16sp"
549
550 android:backgroundTint="@color/boton_no_pulsado"/>
551                 </LinearLayout>
552
553                 <View
554                     android:layout_width="match_parent"
555                     android:layout_height="10sp"
556                     />
557
558             </LinearLayout>
559
560
561         </androidx.cardview.widget.CardView>
562
563
564         <!-- AJUSTAR EL TIEMPO DE MUESTREO -->
565         <androidx.cardview.widget.CardView
566             android:layout_width="match_parent"
567             android:layout_height="wrap_content"
568             android:layout_gravity="center"
569             card_view:cardCornerRadius="10dp"
570             card_view:cardElevation="10dp"
571             android:layout_margin="10sp">
572
573             <LinearLayout
574 xmlns:android="http://schemas.android.com/apk/res/android"
575                 android:layout_width="match_parent"
576                 android:layout_height="match_parent"
577                 xmlns:card_view="http://schemas.android.com/apk/res-
578 auto"
579                 android:gravity="center"
580                 android:orientation="vertical">
581
582                 <View
583                     android:layout_width="match_parent"
584                     android:layout_height="10sp"
585                     />
586
587                 <TextView
588                     android:layout_width="wrap_content"
589                     android:layout_height="wrap_content"
590                     android:text="TIEMPO ENTRE MUESTRA Y MUESTRA"
591                     android:textColor="#000000"
```

```
                    android:textSize="22sp"
                    />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10sp"
                    />

                <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:gravity="center"

xmlns:card_view="http://schemas.android.com/apk/res-auto"
                    android:orientation="horizontal">

                    <EditText

android:id="@+id/texto_configuracion_muestreo_brazo_derecho"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:inputType="number"
                        android:textColor="#000000"
                        android:textSize="22sp" />

                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="ms"
                        android:textColor="#000000"
                        android:textSize="20sp"
                        />

                    <View
                        android:layout_width="10sp"
                        android:layout_height="match_parent"
                        />

                    <Button

android:id="@+id/boton_configuracion_muestreo_brazo_derecho"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="ENVIAR"
                        android:textColor="#000000"
                        android:textSize="16sp"

android:backgroundTint="@color/boton_pulsado"/>

                    <View
                        android:layout_width="30sp"
                        android:layout_height="match_parent"
                        />
```

```
646
647                 </LinearLayout>
648
649                 <View
650                     android:layout_width="match_parent"
651                     android:layout_height="10sp"
652                     />
653
654             </LinearLayout>
655
656         </androidx.cardview.widget.CardView>
657
658     </LinearLayout>
659
660
661
662
663
664
665
666
667     <!-- SECCION DONDE SE MUESTRA EL VALOR DE LOS DOS SENSORES -->
668     <androidx.cardview.widget.CardView
669         android:id="@+id/tarjeta_valores_tiempo_real"
670         android:layout_width="match_parent"
671         android:layout_height="320sp"
672         android:layout_gravity="center"
673         card_view:cardCornerRadius="10dp"
674         card_view:cardElevation="10dp"
675         android:layout_margin="10sp">
676
677         <LinearLayout
678 xmlns:android="http://schemas.android.com/apk/res/android"
679
680             xmlns:tools="http://schemas.android.com/tools"
681             android:id="@+id/LinearLayout5"
682             android:layout_width="match_parent"
683             android:layout_height="320sp"
684             android:orientation="horizontal"
685             android:gravity="center"
686             android:background="@drawable/fondo">
687
688             <View
689                 android:layout_width="20sp"
690                 android:layout_height="match_parent"
691                 />
692
693             <!-- BRAZO IZQUIERDO -->
694             <LinearLayout
695 xmlns:android="http://schemas.android.com/apk/res/android"
696
697                 xmlns:tools="http://schemas.android.com/tools"
698                 android:id="@+id/LinearLayout6"
699                 android:layout_width="0sp"
```

```
700                 android:layout_weight="1"
701                 android:layout_height="match_parent"
702                 android:orientation="vertical">
703 
704 
705                 <View
706                     android:layout_width="match_parent"
707                     android:layout_height="215sp"
708                     />
709 
710                 <!-- EJE X -->
711                 <TextView
712 
713 android:id="@+id/texto_brazo_izquierdo_acelerometro_x"
714                     android:gravity="left"
715                     android:layout_width="wrap_content"
716                     android:layout_height="wrap_content"
717                     android:text="X:"
718                     android:textColor="#000000"
719                     android:textSize="24sp"
720                     />
721 
722                 <!-- EJE Y -->
723                 <TextView
724 
725 android:id="@+id/texto_brazo_izquierdo_acelerometro_y"
726                     android:gravity="left"
727                     android:layout_width="wrap_content"
728                     android:layout_height="wrap_content"
729                     android:text="Y:"
730                     android:textColor="#000000"
731                     android:textSize="24sp"
732                     />
733 
734                 <!-- EJE Z -->
735                 <TextView
736 
737 android:id="@+id/texto_brazo_izquierdo_acelerometro_z"
738                     android:gravity="left"
739                     android:layout_width="wrap_content"
740                     android:layout_height="wrap_content"
741                     android:text="Z:"
742                     android:textColor="#000000"
743                     android:textSize="24sp"
744                     />
745 
746             </LinearLayout>
747 
748 
749             <View
750                 android:layout_width="180sp"
751                 android:layout_height="match_parent"
752                 />
753 
```

```
754
755             <!-- BRAZO DERECHO -->
756             <LinearLayout
757 xmlns:android="http://schemas.android.com/apk/res/android"
758
759                 xmlns:tools="http://schemas.android.com/tools"
760                 android:id="@+id/LinearLayout7"
761                 android:layout_width="0sp"
762                 android:layout_weight="1"
763                 android:layout_height="match_parent"
764                 android:orientation="vertical" >
765
766
767                 <View
768                     android:layout_width="match_parent"
769                     android:layout_height="215sp"
770                     />
771
772
773                 <!-- EJE X -->
774                 <TextView
775
776 android:id="@+id/texto_brazo_derecho_acelerometro_x"
777                     android:gravity="left"
778                     android:layout_width="wrap_content"
779                     android:layout_height="wrap_content"
780                     android:text="X:"
781                     android:textColor="#000000"
782                     android:textSize="24sp"
783                     />
784
785                 <!-- EJE Y -->
786                 <TextView
787
788 android:id="@+id/texto_brazo_derecho_acelerometro_y"
789                     android:gravity="left"
790                     android:layout_width="wrap_content"
791                     android:layout_height="wrap_content"
792                     android:text="Y:"
793                     android:textColor="#000000"
794                     android:textSize="24sp"
795                     />
796
797                 <!-- EJE Z -->
798                 <TextView
799
800 android:id="@+id/texto_brazo_derecho_acelerometro_z"
801                     android:gravity="left"
802                     android:layout_width="wrap_content"
803                     android:layout_height="wrap_content"
804                     android:text="Z:"
805                     android:textColor="#000000"
806                     android:textSize="24sp"
807                     />
```

