UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Control del ejercicio físico sobre una bicicleta elíptica usando dispositivos Android y Arduino.

Grado en Ingeniería Informática

Autor: Jordi Peiró Castelló

Tutor: Floreal Acebrón Linuesa

2016/2017

Resumen

Actualmente, está de moda el “running” pero en el mundo en el que vivimos no todos disponemos de tiempo para salir a correr al aire libre. Precisamente por este motivo nace el presente proyecto, para poder simular un entrenamiento al aire libre desde el salón de nuestra casa o desde nuestro centro deportivo favorito.

En primer lugar, crearemos un dispositivo IMU que llevará puesto el usuario en la muñeca, a partir del movimiento de muñeca del corredor se enviará una señal al segundo dispositivo que tendremos conectado a la bicicleta. Este segundo dispositivo, dotará a la bicicleta de conexión a Internet mediante wifi para poder enviar la información del entrenamiento en tiempo real a la aplicación para televisores Android. En dicha aplicación podremos visualizar imágenes de Google StreetView. Según la información recibida desde la bicicleta elíptica (Velocidad, resistencia, giros, etc.) el corredor se irá desplazando por el mapa consiguiendo así un entrenamiento al aire libre desde casa.

Palabras clave: Android, Arduino, ESP8266, Google Maps, Bicicleta elíptica

Tabla de contenidos

Tabla de ilustraciones

Índice de tablas

1. Introducción

1.1. Objetivos

El principal objetivo de este proyecto es hacer más divertido el simple hecho de correr en una bicicleta elíptica desde casa.

Basándose en esta idea, se desarrollará una aplicación para televisores Android, en la cual, se podrá elegir entre varias ciudades o lugares emblemáticos de la naturaleza. Una vez seleccionado el punto de partida, el usuario se adentrará en una imagen de 360º de Google StreetView. Cuando la aplicación detecte movimiento desde la bicicleta elíptica, empezará a desplazarse entre las imágenes correlativas a la actual, simulado así, la experiencia de salir a correr por la ciudad o punto de la naturaleza seleccionado.

Para que la aplicación pueda detectar el movimiento del corredor, se creará un dispositivo que se conectará a la bicicleta elíptica, en adelante DCB. El DCB será el encargado de dotar a la bicicleta de conexión a internet para poder comunicar la información del entrenamiento, es decir, el DCB obtendrá la velocidad, resistencia y la dirección en la que el corredor quiere ir y lo actualizará en el servidor, para que la aplicación pueda obtener dicha información y actualizarla si es necesario.

Por otra parte, se creará un dispositivo que el corredor llevará en su muñeca, en adelante IMU, para indicar si quiere realizar un giro o cambiar la dirección de la cámara solamente girando la muñeca. El IMU se conectará de forma inalámbrica al DCB, enviará la información del giro al DCB si es que el corredor así lo desea y el DCB será el encargado de actualizar dicho valor en el servidor.

1.2. Motivación

En muchas ocasiones, ya sea a causa de una lesión o de escasez de tiempo, nos vemos obligados en entrenar en el salón de nuestra casa con el aburrimiento que ello conlleva, precisamente para solucionar este problema nació este proyecto.

Con la curiosidad de la idea en mente, se buscaron bicicletas elípticas o cintas de correr, más modernas que la que se tenía, que pudieran integrar un sistema parecido al que se estaba pensando diseñar, pero después de un tiempo de búsqueda, no se encontró nada, no existía nada parecido en el mercado, por tanto, no hubo más remedio que ponerse manos a la obra. Se investigó mucho sobre cómo se podía conectar la bicicleta elíptica a Street View, el famoso servicio de imágenes en 360º que ofrece Google Maps. Conectar la bicicleta a StreetView suponía dotarla de una conexión a internet, para poder enviar la información del entrenamiento en tiempo real al servicio de imágenes de Google. Se decidió buscar información acerca de Arduino, el rey de los microcontroladores "Open Hardware". Arduino no soporta conexión a internet vía wifi, sino que, para poder conectar el microcontrolador a internet se debe de añadir el "Arduino wifi shield". La idea no convenció así que se decidió buscar alternativas a Arduino. Finalmente se eligió NodeMCU, más adelante se explicarán con más detalle los motivos de esta decisión. NodeMCU es una plataforma centrada en el Internet de las cosas basada en el chip ESP8266 de la empresa Espressif Systems.

Teniendo solucionado el problema de la conectividad de la bicicleta, era necesario obtener la información del entrenamiento, es decir, la velocidad y la resistencia. Se buscó información sobre cómo funciona una bicicleta elíptica, en concreto nuestro modelo, Domyos VE430 de Dechatlon. Como la búsqueda no reportó apenas información de utilidad se decidió ir a una tienda Dechatlon para pedir las especificaciones del motor y del sensor de velocidad. Pasados unos días y ante la negativa de Dechatlon a facilitar dicha información se decidió desmontar la bicicleta e intentar averiguar su funcionamiento. En primer lugar, se vio que el sensor de velocidad es un simple sensor de efecto hall que actúa como interruptor y la resistencia se consigue con un motor de corriente continua que acerca o aleja un imán al disco que se gira con el pedaleo, más adelante se detallará el funcionamiento de cada uno de los sensores y componentes de la elíptica.

Cuando se consiguió hacer funcionar el motor a bajo demanda desde el NodeMCU, se pasó a la siguiente fase: ¿Cómo realizar giros desde dentro del StreetView? Para conseguir tal propósito se optó por crear un dispositivo para que el corredor lo lleve en la muñeca y así pueda indicar con un solo giro de muñeca que quiere seguir por la izquierda o por la derecha en la siguiente imagen. Después de un tiempo de investigación se consiguió conectar el chip mpu6050 al módulo wifi ESP8266, creando así el dispositivo IMU.

En este punto ya estaban creados y conectados entre sí el DCB y el IMU, por tanto, el último paso era crear la aplicación para televisores Android que utilizase toda la información registrada por ambos dispositivos.

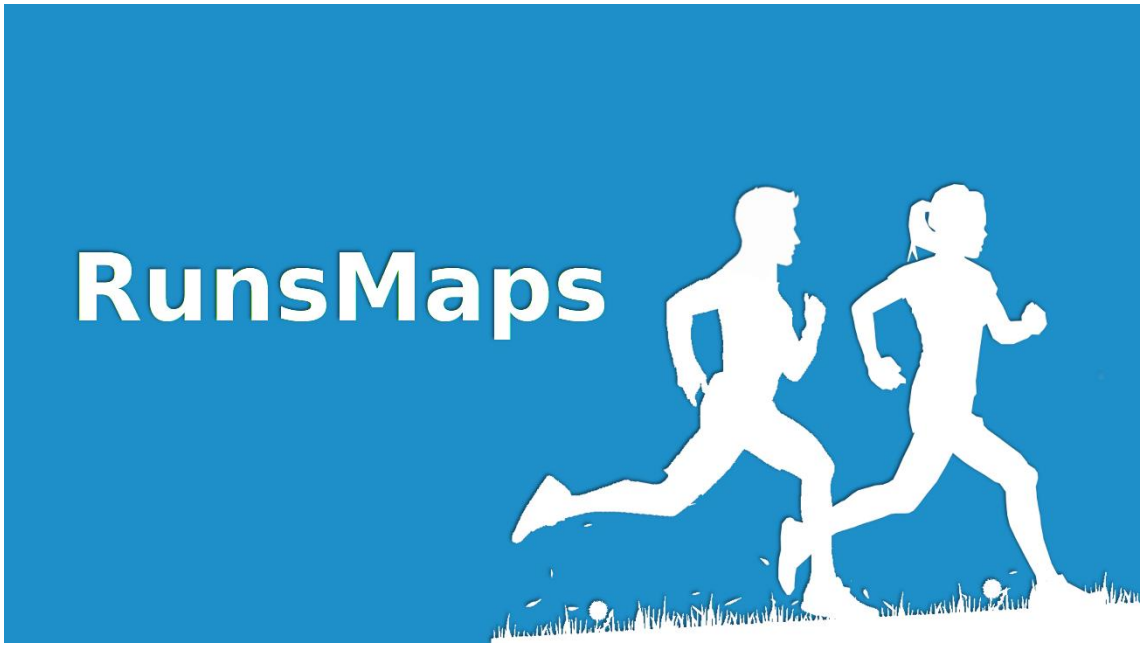


Ilustración 1: Logo de RunsMaps

1.3. Resumen

Actualmente, está de moda el “running” pero en el mundo en el que vivimos no todos disponemos de tiempo para salir a correr al aire libre. Precisamente por este motivo nace el presente proyecto, para poder simular un entrenamiento al aire libre desde el salón de nuestra casa o desde nuestro centro deportivo favorito.

En primer lugar, se creará un dispositivo IMU que llevará puesto el usuario en la muñeca, a partir del movimiento de muñeca del corredor se enviará una señal al segundo dispositivo, el cual estará conectado a la bicicleta. Este segundo dispositivo, dotará a la bicicleta de conexión a Internet mediante wifi para poder enviar la información del entrenamiento en tiempo real a la aplicación para televisores Android. En dicha aplicación se visualizarán imágenes de Google StreetView. Según la información recibida desde la bicicleta elíptica (Velocidad, resistencia, giros, etc.) el corredor se irá desplazando por el mapa consiguiendo así un entrenamiento al aire libre desde casa.

2. Estado del arte

Un tiempo después de empezar el proyecto, se descubrió que en Inglaterra una persona había empezado un proyecto parecido, en este caso, Aaron Puzey utiliza una bicicleta estática, no una elíptica como en este proyecto. Según se puede leer en su página web, Puzey quería recorrer Inglaterra de sur a norte con unas gafas de realidad virtual por una ruta prestablecida previamente. Para ello compro un velocímetro bluetooth, lo instaló en su bicicleta y lo conectó al móvil que utilizaba con las gafas de realidad virtual.

Ésta aplicación está desarrollada con Unity y el funcionamiento es un poco distinto al proyecto que nosotros nos enfrentamos. La idea base del proyecto es moverse libremente por una ciudad, por un parque, etc. En cambio, CycleVR como se llama la aplicación de Aaron Puzey, está centrada en seguir una ruta.

Ilustración 2: Ruta CycleVR

Esto desde nuestro punto de vista tiene ventajas e inconvenientes, a continuación, se listan algunos de ellos:

La ventaja principal es que permite precargar todos los panoramas (Así es como Google llama a las imágenes de StreetView) antes de iniciar la ruta y así se puede mejorar la transición entre ellos, es decir, se puede hacer más fluido el cambio entre panoramas para mejorar la experiencia de usuario. Así mismo, esto es también un inconveniente, ya que, para realizar 30 minutos de ejercicio, se tiene que esperar a que se descarguen todos los panoramas durante unos 10 minutos aproximadamente, siempre basándonos en datos de su página web.

Por otra parte, la principal desventaja encontrada es que se priva al usuario de la liberad de correr por donde quiera. Con el dispositivo IMU y la conexión a internet de la bici con el DCB, se permite al usuario correr libremente, seleccionando solo el punto de partida.

3. Entorno tecnológico

En este apartado se describen todos los componentes que forman parte del proyecto. Este proyecto está dividido en dos bloques, hardware y software. En el primero de ellos se describe todo el hardware utilizado, así como todos los dispositivos creados. En el apartado de software se detallará el proceso y la problemática de crear la aplicación para televisiones Android.

3.1. Hardware

3.1.1. Esquema del hardware utilizado:

Ilustración 3: Esquema hardware utilizado

Como se observa en la *Ilustración 3*, el esquema de componentes esta dividido en tres redes: Red domestica del usuario, red RunsMaps e Internet. A continuaón se procede a explicar cada una de ellas.

3.1.1.1. Red RunsMasp:

Ilustración 4: Red RunsMaps

La red RunsMaps está formada por los dispositivos DCB e IMU. En este caso el DCB actúa como servidor y el IMU como cliente, de esta forma se consigue conectar en tiempo real mediante un "*webSocket*" ambos dispositivos, consiguiendo así, que el DCB pueda actualizar en el servidor de firebase la información de giro, velocidad y resistencia del motor ya que el DCB está conectado a la red doméstica del corredor por la otra interface de red como se verá en el siguiente apartado.

3.1.1.2. Red doméstica:

Ilustración 5: Red Doméstica

La red doméstica del corredor esta formada, además de por el resto de dispositivos conectados, por el DCB, la televisión Android y el router.

En este caso el DCB actúa como cliente conectado a la red. Su función principal es actualizar en el servidor la velocidad y la petición de giro del corredor obtenida por la otra interface de red, que como se ha visto en el punto anterior es la encargada de crear la red RunsMaps. El DCB lee del servidor el nivel de resistencia y compara dicho nivel con el actual de la bici. Según el resultado de la comparación se encarga de subir o bajar la resistencia del motor. Por otro lado, tenemos el televisor que necesita estar conectado a internet para poder obtener los panoramas de Google Street View y toda la información del entrenamiento guadado en el servidor por parte del DCB. Cabe destacar que la aplicacción utiliza la altura de la ubicación de cada uno de los panoramas

para poder calcular la pendiente entre la ubicación actual y el siguiente panorama, para así, poder actualizar en el servidor el nivel de resistencia que debe tener la bicicleta. Mas adelante se ecplicará esta funcionalidad con mas detalle. Por último, el router es el encargado de encaminar todos los dispositivos conectados a la red del corredor a Internet.

3.1.1.3. Internet:

Ilustración 6: Internet

En nuestro caso, esta red solo cuenta con un dispositivo físico, el router, ya que los servidores de la base de datos son un BaaS[1] propiedad de Google. Se trabaja con la base de datos en tiempo real que ofrece firebase, porque nos proporciona integración en tiempo real entre todos nuestros dispositivos, de este modo, desde la aplicación se obtienen los datos inmediatamente después de que el DCB los actualice y viceversa permitiendo así mostrar información del entrenamiento en tiempo real en la pantalla de la app y subir o bajar la resistencia de la bici según la pendiente calculada por la app.

[1] "*Backend as a service (BaaS), también conocido como "mobile backend as a service", es un modelo para proporcionar a los desarrolladores web y de aplicaciones móviles una forma de vincular estas aplicaciones al almacenamiento en nube (cloud storage), servicios analíticos y/o otras características tales como la gestión de usuarios, la posibilidad de enviar notificaciones push y la integración con servicios de redes sociales.*
"

Fuente: Backend as a service - https://es.wikipedia.org

3.1.2. DCB:

El DCB es, como se ha dicho anteriormente, el dispositivo encargado de dotar de conexión de red inalámbrica a la Domyos VE430, además de calcular la velocidad y modificar la resistencia de pedaleo. Este módulo está compuesto por el NodeMCU, la placa "NodeMCU motor shield", el sensor de efecto hall y el motor de la bicicleta. A continuación, se explica que es y cómo se utilizan estos componentes.

3.1.2.1. NodeMCU:

Arduino es un microcontrolador de hardware abierto, desarrollado por Massimo Banzi y Hernando Barragan. Su principal objetivo era crear una placa de bajo coste que pudiera competir con otras alternativas más caras del mercado, para que cualquier estudiante o aficionado a la electrónica pudiera trabajar sin tener que desembolsar una gran cantidad de dinero. Todo esto y la gran aceptación por parte de la comunidad ha propiciado que existan infinidad de placas Arduino a diferentes precios, según el fabricante en cuestión. Por su popularidad y la extensa documentación, en un principio se pensó trabajar con Arduino, pero debido al alto precio de la placa "Arduino wifi shield" (la que permite conectar Arduino a internet de forma inalámbrica) y de su gran tamaño en comparación con otras alternativas como NodeMCU, se llevó a cabo una investigación para comprobar si efectivamente era más conveniente crear el DCB con otra placa. Las alternativas que se estuvieron estudiando fueron: NodeMCU, Adafruit Feather HUZZAH con ESP8266 WiFi, Espruino WiFi, Pyboard, Raspberry Pi, BeagleBone, Gizmo2, MimowBoard.

En una primera búsqueda por Internet se llegó a la conclusión de que la principal alternativa a Arduino era Raspberry Pi, cosa que en nuestra opinión no es cierta, ya que son placas destinadas a distintos usos. Tanto Raspberry Pi, BeagleBone, Gizma2 como MimowBoard son miniordenadores y no microcontroladores. En el caso de los miniordenadores, son capaces de ejecutar un sistema operativo completo como pueda ser Debian o Android, con toda la configuración y tiempo que esto conlleva. Para este proyecto, esto era demasiada potencia desaprovechada ya que en este caso no necesitábamos un miniordenador con un SO sino un microcontrolador en el que poder

controlar entradas y salidas y tener una conexión de red inalámbrica. Por todo esto, se descartaron los miniordenadores y se siguieron buscando alternativas que se acercaran más a las necesidades del proyecto. En este nuevo barrido, se encontraron los siguientes microcontroladores: NodeMCU, Adafruit Feather HUZZAH con ESP8266 WiFi, Espruino WiFi, Pyboard. Como se ha comentado en el encabezado de esta sección se necesita controlar un motor de corriente continua y en el momento de la búsqueda de información, solamente se encontró una placa que permitía gestionar este aspecto, la NodeMCU Motor Shield, lo cual fue de gran influencia en la elección de NodeMCU. De todas formas, a continuación, se muestra una pequeña tabla comparativa de los microcontroladores mencionados arriba.

	NodeMCU	**Adafruit Feather HUZZAH**	**Espruino WiFi**	**Pyboard**
Microcontrolador	Xtensa® Single-Core 32-bit L106	Xtensa® Single-Core 32-bit L106	STM32F411CEU6 32-bit 100MHz ARM Cortex M4 CPU	STM32F405RG
Voltaje de operación	3.3V	3.3V	3.3V	3.3V
Voltaje de entrada (Recomendado)	3.3V – 20V	3.3V – 20V	3.3V – 5V	3.3V 16V
Pines de I/O digital	17 (8 PWM)	9 (I2C y SPI)	20 PWM, 1 Serie, 3 SPI, 3 I2C	24 (2 DAC)
Pines entrada analóg.	1	1 – 1V máx.	8	3
Memoria Flash	4 MB	4 MB	512kb	1024KB
SRAM	32 KB inst. 96 KB datos	32 KB inst. 96 KB datos	128kb	192KB
Frec. de reloj	80 MHz	80 MHz	100MHz	168 MHz
Tamaño (mm)		23 x 51 x 7.2	30 x 23	
Precio / tienda	4.39€	16.57€	29.02€	30.39€

Tabla 1: Comparativa microcontroladores

Como se puede observar en la Tabla 1, NodeMCU no es el microcontrolador más potente, pero para este proyecto es más que suficiente. Tal y como se ha comentado anteriormente, en el momento de la búsqueda de información NodeMCU era el único dispositivo que contaba con un controlador para el motor de corriente continua lo cual facilita mucho el trabajo, de no ser así, se tendría que crear un controlador propio y no es el propósito de este proyecto. También cabe destacar que la idea inicial era crear un módulo barato en el que poder controlar la bici. En la comparativa, se puede observar que el precio es significativamente menor al resto de alternativas y por tanto un valor añadido a la hora de decantarnos por NodeMCU.

A continuación, se muestra una tabla comparativa entre Arduino Uno y NodeMCU.

	Arduino Uno	**NodeMCU**
Microcontrolador	Atmega328	Xtensa® Single-Core 32-bit L106
Voltaje de operación	5V	3.3V
Voltaje de entrada (Recomendado)	5 – 12V	3.3V – 20V
Voltaje de entrada (Límite)	6 – 20V	3.3V – 20V
Pines de entrada/salida digital	14 (6 pueden usarse como salida de PWM)	17 (8 pueden usarse como salida de PWM)
Pines de entrada analógica	6	1
Memoria Flash	32 KB (0.5 KB ocupados por el bootloader)	4 MB
SRAM	2 KB	32 KB Inst. - 96 KB datos
EEPROM	1 KB	
Frecuencia de reloj	16 MHz	80 MHz
CWiFi	Arduino wifi shield	Sí, HT20

Tabla 2: Comparativa entre Arduino uno y NodeMCU

Tal y como se muestra en la Tabla 2, se ha ganado en velocidad ya que la de NodeMCU es considerablemente más alta, en el tamaño de memoria de almacenamiento de software, en conectividad ya que integra wifi de forma nativa y también se ha reducido el tamaño del módulo.

3.1.2.2. NodeMCU Motor Shield:

NodeMCU Motor Shield está diseñado y desarrollado por Shenzhen Doctors of Intelligence & Technology (SZDOIT). Este módulo es compatible con ESP12E Dev Kit y NodeMCU. Gracias a su diseño de conexión superpuesta el motor puede conectarse directamente al NodeMCU.

Esta placa está conducida por el chip L293DD de la famosa compañía de Stmicroelectronics, el cual permite entregar la potencia necesaria a los motores. Gracias a esto es posible controlar dos motores de corriente continua de 2 canales o un motor paso a paso de un canal. La corriente impulsada puede llegar a 1.2A.

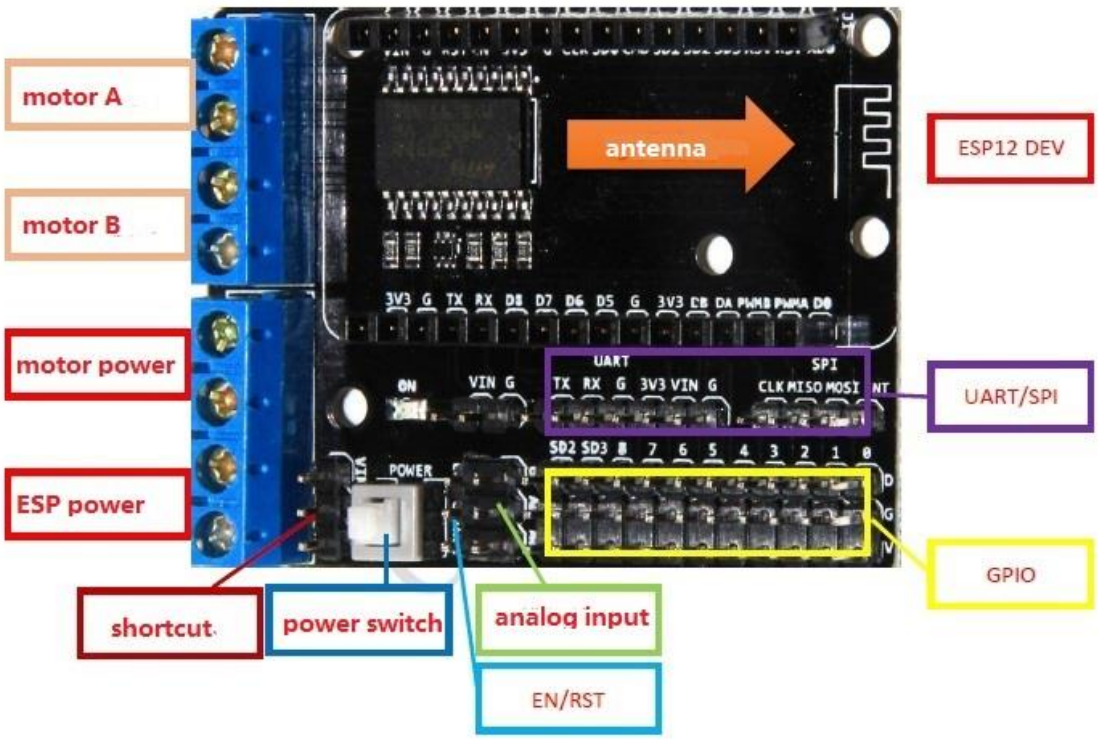


Ilustración 7: NodeMCU Motor Shield

En esta placa de protección de motor, los pines de entrada/salida del ESP-12E Dev Kit se utilizan como pines de control. El chip lógico configurado en el interior puede terminar

con el IC. Por lo tanto, la placa de protección tiene cuatro puertos: D1, D2, D3 y D4, que se utilizan como PWMA (motor A), PWMB (motor B), DA (dirección del motor A) y DB (dirección del motor B), Respectivamente.

Además, cuenta con pines tales como: VIN, 3.3V, DIO, AIO, SDIO, UART, SPI, RST, y EN; lo cual permite conectar el resto de sensores. En este caso el sensor de efecto hall para controlar la velocidad del corredor.

3.1.2.3. Sensor de efecto hall:

"El efecto Hall, descubierto por Edwin C. Hall en 1879, consiste en la producción de una caída de voltaje a través de un conductor o semiconductor con corriente, bajo la influencia de un campo magnético externo. Para esto es necesario que la dirección del campo magnético sea perpendicular a la dirección de flujo de la corriente.

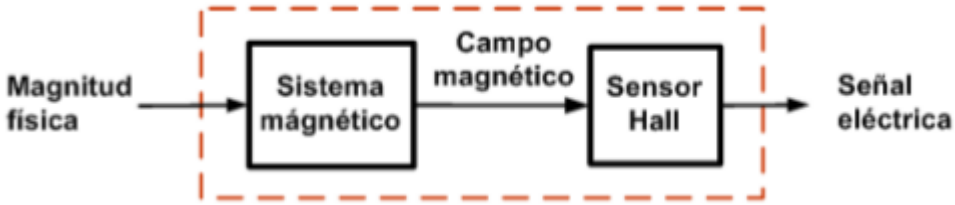


Ilustración 8: Esquema efecto Hall

El campo magnético transversal ejerce una fuerza desviadora (Fuerza de Lorentz) sobre el conductor o semiconductor. Esta fuerza causa la desviación de los portadores de carga que se mueven a través del material. Como resultado, aparece una diferencia de potencial Vxy (denominada voltaje de Hall) entre los extremos del conductor. Este voltaje es proporcional a la intensidad del campo magnético aplicado y su polaridad depende del signo de los portadores de carga.

El efecto Hall se presenta en conductores y en semiconductores. Las diferencias de potencial producidas en tiras metálicas son muy pequeñas, siendo a menudo enmascaradas por el ruido. Por esto, los dispositivos comerciales usan materiales semiconductores especiales, donde el efecto Hall es más notable. En estos casos, el elemento básico es generalmente una tira de arseniuro de galio (GaAs) o de indio (InAs) la cual, cuando se polariza mediante una corriente constante y se

sumerge en un campo magnético transversal a su superficie, genera un voltaje proporcional a la intensidad del campo. Este voltaje es reforzado por un amplificador operacional incorporado en el dispositivo y se procesa para proporcionar una señal de salida útil."

MARTÍN MURDOCCA, R. (N.D.). SENSORES DE EFECTO HALL. [EBOOK] SAN LUIS - ARGENTINA: UNIVERSIDAD NACIONAL DE SAN LUIS. FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES, PP.2-3. AVAILABLE AT: HTTP://WWW0.UNSL.EDU.AR/~INTERFASES/LABS/LAB09.PDF [ACCESO 4 AGOSTO 2017].

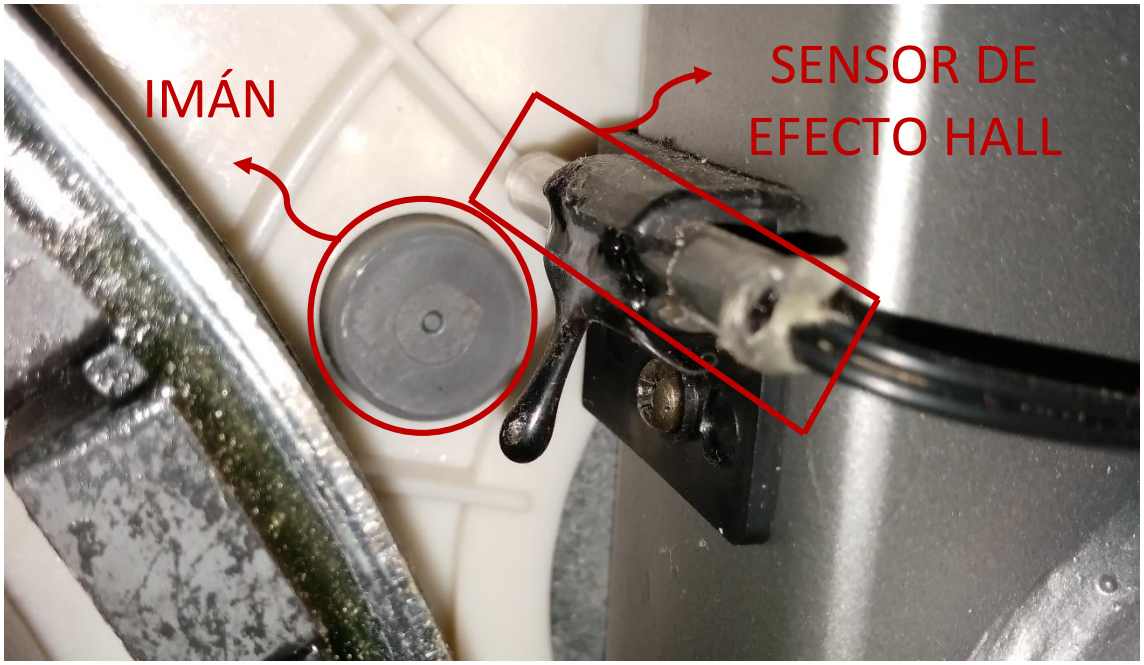


Ilustración 9: Sensor de efecto Hall de la VE430

En la Domyos VE430 el sensor de efecto hall actúa como interruptor, es decir, cuando se aplica la fuerza del campo magnético, deja pasar corriente permitiéndo así saber cuándo pasa el imán por el sensor. En el apartado de software se explicará con más profundidad como se calcula la velocidad gracias este sensor.

3.1.2.4. Motor de corriente continua:

Las bicicletas elípticas funcionan con dos patines que simulan la acción de correr. Al correr los patines mencionados hacen girar un disco metálico que la bici lleva en su interior. Para poder mejorar la experiencia del corredor, el disco metálico de su interior está rodeado por un imán en forma de arco que actúa como resistencia, cuando más cerca está el imán al disco, mayor es la resistencia de pedaleo. El motor es el encargado de acercar y alejar el imán al disco metálico de la bicicleta, esto se consigue invirtiendo la polaridad de la corriente, es decir si la corriente es positiva, el motor gira en sentido horario y si es negativa en el contrario.

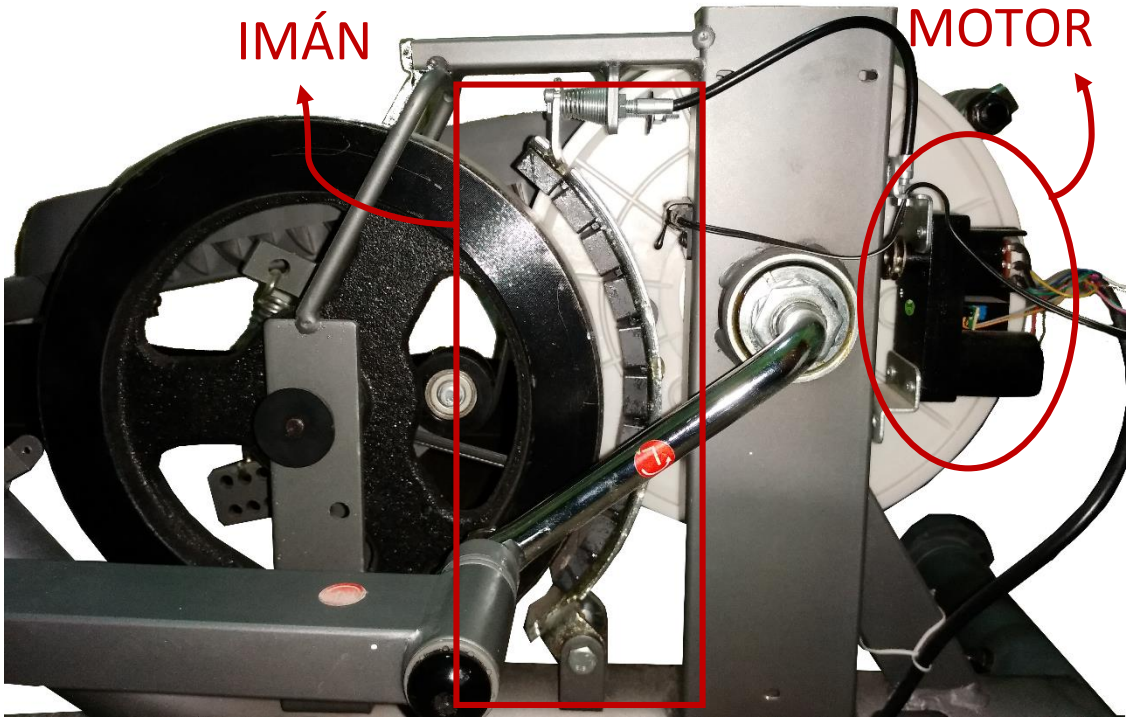


Ilustración 10: Motor e imán de la VE430

3.1.3. IMU:

En este apartado se describe el proceso de creación del dispositivo encargado de detectar los giros de muñeca del corredor para así poder realizar los cambios de dirección o giros de cámara.

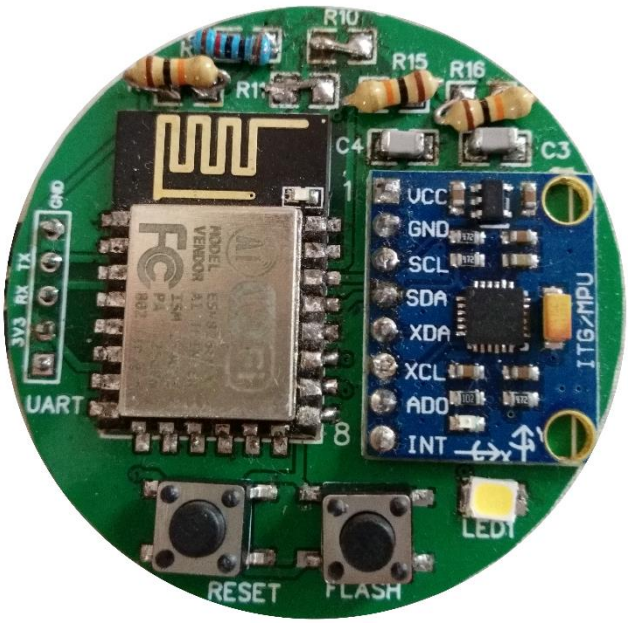


Ilustración 11: Dispositivo IMU

En primer lugar, se procede a detallar los componentes utilizados para su desarrollo. Los dos elementos principales son el módulo ESP8266, que en este caso actúa como microcontrolador y el módulo MPU6050, acelerómetro y giroscopio utilizado para detectar el movimiento de muñeca.

3.1.3.1. ESP8266:

El ESP8266 es un chip Wi-Fi de bajo costo con pila TCP / IP completa y capacidad de MCU (unidad de microcontrolador) producida por el fabricante chino Espressif Systems.

El primer chip (ESP8266 01 o ESP-01) llegó a la atención de los fabricantes occidentales en agosto de 2014 con el módulo ESP-01, fabricado por AI-Thinker. Este pequeño módulo permite que los microcontroladores se conecten a una red Wi-Fi y realicen conexiones TCP / IP simples utilizando comandos de tipo Hayes. El precio muy bajo y el hecho de que había muy pocos componentes externos en el módulo, atrajo a muchos

curiosos para explorar el módulo, el chip y el software en él, así como para traducir La documentación china.

Como se habrán dado cuenta, el NodeMCU mencionado anteriormente, lleva este mismo chip como microcontrolador principal, por ello se decidió investigar sobre cómo integrarlo en nuestra propia placa y utilizarlo como microcontrolador en el dispositivo IMU. Además de por sus prestaciones, se ha elegido este chip porque en la actualidad se puede cargar el firmware de Arduino y trabajar como si de un Arduino se tratase ya que el módulo MPU6050 no es compatible con todos los dispositivos ni dispone de librerías para usarlo en otros entornos. Más adelante en la sección de software se explicará con más detalle todo lo referente al firmware y a como se ha logrado cargarlo en el módulo.

En concreto se está utilizando la versión **ESP8266 12E**. A continuación, se muestra un esquema de los pines de dicho módulo.

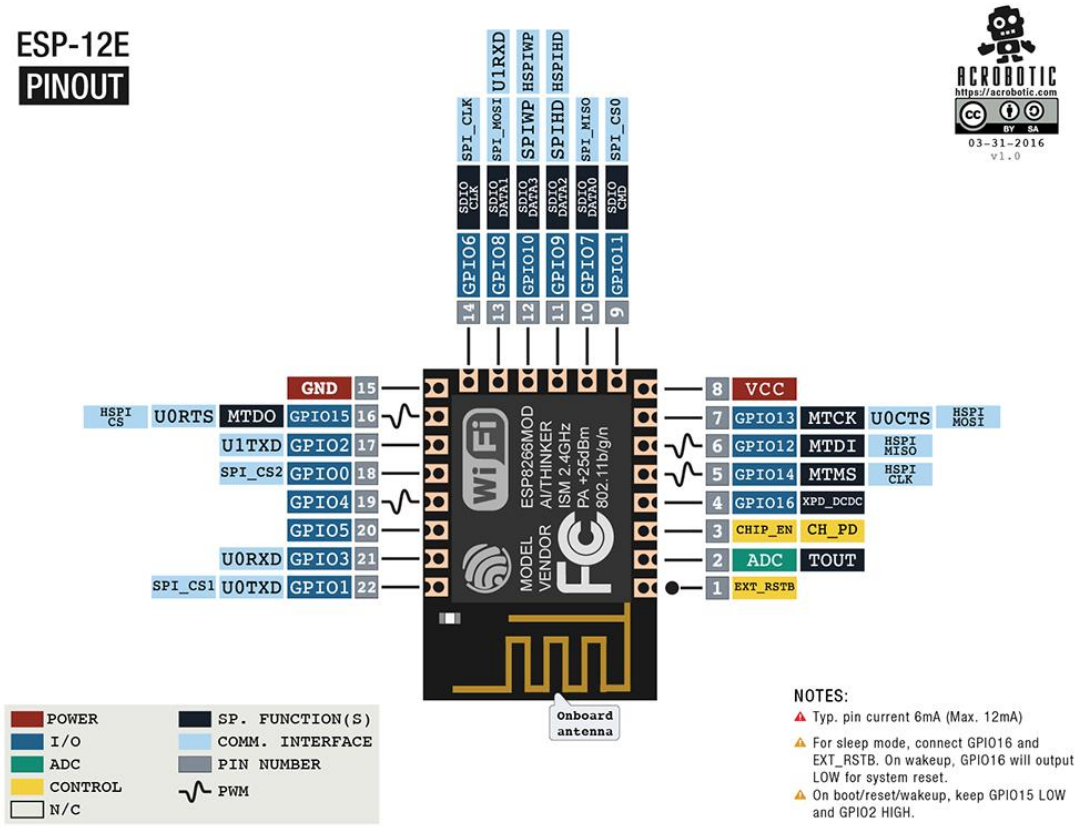


Ilustración 12: Esquema pinout del módulo ESP8266

Cabe destacar la complejidad encontrada para hacer que el módulo funcionase de forma estable, es decir, sin reinicios espontáneos. La siguiente imagen muestra un esquema con la configuración básica en las conexiones.