```
808
809             </LinearLayout>
810
811         </LinearLayout>
812
813     </androidx.cardview.widget.CardView>
814
815
816
817
818
819     <!-- SECCION DONDE SE MUESTRA EL GRÁFICO CON LOS VALORES DE LOS
820 SENSORES -->
821     <androidx.cardview.widget.CardView
822         android:id="@+id/tarjeta_grafico"
823         android:layout_width="match_parent"
824         android:layout_height="wrap_content"
825         android:layout_gravity="center"
826         card_view:cardCornerRadius="10dp"
827         card_view:cardElevation="10dp"
828         android:layout_margin="10sp"
829         >
830
831         <LinearLayout
832 xmlns:android="http://schemas.android.com/apk/res/android"
833             xmlns:tools="http://schemas.android.com/tools"
834             android:layout_width="match_parent"
835             android:layout_height="match_parent"
836             android:orientation="vertical" >
837
838             <!-- CONTROLES PARA INICIAR, PAUSAR O PARAR LA TOMA DE
839 DATOS -->
840             <LinearLayout
841 xmlns:android="http://schemas.android.com/apk/res/android"
842             xmlns:tools="http://schemas.android.com/tools"
843             android:layout_width="match_parent"
844             android:layout_height="match_parent"
845             android:gravity="center"
846             android:orientation="horizontal" >
847
848             <!-- MOSTRAR ACELERACION X BRAZO IZQUIERDO -->
849             <Button
850
851 android:id="@+id/boton_brazo_izquierdo_mostrar_aceleracion_x"
852                 android:layout_width="40sp"
853                 android:layout_height="wrap_content"
854                 android:text="X"
855                 android:textColor="#000000"
856                 android:textSize="16sp"
857
858 android:backgroundTint="@color/brazo_izquierdo_aceleracion_x_pulsado"/
859 >
860
861             <!-- MOSTRAR ACELERACION Y BRAZO IZQUIERDO -->
```

```
862             <Button
863
864   android:id="@+id/boton_brazo_izquierdo_mostrar_aceleracion_y"
865                 android:layout_width="40sp"
866                 android:layout_height="wrap_content"
867                 android:text="Y"
868                 android:textColor="#000000"
869                 android:textSize="16sp"
870
871   android:backgroundTint="@color/brazo_izquierdo_aceleracion_y_pulsado"/
872   >
873
874             <!-- MOSTRAR ACELERACION Z BRAZO IZQUIERDO -->
875             <Button
876
877   android:id="@+id/boton_brazo_izquierdo_mostrar_aceleracion_z"
878                 android:layout_width="40sp"
879                 android:layout_height="wrap_content"
880                 android:text="Z"
881                 android:textColor="#000000"
882                 android:textSize="16sp"
883
884   android:backgroundTint="@color/brazo_izquierdo_aceleracion_z_pulsado"/
885   >
886
887                 <!-- TOMAR MUESTRAS -->
888             <ImageButton
889                 android:id="@+id/boton_brazo_izquierdo_play"
890                 android:layout_width="50sp"
891                 android:layout_height="50sp"
892                 android:background="@drawable/play"/>
893
894             <!-- PAUSAR MUESTRAS -->
895             <ImageButton
896                 android:id="@+id/boton_brazo_izquierdo_pause"
897                 android:layout_width="50sp"
898                 android:layout_height="50sp"
899                 android:background="@drawable/pause"/>
900
901             <!-- FINALIZAR LA TOMA DE MUESTRAS -->
902             <ImageButton
903                 android:id="@+id/boton_brazo_izquierdo_stop"
904                 android:layout_width="50sp"
905                 android:layout_height="50sp"
906                 android:background="@drawable/stop"/>
907
908             <!-- MOSTRAR ACELERACION X BRAZO DERECHO -->
909             <Button
910
911   android:id="@+id/boton_brazo_derecho_mostrar_aceleracion_x"
912                 android:layout_width="40sp"
913                 android:layout_height="wrap_content"
914                 android:text="X"
915                 android:textColor="#000000"
```

```
916                 android:textSize="16sp"
917
918 android:backgroundTint="@color/brazo_derecho_aceleracion_x_pulsado"/>
919
920             <!-- MOSTRAR ACELERACION Y BRAZO DERECHO -->
921             <Button
922
923 android:id="@+id/boton_brazo_derecho_mostrar_aceleracion_y"
924                 android:layout_width="40sp"
925                 android:layout_height="wrap_content"
926                 android:text="Y"
927                 android:textColor="#000000"
928                 android:textSize="16sp"
929
930 android:backgroundTint="@color/brazo_derecho_aceleracion_y_pulsado"/>
931
932             <!-- MOSTRAR ACELERACION Z BRAZO DERECHO -->
933             <Button
934
935 android:id="@+id/boton_brazo_derecho_mostrar_aceleracion_z"
936                 android:layout_width="40sp"
937                 android:layout_height="wrap_content"
938                 android:text="Z"
939                 android:textColor="#000000"
940                 android:textSize="16sp"
941
942 android:backgroundTint="@color/brazo_derecho_aceleracion_z_pulsado"/>
943
944         </LinearLayout>
945
946
947             <!-- GRÁFICO -->
948             <com.jjoe64.graphview.GraphView
949                 android:id="@+id/grafico_brazo_izquierdo"
950                 android:layout_width="match_parent"
951                 android:layout_height="200sp"
952                  />
953
954
955
956         </LinearLayout>
957
958
959
960     </androidx.cardview.widget.CardView>
961
962
963
964 </LinearLayout>
```

7.4 Código Fuente Android: Servicio.java

```
1   package elias.prueba;
2
3   import android.Manifest;
4   import android.app.Service;
5   import android.bluetooth.BluetoothDevice;
6   import android.bluetooth.BluetoothGatt;
7   import android.bluetooth.BluetoothGattCallback;
8   import android.bluetooth.BluetoothGattCharacteristic;
9   import android.bluetooth.BluetoothGattDescriptor;
10  import android.bluetooth.BluetoothProfile;
11  import android.content.BroadcastReceiver;
12  import android.content.Context;
13  import android.content.Intent;
14  import android.content.pm.PackageManager;
15  import android.os.Binder;
16  import android.os.Bundle;
17  import android.os.IBinder;
18  import android.util.Log;
19
20  import androidx.core.app.ActivityCompat;
21  import androidx.core.content.ContextCompat;
22
23  import java.nio.charset.StandardCharsets;
24  import java.util.UUID;
25
26  public class BluetoothLeService extends Service {
27
28      // BRAZO IZQUIERDO
29          private BluetoothGatt mBluetoothGatt_brazo_izquierdo;
30
31          private BluetoothGattCharacteristic
32  caracteristica_acelerometro_x_brazo_izquierdo;
33          private BluetoothGattCharacteristic
34  caracteristica_acelerometro_y_brazo_izquierdo;
35          private BluetoothGattCharacteristic
36  caracteristica_acelerometro_z_brazo_izquierdo;
37          private BluetoothGattCharacteristic
38  caracteristica_acelerometro_sensibilidad_brazo_izquierdo;
39
40          private BluetoothGattCharacteristic
41  caracteristica_configuracion_nombre_brazo_izquierdo;
42          private BluetoothGattCharacteristic
43  caracteristica_configuracion_muestreo_brazo_izquierdo;
44          private BluetoothGattCharacteristic
45  caracteristica_configuracion_sensibilidad_brazo_izquierdo;
46
47      // BRAZO DERECHO
48          private BluetoothGatt mBluetoothGatt_brazo_derecho;
49
50          private BluetoothGattCharacteristic
51  caracteristica_acelerometro_x_brazo_derecho;
```