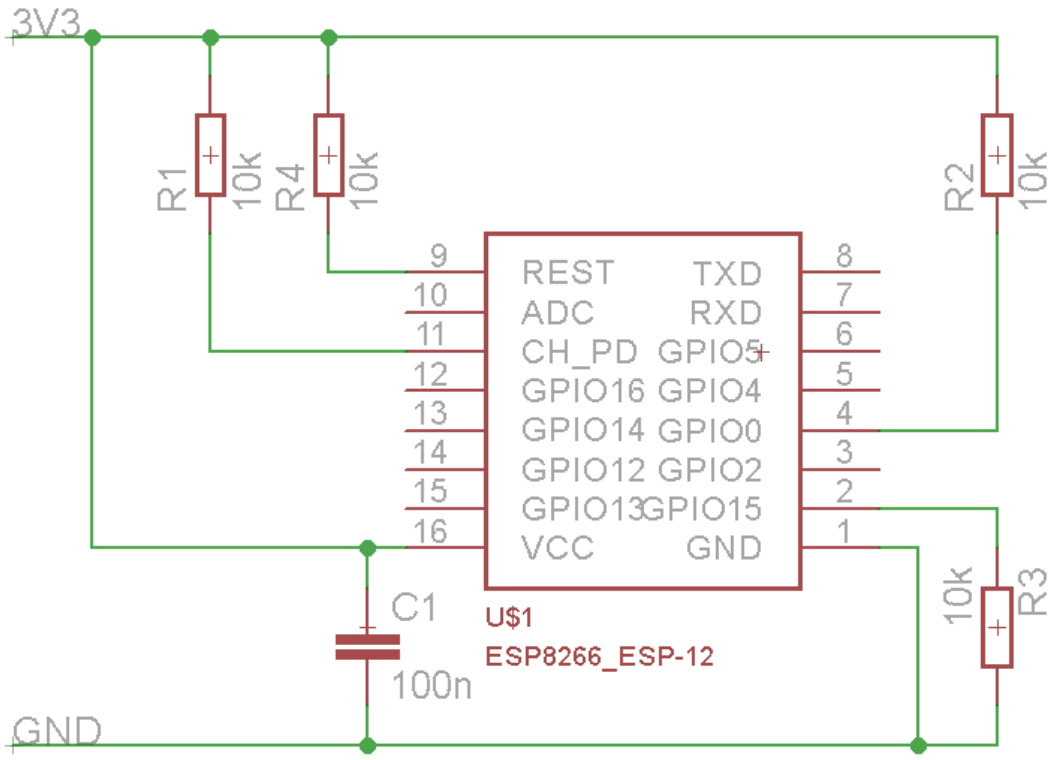


Ilustración 13: Conexiones para un correcto funcionamiento del ESP8266

El módulo ESP comprueba en cada inicio los pines 0, 2 y 15. Basándose en el estado de estos, puede arrancar en uno de los siguientes modos.

Modo	GPIO 15	GPIO 0	GPIO 2
Uart Bootloader	0V	0V	3.3V
Boot sketch (SPI flash)	0V	3.3V	3.3V
SDIO mode (no puede usarse en Arduino)	3.3V	X	X

Tabla 3: Modos ESP8266

Para este proyecto se ha utilizado el modo "*Uart Bootloader*", para iniciar el módulo en modo normal, es decir, que ejecute el software instalado en él y el modo "*Boot sketch*" para cargar nuestro software en el módulo.

3.1.3.2. MPU6050:

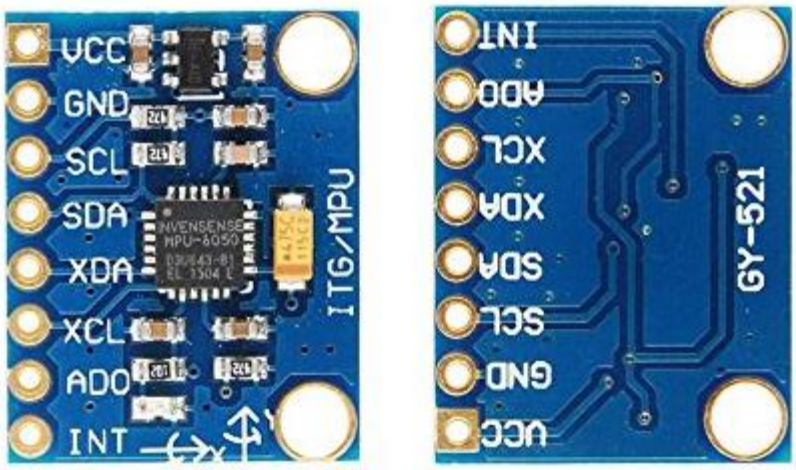


Ilustración 14: MPU-6050

El MPU-6050 ™ es el primer dispositivo MotionTracking diseñado para los requisitos de bajo consumo, bajo costo y alto rendimiento de los teléfonos inteligentes, tabletas y sensores portátiles.

La MPU-6050 incorpora el MotionFusion ™ de InvenSense y el firmware de calibración en tiempo de ejecución que permite a los fabricantes eliminar la costosa y compleja selección, calificación e integración a nivel de sistema de dispositivos discretos en productos habilitados para movimiento.

Los dispositivos MPU-6050 combinan un giroscopio de 3 ejes y un acelerómetro de 3 ejes en la misma matriz de silicio, junto con un Digital Motion Processor ™ integrado (DMP ™), que procesa complejos algoritmos MotionFusion de 6 ejes. El dispositivo puede acceder a magnetómetros externos o a otros sensores a través de un bus maestro auxiliar I^2C, permitiendo a los dispositivos reunir un conjunto completo de datos de sensores sin intervención del procesador del sistema. Los dispositivos se ofrecen en un paquete QFN de 4 mm x 4 mm x 0,9 mm.

Ilustración 15: Esquema MPU-6050

La plataforma InvenSense MotionApps ™, que viene con la MPU-6050, abstrae las complejidades basadas en el movimiento, descarga la gestión de sensores desde el sistema operativo y proporciona un conjunto estructurado de API para el desarrollo de aplicaciones.

Para el seguimiento de precisión de los movimientos rápidos y lentos, las piezas cuentan con un girocompás programable por el usuario escala completa de ± 250, ± 500, ± 1000, y ± 2000 ° / s (dps), y un acelerómetro programable por el usuario, Escala de ± 2g, ± 4g, ± 8g, y ± 16g. Las características adicionales incluyen un sensor de temperatura incorporado y un oscilador en el chip con ± 1% de variación en el rango de temperatura de funcionamiento.

Este dispositivo, cuenta con la posibilidad de trabajar con interrupciones, lo cual es mucho más eficiente a la hora de detectar cambios de posición en la muñeca del corredor. Todo esto se verá con más detalle en el apartado de software, también se explicará cómo se ha logrado integrar el ESP8266 y el MPU6050.

3.1.3.3. Placa base y hardware necesario:

Para poder crear un dispositivo compacto que el corredor pudiera llevar en su muñeca como si de un reloj se tratase, se optó por diseñar un circuito impreso. Para diseñarlo se utilizaron las herramientas proporcionadas por https://easyeda.com/es y posteriormente se pidió a easyeda, fabricante de componentes electrónicos en china, que fabricara la placa que se había diseñado.

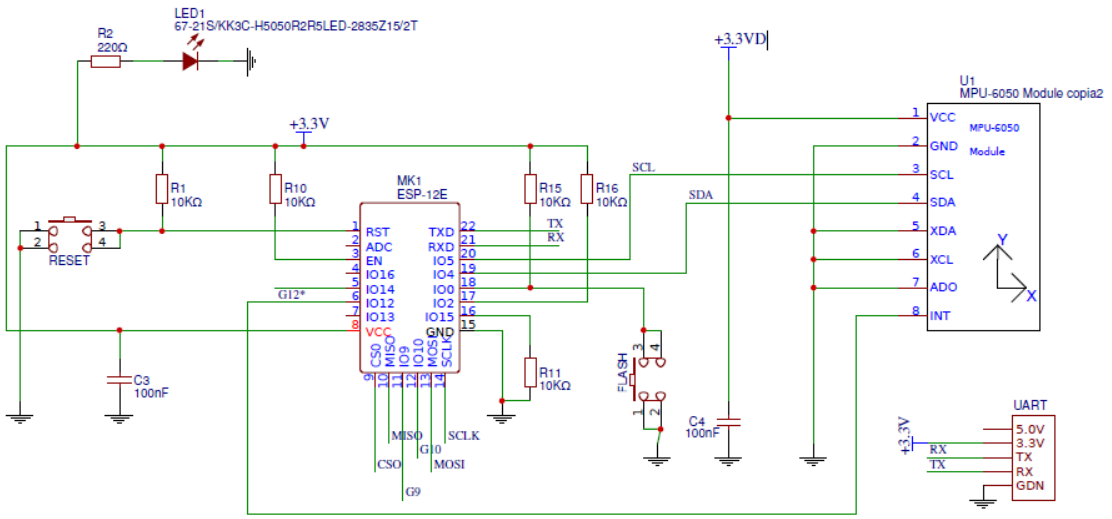
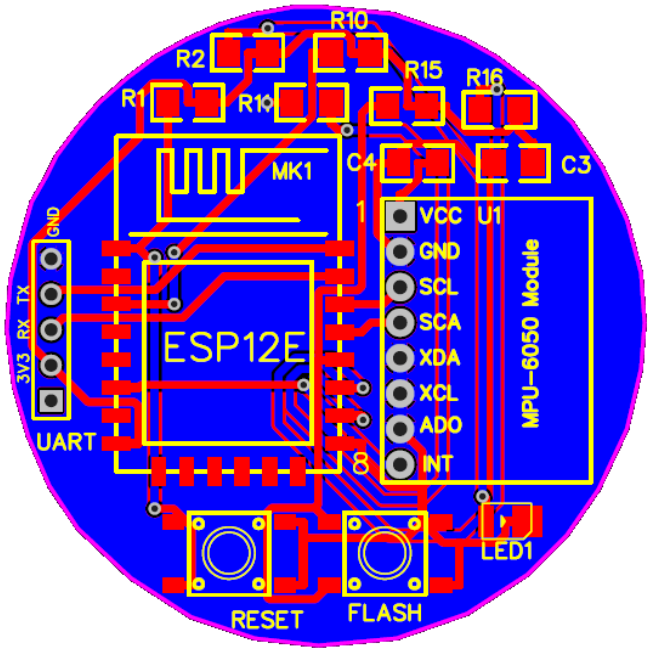


Ilustración 16:Circuito del IMU

A continuación, se explica el funcionamiento del circuito de la Ilustración 16. El circuito se alimenta mediante las patillas 3.3V y GND del componente "UART" el cual es el encargado de comunicar el circuito con el ordenador mediante un conversor UART-USB. Para ello solamente tendremos que establecer una conexión cruzada de los pines TX y RX, es decir, el pin TX del componente UART al pin RX del conversor y el pin RX del UART al TX del conversor. Es posible alimentar el circuito sin tener que conectar el conversor UART-USB, para ello bastará con conectar el pin 3.3V al borne positivo de una batería o alimentador y el pin GND al borne negativo. El dispositivo cuenta con dos pulsadores, el primero, identificado como "RESET", es como su nombre indica, el encargado de reiniciar el dispositivo ESP8266. El segundo ofrece la posibilidad de iniciar el chip ESP12e en modo flash o en modo normal, para ello, será suficiente con mantener o no presionado el pulsador identificado como "*FLASH*".

Llegados a este punto, se muestra como se comunican entre sí los dos principales componentes del dispositivo, el módulo ESP12e y el MPU6050. Como se ha comentado anteriormente el MPU6050 cuenta con la posibilidad de trabajar con interrupciones y el ESP8266, si se configura una patilla para tal efecto también (No todos los pines se pueden utilizar para recibir interrupciones). Aprovechando esta funcionalidad se ha conectado el pin GPIO12 del microcontrolador al pin INT del MPU, de este modo cada vez que el sensor detecte movimiento enviará una interrupción al ESP12e. Cuando éste

reciba dicha interrupción pedirá al sensor que le envía la información que tiene guardada, para esto, se utilizan los pines SCL y SDA del MPU y los pines GPIO5 y GPIO4 del ESP respectivamente.



A continuación, vemos el PCB que se encargó al fabricante Easyeda.

Ilustración 17: Diseño del PCB

Desglose de precios:

Desglose de precios del dispositivo IMU:

Componente	Unidades	Precio	Importe	Proveedor	
Resistencia 10KΩ	4	0.0017€	0.0068€	lcsc	
Condensador 100nF	2	0.1049€	0.2098€	lcsc	
Led	1	0.0281€	0.0281€	lcsc	
Pulsador	2	0.0178€	0.0356€	lcsc	
ESP8266	1	3.29€	3.29€	solectroShop	
MPU-6050	1	2.40€	2.40€	solectroShop	
		TOTAL:	**5.9703€**		

Tabla 4:listado de precios y componentes del IMU

3.1.4. Router:

El router es el dispositivo encargado de interconectar todos los dispositivos de la red doméstica del corredor y comunicarlos por internet con los servidores de base de datos de firebase. Basta con tener un router que ofrezca conexión wifi, el que instala el proveedor ISP puede ser perfectamente válido.

3.1.5. Televisor Android

En este caso se dispone de un televisor Philips 50PUH6400 con sistema operativo Android, pero sería perfectamente válido cualquier televisor con SO Android o un dispositivo Android tv box.

3.2. Software

Antes de empezar a explicar cómo se ha desarrollado el software de cada una de las partes del proyecto, es conveniente explicar el entorno de desarrollo que se ha utilizado para programar el DCB y el IMU, ya que no se ha desarrollado con la plataforma proporcionada por el ecosistema de Arduino, sino que se ha utilizado PlatformIO.

Ilustración 18: Logo PlatformIO

PlatformIO como en su página web explican "*es un ecosistema de código abierto para el desarrollo del Internet de las cosas* con sistema de compilación multiplataforma, administrador de bibliotecas y soporte completo para el desarrollo de Espressif (ESP8266)". Esta plataforma de desarrollo aporta una serie de ventajas respecto al entorno de desarrollo que ofrece Arduino, a continuación, se explican las más relevantes:

- Ofrece "PIO Unified Debugger", depurador con soporte para múltiples arquitecturas y plataformas de desarrollo. Permite depurar más de 100 tarjetas incrustadas con Zero-Configuration. El depurador no soporta ESP8266, por tanto, no se ha podido utilizar para este proyecto.
- Sistema de compilación multiplataforma sin dependencias externas para el software del sistema operativo, permite compilar software para: más de 350 placas incrustadas, más de 15 plataformas de desarrollo y más de 10 marcos. Entre ellas está la plataforma Arduino, que se necesita para este proyecto.
- Añade funcionalidad para autocompletar código C o C++ en el IDE y también, "Smart Code Linter" que ofrece la posibilidad de detectar errores en el código antes de compilar.

- Incorpora un potente terminal con PlatformIO Core y un potente monitor de puerto serie para mejorar la depuración de código.
- Está disponible en multitud de editores de texto avanzados, como Atom, Sublime Text, VSCode, etc. Por tanto, se puede programar en el IDE que más se conozca o más cómodo se esté.

Ha llegado el momento de explicar cómo se ha configurado el editor para poder trabajar con esta plataforma de desarrollo. En este caso el editor de texto elegido ha sido Atom, por la facilidad que ofrece para añadir funcionalidad nueva y personalizaciones. Aunque en la página de descarga de complementos de Atom existen diferentes alternativas, es importante instalar los desarrollados por PlatformIO, ya que son los únicos que disponen de soporte oficial.

En primer lugar, se debe instalar PlarformIO IDE, el cual dota a Atom de una nueva barra de herramientas. Esta barra de herramientas ofrece la posibilidad de compilar, depurar y subir el software a la placa de forma rápida, también dispone de accesos directos para abrir el monitor del puerto serie, abrir el terminal, etc.

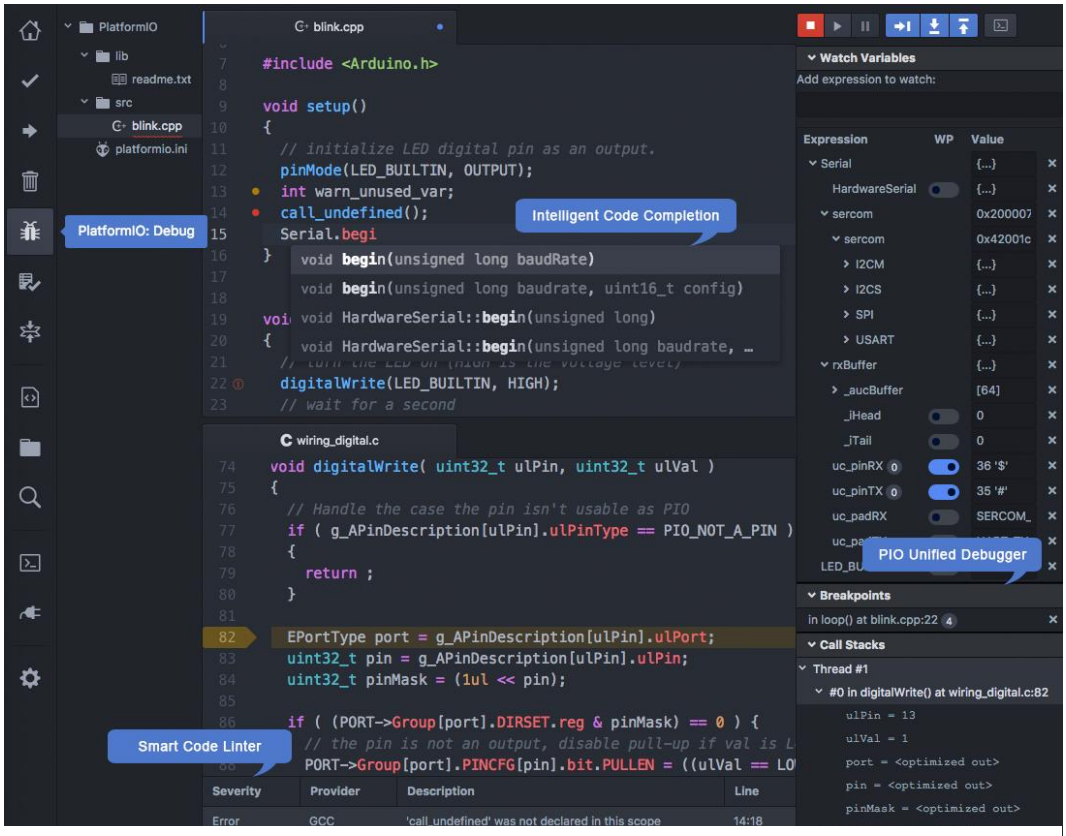


Ilustración 19:PlatformIO IDE para Atom

Resulta de interés instalar también PlatformIO IDE Terminal, el cual hace posible utilizar el terminal de PlatformIO Core directamente en Atom.