```
52          private BluetoothGattCharacteristic
53  caracteristica_acelerometro_y_brazo_derecho;
54          private BluetoothGattCharacteristic
55  caracteristica_acelerometro_z_brazo_derecho;
56          private BluetoothGattCharacteristic
57  caracteristica_acelerometro_sensibilidad_brazo_derecho;
58
59          private BluetoothGattCharacteristic
60  caracteristica_configuracion_nombre_brazo_derecho;
61          private BluetoothGattCharacteristic
62  caracteristica_configuracion_muestreo_brazo_derecho;
63          private BluetoothGattCharacteristic
64  caracteristica_configuracion_sensibilidad_brazo_derecho;
65
66
67
68      // UUID
69
70          // ACELEROMETRO
71                           private static final UUID
72  uuid_servicio_acelerometro = UUID.fromString("00000001-0001-0001-0001-
73  000000000001");
74                  private static final UUID
75  uuid_caracteristica_acelerometro_x = UUID.fromString("00000001-0001-
76  0001-0001-000000000002");
77                  private static final UUID
78  uuid_caracteristica_acelerometro_y = UUID.fromString("00000001-0001-
79  0001-0001-000000000003");
80                  private static final UUID
81  uuid_caracteristica_acelerometro_z = UUID.fromString("00000001-0001-
82  0001-0001-000000000004");
83      private static final UUID
84  uuid_caracteristica_acelerometro_sensibilidad =
85  UUID.fromString("00000001-0001-0001-0001-000000000005");
86
87
88          // MUESTREO Y SENSIBILIDAD
89                          private static final UUID
90  uuid_servicio_configuracion = UUID.fromString("00000001-0001-0001-
91  0004-000000000001");
92         private static final UUID
93  uuid_caracteristica_configuracion_muestreo =
94  UUID.fromString("00000001-0001-0001-0004-000000000003");
95     private static final UUID
96  uuid_caracteristica_configuracion_sensibilidad =
97  UUID.fromString("00000001-0001-0001-0004-000000000004");
98
99
100
101     // BRAZO IZQUIERDO
102      public final static String ACTION_GATT_CONNECTED_brazo_izquierdo =
103 "com.example.bluetooth.le.ACTION_GATT_CONNECTED_brazo_izquierdo";
```

```
104     public final static String
105 ACTION_GATT_DISCONNECTED_brazo_izquierdo =
106 "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED_brazo_izquierdo";
107     public final static String
108 ACTION_GATT_SERVICES_DISCOVERED_brazo_izquierdo =
109 "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED_brazo_izquie
110 rdo";
111     public final static String ACTION_DATA_AVAILABLE_brazo_izquierdo =
112 "com.example.bluetooth.le.ACTION_DATA_AVAILABLE_brazo_izquierdo";
113
114     // BRAZO DERECHO
115     public final static String ACTION_GATT_CONNECTED_brazo_derecho =
116 "com.example.bluetooth.le.ACTION_GATT_CONNECTED_brazo_derecho";
117     public final static String ACTION_GATT_DISCONNECTED_brazo_derecho
118 = "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED_brazo_derecho";
119     public final static String
120 ACTION_GATT_SERVICES_DISCOVERED_brazo_derecho =
121 "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED_brazo_derech
122 o";
123     public final static String ACTION_DATA_AVAILABLE_brazo_derecho =
124 "com.example.bluetooth.le.ACTION_DATA_AVAILABLE_brazo_derecho";
125
126
127
128     // BRAZO IZQUIERDO
129     private final BluetoothGattCallback mGattCallback_brazo_izquierdo
130 = new BluetoothGattCallback() {
131
132         // este método se ejecuta cuando haya cambiado el estado de la
133 conexion con el dispositivo (estado: conectado o desconectado)
134         @Override
135         public void onConnectionStateChange(BluetoothGatt gatt, int
136 status, int newState) {
137             String intentAction;
138
139             // me he conectado al dispositivo
140             if (newState == BluetoothProfile.STATE_CONNECTED) {
141
142                 // mando un broadcast diciendo que me he conectado
143                 intentAction = ACTION_GATT_CONNECTED_brazo_izquierdo;
144                 Log.i("problema", "servicio: onConnectionStateChange
145 CONNECTED a izquierdo, envio intent
146 ACTION_GATT_CONNECTED_brazo_izquierdo");
147
148
149                 broadcastUpdate(intentAction);
150
151                 if
152 (ContextCompat.checkSelfPermission(BluetoothLeService.this,
153 Manifest.permission.BLUETOOTH)
154                         != PackageManager.PERMISSION_GRANTED) {
155                     // Permission is not granted
156                 }
157
```

```
158                 // descubro los servicios del dispositivo
159                 Log.i("problema", "servicio: onConnectionStateChange
160   CONNECTED a izquierdo, descubro los servicios");
161                 mBluetoothGatt_brazo_izquierdo.discoverServices();
162
163                 // me he desconectado del dispositivo
164             } else if (newState ==
165   BluetoothProfile.STATE_DISCONNECTED) {
166
167
168                 // mando un broadcast diciendo que me he desconectado
169                 intentAction =
170   ACTION_GATT_DISCONNECTED_brazo_izquierdo;
171                 Log.i("problema", "servicio: onConnectionStateChange
172   DISCONNECTED a izquierdo, envio intent
173   ACTION_GATT_DISCONNECTED_brazo_izquierdo");
174
175                 broadcastUpdate(intentAction);
176             }
177         }
178
179         // este método se ejecuta cuando he terminado de descubrir los
180   servicios del dispositivo
181         @Override
182         public void onServicesDiscovered(BluetoothGatt gatt, int
183   status) {
184
185             // Comprobamos los permisos
186                 final int CODIGO_PERMISOS_BLUETOOTH = 1;
187                 final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
188                 final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
189                 final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
190
191                 // BLUETOOTH
192                 int estadoDePermiso_1 =
193   ContextCompat.checkSelfPermission(BluetoothLeService.this,
194   Manifest.permission.BLUETOOTH);
195                 if(estadoDePermiso_1
196   ==PackageManager.PERMISSION_GRANTED) {
197                     // Aquí el usuario dio permisos para acceder al
198   bluetooth
199                 } else {
200                 }
201
202                 // BLUETOOTH_ADMIN
203                 int estadoDePermiso_2 =
204   ContextCompat.checkSelfPermission(BluetoothLeService.this,
205   Manifest.permission.BLUETOOTH_ADMIN);
206                 if(estadoDePermiso_2
207   ==PackageManager.PERMISSION_GRANTED) {
208                     // Aquí el usuario dio permisos para acceder al
209   bluetooth
210                 } else {
211                 }
```

```
212
213                 // BLUETOOTH_CONNECT
214                 int estadoDePermiso_3 =
215 ContextCompat.checkSelfPermission(BluetoothLeService.this,
216 Manifest.permission.BLUETOOTH_CONNECT);
217                 if(estadoDePermiso_3
218 ==PackageManager.PERMISSION_GRANTED) {
219                     // Aquí el usuario dio permisos para acceder al
220 bluetooth
221                 } else {
222                 }
223
224                 // BLUETOOTH_SCAN
225                 int estadoDePermiso_4 =
226 ContextCompat.checkSelfPermission(BluetoothLeService.this,
227 Manifest.permission.BLUETOOTH_SCAN);
228                 if(estadoDePermiso_4
229 ==PackageManager.PERMISSION_GRANTED) {
230                     // Aquí el usuario dio permisos para acceder al
231 bluetooth
232                 } else {
233                 }
234
235
236             if (status == BluetoothGatt.GATT_SUCCESS) {
237                 // servicios descubiertos correctamente
238
239 broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED_brazo_izquierdo);
240
241                 // inicializo la caracteristica de la que quiero
242 recibir notificaciones
243                 caracteristica_acelerometro_x_brazo_izquierdo =
244 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_acelerometro).
245 getCharacteristic(uuid_caracteristica_acelerometro_x);
246                 caracteristica_acelerometro_y_brazo_izquierdo =
247 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_acelerometro).
248 getCharacteristic(uuid_caracteristica_acelerometro_y);
249                 caracteristica_acelerometro_z_brazo_izquierdo =
250 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_acelerometro).
251 getCharacteristic(uuid_caracteristica_acelerometro_z);
252
253 caracteristica_acelerometro_sensibilidad_brazo_izquierdo =
254 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_acelerometro).
255 getCharacteristic(uuid_caracteristica_acelerometro_sensibilidad);
256
257                 caracteristica_configuracion_muestreo_brazo_izquierdo
258 =
259 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_configuracion)
260 .getCharacteristic(uuid_caracteristica_configuracion_muestreo);
261
262 caracteristica_configuracion_sensibilidad_brazo_izquierdo =
263 mBluetoothGatt_brazo_izquierdo.getService(uuid_servicio_configuracion)
264 .getCharacteristic(uuid_caracteristica_configuracion_sensibilidad);
265
```

```
266
267                 // habilito las notificaciones, para que cada vez que
268 reciba una notificacion se active el callback onCharacteristicChanged
269
270 mBluetoothGatt_brazo_izquierdo.setCharacteristicNotification(caracteri
271 stica_acelerometro_x_brazo_izquierdo, true);
272
273 mBluetoothGatt_brazo_izquierdo.setCharacteristicNotification(caracteri
274 stica_acelerometro_y_brazo_izquierdo, true);
275
276 mBluetoothGatt_brazo_izquierdo.setCharacteristicNotification(caracteri
277 stica_acelerometro_z_brazo_izquierdo, true);
278
279
280
281             } else {
282                 // error descubriendo los servicios
283             }
284         }
285
286         // este método se ejecuta cuando he leido una caracteristica
287         @Override
288         public void onCharacteristicRead(BluetoothGatt gatt,
289 BluetoothGattCharacteristic characteristic, int status) {
290             if (status == BluetoothGatt.GATT_SUCCESS) {
291                 broadcastUpdate(ACTION_DATA_AVAILABLE_brazo_izquierdo,
292 characteristic);
293             }
294         }
295
296         // este método se ejecuta cuando una caracteristica ha
297 cambiado
298         @Override
299         public void onCharacteristicChanged(BluetoothGatt gatt,
300 BluetoothGattCharacteristic characteristic) {
301             broadcastUpdate(ACTION_DATA_AVAILABLE_brazo_izquierdo,
302 characteristic);
303         }
304     };
305
306
307
308
309     // BRAZO DERECHO
310     private final BluetoothGattCallback mGattCallback_brazo_derecho =
311 new BluetoothGattCallback() {
312
313         // este método se ejecuta cuando haya cambiado el estado de la
314 conexion con el dispositivo (estado: conectado o desconectado)
315         @Override
316         public void onConnectionStateChange(BluetoothGatt gatt, int
317 status, int newState) {
318             String intentAction;
319
```

```
320             // me he conectado al dispositivo
321             if (newState == BluetoothProfile.STATE_CONNECTED) {
322 
323                 // mando un broadcast diciendo que me he conectado
324                 intentAction = ACTION_GATT_CONNECTED_brazo_derecho;
325                 Log.i("problema", "servicio: onConnectionStateChange
326 CONNECTED a derecho, envio intent
327 ACTION_GATT_CONNECTED_brazo_derecho");
328 
329                 broadcastUpdate(intentAction);
330 
331                 if
332 (ContextCompat.checkSelfPermission(BluetoothLeService.this,
333 Manifest.permission.BLUETOOTH)
334                         != PackageManager.PERMISSION_GRANTED) {
335                     // Permission is not granted
336                 }
337 
338                 // descubro los servicios del dispositivo
339                 Log.i("problema", "servicio: onConnectionStateChange
340 CONNECTED a derecho, descubro los servicios");
341                 mBluetoothGatt_brazo_derecho.discoverServices();
342 
343                 // me he desconectado del dispositivo
344             } else if (newState ==
345 BluetoothProfile.STATE_DISCONNECTED) {
346 
347 
348                 // mando un broadcast diciendo que me he desconectado
349                 intentAction = ACTION_GATT_DISCONNECTED_brazo_derecho;
350                 Log.i("problema", "servicio: onConnectionStateChange
351 DISCONNECTED a derecho, envio intent
352 ACTION_GATT_DISCONNECTED_brazo_derecho");
353 
354                 broadcastUpdate(intentAction);
355             }
356         }
357 
358         // este método se ejecuta cuando he terminado de descubrir los
359 servicios del dispositivo
360         @Override
361         public void onServicesDiscovered(BluetoothGatt gatt, int
362 status) {
363 
364             // Comprobamos los permisos
365             final int CODIGO_PERMISOS_BLUETOOTH = 1;
366             final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
367             final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
368             final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
369 
370             // BLUETOOTH
371             int estadoDePermiso_1 =
372 ContextCompat.checkSelfPermission(BluetoothLeService.this,
373 Manifest.permission.BLUETOOTH);
```

```
374             if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
375 {
376                 // Aquí el usuario dio permisos para acceder al
377 bluetooth
378             } else {
379             }
380 
381             // BLUETOOTH_ADMIN
382             int estadoDePermiso_2 =
383 ContextCompat.checkSelfPermission(BluetoothLeService.this,
384 Manifest.permission.BLUETOOTH_ADMIN);
385             if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
386 {
387                 // Aquí el usuario dio permisos para acceder al
388 bluetooth
389             } else {
390             }
391 
392             // BLUETOOTH_CONNECT
393             int estadoDePermiso_3 =
394 ContextCompat.checkSelfPermission(BluetoothLeService.this,
395 Manifest.permission.BLUETOOTH_CONNECT);
396             if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
397 {
398                 // Aquí el usuario dio permisos para acceder al
399 bluetooth
400             } else {
401             }
402 
403             // BLUETOOTH_SCAN
404             int estadoDePermiso_4 =
405 ContextCompat.checkSelfPermission(BluetoothLeService.this,
406 Manifest.permission.BLUETOOTH_SCAN);
407             if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
408 {
409                 // Aquí el usuario dio permisos para acceder al
410 bluetooth
411             } else {
412             }
413 
414 
415             if (status == BluetoothGatt.GATT_SUCCESS) {
416                 // servicios descubiertos correctamente
417 
418 broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED_brazo_derecho);
419 
420                 // inicializo la caracteristica de la que quiero
421 recibir notificaciones
422                 caracteristica_acelerometro_x_brazo_derecho =
423 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_acelerometro).ge
424 tCharacteristic(uuid_caracteristica_acelerometro_x);
425                 caracteristica_acelerometro_y_brazo_derecho =
426 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_acelerometro).ge
427 tCharacteristic(uuid_caracteristica_acelerometro_y);
```

```
428                 caracteristica_acelerometro_z_brazo_derecho =
429 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_acelerometro).ge
430 tCharacteristic(uuid_caracteristica_acelerometro_z);
431                 caracteristica_acelerometro_sensibilidad_brazo_derecho
432 =
433 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_acelerometro).ge
434 tCharacteristic(uuid_caracteristica_acelerometro_sensibilidad);
435
436                 caracteristica_configuracion_muestreo_brazo_derecho =
437 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_configuracion).g
438 etCharacteristic(uuid_caracteristica_configuracion_muestreo);
439
440 caracteristica_configuracion_sensibilidad_brazo_derecho =
441 mBluetoothGatt_brazo_derecho.getService(uuid_servicio_configuracion).g
442 etCharacteristic(uuid_caracteristica_configuracion_sensibilidad);
443
444
445                 // habilito las notificaciones, para que cada vez que
446 reciba una notificacion se active el callback onCharacteristicChanged
447
448 mBluetoothGatt_brazo_derecho.setCharacteristicNotification(caracterist
449 ica_acelerometro_x_brazo_derecho, true);
450
451 mBluetoothGatt_brazo_derecho.setCharacteristicNotification(caracterist
452 ica_acelerometro_y_brazo_derecho, true);
453
454 mBluetoothGatt_brazo_derecho.setCharacteristicNotification(caracterist
455 ica_acelerometro_z_brazo_derecho, true);
456
457
458
459             } else {
460                 // error descubriendo los servicios
461             }
462         }
463
464         // este método se ejecuta cuando he leido una caracteristica
465         @Override
466         public void onCharacteristicRead(BluetoothGatt gatt,
467 BluetoothGattCharacteristic characteristic, int status) {
468             if (status == BluetoothGatt.GATT_SUCCESS) {
469                 broadcastUpdate(ACTION_DATA_AVAILABLE_brazo_derecho,
470 characteristic);
471             }
472         }
473
474         // este método se ejecuta cuando una caracteristica ha
475 cambiado
476         @Override
477         public void onCharacteristicChanged(BluetoothGatt gatt,
478 BluetoothGattCharacteristic characteristic) {
479             broadcastUpdate(ACTION_DATA_AVAILABLE_brazo_derecho,
480 characteristic);
481         }
```