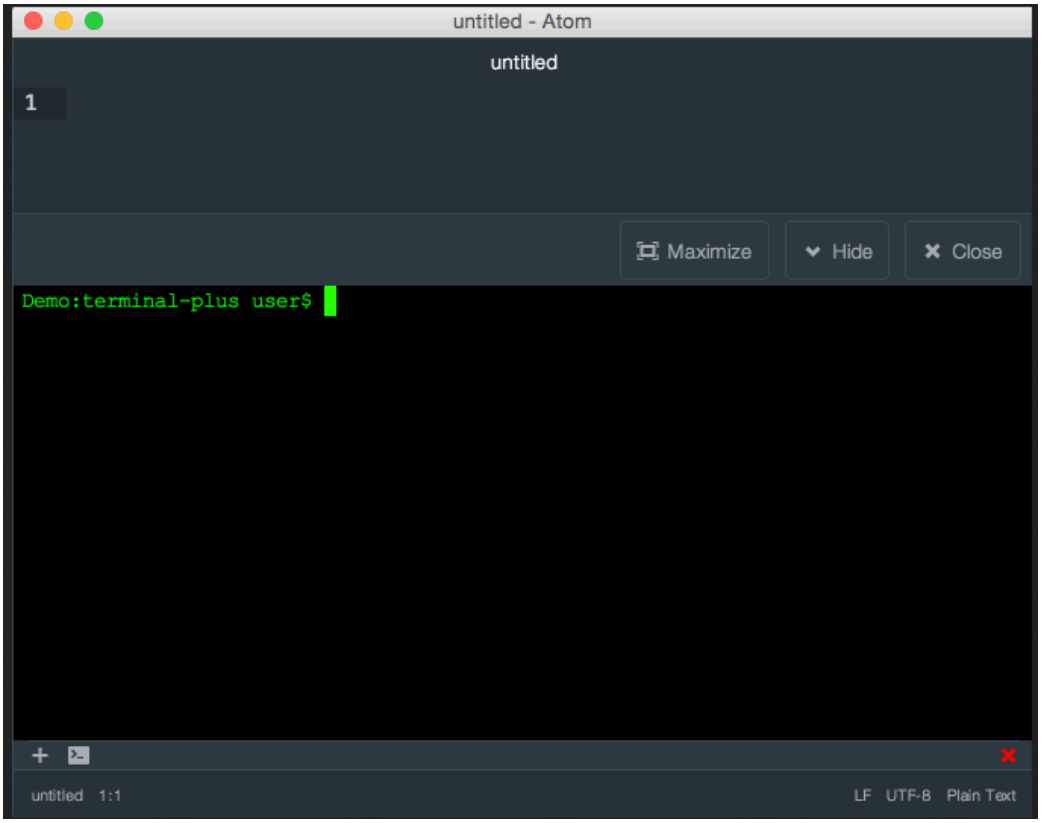


Ilustración 20: Terminal de PlatformIO para Atom

PlatformIO integra Arduino core para ESP8266 gracias al cual se ha podido desarrollar en NodeMCU como si de un Arduino se tratase. Arduino core para ESP8266 es un firmware de código abierto, el cual viene con librerías para comunicar por WiFi usando TCP y UDP, configurar HTTP, mDNS, SSDP y servidores DNS, hacer actualizaciones OTA, utilizar un sistema de ficheros en la memoria flash, trabajar con tarjetas SD, servomotores y periféricos SPI y I2C.

3.2.1. DCB:

En este apartado se detallan los aspectos más destacados en el proceso de desarrollo de software para el DCB y los problemas encontrados.

El primer paso para poder empezar a desarrollar software para NodeMCU como si de un Arduino se tratase, es indicar al PlatformIO IDE con qué plataforma, qué placa y qué framework se va a trabajar. Esto permite al "Smart Code Linter" identificar en tiempo real si el código introducido es correcto o no. La siguiente imagen muestra el fichero de configuración utilizado para desarrollar el DCB.

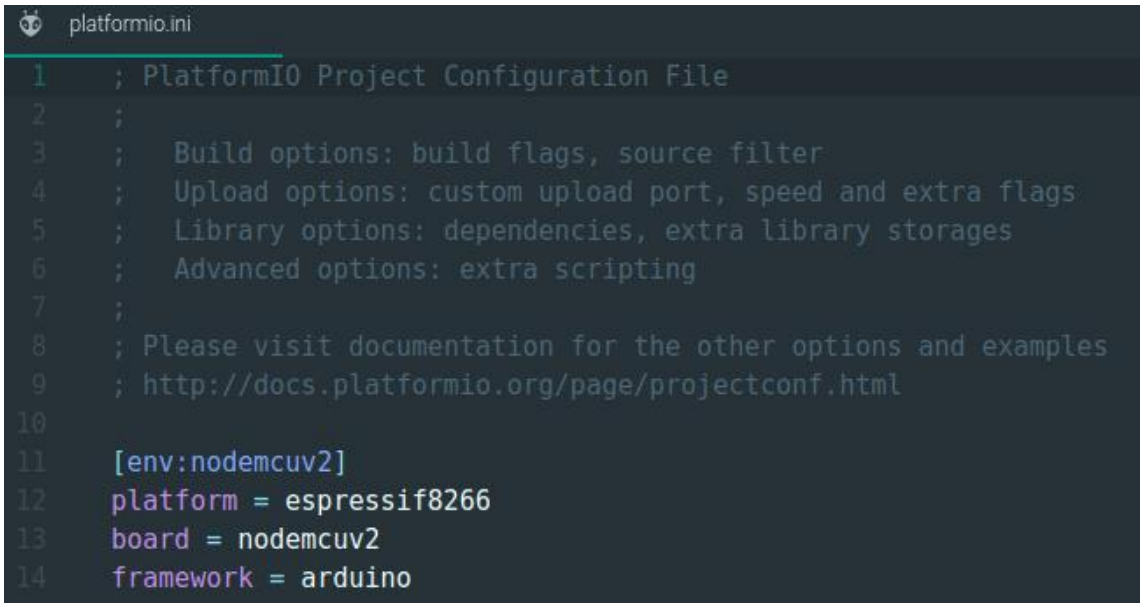


Ilustración 21: Fichero de configuración platform.ini

Como se puede observar en la Ilustración 21, se utiliza la variable de entorno nodemcuv2 (linea 11), en la cual se guarda la configuración sobre la plataforma (linea 12), la placa (linea 13), y el framework (linea 14). En este caso se va a trabajar con la plataforma expressif8266 ya que NodeMCU utiliza el chip ESP8266 y en el apartado de framwork se indica que se va a desarrollar con el framework de Arduino proporcionado por Arduino core para ESP8266.

Una vez configurado el entorno de desarrollo ya se puede empezar a programar toda la funcionalidad deseada para el módulo DCB, a continuación, se detallan algunas de las partes más importantes de dicho dispositivo.

3.2.1.1. Conexión de inalámbrica para la bicicleta:

Para facilitar al usuario la conexión de la bicicleta a la red wifi de su casa, se ha desarrollado la funcionalidad para configurar la red inalámbrica del DCB sin necesidad de conectarlo a ningún ordenador, para ello se ha utilizado la librería WifiManager, la cual fue creada por "tzapu", así es como se hace llamar el desarrollador en GitHub.

Esta librería facilita la creación de un configurador para la red inalámbrica con muy pocas líneas de código.

En primer lugar, se deben de añadir al proyecto las librerías de las que esta depende:

```
5   #include <ESP8266WiFi.h>
6
7   //needed for library
8   #include <DNSServer.h>
9   #include <ESP8266WebServer.h>
10  #include "WiFiManager.h"
```

Ilustración 22: Librerías necesarias para WiFiManager

Librería	**Función**
ESP8266WiFi	Librería wifi del core de ESP8266
DNSServer	Servidor DNS local utilizado para redirigir todas las peticiones al portal de configuración
ESP8266WebServer	Servidor web local utilizado para servir el portal de configuración.

Tabla 5: Uso de las librerías utilizadas por WiFiManager

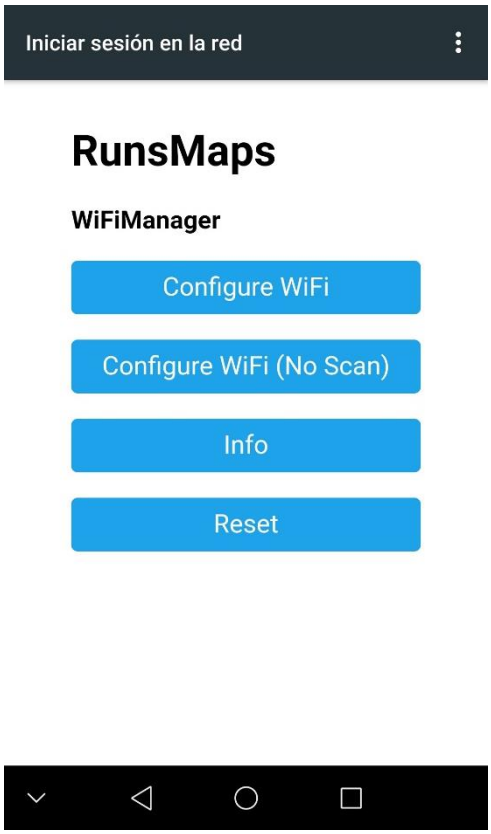
El siguiente paso es inicializar la librería, para ello se crea un objeto WiFiManager. Una vez creado el objeto, se debe asignar el callback, método al que se llamará si falla la creación del objeto WiFiManager. Como se puede observar en la línea 204 de la siguiente imagen se intenta realizar la conexión, pero si esta no es satisfactoria, WiFiManager se encarga de ejecutar el servidor web con el ssid y contraseña pasados como parámetros a la función *autoConnect*.

```
//WiFiManager
// Inicialización local. Una vez terminada la actividad,
// no és necesarip mantenerlo actualizado.
WiFiManager wifiManager;
//Reinicia la configuracion del wifi (Para desarrollo)
//wifiManager.resetSettings();

// Establece el callback por si la configuración del wifi falla y establece
// del punto de acceso
wifiManager.setAPCallback(configModeCallback);

// Busca ssid y contraseña e intenta conectarse si no lo conecta inicia un
// punto de acceso con el nombre especificado aquí "AutoConnectAP" y
// entra en un bucle de bloqueo esperando la configuración
if(!wifiManager.autoConnect(ssid, password)) {
    Serial.println("No se ha podido conectar y se ha superado " +
                   "el tiempo de espera");
  // Reinicia y vuelve a intetnarlo
  ESP.reset();
  delay(1000);
}
//Se se llega aquí, se ha conectado correctamente
// Tenemos que activar ambos modos de funcionamiento del wifi para poder
// trabajar con el IMU y con la red doméstica
WiFi.mode(WIFI_AP_STA);
Serial.println("Conectado a la red!");
```

Ilustración 23: Código para la inicialización de la librería

En el caso de que no pueda establecerse la conexión a ninguna de las redes guardadas previamente o es la primera vez que se inicia el DCB, el usuario puede configurar la red wifi de su casa de forma muy sencilla. Una vez conectado el DCB a la bicicleta bastará con conectarse desde un Smartphone o un ordenador portátil a la red creada por el DCB. En este caso el nombre de la red por defecto es "*RunsMaps"* y la contraseña "*RunsMaps123"* sin las comillas.

A continuación, se muestra paso a paso el proceso que debe seguir el corredor para configurar la red desde un Smartphone Android.



Esta es la primera pantalla que aparece al conectarse a la red wifi RunsMaps. A ella se llega de forma automática gracias al servidor DNS que redirige todas las peticiones al servidor web creado para configurar una nueva conexión inalámbrica.

Como se puede apreciar en la Ilustración 24, presenta una interfaz sencilla, solamente cuenta con cuatro botones. Los dos primeros sirven para configurar una nueva conexión, el tercero para ver la información de red del DCB y por último un botón para reiniciar toda la configuración, es decir, borrar todas las redes guardadas previamente.

Ilustración 24: Pantalla principal de WiFiManager

Para configurar una red partiendo de la lista con todas las redes disponibles, se debe seleccionar la primera opción. En la Ilustración 25 se puede ver la pantalla de configuración de red, con la lista de las redes disponibles. Basta con seleccionar el nombre de la red wifi que se desee configurar e introducir la contraseña para dicha red. Si por el contrario se desea configurar una red oculta, se debe hacer desde la opción “Configurar wifi (Sin escanear)”.

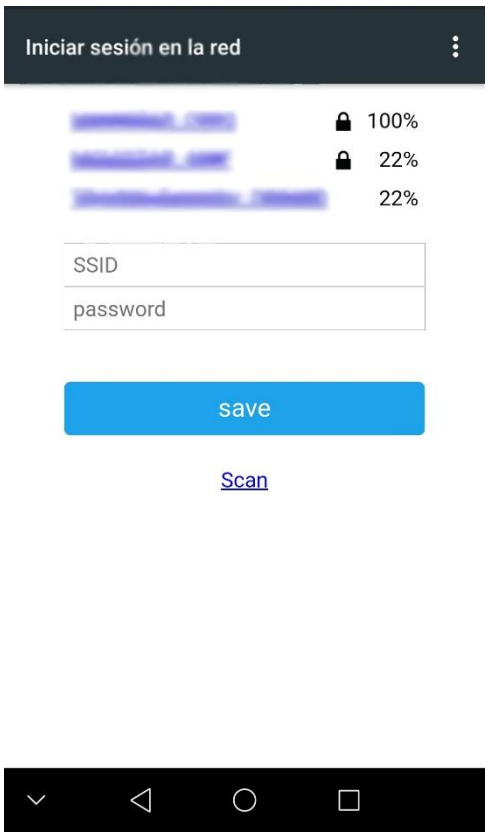


Ilustración 25: Pantalla de introducción de red

Una vez guardada la red wifi, el DCB se reinicia y se conecta automáticamente a la red configurada, esta configuración queda almacenada en la memoria del ESP8266 permitiendo desconectar el DCB de la corriente electrica sin perder las configuraciones.

En la siguiente imagen se puede observar la pantalla de información de red del DCB, para acceder a ella bastará con presionar sobre el botón “Info” de la pantalla principal.

Ilustración 26: Pantalla de información de red

3.2.1.2. DomyosVE430:

En Arduino es posible crear librerías utilizando el lenguaje de programación C o C++. Estos dos lenguajes no se basan en el mismo estilo de programación. El primero es un lenguaje declarativo, mientras que el segundo es un lenguaje orientado a objetos.

Las librerías en estos dos lenguajes presentan similitudes a nivel de construcción. Todas se dividen en dos archivos, un archivo de encabezado, o *header*, que contiene todas las definiciones de las funciones, es decir, enumera el conjunto de funciones propias de la librería, y un archivo fuente que contiene todo el código que describe estas funciones. A continuación, se detalla el proceso de creación de librerías C++ en Arduino.

El lenguaje orientado a objetos se basa en la creación y uso de objetos software. Estos objetos son una representación de un concepto o de una idea que queremos

implementar. Deben contener en su interior todas las funciones o variables que permitan su utilización.

Una clase es una estructura que sirve para describir estos objetos. Dichas clases tienen un nombre y contienen las variables y métodos que forman los objetos. Para utilizar estas clases es necesario instanciarlas mediante un método constructor.

Una librería en C++ está formada por un archivo de encabezado con extensión .h y de un archivo de código fuente con extensión .cpp. El archivo de encabezado contiene una descripción de la clase. Las variables y funciones de la clase deben dividirse en dos partes distintas, la parte *public* y la parte *private*. Estas dos partes determinan su accesibilidad. De hecho, es necesario ocultar alguno de estos elementos a las otras partes del programa principal. Los elementos contenidos en la parte *private* solo son accesibles por la clase en sí misma, mientras que los elementos contenidos en la parte *pubic* lo son por todo el mundo. Como se ha mencionado, las clases necesitan un método de construcción. Éste método debe tener el mismo nombre que la clase y no devolver ningún valor. Por el contrario, puede recibir argumentos que se pueden utilizar, por ejemplo, para asignar valores por iniciales a variables internas.

Este archivo de encabezado debe de tener las directivas que incluyan la librería estándar utilizada por Arduino, así como las directivas que garantizan la unicidad de la librería durante su utilización. El archivo de cabecera creado para la librería de control de la Domyos VE430 es el siguiente:

```
/**
 * @file DomyosVE430.h
 * @date 24.01.2017
 * @author Jordi Peiró Castelló
 *
 * DomyosVE430 es una librería para trabajar con la bicicleta estatica de
 * Decathlon, Domyos VE430. Ésta libreria nos permite saber las pulsaciones del
 * correrdor así como a la velocidad que está funcionando la bicicelta. Además
 * también podremos subir y bajar la resistencia de esta.
 */

#ifndef DomyosVE430_h
#define DomyosVE430_h

#include <ESP8266WiFi.h>
```

```
#include <ESP8266WiFi.h>

class DomyosVE430
{
  public:
    DomyosVE430 (int pinVelocidad, double radioRueda);

    //getters
    //Obtiene la distancia recorrida
    double  getDistancia();

    //Método lanzadera para obtener la velocidad
    double  getVelocidad();

    //setters
    //Habilita o desabilita el modo desarrollador
    void    setDebugOutput(bool debug);

    //resto de métodos
    //Baja la resisitencia según el decremento pasado como parámetro
    void    bajarResistencia();
    //Sube la resisitencia según el incremento pasado como parámetro
    void    subirResistencia();
    //Finaliza el proceso de subir/bajar resistencia
    void    pararMovimientoResistencia();

  private:
    bool _debug = false; //modo debug activado por defecto
    template <typename Generic>
    void          DEBUG_VE430(Generic text); //función debug
    double getPerimetro();
    double calcularVelocidad();

    int           contador = 0;
    double        distanciaKM = 0;
    double        distanciaM = 0;
    int           estadoActual1 = 0;
    int           estadoActual2 = 0;
    int           estadoUltimo = 0;
    double        perimetroRueda;
    int           pinPulsaciones;
    int           pinResistencia;
    int           pinVelocidad;
    double        pi = 3.14159268359;
    double        pulsaciones;
    double        radioRueda;
    double        resistencia;
    double        tiempoEntreVueltas = 0; //milisegundos
    double        tiempoEnHoras = 0; //horas
    double        tiempoVueltaImpar = 0; //milisegundos
    double        tiempoVueltaPar = 0; //milisegundos
    double        velocidad;
};
#endif
```

Ilustración 27: DomyosVE430.h Fichero de cabecera de la librería

Una vez desarrollado el fichero de cabecera hay que crear el archivo de código fuente DomyosVE430.cpp. Este archivo además del método constructor de la clase, debe contener la declaración de todos los métodos de la clase mencionados en el fichero de encabezado.

```
#include "DomyosVE430.h"

/**
* Permite imprimir mensajes por el puerto serie cuando el modo desarrollador
* está activado. (Por defecto se encuentra activo)
*/
template <typename Generic>
/**
 * Si el modo debug esta activo (valor por defecto) la aplicación irá
 * imprimiendo mensajes a modo de traza
 * @method DomyosVE430::DEBUG_VE430
 * @param  text                    Mensaje que comenzará por *VE430:
 */
void DomyosVE430::DEBUG_VE430(Generic text) {
  if (_debug) {
    Serial.print("*VE430: ");
    Serial.println(text);
  }
}

DomyosVE430::DomyosVE430 (int pinVelocidad, double radioRueda){
  // MOTOR
  pinMode(D1, OUTPUT); // Velocidad del motor
  pinMode(D3, OUTPUT); // Dirección del motor
  // No utilizar D1,D2,D3,D4
  this->pinVelocidad = pinVelocidad; //input
  pinMode(pinVelocidad, INPUT);
  this->radioRueda = radioRueda;
  perimetroRueda = this->getPerimetro();

  DEBUG_VE430("Clase DomyosVE430 instanciada correctamente");
}
/**
 * Baja la resistencia de la bicicleta durante el tiempo pasado por parametro
 * @method DomyosVE430::bajarResistencia
 * @param  decremento tiempo de movimiento del motor en segundos
 */
void DomyosVE430::bajarResistencia(){
  analogWrite(D1, 1023);
  digitalWrite(D3, LOW);
}

/**
 * sube la resistencia de la bicicleta durante el tiempo pasado por parametro
 * @method DomyosVE430::bajarResistencia
 * @param  decremento tiempo de movimiento del motor en segundos
 */
```

```
48  void DomyosVE430::subirResistencia(){
49    analogWrite(D1, 1023);
50    digitalWrite(D3, HIGH);
51  }
52
53  /**
54  * Paraliza el proceso de subir/bajar la resistencia
55  */
56  void DomyosVE430::pararMovimientoResistencia(){
57    analogWrite(D1, 0);
58    digitalWrite(D3, LOW);
59  }
60
61  /**
62   * Calcula la velocidad de la bicicleta
63   * @method calcularVelocidad
64   * @return double
65   */
66  double DomyosVE430::calcularVelocidad(){
67    if ((contador % 2) == 0){ //vuelta par
68      // guarda el tiempo transcurrido desde el inicio hasta esta vuelta
69      tiempoVueltaPar = millis();
70      DEBUG_VE430("Tiempo vuelta par: " + String(tiempoVueltaPar));
71    }else{
72      // guarda el tiempo transcurrido desde el inicio hasta esta vuelta
73      tiempoVueltaImpar = millis();
74       DEBUG_VE430("Tiempo vuelta impar: " + String(tiempoVueltaImpar));
75     }
76     // tiempo trnascurrido entre vueltas
77     tiempoEntreVueltas = abs(tiempoVueltaImpar - tiempoVueltaPar); //milisegundos
78     tiempoEnHoras =  (((tiempoEntreVueltas / 1000.0) / 60 ) / 60);
79     DEBUG_VE430("Tiempo entre vueltas: " + String(tiempoEntreVueltas));
80     DEBUG_VE430("Velocidad: " + String((this->perimetroRueda / 1000)
81                                        / tiempoEnHoras));
82     // perimetro rueda en kilometros / tiempo en  horas
83
84     return (this->perimetroRueda / 1000) / tiempoEnHoras; //velocidad
85   }
86
87   /**
88    * Obtiene la disancia recorrida en kilometros
89    * @method DomyosVE430::getDistancia
90    * @return double
91    */
92   double DomyosVE430::getDistancia(){
93     double distRec = (perimetroRueda * contador)/1000;
94     DEBUG_VE430("Distancia recorrida = " + String(distRec));
95     return distRec;
96   }
97
98   /**
99    * Obtiene el perimetro de la rueda en centimetros
100   * @method getPerimetro
```

```
 * @return double
 */
double DomyosVE430::getPerimetro(){
  return (2 * this->pi * this->radioRueda) / 100;
}

/**
 * Método lanzadera que devuelve a la velocidad a la que se está corriendo.
 * @method DomyosVE430::getVelocidad
 * @return Velocidad actual de la bicicelta
 */
double DomyosVE430::getVelocidad(){
  estadoActual1 = digitalRead(pinVelocidad);
  delay(10);
  estadoActual2 = digitalRead(pinVelocidad);
  //Si los estados no son iguales el sketch no hace nada
  DEBUG_VE430("Comprobación de estados: estadoActual1 = " +
              String(estadoActual1) + " estadoActual2 = " +
              String(estadoActual2));
  if (estadoActual1 == estadoActual2){
    DEBUG_VE430("estados iguales");
    if (estadoActual1 != estadoUltimo) {
      if (estadoActual1 == LOW){
        contador++;
        DEBUG_VE430("Vueltas: " + String(contador));
        velocidad = this->calcularVelocidad();
      }
    }
  }
  estadoUltimo = estadoActual1;
  return velocidad;
}

```

Ilustración 28: Implementación de la librería DomyosVE430

Como se puede apreciar en la Ilustración 28 el nombre de los métodos va precedido por «DomyosVE430: :». Esto indica al compilador que el método que sigue forma parte de la clase DomyosVE430.