```
482     };
483 
484 
485 
486 
487     // metodo para enviar un intent cuando nos conectemos,
488 desconectemos o hayamos descubierto los servicios del dispositivo
489     private void broadcastUpdate(final String action) {
490         final Intent intent = new Intent(action);
491         sendBroadcast(intent);
492     }
493 
494 
495     // metodo para enviar un intent cuando hayamos leido una
496 caracteristica del dispositivo , o una caracteristica haya cambiado
497     private void broadcastUpdate(final String action, final
498 BluetoothGattCharacteristic characteristic) {
499         final Intent intent = new Intent(action);
500         final Bundle extras = new Bundle();
501 
502         // extraigo el byte de datos que he recibido
503         final byte[] data = characteristic.getValue();
504 
505 
506         // ACELEROMETRO
507 
508             // comprobamos si la caracteristica de la que hemos
509 recibido la notificacion es la del acelerometro x
510             if
511 (uuid_caracteristica_acelerometro_x.equals(characteristic.getUuid()))
512 {
513 
514 
515                 extras.putString("recibido_acelerometro_x", new
516 String(data));
517                 extras.putString("recibido_quien",
518 "recibido_acelerometro_x");
519                 intent.putExtras(extras);
520 
521 
522                 sendBroadcast(intent);
523 
524             }
525 
526             // comprobamos si la caracteristica de la que hemos
527 recibido la notificacion es la del acelerometro y
528             if
529 (uuid_caracteristica_acelerometro_y.equals(characteristic.getUuid()))
530 {
531 
532                 extras.putString("recibido_acelerometro_y", new
533 String(data));
534                 extras.putString("recibido_quien",
535 "recibido_acelerometro_y");
```

```
536                 intent.putExtras(extras);
537                 sendBroadcast(intent);
538 
539             }
540 
541             // comprobamos si la caracteristica de la que hemos
542 recibido la notificacion es la del acelerometro z
543             if
544 (uuid_caracteristica_acelerometro_z.equals(characteristic.getUuid()))
545 {
546 
547                 extras.putString("recibido_acelerometro_z", new
548 String(data));
549                 extras.putString("recibido_quien",
550 "recibido_acelerometro_z");
551                 intent.putExtras(extras);
552                 sendBroadcast(intent);
553 
554             }
555 
556 
557 
558         // MUESTREO
559             // comprobamos si la caracteristica de la que hemos
560 recibido el valor es la del muestreo
561             if
562 (uuid_caracteristica_configuracion_muestreo.equals(characteristic.getU
563 uid())) {
564 
565                 extras.putString("recibido_configuracion_muestreo",
566 new String(data));
567                 extras.putString("recibido_quien",
568 "recibido_configuracion_muestreo");
569                 intent.putExtras(extras);
570                 sendBroadcast(intent);
571 
572             }
573 
574 
575         // SENSIBILIDAD
576             // comprobamos si la caracteristica de la que hemos
577 recibido el valor es la de la sensibilidad
578             if
579 (uuid_caracteristica_configuracion_sensibilidad.equals(characteristic.
580 getUuid())) {
581 
582 
583 extras.putString("recibido_configuracion_sensibilidad", new
584 String(data));
585                 extras.putString("recibido_quien",
586 "recibido_configuracion_sensibilidad");
587                 intent.putExtras(extras);
588                 sendBroadcast(intent);
589 
```

```
590             }
591
592
593         }
594
595
596         // BRAZO IZQUIERDO
597         // metodo para escribir una caracteristica
598         public void writeCharacteristic_brazo_izquierdo(String
599     quien_quiere_escribir, String mensaje_a_enviar){
600
601             // Comprobamos los permisos
602                 final int CODIGO_PERMISOS_BLUETOOTH = 1;
603                 final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
604                 final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
605                 final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
606
607                 // BLUETOOTH
608                 int estadoDePermiso_1 =
609     ContextCompat.checkSelfPermission(BluetoothLeService.this,
610     Manifest.permission.BLUETOOTH);
611                 if (estadoDePermiso_1 ==
612     PackageManager.PERMISSION_GRANTED) {
613                     // Aquí el usuario dio permisos para acceder al
614     bluetooth
615                 } else {
616                 }
617
618                 // BLUETOOTH_ADMIN
619                 int estadoDePermiso_2 =
620     ContextCompat.checkSelfPermission(BluetoothLeService.this,
621     Manifest.permission.BLUETOOTH_ADMIN);
622                 if (estadoDePermiso_2 ==
623     PackageManager.PERMISSION_GRANTED) {
624                     // Aquí el usuario dio permisos para acceder al
625     bluetooth
626                 } else {
627                 }
628
629                 // BLUETOOTH_CONNECT
630                 int estadoDePermiso_3 =
631     ContextCompat.checkSelfPermission(BluetoothLeService.this,
632     Manifest.permission.BLUETOOTH_CONNECT);
633                 if (estadoDePermiso_3 ==
634     PackageManager.PERMISSION_GRANTED) {
635                     // Aquí el usuario dio permisos para acceder al
636     bluetooth
637                 } else {
638                 }
639
640                 // BLUETOOTH_SCAN
641                 int estadoDePermiso_4 =
642     ContextCompat.checkSelfPermission(BluetoothLeService.this,
643     Manifest.permission.BLUETOOTH_SCAN);
```

```
644             if (estadoDePermiso_4 ==
645   PackageManager.PERMISSION_GRANTED) {
646                 // Aquí el usuario dio permisos para acceder al
647   bluetooth
648             } else {
649             }
650
651
652           // ACELEROMETRO
653
654               if(quien_quiere_escribir.equals("boton_2G")) {
655
656   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setValue("1")
657   ;
658
659   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setWriteType(
660   BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
661
662   mBluetoothGatt_brazo_izquierdo.writeCharacteristic(caracteristica_acel
663   erometro_sensibilidad_brazo_izquierdo);
664               }
665
666               if(quien_quiere_escribir.equals("boton_4G")) {
667
668   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setValue("2")
669   ;
670
671   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setWriteType(
672   BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
673
674   mBluetoothGatt_brazo_izquierdo.writeCharacteristic(caracteristica_acel
675   erometro_sensibilidad_brazo_izquierdo);
676               }
677
678               if(quien_quiere_escribir.equals("boton_8G")) {
679
680   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setValue("3")
681   ;
682
683   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setWriteType(
684   BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
685
686   mBluetoothGatt_brazo_izquierdo.writeCharacteristic(caracteristica_acel
687   erometro_sensibilidad_brazo_izquierdo);
688               }
689
690               if(quien_quiere_escribir.equals("boton_16G")) {
691
692   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setValue("4")
693   ;
694
695   caracteristica_acelerometro_sensibilidad_brazo_izquierdo.setWriteType(
696   BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
```