Para añadir esta librería al proyecto bastará con incluir el archivo de cabecera al principio del archivo .ino.

```
#include "DomyosVE430.h"
```

Ilustración 29: Utilización de la librería en el proyecto.

La librería DomyosVE430 dota al proyecto de la funcionalidad necesaria para poder obtener la velocidad y variar el grado de resistencia. A continuación, se detallan los métodos necesarios.

3.2.1.2.1. Velocidad:

```
/**
 * Método lanzadera que devuelve a la velocidad a la que se está corriendo.
 * @method DomyosVE430::getVelocidad
 * @return Velocidad actual de la bicicelta
 */
double DomyosVE430::getVelocidad(){
  estadoActual1 = digitalRead(pinVelocidad);
  delay(10);
  estadoActual2 = digitalRead(pinVelocidad);
  //Si los estados no son iguales el sketch no hace nada
  DEBUG_VE430("Comprobación de estados: estadoActual1 = " +
              String(estadoActual1) + " estadoActual2 = " +
              String(estadoActual2));
  if (estadoActual1 == estadoActual2){
    DEBUG_VE430("estados iguales");
    if (estadoActual1 != estadoUltimo) {
      if (estadoActual1 == LOW){
        contador++;
        DEBUG_VE430("Vueltas: " + String(contador));
        velocidad = this->calcularVelocidad();
      }
    }
  }
  estadoUltimo = estadoActual1;
  return velocidad;
}
```

Ilustración 30: Método getVelocidad()

Para empezar con el cálculo de la velocidad cabe destacar el uso del método *getVelocidad()* el cual se encarga de eliminar el efecto "*bounce*" o rebote. Este fenómeno se produce al efectuar el conteo de vueltas. Durante el primer milisegundo de cada conteo el sensor no reacciona correctamente lo que produce pequeñas variaciones de la señal de entrada. Como en el caso de los pulsadores, este fenómeno hace que los valores de entrada alternen rápidamente entre HIGH y LOW, por tanto, el valor obtenido puede no ser correcto. Para solucionar esto se lee el valor obtenido por el sensor de efecto hall dos veces en un intervalo de 10 milisegundos. El siguiente paso es comparar los dos valores, si son iguales significa que el sensor se encuentra en posición estable y por tanto se procede a ejecutar toda la lógica del conteo de vueltas.

```
/**
 * Calcula la velocidad de la bicicleta
 * @method calcularVelocidad
 * @return double
 */
double DomyosVE430::calcularVelocidad(){
  if ((contador % 2) == 0){ //vuelta par
    // guarda el tiempo transcurrido desde el inicio hasta esta vuelta
    tiempoVueltaPar = millis();
    DEBUG_VE430("Tiempo vuelta par: " + String(tiempoVueltaPar));
  }else{
    // guarda el tiempo transcurrido desde el inicio hasta esta vuelta
    tiempoVueltaImpar = millis();
    DEBUG_VE430("Tiempo vuelta impar: " + String(tiempoVueltaImpar));
  }
  // tiempo trnascurrido entre vueltas
  tiempoEntreVueltas = abs(tiempoVueltaImpar - tiempoVueltaPar); //milisegundos
  tiempoEnHoras =  (((tiempoEntreVueltas / 1000.0) / 60 ) / 60);
  DEBUG_VE430("Tiempo entre vueltas: " + String(tiempoEntreVueltas));
  DEBUG_VE430("Velocidad: " + String((this->perimetroRueda / 1000) /
                                     tiempoEnHoras));
  // perimetro rueda en kilometros / tiempo en  horas

  return (this->perimetroRueda / 1000) / tiempoEnHoras; //velocidad
}
```

Ilustración 31: Método calcularVelocidad().

Cuando el método getVelocidad() detecta que se ha dado una vuelta completa llama al método calcularVelocidad() para calcular a la velocidad que se está corriendo. Para ello se guarda el tiempo que ha transcurrido desde el inicio hasta la vuelta actual, separando entre vueltas pares e impares. Después de guardar el tiempo entre vueltas, en adelante TEV, se calcula el valor absoluto de la diferencia de tiempo entre la vuelta actual y la anterior en milisegundos. Para calcular la velocidad, se divide el perímetro de la rueda en kilómetros entre el TEV en horas. El valor obtenido en la división es la velocidad a la que circula el corredor en km/h.

3.2.1.2.2. Gestión de la resistencia:

Como se muestra en la siguiente ilustración, para gestionar el nivel de resistencia de pedaleo, se han desarrollado tres métodos.

```
/**
 * Baja la resistencia de la bicicleta durante el tiempo pasado por parametro
 * @method DomyosVE430::bajarResistencia
 * @param  decremento tiempo de movimiento del motor en segundos
 */
void DomyosVE430::bajarResistencia(){
  analogWrite(D1, 1023);
  digitalWrite(D3, LOW);
}

/**
 * sube la resistencia de la bicicleta durante el tiempo pasado por parametro
 * @method DomyosVE430::bajarResistencia
 * @param  decremento tiempo de movimiento del motor en segundos
 */
void DomyosVE430::subirResistencia(){
  analogWrite(D1, 1023);
  digitalWrite(D3, HIGH);
}

/**
* Paraliza el proceso de subir/bajar la resistencia
*/
void DomyosVE430::pararMovimientoResistencia(){
  analogWrite(D1, 0);
  digitalWrite(D3, LOW);
}
```

Ilustración 32: Métodos para gestión de la resistencia

Para entender cómo funcionan estos métodos, hay que tener en cuenta el funcionamiento del *NodeMCU Motor Shield*. Éste se encarga de hacer que el motor gire en una dirección u otra según el sentido de la corriente. A consecuencia de este movimiento se enrolla o desenrolla un cable metálico permitiendo así acercar o alejar el imán a la rueda metálica que el corredor gira al correr. Teniendo en cuenta este funcionamiento, desde el software desarrollado para subir o bajar la resistencia se debe de indicar a *NodeMCU Motor Shiel* en qué dirección debe circular la corriente hacia el motor. Para ello dispone de un pin especifico, D3, al cual se le pasa una señal en estado HIGH para que la corriente gire en sentido de las agujas del reloj o LOW para que gire en sentido contrario. Además se debe indicar la intensidad de corriente enviada al

motor, instrucción *analogWrite(D1, 0)* para el motor e instrucción *analogWrite(D1, 1023)* pone en marcha el motor.

La gestión del tiempo de movimiento del motor se debe hacer desde el programa principal, pudiéndose hacer como el programador desee o según las características del proyecto. En este caso se ha dividido el recorrido del motor en 10 intervalos permitiendo así simular el funcionamiento de un motor paso a paso en el que se puede incrementar el nivel de resistencia de uno en uno. En el ecosistema de Arduino existen varias funciones para la gestión del tiempo: las funciones de pausa y las funciones contadoras.

- Funciones de pausa: *delay(tiempo)* y *delayMicroseconds(tiempo)*. Como su propio nombre indica, permiten realizar pausas en el programa. Estas funciones presentan un inconveniente muy importante. Su uso interrumpe el resto de acciones del programa, paran la ejecución de las instrucciones al nivel de llamada de una de estas funciones durante el tiempo pasado por parámetro.
- Funciones contadoras: millis() y micros(). Estas funciones permiten saber el tiempo que ha transcurrido desde la puesta en marcha del programa presente en controlador.

Como ya se ha descrito anteriormente el DCB debe de controlar el motor, pero también es el encargado de controlar la velocidad, recibir la información de giro, etc. Todas estas funciones no pueden quedar a la espera de que el motor suba o baje uno o varios niveles. Imagine que se desean subir 6 niveles de resistencia a 0,9 segundos por nivel, esto desencadenaría una pausa de 5,1 segundos. Para el proyecto en cuestión esto supone un desastre absoluto, ya que el DCB estaría 5,1 segundos sin calcular la velocidad, sin actualizar información en la base de datos de firebase, etc.

En el DCB para desarrollar la gestión del tiempo de movimiento del motor se han utilizado las funciones contadoras y no las de pausa. A continuación, se explica con más detalle cómo funciona la gestión del tiempo de movimiento del motor.

```
  //resistencia motor
int nivelResistenciaActual = 10;
// Guarada el instante en que se empezó a subir/bajar la resistencia, es decir,
// cuando se activó el motor
unsigned long tiempoInicioMoverResistencia = 0;
// Estará a verdadero si se está subiendo la resistencia de la bicicleta
bool moviendoResistencia = false;
// Determina el timepo que tiene que estar el motor desplazando para subir
// la resistencia un nivel.
int tiempoSubiendoPorNivel= 900; // milisegundos
int tiempoBajandoPorNivel= 1700; // milisegundos
// Guarda el tiempo que debe de estar el motor en funcionamiento.
unsigned long tiempoMoviendoReistencia = 0;
```

Ilustración 33: Variables necesarias para el control del motor

Cuando se inicia el programa se presupone que la resistencia está al máximo, nivel 10, por tanto la variable *nivelResistenciaActual* se inicializa con dicho valor. Esto permite reiniciar siempre el nivel de la resistencia a cero. Para ello basta con llamar a la función *bajarNivelResistencia(0)* desde la función *setup()* del programa principal. Esto bajará el nivel de resistencia de la bici desde la posisición en que se quedó en su último uso a nivel cero. La variable *tiempoInicioMoverResistencia* se utiliza para guardar el instatnte en el que el motor se pone en funcionamiento, *moviendoResistencia* determina si el motor esta en movimiento o no, *tiempoSubiendoPorNivel* y *tiempoBajandoPorNivel* indican el tiempo que debe estar girando el motor para subir o bajar un nivel, por último *tiempoMoviendoResistencia* indica el tiempo que debe de estar el motor en movimiento.

```
void subirNivelResistencia(int aNivel){
  if(!moviendoResistencia){
    if (aNivel > 10) {
      aNivel = 10;
    }
    Serial.println("Subiendo resistencia...");
    // Calculamos los niveles a subir
    int nivelesASubir = aNivel - nivelResistenciaActual;
    // Calculamos el tiempo que tiene que estar activo el motor, para subir los
    // niveles de resistencia deseados.
    tiempoMoviendoReistencia += (nivelesASubir * tiempoSubiendoPorNivel);
    moviendoResistencia = true;
    //Guardamos el instante que empezamos a subir la resistencia
    tiempoInicioMoverResistencia = millis();
    bici.subirResistencia();
    nivelResistenciaActual = aNivel;
  }
}
```

Ilustración 34: Método para subir la resistencia

```
84    /**
85    * Baja el nivel de resistencia hasta el valor pasado por parámetro,
86    * llengando como mínimo al nivel 0
87    */
88    void bajarNivelResistencia(int aNivel){
89      if(!moviendoResistencia){
90        Serial.println("Bajando resistencia...");
91         // calculamos los niveles a bajar
92        int nivelesABajar = nivelResistenciaActual - aNivel;
93        if (nivelesABajar > nivelResistenciaActual){
94          nivelesABajar = nivelResistenciaActual; // Como máximo bajamos a nivel 0
95        }
96        tiempoMoviendoReistencia += (nivelesABajar * tiempoBajandoPorNivel);
97        // Indicamos que vamos a mover la resistencia
98        moviendoResistencia = true;
99        // Guardamos el instante en que empezamos a mover la resistencia
100       tiempoInicioMoverResistencia = millis();
101       // empezamos a bajar la resistencia
102       bici.bajarResistencia();
103       nivelResistenciaActual = aNivel;
104     }
105   }
```

Ilustración 35: Método bajarNivelResistencia()

Los métodos *subirNivelResistencia(int Nivel)* y *bajarNivelResistencia(int Nivel)* se encargan de calcular el tiempo de movimiento del motor y subir o bajar la resistencia respectivamente, además de poner a *true* la variable *moviendoResistencia*, lo cual indica que la resistencia se está desplazando. Estos métodos no detienen el movimiento del motor por tanto una vez llamados el motor se pone en marcha, pero no se detiene. Para ello hay que efectuar una llamada al método *pararMovimientoResistencia()*. Esta llamada se efectúa desde la función *loop()* del programa principal, es decir cada iteración del bucle del programa Arduino. Dicho método comprueba si el tiempo de movimiento del motor es mayor o igual al tiempo que debe moverse el motor, si es así, llama al método de la librería que se encarga de detener el movimiento del motor.

```
107   /**
108   * Deja de mover la resistencia porque se ha llegado al límite de tiempo.
109   */
110   void pararMovimientoResistencia(){
111     if ((millis() >= (tiempoMoviendoReistencia + tiempoInicioMoverResistencia))
112         && moviendoResistencia){
113       Serial.println("parando movimiento resistencia...");
114       moviendoResistencia = false;
115       bici.pararMovimientoResistencia();
116       tiempoMoviendoReistencia = 0;
117       tiempoInicioMoverResistencia = 0;
```

Ilustración 36: Método pararMovimientoResistencia()

Como se muestra en los métodos anteriores *pararMovimientoResistencia()*, *bajarNivelResistencia()* y *subirNivelResistencia()* se utiliza la función contadora *millis()* para evitar que mientras se está cambiando la resistencia se pause la ejecución de la aplicación.

3.2.1.3. Comunicación con el IMU mediante WebSocket:

Para conseguir comunicación inalámbrica en tiempo real entre el dispositivo IMU y el DCB sin tener que añadir un módulo bluetooth a ambos dispositivos, con el incremento de tamaño y de coste que esto conllevaría, se ha decidido conectarlos vía wifi mediante un webSocket. En este caso el DCB actúa como servidor y por tanto el IMU como cliente. Para conseguir esta funcionalidad se ha utilizado la librería ArduinoWebSockets desarrollada por Links2004. Ésta librería permite la creación de un servidor webSocket.

Llegados a este punto cabe destacar que el framework de Arduino para ESP8266 permite configurar el ESP8266 como estación, como punto de acceso o ambos. En modo estación se conecta a una red wifi creada por otro punto de acceso y en modo punto de acceso el dispositivo crea una red wifi a la que cualquiera se puede conectar. En este caso se ha configurado para usar ambos, el modo estación para conectar el DCB a la red wifi del corredor y el modo punto de acceso para crear una red llamada “RunsMaps” a la que solamente se conecta el dispositivo IMU para conectar al webSocket y enviar la información de giro.

Para empezar a utilizar la librería se debe de instanciar la clase *WebSocketsServer* pasando como parámetro del método constructor el puerto en que el webSocket debe escuchar.

```
57    WebSocketsServer webSocket = WebSocketsServer(81);
```

Ilustración 37: Cración de un objeto WebSocketsServer

Una vez instanciada ya se puede acceder a todos los métodos de la clase. Para ello desde la función *setup()* de Arduino se debe de indicar al objeto mediante el método *begin()* para que empiece a escuchar por el puerto configurado y añadir mediante el método *onEvent()* el callback que se ejecutará cuando se dispare algún evento del webSocket.

```
225     webSocket.begin();
226     webSocket.onEvent(webSocketEvent);
```

Ilustración 38: Métodos de inicialización del webSocket

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload,
                    size_t length) {
    switch(type) {
        case WStype_DISCONNECTED:
              Serial.printf("[%u] Disconnected!\n", num);
            break;
        case WStype_CONNECTED:
            {
                IPAddress ip = webSocket.remoteIP(num);
                Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s\n",
                              num, ip[0], ip[1], ip[2], ip[3], payload);
                // send message to client
                webSocket.sendTXT(num, "Connected");
            }
            break;
        case WStype_TEXT:
            Serial.printf("[%u] get Text: %s\n", num, payload);
            // Insertamos en Firebase database los valores del IMU
            //json

            Firebase.set(BIKE, preparaDatosFirebase(payload));
            if(Firebase.failed()){
              Serial.print("set /json failed:");
              Serial.println(Firebase.error());
              return;
            }
            break;
        case WStype_BIN:
            Serial.printf("[%u] get binary length: %u\n", num, length);
            hexdump(payload, length);
            // send message to client
            // webSocket.sendBIN(num, payload, length);
            break;
    }

}
```

Ilustración 3940: Método que gestiona los eventos del webSocket

La Ilustración 39 muestra el método que se ejecuta cuando se dispara algún evento en el webSocket. Siguiendo el orden de programación, este método se ejecutará cada vez que se desconecte o conecte un cliente, se reciba un texto o se reciba un binario por el webSocket. Cabe destacar que en el caso que se envíe un texto, se procede a preparar los datos para posteriormente enviarlos a la base de datos en tiempo real de firebase.

```
JsonObject& preparaDatosFirebase(uint8_t * payload){
  char json[200];
  for (size_t i = 0; i < 200; i++) {
    json[i] = payload[i];
  }
  Serial.printf("%s", json);
  DynamicJsonBuffer jsonBuffer;
  JsonObject& root = jsonBuffer.parseObject(String(json));
  return root;
```

Ilustración 39: Transforma un array codificada en json en un array de Arduino

El método prepararDatosFirebase() transforma la cadena de texto codificado como un array json recibido por el webSocket en un array de Arduino, para conseguir esto se utiliza la librería ArduinoJSON desarrollada por "bblanchon". Posteriormente, se actualizan los valores en la base de datos de firebase como se verá en el siguiente apartado.

3.2.1.4. Comunicación en tiempo real utilizando firebase.

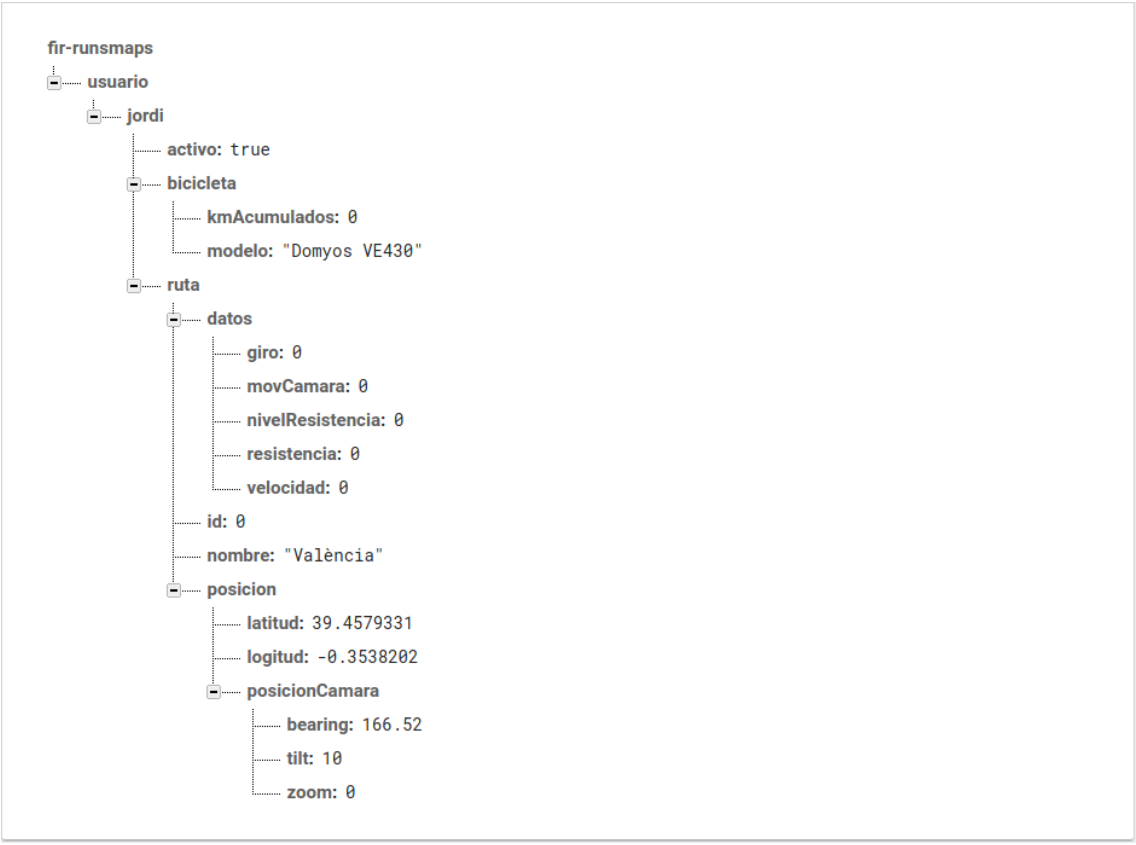


Ilustración 41: Estructura de la base de datos en tiempo real.

Las bases de datos en tiempo real ofrecidas por Google a través de la plataforma firebase, se pueden utilizar como si fueran webSockets, es decir, se pueden obtener sus actualizaciones de estado en tiempo real desde todos los dispositivos conectados y configurados para tal efecto. En este proyecto se utilizan para comunicar la bicicleta elíptica con la aplicación Android, de este modo permite una comunicación bidireccional entre ambas partes del proyecto. Para poder trabajar con firebase desde Arduino, la comunidad ha creado una librería, que se encuentra actualmente en fase de desarrollo, que permite utilizar las bases de datos mencionadas de forma sencilla para el programador final.

Esta librería permite actualizar o leer valores desde la base de datos según el tipo de dato que se va a leer o a actualizar, se pueden incluso actualizar distintos campos al mismo tiempo a través de un array json. Los métodos utilizados son los siguientes:

Actualización	String `pushInt`(*const* String &*path*, int *value*)
	String `pushFloat`(*const* String &*path*, float *value*)
	String `pushBool`(*const* String &*path*, bool *value*)
	String `pushString`(*const* String &*path*, *const* String &*value*)
	String `push`(*const* String &*path*, *const* JsonVariant &*value*)
Inserción	void `setInt`(*const* String &*path*, int *value*)
	void `setFloat`(*const* String &*path*, float *value*)
	void `setBool`(*const* String &*path*, bool *value*)
	void `setString`(*const* String &*path*, *const* String &*value*)
	void `set`(*const* String &*path*, *const* JsonVariant &*value*)
Obtención	int `getInt`(*const* String &*path*)
	float `getFloat`(*const* String &*path*)
	String `getString`(*const* String &*path*)
	bool `getBool`(*const* String &*path*)
	FirebaseObject `get`(*const* String &*path*)

Tabla 6: Métodos para interactuar con firebase

La Ilustración 42 es un ejemplo de cómo se lee un campo entero en firebase desde Arduino.

```
237       // obtenemos la resistencia de la base de datos en tiempo real
238       int resFire = Firebase.getInt("usuario/jordi/ruta/datos/nivelResistencia");
239       if (resFire > nivelResistenciaActual){
240         subirNivelResistencia(resFire);
241       }else if(resFire < nivelResistenciaActual){
242         bajarNivelResistencia(resFire);
243       }
```

Ilustración 42: Obtención de un campo entero de firebase

En el caso anterior se obtiene el valor que debe de tener resistencia. Dicho valor lo actualiza la aplicación Android en la base de datos. Si el valor leído es mayor que la resistencia actual de la bici entonces se llama al método que se encarga de subir la

resistencia, por el contrario, si es menor se llama al método que se encarga de bajar dicha resistencia.

```
249         // actualizamos la velocidad en la base de datos en tiempo real
250         Firebase.setFloat("usuario/jordi/ruta/datos/velocidad", velocidad);
251         // handle error
252         if (Firebase.failed()) {
253             Serial.print("setting /number failed:");
254             Serial.println(Firebase.error());
255             return;
256         }
```

Ilustración 43: Actualización de un float en firebase

En la porción anterior de código se puede observar cómo se actualiza la velocidad en la base de datos, o lo que es lo mismo, como se actualiza un valor en coma flotante, para que posteriormente la aplicación pueda interpretar dicha velocidad y así avanzar entre panoramas de StreetView. Firebase dispone del método *failed()*, el cual nos ofrece información acerca del error producido en el caso de que se hubiera producido alguno.

3.2.2. IMU:

3.2.2.1. Configuración del mpu6050

El principal desafío que se ha encontrado en este apartado ha sido el de integrar el mpu6050 junto con el ESP8266. Existen diferentes formas de trabajar con el giroscopio y acelerómetro incorporados en el chip mpu6050, en este caso, se ha decido trabajar con la librería i2cdevlib desarrollada por "jrowberg". Se eligió esta librería frente a otras alternativas disponibles, por la documentación incluida en la página web del proyecto y por su facilidad de personalización. Explicar el funcionamiento completo de la librería podría dar lugar a un proyecto como este, por tanto, a continuación, se detallan los aspectos principales sobre cómo se ha utilizado dicha librería para configurar el IMU.

Esta librería se encarga de configurar el módulo mpu6050. Como la memoria de dicho modulo es volátil, es necesario configurarlo con cada inicio del programa. Para su correcta configuración, la librería se encarga de insertar el valor correcto en hexadecimal en cada registro de la memoria del chip. Pudiéndose modificar el valor de cada uno de estos registros por el programador. A continuación, se muestra una fracción de la matriz que posteriormente se volcará a la memoria del mpu6050.

```
/* =========================================================================================== *
 | Default MotionApps v2.0 42-byte FIFO packet structure:                                      |
 |                                                                                             |
 | [QUAT W][      ][QUAT X][      ][QUAT Y][      ][QUAT Z][      ][GYRO X][      ][GYRO Y][      ] |
 |   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  |
 |                                                                                             |
 | [GYRO Z][      ][ACC X ][      ][ACC Y ][      ][ACC Z ][      ][      ]                    |
 |  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41                     |
 * =========================================================================================== */

// this block of memory gets written to the MPU on start-up, and it seems
// to be volatile memory, so it has to be done each time (it only takes ~1
// second though)
const unsigned char dmpMemory[MPU6050_DMP_CODE_SIZE] PROGMEM = {
    // bank 0, 256 bytes
    0xFB, 0x00, 0x00, 0x3E, 0x00, 0x0B, 0x00, 0x36, 0x00, 0x01, 0x00, 0x02, 0x00, 0x03, 0x00, 0x00,
    0x00, 0x65, 0x00, 0x54, 0xFF, 0xEF, 0x00, 0x00, 0xFA, 0x80, 0x00, 0x0B, 0x12, 0x82, 0x00, 0x01,
    0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x28, 0x00, 0x00, 0xFF, 0xFF, 0x45, 0x81, 0xFF, 0xFF, 0xFA, 0x72, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x03, 0xE8, 0x00, 0x00, 0x00, 0x01, 0x00, 0x01, 0x7F, 0xFF, 0xFF, 0xFE, 0x80, 0x01,
    0x00, 0x1B, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x3E, 0x03, 0x30, 0x40, 0x00, 0x00, 0x00, 0x02, 0xCA, 0xE3, 0x09, 0x3E, 0x80, 0x00, 0x00,
    0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00,
    0x41, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x0B, 0x2A, 0x00, 0x00, 0x16, 0x55, 0x00, 0x00, 0x21, 0x82,
    0xFD, 0x87, 0x26, 0x50, 0xFD, 0x80, 0x00, 0x00, 0x00, 0x1F, 0x00, 0x00, 0x00, 0x05, 0x80, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00,
    0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x6F, 0x00, 0x02, 0x65, 0x32, 0x00, 0x00, 0x5E, 0xC0,
```

Ilustración 44: Fracción de código de la librería MPU6050_6Axis_MotionApps20

Cabe destacar, que para el correcto funcionamiento del chip no basta solo con utilizar la librería de configuración, es necesario utilizar un scketch de calibración, el cual obtiene los valores de compensación necesarios para la configuración del módulo.

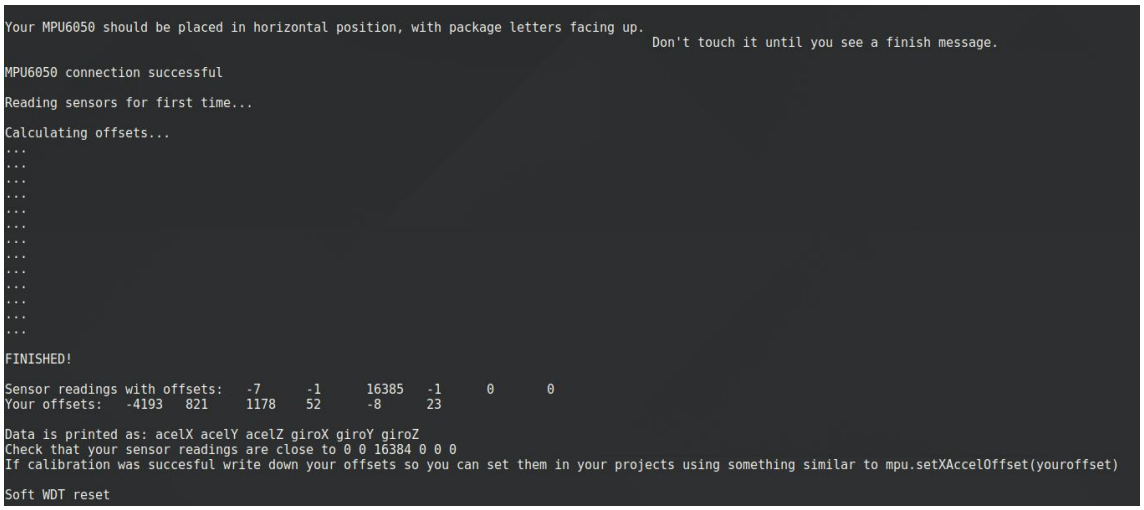


Ilustración 45: Resultado de la calibración del mpu6050

Como se observa en la Ilustración 45, lo primero es mantener el chip en posición horizontal y sin movimiento. A partir de ahí, el sketch empieza con el cálculo de los valores de compensación que posteriormente se han de utilizar para configurar el chip. Los valores de compensación se utilizan directamente en el programa principal ya que estos se pasan como parámetro en los métodos habilitados para configurar dichos valores, esto es así, porque los valores de compensación no son los mismos para todos los dispositivos, de ahí la existencia del scketch de calibración.

```
#define ACCEL_X -4193
#define ACCEL_Y 821
#define ACCEL_Z 1178
#define GYRO_X 52
#define GYRO_Y -8
#define GYRO_Z 23
```

Ilustración 46: Constantes con los valores de compensación

```
mpu.setXGyroOffset(GYRO_X);
mpu.setYGyroOffset(GYRO_Y);
mpu.setZGyroOffset(GYRO_Z);
mpu.setXAccelOffset(ACCEL_X);
mpu.setYAccelOffset(ACCEL_Y);
mpu.setZAccelOffset(ACCEL_Z); // 1688 factory default for my test chip
```

Ilustración 47: Asignación de los valores de compensación

Una vez instanciada la clase de la librería, entre otras cosas, es necesario introducir los valores de compensación en el chip, para ello se ejecutan los métodos de la Ilustración 47 desde la función *setup*() del programa principal. Esto permite al chip calcular con precisión los giros de muñeca.

Otro aspecto a tener en cuenta a la hora de configurar el mpu6050 es la frecuencia con la que el chip enviará las interrupciones, por el pin INT, al módulo ESP8266. Como se ha explicado en el apartado de hardware ambos chips están conectados por tres pins, el pin de interrupciones y los dos de transferencia de información. En este caso las interrupciones se utilizan para que el chip mpu6050 avise al ESP8266, que es el que ejecuta el programa principal, de que tiene información nueva y por tanto debe de ir a consultar los registros que contienen dicha información, si en ese momento la información que contiene el mpu6050 es relevante para el funcionamiento del programa. Para configurar correctamente la frecuencia con la que el chip mpu6050 envía interrupciones al ESP8266, se tiene que modificar el valor de la librería como se explica a continuación.

```
const unsigned char dmpConfig[MPU6050_DMP_CONFIG_SIZE] PROGMEM = {
//  BANK    OFFSET  LENGTH  [DATA]
    0x03,   0x7B,   0x03,   0x4C, 0xCD, 0x6C,         // FCFG_1 inv_set_gyro_calibration
    0x03,   0xAB,   0x03,   0x36, 0x56, 0x76,         // FCFG_3 inv_set_gyro_calibration
    0x00,   0x68,   0x04,   0x02, 0xCB, 0x47, 0xA2,   // D_0_104 inv_set_gyro_calibration
    0x02,   0x18,   0x04,   0x00, 0x05, 0x8B, 0xC1,   // D_0_24 inv_set_gyro_calibration
    0x01,   0x0C,   0x04,   0x00, 0x00, 0x00, 0x00,   // D_1_152 inv_set_accel_calibration
    0x03,   0x7F,   0x06,   0x0C, 0xC9, 0x2C, 0x97, 0x97, 0x97, // FCFG_2 inv_set_accel_calibration
    0x03,   0x89,   0x03,   0x26, 0x46, 0x66,         // FCFG_7 inv_set_accel_calibration
    0x00,   0x6C,   0x02,   0x20, 0x00,               // D_0_108 inv_set_accel_calibration
    0x02,   0x40,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_00 inv_set_compass_calibration
    0x02,   0x44,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_01
    0x02,   0x48,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_02
    0x02,   0x4C,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_10
    0x02,   0x50,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_11
    0x02,   0x54,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_12
    0x02,   0x58,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_20
    0x02,   0x5C,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_21
    0x02,   0xBC,   0x04,   0x00, 0x00, 0x00, 0x00,   // CPASS_MTX_22
    0x01,   0xEC,   0x04,   0x00, 0x00, 0x40, 0x00,   // D_1_236 inv_apply_endian_accel
    0x03,   0x7F,   0x06,   0x0C, 0xC9, 0x2C, 0x97, 0x97, 0x97, // FCFG_2 inv_set_mpu_sensors
    0x04,   0x02,   0x03,   0x0D, 0x35, 0x5D,         // CFG_MOTION_BIAS inv_turn_on_bias_from_no_motion
    0x04,   0x09,   0x04,   0x87, 0x2D, 0x35, 0x3D,   // FCFG_5 inv_set_bias_update
    0x00,   0xA3,   0x01,   0x00,                     // D_0_163 inv_set_dead_zone
    0x00,   0x00,   0x00,   0x01, // SET INT_ENABLE at i=22, SPECIAL INSTRUCTION
    0x07,   0x86,   0x01,   0xFE,                     // CFG_6 inv_set_fifo_interupt
    0x07,   0x41,   0x05,   0xF1, 0x20, 0x28, 0x30, 0x38, // CFG_8 inv_send_quaternion
    0x07,   0x7E,   0x01,   0x30,                     // CFG_16 inv_set_footer
    0x07,   0x46,   0x01,   0x9A,                     // CFG_GYRO_SOURCE inv_send_gyro
    0x07,   0x47,   0x04,   0xF1, 0x28, 0x30, 0x38,   // CFG_9 inv_send_gyro -> inv_construct3_fifo
    0x07,   0x6C,   0x04,   0xF1, 0x28, 0x30, 0x38,   // CFG_12 inv_send_accel -> inv_construct3_fifo
    0x02,   0x16,   0x02,   0x00, 0xFF                // D_0_22 inv_set_fifo_rate
};
```

Ilustración 48: Matriz de configuración del mpu6050

Como se observa en la ilustración 48, en el caso de RunsMaps se utiliza un valor 0xFF para la frecuencia de interrupción. Los valores validos están comprendidos entre 0x00 y 0xFF, siendo 0x00 la frecuencia más rápida y 0xFF la más lenta. 0x00 es equivalente a 200Hz según la documentación, por tanto, haciendo uso de la siguiente formula se puede calcular la frecuencia que se utiliza en el IMU.

$$\frac{200Hz}{1 + valor\ en\ decimal}$$

200Hz es la frecuencia de interrupción en el caso de pasar el valor 0x00, por tanto, en el caso de utilizar un valor de 0xFF, la frecuencia es de:

$$\frac{200Hz}{1 + 15} = 12.5Hz$$

12.5Hz es una frecuencia de interrupción más que suficiente para RunsMaps, ya que se tendría acceso a la información del giro de muñeca del corredor 12.5 veces por segundo.

Teniendo el módulo mpu6050 configurado, es hora de ver como se obtienen los valores que dicho chip obtiene según el movimiento de muñeca del corredor. Como es sabido por todos al correr se hace un movimiento involuntario para evitar que este movimiento interfiera en la aplicación se ha decido descartar todos los valores que estén por debajo de los 45º, es decir, todo valor que esté por debajo de ±45º no será utilizado para realizar giros en la ruta. Según el orden de ejecución se van a explicar los métodos utilizados para obtener y manipular dichos valores.

Con cada iteración del bucle *loop()* del programa principal y teniendo siempre en cuenta que el webSocket esté conectado se llama al método trabajarConIMU().

```
207   void trabajarConIMU(){
208     noInterrupts();
209     if (mpuInterrupt) {
210       getIMUValues();
211       if(checkMPUValues() != 0){
212           sendIMUvalues();
213       }
214
215       mpuInterrupt = false;
216     }
217     interrupts();
218   }
```

Ilustración 49: Método trabajarConIMU

Éste método lo primero que hace es desactivar las interrupciones ya que el envío de información mediante webSocket es un proceso crítico y lo recomendable es hacerlo sin interrupciones. El siguiente paso es comprobar si se ha recibido alguna interrupción, si es así se llama a la función encargada de leer los datos del mpu6050, una vez obtenidos los datos se comprueban con el método checkMPUValues() y si se ha detectado algún giro se envía al DCB mediante el webSocket.

Al recibir una interrupción por parte el mpu6050, se ejecuta el método *dmpDataReady()* tal y como se indica en la siguiente ilustración.