```
697
698   mBluetoothGatt_brazo_izquierdo.writeCharacteristic(caracteristica_acel
699   erometro_sensibilidad_brazo_izquierdo);
700               }
701
702
703
704           // MUESTREO
705
706
707   if(quien_quiere_escribir.equals("boton_configuracion_muestreo")) {
708
709   caracteristica_configuracion_muestreo_brazo_izquierdo.setValue(mensaje
710   _a_enviar);
711
712   caracteristica_configuracion_muestreo_brazo_izquierdo.setWriteType(Blu
713   etoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
714
715   mBluetoothGatt_brazo_izquierdo.writeCharacteristic(caracteristica_conf
716   iguracion_muestreo_brazo_izquierdo);
717               }
718
719
720
721       };
722
723
724
725
726       // BRAZO DERECHO
727       // metodo para escribir una caracteristica
728       public void writeCharacteristic_brazo_derecho(String
729   quien_quiere_escribir, String mensaje_a_enviar){
730
731           // Comprobamos los permisos
732           final int CODIGO_PERMISOS_BLUETOOTH = 1;
733           final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
734           final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
735           final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
736
737           // BLUETOOTH
738           int estadoDePermiso_1 =
739   ContextCompat.checkSelfPermission(BluetoothLeService.this,
740   Manifest.permission.BLUETOOTH);
741           if (estadoDePermiso_1 == PackageManager.PERMISSION_GRANTED) {
742               // Aquí el usuario dio permisos para acceder al bluetooth
743           } else {
744           }
745
746           // BLUETOOTH_ADMIN
747           int estadoDePermiso_2 =
748   ContextCompat.checkSelfPermission(BluetoothLeService.this,
749   Manifest.permission.BLUETOOTH_ADMIN);
750           if (estadoDePermiso_2 == PackageManager.PERMISSION_GRANTED) {
```

```
751             // Aquí el usuario dio permisos para acceder al bluetooth
752         } else {
753         }
754 
755         // BLUETOOTH_CONNECT
756         int estadoDePermiso_3 =
757 ContextCompat.checkSelfPermission(BluetoothLeService.this,
758 Manifest.permission.BLUETOOTH_CONNECT);
759         if (estadoDePermiso_3 == PackageManager.PERMISSION_GRANTED) {
760             // Aquí el usuario dio permisos para acceder al bluetooth
761         } else {
762         }
763 
764         // BLUETOOTH_SCAN
765         int estadoDePermiso_4 =
766 ContextCompat.checkSelfPermission(BluetoothLeService.this,
767 Manifest.permission.BLUETOOTH_SCAN);
768         if (estadoDePermiso_4 == PackageManager.PERMISSION_GRANTED) {
769             // Aquí el usuario dio permisos para acceder al bluetooth
770         } else {
771         }
772 
773 
774         // ACELEROMETRO
775 
776         if(quien_quiere_escribir.equals("boton_2G")) {
777 
778 caracteristica_acelerometro_sensibilidad_brazo_derecho.setValue("1");
779 
780 caracteristica_acelerometro_sensibilidad_brazo_derecho.setWriteType(Bl
781 uetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
782 
783 mBluetoothGatt_brazo_derecho.writeCharacteristic(caracteristica_aceler
784 ometro_sensibilidad_brazo_derecho);
785         }
786 
787         if(quien_quiere_escribir.equals("boton_4G")) {
788 
789 caracteristica_acelerometro_sensibilidad_brazo_derecho.setValue("2");
790 
791 caracteristica_acelerometro_sensibilidad_brazo_derecho.setWriteType(Bl
792 uetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
793 
794 mBluetoothGatt_brazo_derecho.writeCharacteristic(caracteristica_aceler
795 ometro_sensibilidad_brazo_derecho);
796         }
797 
798         if(quien_quiere_escribir.equals("boton_8G")) {
799 
800 caracteristica_acelerometro_sensibilidad_brazo_derecho.setValue("3");
801 
802 caracteristica_acelerometro_sensibilidad_brazo_derecho.setWriteType(Bl
803 uetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
```

```
804
805     mBluetoothGatt_brazo_derecho.writeCharacteristic(caracteristica_aceler
806     ometro_sensibilidad_brazo_derecho);
807             }
808
809             if(quien_quiere_escribir.equals("boton_16G")) {
810
811     caracteristica_acelerometro_sensibilidad_brazo_derecho.setValue("4");
812
813     caracteristica_acelerometro_sensibilidad_brazo_derecho.setWriteType(Bl
814     uetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
815
816     mBluetoothGatt_brazo_derecho.writeCharacteristic(caracteristica_aceler
817     ometro_sensibilidad_brazo_derecho);
818             }
819
820
821
822             // MUESTREO
823
824
825     if(quien_quiere_escribir.equals("boton_configuracion_muestreo")) {
826
827     caracteristica_configuracion_muestreo_brazo_derecho.setValue(mensaje_a
828     _enviar);
829
830     caracteristica_configuracion_muestreo_brazo_derecho.setWriteType(Bluet
831     oothGattCharacteristic.WRITE_TYPE_NO_RESPONSE);
832
833     mBluetoothGatt_brazo_derecho.writeCharacteristic(caracteristica_config
834     uracion_muestreo_brazo_derecho);
835             }
836
837
838
839         };
840
841
842
843         public class LocalBinder extends Binder {
844             BluetoothLeService getService() {
845                 return BluetoothLeService.this;
846             }
847         }
848
849         @Override
850         public IBinder onBind(Intent intent) {
851             return mBinder;
852         }
853
854         @Override
855         public boolean onUnbind(Intent intent) {
856             // si la interfaz de usuario se desconecta del servicio,
857     cierro correctamente la conexion para liberar recursos
```

```
858             close();
859             return super.onUnbind(intent);
860         }
861 
862         private final IBinder mBinder = new LocalBinder();
863 
864 
865         // BRAZO IZQUIERDO
866         // metodo para conectarme al dispositivo bluetooth
867         public boolean connect_brazo_izquierdo(final String address) {
868 
869             // Comprobamos los permisos
870                 final int CODIGO_PERMISOS_BLUETOOTH = 1;
871                 final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
872                 final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
873                 final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
874 
875                 // BLUETOOTH
876                 int estadoDePermiso_1 =
877     ContextCompat.checkSelfPermission(BluetoothLeService.this,
878     Manifest.permission.BLUETOOTH);
879                 if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
880     {
881                     // Aquí el usuario dio permisos para acceder al
882     bluetooth
883                 } else {
884                 }
885 
886                 // BLUETOOTH_ADMIN
887                 int estadoDePermiso_2 =
888     ContextCompat.checkSelfPermission(BluetoothLeService.this,
889     Manifest.permission.BLUETOOTH_ADMIN);
890                 if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
891     {
892                     // Aquí el usuario dio permisos para acceder al
893     bluetooth
894                 } else {
895                 }
896 
897                 // BLUETOOTH_CONNECT
898                 int estadoDePermiso_3 =
899     ContextCompat.checkSelfPermission(BluetoothLeService.this,
900     Manifest.permission.BLUETOOTH_CONNECT);
901                 if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
902     {
903                     // Aquí el usuario dio permisos para acceder al
904     bluetooth
905                 } else {
906                 }
907 
908                 // BLUETOOTH_SCAN
909                 int estadoDePermiso_4 =
910     ContextCompat.checkSelfPermission(BluetoothLeService.this,
911     Manifest.permission.BLUETOOTH_SCAN);
```