```
309     attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
```

Ilustración 50: Habilitación de interrupciones.

El método *dmpDataReady()* solamente pone a true la variable global *mpuInterrupt.* Se ha decidido trabajar de esta forma porque las interrupciones, como su propio nombre indica, interrumpen la ejecución normal del programa para ejecutar el código del método asignado al evento de la interrupción para después volver al punto de la ejecución que se encontraba el programa. Como se ha visto anteriormente la frecuencia mínima de interrupciones son 12.5 veces por segundo, por tanto, si el código ejecutado por la interrupción pausa la ejecución durante un tiempo prolongado, no se podría ejecutar el flujo normal de la aplicación correctamente.

Como se ha visto en la Ilustración 49 si ha habido alguna interrupción, lo cual indica que hay datos preparados para su lectura en el mpu6050, se procede a su lectura llamando al método *getIMUValues()*.

```
161     void getIMUValues(){
162         mpu.dmpGetQuaternion(&q, fifoBuffer);
163         mpu.dmpGetGravity(&gravity, &q);
164         mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
165         x = ypr[2] * 180/M_PI;
166         y = ypr[1] * 180/M_PI;
167         z = ypr[0] * 180/M_PI;
168     }
```

Ilustración 51: Función getIMUValues()

Obtiene los datos del mpu, los convierte en grados y guarda cada eje del plano en la variable con su nombre.

El siguiente método por orden de ejecución es *checkMPUValues()* el cual se encarga de comprobar si el movimiento se ha de tener en cuenta como un giro o no.

```
/**
* comprobamos si la muñeca ha girado, para ello, comprobamos si el eje y del
* IMU ha variado significativamente. Los valores normales oscilan entre 60º y 90º
* @method checkMPUValues
* @return -1 giro a la izquierda, 0 no hay giro y 1 giro a la derecha
*/
int checkMPUValues(){
  int izquierda = -1;
  int noGiro = 0;
  int derecha = 1;
  if (x <= -40.00 && x >= -140.00){
    Serial.println("No giro");
    return noGiro;
  }else if (x < -140.0) {
    Serial.println("Derecha");
    return derecha;
  }else{
    Serial.println("Izquierda");
    return izquierda;
  }
}
```

Ilustración 52: Función checkMPUValues()

Una vez teminado el proceso de obtención de datos del mpu6050 y comprobados dichos datos, solo queda notificar al DCB que el corredor desea hacer un giro. Aquí entra en juego el websocket.

3.2.2.2. WebSocket cliente:

El IMU como ya se ha mencionado anteriormente, actúa como cliente de webSocket, conectándose al DCB que actúa como servidor. Para conseguir esta funcionalidad es necesario conectar el dispositivo IMU a la red wifi “RunsMaps” creada por el DCB.

```
334         WiFi.begin("RunsMaps", "RunsMaps123");
335         while (WiFi.status() != WL_CONNECTED){
336           /* code */
337           delay(500);
338           Serial.println("Conectando a la red wifi");
339         }
```

Ilustración 53: Código de conexión a la red RunsMaps

El código anterior muestra cómo se conecta el dispositivo IMU a la red wifi del DCB. Para ello basta con pasar como parámetros del método *begin()* el nombre y la contraseña de la red wifi. El bucle posterior solamente sirve para pausar el programa hasta que el dispositivo esté correctamente conectado a la red.

Para poder enviar la información de giro al webSocket del DCB, se debe utilizar la librería WebSocketsClient, también desarrollada por "Links2004". En este caso, para crear un objeto webSocket se debe instanciar la clase WebSocketsClient.

```
91    WebSocketsClient webSocket;
```

Ilustración 54: Creación del objeto webSocket

Una vez creado el objeto solo queda conectar al servidor mediante el método *begin()* y como en el caso anterior, configurar la función que captura los eventos asignados al webSocket, para ello la librería cuenta con el método *onEvent()*. Para este caso en concreto se conecta a la IP 192.168.4.1, que es la que obtiene el servidor por defecto y al puerto 81, que es el puerto por el que el servidor está escuchando.

```
347         webSocket.begin("192.168.4.1", 81);
348         webSocket.onEvent(webSocketEvent);
```

Ilustración 55: Configuración del webSocket cliente

Para terminar con la configuración del webSocket cliente solo queda llamar a la función *loop()* del webSocket dentro del a función *loop()* del programa principal. Esto permite al objeto webSocket estar siempre activo, es decir, con cada iteración del programa principal, se reactiva la conexión del webSocket. Si no se llama al método *loop()* del webSocket con cada iteración del programa principal este estaría conectado pero no sería posible enviar información a través de él.

Llegados a este punto solo que mostrar cómo se envía la información a través del webSocket. Tal y como se puede observar en la siguiente imagen, se utiliza el método sendTXT() para enviar una matriz json con los valores x,y,z. Como se ha visto en el punto anterior, el DCB, convierte dicha matriz en una matriz Arduino para poder trabajar con estos datos.

```
192   void sendIMUvalues(){
193           // display Euler angles in degrees
194           char strX[8], strY[8], strZ[8];
195           dtostrf(x, 6, 2, strX); // Leave room for too large numbers!
196           dtostrf(y, 6, 2, strY); // Leave room for too large numbers!
197           dtostrf(z, 6, 2, strZ); // Leave room for too large numbers!
198           JsonObject& root = jsonBuffer.createObject();
199           root["x"] = x;
200           root["y"] = y;
201           root["z"] = z;
202           char buffer[bufferSize];
203           root.printTo(buffer, sizeof(buffer));
204           webSocket.sendTXT(buffer);
205   }
```

Ilustración 56: Envío de información a través del webSocket

3.2.3. Aplicación:

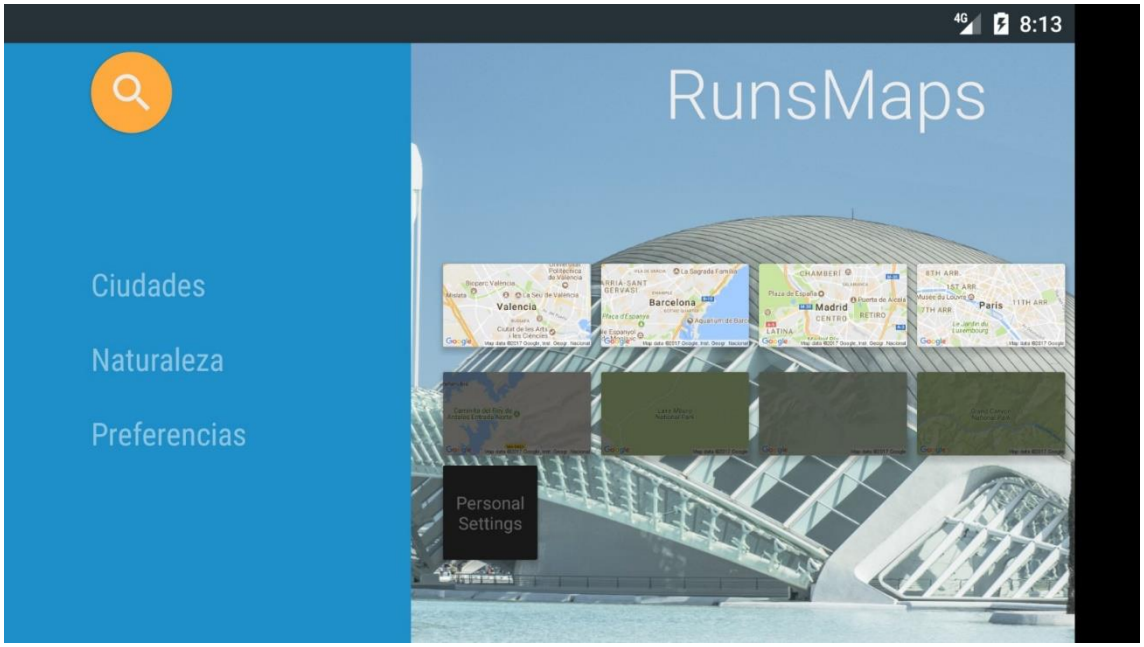


Ilustración 57: Pantalla principal de RunsMaps

Debido a que la interfaz de usuario es una de las mayores diferencias entre las aplicaciones para dispositivos móviles Android y las aplicaciones para Android TV, se debe diseñar una interfaz de usuario adecuada para el uso en la TV. Por ejemplo, se debe crear una aplicación en la que se pueda navegar a través de sus menús utilizando sólo las teclas de dirección ↑ ↓ → ←, en lugar de navegación touchpad, ya que el usuario utiliza el mando a distancia, y no puede utilizar la función de "pantalla táctil" en la televisión. Para facilitar este diseño, el proyecto de código abierto de Android proporciona la biblioteca de soporte de Leanback (android.support.v17.leanback) para que los desarrolladores puedan implementar fácilmente la interfaz de usuario que satisface estos requisitos y, por tanto, adecuado para el uso en la TV.

Se ha partido de la aplicación Leanback base que genera automáticamente Android Studio. RunsMaps ha sido desarrollada utilizado la API 26 del SDK, es decir, Android 8.0 (O).

Al indicar a Android Studio que se quiere desarrollar una aplicación para televisiones, este nos propone utilizar una aplicación base, que como se ha comentado arriba, utiliza el patrón de diseño Leanback. Cuando se crea dicha aplicación base ya están

implementadas la pantalla principal y la de detalles, pero sin funcionalidad alguna, solamente está la estructura básica que debe ser igual para todas las aplicaciones.

Se puede decir que RunsMaps se divide en tres pantallas, la pantalla principal, donde el usuario puede elegir entre distintas ciudades o lugares para empezar a correr. La pantalla de detalles en la que se puede leer más información acerca de la ruta elegida y, por último, está la pantalla de seguimiento de ruta, donde el usuario se va a desplazar por los diferentes panoramas de Google Street View.

Antes de empezar a detallar cada una de las partes de la aplicación, se van a presentar los aspectos básicos del desarrollo de aplicaciones Android.

3.2.3.1. Aspectos básicos del desarrollo en Android:

Actividades:

Una Activity es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción, como marcar un número telefónico, tomar una foto, enviar un correo electrónico o ver un mapa. A cada actividad se le asigna una ventana en la que se puede dibujar su interfaz de usuario. La ventana generalmente abarca toda la pantalla, pero en ocasiones puede ser más pequeña que esta y quedar "flotando" encima de otras ventanas.

Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. Normalmente, una actividad en una aplicación se especifica como la actividad "principal" que se presenta al usuario cuando este inicia la aplicación por primera vez. Cada actividad puede a su vez iniciar otra actividad para poder realizar diferentes acciones. Cada vez que se inicia una actividad nueva, se detiene la actividad anterior, pero el sistema conserva la actividad en una pila (la "pila de actividades"). Cuando se inicia una actividad nueva, se la incluye en la pila de actividades y capta el foco del usuario. La pila de actividades cumple con el mecanismo de pila "el último en entrar es el primero en salir", por lo que, cuando el usuario termina de interactuar con la actividad actual y presiona el botón Atrás, se quita de la pila (y se destruye) y se reanuda la actividad anterior.

Cuando se detiene una actividad porque se inicia otra, se notifica el cambio de estado a través de los métodos callback del ciclo de vida de la actividad. Existen diferentes métodos callback que podría recibir una actividad como consecuencia de un cambio de estado (ya sea que el sistema la esté creando, deteniendo, reanudando o destruyendo) y cada callback te da la oportunidad de realizar una tarea específica que resulta adecuada para ese cambio de estado. Por ejemplo, cuando se detiene una actividad, esta debería liberar los objetos grandes, como una conexión de red o a la base de datos. Cuando se reanuda la actividad, puedes volver a adquirir los recursos necesarios y reanudar las acciones que se interrumpieron. Todas estas transiciones de estado forman parte del ciclo de vida de la actividad.

Fuente: Android developers.

https://developer.android.com/guide/components/activities.html?hl=es-419#Creating

Fragmentos:

Un Fragment representa un comportamiento o una parte de la interfaz de usuario en una Activity. Puedes combinar múltiples fragmentos en una sola actividad para crear una IU multipanel y volver a usar un fragmento en múltiples actividades. Puedes pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio, recibe sus propios eventos de entrada y que puedes agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que puedes volver a usar en diferentes actividades).

Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos. Sin embargo, mientras una actividad se está ejecutando (está en el estado del ciclo de vida reanudada), puedes manipular cada fragmento de forma independiente; por ejemplo, para agregarlos o quitarlos. Cuando realizas una transacción de fragmentos como esta, también puedes agregarlos a una pila de actividades administrada por la actividad; cada entrada

de la pila de actividades en la actividad es un registro de la transacción de fragmentos realizada. La pila de actividades le permite al usuario invertir una transacción de fragmentos (navegar hacia atrás) al presionar el botón Atrás.

Cuando agregas un fragmento como parte del diseño de tu actividad, este se ubica en ViewGroup, dentro de la jerarquía de vistas de la actividad y el fragmento define su propio diseño de vista. Puedes insertar un fragmento en el diseño de tu actividad declarando el fragmento en el archivo de diseño de la actividad como elemento <fragment> o agregándolo a un ViewGroup existente desde el código de tu aplicación. Sin embargo, no es necesario que un fragmento forme parte del diseño de la actividad; también puedes usar un fragmento con su IU propia, como un trabajador invisible para la actividad.

Este documento describe cómo crear tu aplicación para usar fragmentos, incluido cómo los fragmentos pueden mantener su estado cuando se los agrega a la pila de actividades de la actividad, cómo pueden compartir eventos con la actividad y con otros fragmentos en la actividad, cómo pueden contribuir con la barra de acciones de la actividad, etc.

Fuente: Android developers.
https://developer.android.com/guide/components/fragments.html?hl=es-419#Design

Intents y filtros de intents

Una Intent es un objeto de acción que puedes usar para solicitar una acción de otro componente de la aplicación. Aunque las intents facilitan la comunicación entre los componentes de muchas maneras, existen tres casos de uso fundamentales:

Para comenzar una actividad:

Una Activity representa una única pantalla en una aplicación. Puedes iniciar una nueva instancia de una Activity pasando una Intent a startActivity(). La Intent describe la actividad que se debe iniciar y contiene los datos necesarios para ello.

Si deseas recibir un resultado de la actividad cuando finalice, llama a startActivityForResult(). La actividad recibe el resultado como un objeto Intent separado en el callback de onActivityResult() de la actividad. Para obtener más información, consulta la guía Actividades.

Para iniciar un servicio:

Un Service es un componente que realiza operaciones en segundo plano sin una interfaz de usuario. Puede iniciar un servicio para realizar una operación única (como descargar un archivo) pasando una Intent a startService(). La Intent describe el servicio que se debe iniciar y contiene los datos necesarios para ello.

Si el servicio está diseñado con una interfaz cliente-servidor, puedes establecer un enlace con el servicio de otro componente pasando una Intent a bindService(). Para obtener más información, consulte la guía Servicios.

Para entregar un mensaje:

Un mensaje es un aviso que cualquier aplicación puede recibir. El sistema entrega varios mensajes de eventos del sistema, como cuando el sistema arranca o el dispositivo comienza a cargarse. Puedes enviar un mensaje a otras apps pasando una Intent a sendBroadcast(), sendOrderedBroadcast() o sendStickyBroadcast().

Fuente: Android developers.
https://developer.android.com/guide/components/intents-filters.html?hl=es-419

Una vez vistos los aspectos básicos necesarios para entender cómo funcionan las aplicaciones Android, se van a detallar los aspectos más importantes de cada una de las partes de la aplicación.

3.2.3.2. Pantalla principal:

La pantalla principal se compone de una actividad, la cual, está al mismo tiempo compuesta por un fragmento.

```
26  public class MainActivity extends Activity {
27      /**
28       * Called when the activity is first created.
29       */
30
31      @Override
32      public void onCreate(Bundle savedInstanceState) {
33          super.onCreate(savedInstanceState);
34          setContentView(R.layout.activity_main);
35      }
36  }
37
```

Ilustración 58: MainActivity.java

En la Ilustración 58 se muestra el código de la actividad principal de la aplicación, como se puede observar, solamente se asigna la vista que debe mostrar por pantalla. Dicha vista es la mostrada por la Ilustración 59, la cual solamente indica que va a tener un fragmento.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <fragment xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:id="@+id/main_browse_fragment"
5       android:name="com.pc.jordi.runsmaps_app.ui.MainFragment"
6       android:layout_width="match_parent"
7       android:layout_height="match_parent"
8       tools:context=".ui.MainActivity"
9       tools:deviceIds="tv"
10      tools:ignore="MergeRootFrame" />
11
```

Ilustración 59: activity_main.xml

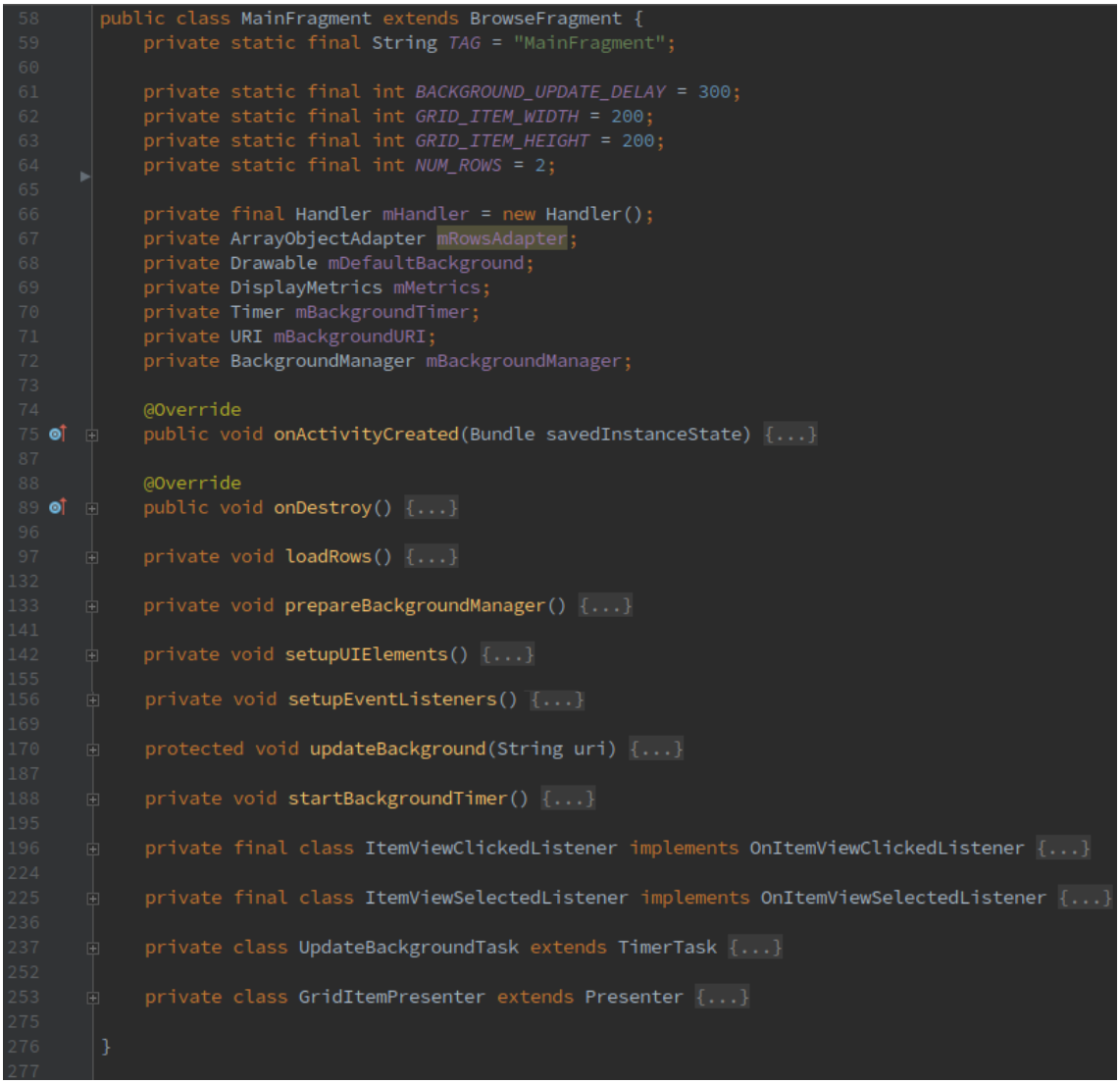


Ilustración 61: Estructura de MainFragment.java

Como se puede observar en la ilustración anterior, el fragmento utilizado en la actividad principal de RunsMaps, extiende la clase *BrowseFragment*, la cual forma parte de la librería *android.support.v17.leanback* y aporta todo el diseño de la pantalla además de la funcionalidad básica. Al extender *MainFragment* de *BrowseFragment* es necesario sobrescribir los métodos *onActivityCreated* y *onDestroy* para poder así, asignar la funcionalidad deseada al crear o destruir el fragmento.