```
912             if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
913 {
914                 // Aquí el usuario dio permisos para acceder al
915 bluetooth
916             } else {
917             }
918 
919 
920         // creamos el dispositivo bluetooth al que nos vamos a
921 conectar
922         final BluetoothDevice device =
923 principal.mBluetoothAdapter.getRemoteDevice(address);
924 
925         // nos conectamos al dispositivo
926         mBluetoothGatt_brazo_izquierdo = device.connectGatt(this,
927 false, mGattCallback_brazo_izquierdo);
928         Log.i("problema", "servicio: creado dispositivo izquierdo " +
929 mBluetoothGatt_brazo_izquierdo.getDevice());
930 
931         return true;
932     }
933 
934 
935 
936     // BRAZO DERECHO
937     // metodo para conectarme al dispositivo bluetooth
938     public boolean connect_brazo_derecho(final String address) {
939 
940         // Comprobamos los permisos
941         final int CODIGO_PERMISOS_BLUETOOTH = 1;
942         final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
943         final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
944         final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
945 
946         // BLUETOOTH
947         int estadoDePermiso_1 =
948 ContextCompat.checkSelfPermission(BluetoothLeService.this,
949 Manifest.permission.BLUETOOTH);
950         if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED) {
951             // Aquí el usuario dio permisos para acceder al bluetooth
952         } else {
953         }
954 
955         // BLUETOOTH_ADMIN
956         int estadoDePermiso_2 =
957 ContextCompat.checkSelfPermission(BluetoothLeService.this,
958 Manifest.permission.BLUETOOTH_ADMIN);
959         if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED) {
960             // Aquí el usuario dio permisos para acceder al bluetooth
961         } else {
962         }
963 
964         // BLUETOOTH_CONNECT
```

```
965         int estadoDePermiso_3 =
966 ContextCompat.checkSelfPermission(BluetoothLeService.this,
967 Manifest.permission.BLUETOOTH_CONNECT);
968         if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED) {
969             // Aquí el usuario dio permisos para acceder al bluetooth
970         } else {
971         }
972
973         // BLUETOOTH_SCAN
974         int estadoDePermiso_4 =
975 ContextCompat.checkSelfPermission(BluetoothLeService.this,
976 Manifest.permission.BLUETOOTH_SCAN);
977         if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED) {
978             // Aquí el usuario dio permisos para acceder al bluetooth
979         } else {
980         }
981
982
983         // creamos el dispositivo bluetooth al que nos vamos a
984 conectar
985         final BluetoothDevice device =
986 principal.mBluetoothAdapter.getRemoteDevice(address);
987
988         // nos conectamos al dispositivo
989         mBluetoothGatt_brazo_derecho = device.connectGatt(this, false,
990 mGattCallback_brazo_derecho);
991         Log.i("problema", "servicio: creado dispositivo derecho " +
992 mBluetoothGatt_brazo_derecho.getDevice());
993
994         return true;
995     }
996
997
998
999
1000     // BRAZO IZQUIERDO
1001     public void conocer_muestreo_brazo_izquierdo(){
1002
1003         // Comprobamos los permisos
1004             final int CODIGO_PERMISOS_BLUETOOTH = 1;
1005             final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1006             final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1007             final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1008
1009             // BLUETOOTH
1010             int estadoDePermiso_1 =
1011 ContextCompat.checkSelfPermission(BluetoothLeService.this,
1012 Manifest.permission.BLUETOOTH);
1013             if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
1014 {
1015                 // Aquí el usuario dio permisos para acceder al
1016 bluetooth
1017             } else {
1018             }
```

```
1019
1020            // BLUETOOTH_ADMIN
1021            int estadoDePermiso_2 =
1022  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1023  Manifest.permission.BLUETOOTH_ADMIN);
1024            if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
1025  {
1026                // Aquí el usuario dio permisos para acceder al
1027  bluetooth
1028            } else {
1029            }
1030
1031            // BLUETOOTH_CONNECT
1032            int estadoDePermiso_3 =
1033  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1034  Manifest.permission.BLUETOOTH_CONNECT);
1035            if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
1036  {
1037                // Aquí el usuario dio permisos para acceder al
1038  bluetooth
1039            } else {
1040            }
1041
1042            // BLUETOOTH_SCAN
1043            int estadoDePermiso_4 =
1044  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1045  Manifest.permission.BLUETOOTH_SCAN);
1046            if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
1047  {
1048                // Aquí el usuario dio permisos para acceder al
1049  bluetooth
1050            } else {
1051            }
1052
1053
1054
1055  mBluetoothGatt_brazo_izquierdo.readCharacteristic(caracteristica_confi
1056  guracion_muestreo_brazo_izquierdo);
1057      }
1058
1059
1060
1061      // BRAZO DERECHO
1062      public void conocer_muestreo_brazo_derecho(){
1063
1064          // Comprobamos los permisos
1065              final int CODIGO_PERMISOS_BLUETOOTH = 1;
1066              final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1067              final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1068              final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1069
1070              // BLUETOOTH
```

```
1071            int estadoDePermiso_1 =
1072  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1073  Manifest.permission.BLUETOOTH);
1074            if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
1075  {
1076                // Aquí el usuario dio permisos para acceder al
1077  bluetooth
1078            } else {
1079            }
1080
1081            // BLUETOOTH_ADMIN
1082            int estadoDePermiso_2 =
1083  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1084  Manifest.permission.BLUETOOTH_ADMIN);
1085            if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
1086  {
1087                // Aquí el usuario dio permisos para acceder al
1088  bluetooth
1089            } else {
1090            }
1091
1092            // BLUETOOTH_CONNECT
1093            int estadoDePermiso_3 =
1094  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1095  Manifest.permission.BLUETOOTH_CONNECT);
1096            if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
1097  {
1098                // Aquí el usuario dio permisos para acceder al
1099  bluetooth
1100            } else {
1101            }
1102
1103            // BLUETOOTH_SCAN
1104            int estadoDePermiso_4 =
1105  ContextCompat.checkSelfPermission(BluetoothLeService.this,
1106  Manifest.permission.BLUETOOTH_SCAN);
1107            if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
1108  {
1109                // Aquí el usuario dio permisos para acceder al
1110  bluetooth
1111            } else {
1112            }
1113
1114
1115
1116  mBluetoothGatt_brazo_derecho.readCharacteristic(caracteristica_configu
1117  racion_muestreo_brazo_derecho);
1118      }
1119
1120
1121
1122
1123      // BRAZO IZQUIERDO
1124      public void conocer_sensibilidad_brazo_izquierdo(){
```

```
1125
1126            // Comprobamos los permisos
1127                final int CODIGO_PERMISOS_BLUETOOTH = 1;
1128                final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1129                final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1130                final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1131
1132                // BLUETOOTH
1133                int estadoDePermiso_1 =
1134    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1135    Manifest.permission.BLUETOOTH);
1136                if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
1137    {
1138                    // Aquí el usuario dio permisos para acceder al
1139    bluetooth
1140                } else {
1141                }
1142
1143                // BLUETOOTH_ADMIN
1144                int estadoDePermiso_2 =
1145    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1146    Manifest.permission.BLUETOOTH_ADMIN);
1147                if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
1148    {
1149                    // Aquí el usuario dio permisos para acceder al
1150    bluetooth
1151                } else {
1152                }
1153
1154                // BLUETOOTH_CONNECT
1155                int estadoDePermiso_3 =
1156    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1157    Manifest.permission.BLUETOOTH_CONNECT);
1158                if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
1159    {
1160                    // Aquí el usuario dio permisos para acceder al
1161    bluetooth
1162                } else {
1163                }
1164
1165                // BLUETOOTH_SCAN
1166                int estadoDePermiso_4 =
1167    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1168    Manifest.permission.BLUETOOTH_SCAN);
1169                if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
1170    {
1171                    // Aquí el usuario dio permisos para acceder al
1172    bluetooth
1173                } else {
1174                }
1175
1176
1177    mBluetoothGatt_brazo_izquierdo.readCharacteristic(caracteristica_confi
1178    guracion_sensibilidad_brazo_izquierdo);
```

```
1179        }
1180
1181
1182
1183
1184        // BRAZO DERECHO
1185        public void conocer_sensibilidad_brazo_derecho(){
1186
1187            // Comprobamos los permisos
1188            final int CODIGO_PERMISOS_BLUETOOTH = 1;
1189            final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1190            final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1191            final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1192
1193            // BLUETOOTH
1194            int estadoDePermiso_1 =
1195    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1196    Manifest.permission.BLUETOOTH);
1197            if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED) {
1198                // Aquí el usuario dio permisos para acceder al bluetooth
1199            } else {
1200            }
1201
1202            // BLUETOOTH_ADMIN
1203            int estadoDePermiso_2 =
1204    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1205    Manifest.permission.BLUETOOTH_ADMIN);
1206            if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED) {
1207                // Aquí el usuario dio permisos para acceder al bluetooth
1208            } else {
1209            }
1210
1211            // BLUETOOTH_CONNECT
1212            int estadoDePermiso_3 =
1213    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1214    Manifest.permission.BLUETOOTH_CONNECT);
1215            if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED) {
1216                // Aquí el usuario dio permisos para acceder al bluetooth
1217            } else {
1218            }
1219
1220            // BLUETOOTH_SCAN
1221            int estadoDePermiso_4 =
1222    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1223    Manifest.permission.BLUETOOTH_SCAN);
1224            if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED) {
1225                // Aquí el usuario dio permisos para acceder al bluetooth
1226            } else {
1227            }
1228
1229
1230    mBluetoothGatt_brazo_derecho.readCharacteristic(caracteristica_configu
1231    racion_sensibilidad_brazo_derecho);
1232        }
```

```
1233
1234
1235
1236
1237        // metodo para cerrar la conexion correctamente con el dispositivo
1238    bluetooth, y asi poder liberar recursos
1239        public void close() {
1240            Log.i("problema", "servicio: estoy dentro de Close");
1241            if (mBluetoothGatt_brazo_izquierdo == null) {
1242                Log.i("problema", "servicio: Close:
1243    mBluetoothGatt_brazo_izquierdo es Nulo");
1244                return;
1245            }
1246
1247            if (mBluetoothGatt_brazo_derecho == null) {
1248                Log.i("problema", "servicio: Close:
1249    mBluetoothGatt_brazo_derecho es Nulo");
1250                return;
1251            }
1252
1253            // Comprobamos los permisos
1254                final int CODIGO_PERMISOS_BLUETOOTH = 1;
1255                final int CODIGO_PERMISOS_BLUETOOTH_ADMIN = 2;
1256                final int CODIGO_PERMISOS_BLUETOOTH_CONNECT = 3;
1257                final int CODIGO_PERMISOS_BLUETOOTH_SCAN = 4;
1258
1259                // BLUETOOTH
1260                int estadoDePermiso_1 =
1261    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1262    Manifest.permission.BLUETOOTH);
1263                if(estadoDePermiso_1 ==PackageManager.PERMISSION_GRANTED)
1264    {
1265                    // Aquí el usuario dio permisos para acceder al
1266    bluetooth
1267                } else {
1268                }
1269
1270                // BLUETOOTH_ADMIN
1271                int estadoDePermiso_2 =
1272    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1273    Manifest.permission.BLUETOOTH_ADMIN);
1274                if(estadoDePermiso_2 ==PackageManager.PERMISSION_GRANTED)
1275    {
1276                    // Aquí el usuario dio permisos para acceder al
1277    bluetooth
1278                } else {
1279                }
1280
1281                // BLUETOOTH_CONNECT
1282                int estadoDePermiso_3 =
1283    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1284    Manifest.permission.BLUETOOTH_CONNECT);
1285                if(estadoDePermiso_3 ==PackageManager.PERMISSION_GRANTED)
1286    {
```

```
1287                    // Aquí el usuario dio permisos para acceder al
1288    bluetooth
1289                } else {
1290                }
1291
1292                // BLUETOOTH_SCAN
1293                int estadoDePermiso_4 =
1294    ContextCompat.checkSelfPermission(BluetoothLeService.this,
1295    Manifest.permission.BLUETOOTH_SCAN);
1296                if(estadoDePermiso_4 ==PackageManager.PERMISSION_GRANTED)
1297    {
1298                    // Aquí el usuario dio permisos para acceder al
1299    bluetooth
1300                } else {
1301                }
1302
1303            mBluetoothGatt_brazo_izquierdo.close();
1304            mBluetoothGatt_brazo_izquierdo = null;
1305
1306            mBluetoothGatt_brazo_derecho.close();
1307            mBluetoothGatt_brazo_derecho = null;
1308
1309        }
1310
1311
1312
1313
1314
1315    }
```

7.5 Código Fuente Android: AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android">
3
4      <uses-permission android:name="android.permission.BLUETOOTH" />
5      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
6  />
7      <uses-permission
8  android:name="android.permission.BLUETOOTH_CONNECT" />
9      <uses-permission android:name="android.permission.BLUETOOTH_SCAN"
10 />
11     <uses-permission android:name="android.permission.INTERNET" />
12     <uses-permission
13 android:name="android.permission.ACCESS_NETWORK_STATE" />
14
15     <application
16         android:allowBackup="true"
17         android:icon="@drawable/icono_reloj"
18         android:label="@string/app_name"
19         android:roundIcon="@drawable/icono_reloj"
20         android:supportsRtl="true"
21         android:theme="@style/Theme.Prueba">
22
23         <activity
24             android:name=".principal"
25             android:exported="true">
26             <intent-filter>
27                 <action android:name="android.intent.action.MAIN" />
28                 <category
29 android:name="android.intent.category.LAUNCHER" />
30             </intent-filter>
31         </activity>
32
33         <service
34             android:name=".BluetoothLeService"
35             android:enabled="true"
36             android:exported="true" >
37         </service>
38
39     </application>
40
41 </manifest>
```

7.6 Código Fuente Android: Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>

    <color name="boton_pulsado">#97F896</color>
    <color name="boton_no_pulsado">#D5FFD5</color>

    <color name="sensor_conectado">#97F896</color>
    <color name="sensor_no_conectado">#FFB0B0</color>

    <color name="fondo">#E7F5FF</color>


    <color
name="brazo_izquierdo_aceleracion_x_pulsado">#55FF00</color>
    <color
name="brazo_izquierdo_aceleracion_x_no_pulsado">#DFFFCF</color>

    <color
name="brazo_izquierdo_aceleracion_y_pulsado">#FF002B</color>
    <color
name="brazo_izquierdo_aceleracion_y_no_pulsado">#FFD6DD</color>

    <color
name="brazo_izquierdo_aceleracion_z_pulsado">#0095FF</color>
    <color
name="brazo_izquierdo_aceleracion_z_no_pulsado">#D5EDFF</color>


    <color name="brazo_derecho_aceleracion_x_pulsado">#FFE600</color>
    <color
name="brazo_derecho_aceleracion_x_no_pulsado">#FFF9C8</color>

    <color name="brazo_derecho_aceleracion_y_pulsado">#9D00FF</color>
    <color
name="brazo_derecho_aceleracion_y_no_pulsado">#EDCFFF</color>

    <color name="brazo_derecho_aceleracion_z_pulsado">#00FFC4</color>
    <color
name="brazo_derecho_aceleracion_z_no_pulsado">#C9FFF3</color>



</resources>
```