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    Log.i(TAG, "onCreate");
    super.onActivityCreated(savedInstanceState);
    prepareBackgroundManager();
    setupUIElements();
    loadRows();
    setupEventListeners();
}
```

Ilustración 60: Método onActivityCreated()

La Ilustración 60 muestra la funcionalidad añadida al método onActivityCreated(), además de llamar a su padre para ejecutar toda su funcionalidad, llama al método prepareBackgroundManager(), setupUIElements(), loadRows() y setupEventListeners().

Crear tarjetas con las ciudades y lugares:

```
93      private void loadRows() {
94          List<Route> list = RouteList.setupRoutes();
95          int listSize = list.size();
96          mRowsAdapter = new ArrayObjectAdapter(new ListRowPresenter());
97          CardPresenter cardPresenter = new CardPresenter();
98          int i;
99          for (i = 0; i < NUM_ROWS; i++) {
100             if (i != 0) {
101                 Collections.shuffle(list);
102             }
103             ArrayObjectAdapter listRowAdapter = new ArrayObjectAdapter(cardPresenter);
104             for (int j = 0; j < listSize; j++) {
105                 Route route = list.get(j);
106                 if(RouteList.ROUTE_CATEGORY[i].toString().equals(route.getCategory().toString())){
107                     listRowAdapter.add(route);
108                 }
109             }
110             HeaderItem header = new HeaderItem(i, RouteList.ROUTE_CATEGORY[i]);
111             mRowsAdapter.add(new ListRow(header, listRowAdapter));
112         }
113         HeaderItem gridHeader = new HeaderItem(i, "Preferencias");
114         GridItemPresenter mGridPresenter = new GridItemPresenter();
115         ArrayObjectAdapter gridRowAdapter = new ArrayObjectAdapter(mGridPresenter);
116         gridRowAdapter.add("Personal Settings");
117         mRowsAdapter.add(new ListRow(gridHeader, gridRowAdapter));
118         setAdapter(mRowsAdapter);
119     }
```

Ilustración 62: Método loadRows()

El método loadRows() es el encargado de crear las tarjetas con las ciudades o lugares. La información acerca de las tarjetas se obtiene de la clase Route, la cual implementa el modelo de datos, ya que en la actualidad no se dispone de una base de datos con dicha información. Una vez creada la lista con todas las rutas, se procede a crear las cabeceras de cada sección y dentro de cada cabecera, el conjunto de tarjetas que pertenecen a dicha cabecera o categoría. En la siguiente ilustración se explica gráficamente la relación entre las tarjetas y su categoría.

Ilustración 63: Relación categoría-tarjeta

Gestión de eventos en las tarjetas:

Para poder identificar que tarjeta ha seleccionado el usuario, es necesario añadir escuchadores a sus eventos *ItemViewClicked* e *ItemViewSelected*, el método encargado de añadir dichos escuchadores es *setupEventListeners().*

Los métodos setOnItemViewClickedListener() y setOnItemViewSelectedListener() pertenecientes a la clase BorwseFragment se utilizan para asignar el callback que se ejecutará cuando se disparen los eventos de selección o clicado de la tarjeta.

```
143     private void setupEventListeners() {
144         setOnItemViewClickedListener(new ItemViewClickedListener());
145         setOnItemViewSelectedListener(new ItemViewSelectedListener());
146     }
```

Ilustración 64: Asignación de callbacks para los eventos de las tarjetas

Para poder asignar el callback a un evento, es necesario crear una clase que implemente la clase abstracta del evento en cuestión. Para este caso concreto es necesario crear una clase que implemente la clase abstracta *OnItemViewClickedListener()* para el evento *onClick* y otra que implemente *OnItemViewSelectedListener()* para el evento *onSelect.*

A continuación, se muestran la implementación de dichas clases.

```
148     private final class ItemViewClickedListener implements OnItemViewClickedListener {
149         @Override
150         public void onItemClicked(Presenter.ViewHolder itemViewHolder, Object item,
151                                   RowPresenter.ViewHolder rowViewHolder, Row row) {
152 
153             if (item instanceof Route) {
154                 Route route = (Route) item;
155                 Log.d(TAG, "Item: " + item.toString());
156                 // en el intent debemos de pasar la clase de la actividad que deseamos abrir
157                 Intent intent = new Intent(getActivity(), DetailsActivity.class);
158                 intent.putExtra(DetailsActivity.ROUTE, route);
159 
160                 Bundle bundle = ActivityOptionsCompat.makeSceneTransitionAnimation(
161                         getActivity(),
162                         ((ImageCardView) itemViewHolder.view).getMainImageView(),
163                         DetailsActivity.SHARED_ELEMENT_NAME).toBundle();
164                 getActivity().startActivity(intent, bundle);
165             } else if (item instanceof String) {
166                 if (((String) item).indexOf(getString(R.string.error_fragment)) >= 0) {
167                     Intent intent = new Intent(getActivity(), BrowseErrorActivity.class);
168                     startActivity(intent);
169                 } else {
170                     Toast.makeText(getActivity(), ((String) item), Toast.LENGTH_SHORT)
171                             .show();
172                 }
173             }
174         }
175     }
```

Ilustración 65: Implementación del evento onClick

Cuando el usuario hace clic sobre una tarjeta, se ejecuta el método anterior, permitiendo así abrir el detalle de la ruta elegida. Como se observa en la implementación del método, se comprueba si se ha seleccionado una ruta o no, si se ha seleccionado una ruta, se añade al intent que abre la actividad DetailsActivity y se inicia la actividad.

```
177     private final class ItemViewSelectedListener implements OnItemViewSelectedListener {
178         @Override
179         public void onItemSelected(Presenter.ViewHolder itemViewHolder, Object item,
180                                    RowPresenter.ViewHolder rowViewHolder, Row row) {
181             if (item instanceof Route) {
182                 mBackgroundURI = ((Route) item).getBackgroundImageURI();
183                 startBackgroundTimer();
184             }
185 
186         }
187     }
```

Ilustración 66: Implementación del evento onSelect

Cuando se selecciona una tarjeta se ejecuta el método mostrado en la Ilustración 66, el cual permite, si es una ruta, obtener la url correspondiente a la imagen de fondo para dicha ruta y ejecutar una tarea para efectuar el cambio de la imagen de fondo de la actividad.

Gestión de la imagen de fondo:

Actualmente existen diferentes librerías para la gestión de imágenes en Android, las dos más conocidas son Picasso y Glide, ambas disponen de una API muy similar. Estas librerías permiten cargar imágenes desde una url y se encargan de guardarlas en la caché del dispositivo para cargarlas más rápidamente en un futuro. En el caso de RunsMaps se ha utilizado la librería Glide simplemente porque en la documentación acerca de las aplicaciones Leanback se recomienda su uso. No se ha hecho una comparativa entre ambas librerías, porque al buscar información sobre ellas se vio que eran equivalentes, no presentaba una diferencia notable la una sobre la otra. A continuación, se muestra la implementación de la lógica para el cambio de la imagen de fondo.

```
189     private void startBackgroundTimer() {
190         if (null != mBackgroundTimer) {
191             mBackgroundTimer.cancel();
192         }
193         mBackgroundTimer = new Timer();
194         mBackgroundTimer.schedule(new UpdateBackgroundTask(), BACKGROUND_UPDATE_DELAY);
195     }
196
197     private class UpdateBackgroundTask extends TimerTask {
198
199         @Override
200         public void run() {
201             mHandler.post(new Runnable() {
202                 @Override
203                 public void run() {
204                     if (mBackgroundURI != null) {
205                         updateBackground(mBackgroundURI.toString());
206                     }
207                 }
208             });
209
210         }
211     }
212
213     protected void updateBackground(String uri) {
214         int width = mMetrics.widthPixels;
215         int height = mMetrics.heightPixels;
216         Glide.with(getActivity())
217                 .load(uri)
218                 .centerCrop()
219                 .error(mDefaultBackground)
220                 .into(new SimpleTarget<GlideDrawable>(width, height) {
221                     @Override
222                     public void onResourceReady(GlideDrawable resource,
223                                                 GlideAnimation<? super GlideDrawable>
224                                                         glideAnimation) {
225                         mBackgroundManager.setDrawable(resource);
226                     }
227                 });
228         mBackgroundTimer.cancel();
229     }
```

Ilustración 67: Lógica para cambiar la imagen de fondo

Según el orden de ejecución el proceso de cambio de la imagen de fondo sigue de la siguiente forma: En primer lugar se dispara el evento de selección de una tarjeta, si dicha tarjeta es una ruta se llama al método startBackgroundTimer(). Éste método comprueba si existe alguna tarea de cambio de imagen y si es así, la cancela. A continuación, crea un nuevo temporizador para que la imagen de fondo no se cambie inmediatamente y permita desplazarse por las tarjetas sin que el fondo este continuamente cambiando. Cuando el temporizador finaliza, se ejecuta el método updateBackgroud() el cual, utilizando la librería Glide, se encarga de obtener la imagen de fondo asociada a la ruta en cuestión y asignarla como fondo en la actividad principal, una vez asignada la imagen, cancela el temporizador.

3.2.3.3. Pantalla de detalles:

Ilustración 68: Pantalla de detalle de ruta

La actividad DetailsActivity muestra una pequeña descripción de la ruta seleccionada un botón para acceder al Street View de la ruta y un listado con rutas relacionadas.

En este caso el solamente se ha de destacar la interacción con el botón, ya que la gestión de imágenes es la misma que en la pantalla principal de la aplicación y la lista de rutas relacionadas va implícita en el fragmento del que extiende.

```java
public class DetailsActivity extends Activity {
    public static final String SHARED_ELEMENT_NAME = "hero";
    public static final String ROUTE = "Route";

    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);
    }

}
```

Ilustración 69: DetailsActivity.java

Como en el caso anterior esta actividad contiene una vista la cual contiene un fragmento, en la Ilustración 69, se muestra la implementación de la actividad. Como se puede observar solamente asigna el recurso de la vista a la actividad. En la Ilustración 70, se puede observar que la vista solamente contiene la referencia a un fragmento.

```xml
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/details_fragment"
    android:name="com.pc.jordi.runsmaps_app.ui.RouteDetailsFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.DetailsActivity"
    tools:deviceIds="tv" />
```

Ilustración 70: activity_details.xml

La vista hace referencia al fragmento details_fragment, como en el caso anterior, dicho fragmento es heredado de la clase android.support.v17.leanback.

Como se ha dicho al principio de este punto, el aspecto más destacado de esta pantalla es la asignación del evento onClick al botón que abre la pantalla para seguir la ruta. Como se muestra en la ilustración 71, además de añadir la imagen de fondo y la imagen del mapa en el detalle de la ruta, se crea una matriz de objetos en la cual se introducen todas las acciones que se necesiten en el detalle de la ruta, las acciones, en este ámbito se refieren a botones. En este caso solamente se necesita un botón para abrir la actividad que contiene Google Street View. Posteriormente se añade la matriz mediante el método setActionsAdapter(), de esta forma se ha creado una acción la cual desencadena un evento onClick que posteriormente se captura como se explica en la Ilustración 72.

```java
private void setupDetailsOverviewRow() {
    Log.d(TAG, "doInBackground: " + mSelectedRoute.toString());
    final DetailsOverviewRow row = new DetailsOverviewRow(mSelectedRoute);
    row.setImageDrawable(ContextCompat.getDrawable(getActivity(), R.drawable.default_background));
    int width = Utils.convertDpToPixel(getActivity()
            .getApplicationContext(), DETAIL_THUMB_WIDTH);
    int height = Utils.convertDpToPixel(getActivity()
            .getApplicationContext(), DETAIL_THUMB_HEIGHT);
    Glide.with(getActivity())
            .load(mSelectedRoute.getCardImageUrl())
            .centerCrop()
            .error(R.drawable.default_background)
            .into(new SimpleTarget<GlideDrawable>(width, height) {
                @Override
                public void onResourceReady(GlideDrawable resource,
                                            GlideAnimation<? super GlideDrawable>
                                                    glideAnimation) {
                    Log.d(TAG, "details overview card image url ready: " + resource);
                    row.setImageDrawable(resource);
                    mAdapter.notifyArrayItemRangeChanged(0, mAdapter.size());
                }
            });
    // Añadir acciones a la vista detalle
    SparseArrayObjectAdapter adapter = new SparseArrayObjectAdapter();
    adapter.set(ACTION_STREET_VIEW, new Action(ACTION_STREET_VIEW,
            "STREET VIEW",
            "Corre por sus calles"));
    row.setActionsAdapter(adapter);
    mAdapter.add(row);
}
```

Ilustración 71: método setupDetailsOverviewRow

El método siguiente, muestra cómo se asigna el evento onClick() a las acciones.

```java
private void setupDetailsOverviewRowPresenter() {
    // Set detail background and style.
    DetailsOverviewRowPresenter detailsPresenter =
            new DetailsOverviewRowPresenter(new DetailsDescriptionPresenter());
    detailsPresenter.setBackgroundColor(ContextCompat.getColor(getActivity(), R.color.selected_background));
    detailsPresenter.setStyleLarge(true);

    // Hook up transition element.
    detailsPresenter.setSharedElementEnterTransition(getActivity(),
            DetailsActivity.SHARED_ELEMENT_NAME);

    detailsPresenter.setOnActionClickedListener(new OnActionClickedListener() {
        @Override
        public void onActionClicked(Action action) {
            // implementación de los botones
            if (action.getId() == ACTION_STREET_VIEW) {
                Intent intent = new Intent(getActivity(), StreetViewActivity.class);
                intent.putExtra(DetailsActivity.ROUTE, mSelectedRoute);
                startActivity(intent);
            }
        }
    });
    mPresenterSelector.addClassPresenter(DetailsOverviewRow.class, detailsPresenter);
}
```

Ilustración 72: método setOnActionClickedListener()

Para poder capturar el evento desencadenado por las acciones es necesario sobrescribir el método onActionClicked(Action action) de la clase OnActionClickedListener() y pasar una instancia de ella como parámetro al método setOnActionClickedListener(). Para este

caso concreto, el método onActionClicked() comprueba que la acción es ACTION_STREET_VIEW y si es así, crea un intent añadiéndole como extra la ruta seleccionada e inicia la actividad que contiene Google Street View.

3.2.3.4. Pantalla de seguimiento de ruta:

Este apartado, aunque a priori puede parecer, al contrario, es el más interesante en cuanto al desarrollo y funcionalidad se refiere. En él, se ha desarrollado la funcionalidad para poder simular el entrenamiento al aire libre, a continuación, se explica con más detalle todo el proceso.

Google Maps api key:

Android studio facilita mucho la tarea que representa añadir una actividad que implemente la funcionalidad de google Maps, basta con crear la actividad desde el asistente desarrollado para tal efecto. Esto crea la clase de la actividad, la vista asociada y un nuevo recurso llamado google_maps_api_key.xml.

```
4
5       To get one, follow this link, follow the directions and press "Create" at the end:
6
7       https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_
8
9       You can also add your credentials to an existing key, using these values:
10
11      Package name:
12      46:96:73:5B:8E:63:7C:C8:68:EF:68:7E:8B:B0:A3:72:F2:A6:3C:AB
13
14      SHA-1 certificate fingerprint:
15      46:96:73:5B:8E:63:7C:C8:68:EF:68:7E:8B:B0:A3:72:F2:A6:3C:AB
16
17      Alternatively, follow the directions here:
18      https://developers.google.com/maps/documentation/android/start#get-key
19
20      Once you have your key (it starts with "AIza"), replace the "google_maps_key"
21      string in this file.
22      -->
23      <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">
24
25      </string>
26  </resources>
```

Ilustración 73: google_maps_api_key.xml

Como se observa en la ilustración 73, el archivo contiene una string llamada google_maps_key, en la que se tiene que guardar la clave de la API de google Maps con restricción para aplicaciones Android. Para obtener dicha clave basta con acceder a la url enmarcada el imagen anterior (no aparece completa por motivos de seguridad) y

seguir los pasos indicado en la web, una vez obtenida la clave, hay que guardarla en la string google_maps_key ya que posteriormente se usa para identificar la aplicación en el servidor. Con la clave, la vista y la actividad creada, solo quede implementar la funcionalidad deseada.

Creacion del StreetViewPanoramaFragment

```
<fragment android:id="@+id/streetviewpanorama"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    class="com.google.android.gms.maps.StreetViewPanoramaFragment"
    xmlns:android="http://schemas.android.com/apk/res/android" />
```

Ilustración 74: activity_street_view.xml

La ilustración anterior muestra el código de la actividad activity_street_view.xml la cual contiene un fragmento de StreetViewPanoramaFragment.

```
37  public class StreetViewActivity extends FragmentActivity
38      implements OnStreetViewPanoramaReadyCallback{
```

Ilustración 75: Clase StreetViewActivity.java

La ilustración 75 muestra la declaración de la clase StreetViewActivity, esta calse extiende de FragmentActivity para poder albergar un objeto fragment en su vista e implementa la interfaz OnStreetViewPanoramaReadyCallback. A continuación, se muestran los métodos sobrescritos de la clase fragmentActivity y los implementados de la interfaz.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_street_view);
    StreetViewPanoramaFragment streetViewPanoramaFragment =
            (StreetViewPanoramaFragment) getFragmentManager()
                    .findFragmentById(R.id.streetviewpanorama);
    streetViewPanoramaFragment.getStreetViewPanoramaAsync(this);
    mSelectedRoute = (Route) getIntent().getSerializableExtra(DetailsActivity.ROUTE);
    // Write a message to the database
    database = FirebaseDatabase.getInstance();
    refUsuario = database.getReference(USUARIO);
    refRuta = refUsuario.child("ruta");
    refDatosRuta = refRuta.child("datos");
    refPosRuta = refRuta.child("posicion");
    refRuta.child("id").setValue(mSelectedRoute.getId());
    refRuta.child("nombre").setValue(mSelectedRoute.getTitle());
    setupFirebaseListeners();
}
```

Ilustración 76: método onCreate()

Tal y como se puede observar en la ilustración 76, el método *onCreate*() sobrescribe al de la clase *FragmentActivity.* El primer paso, después de llamar al método *onCreate* de la clase padre, es añadir la vista con la se va a trabajar. Ahora, para poder empezar a trabajar con los panoramas, se crea un objeto *StreetViewPanoramaFragment* y se le asigna el fragmento contenido en la vista, a continuación, se le indica donde se encuentra el callback que se va a ejecutar cuando el panorama esté listo para su uso. En este caso, como se implementa la interfaz *OnStreetViewPanoramaReadyCallback* se pasa *this* como parámetro para que se ejecute el método implementado en esta clase. Al crear el fragmento, también se obtiene la ruta que se había pasado como extra en el intent de la pantalla de detalles de ruta y se guarda en la variable global mSelectedRoute. El resto de líneas sirven para actualizar en firebase la ruta que se ha seleccionada e inicializar los listeneres para obtener actualizaciones de estado de los campos de la base de datos. Este último punto, se verá con más profundidad más adelante.

```
@Override
protected void onDestroy(){
    super.onDestroy();
    resetDatosRuta();
}
```

Ilustración 77: Método onDestroy()

Se sobrescribe el método onDestroy para poder reiniciar los datos de la ruta en firebase, es decir, cuando se termina de correr, el valor de los campos velocidad, nivelResistencia, movCamara y giro pasan a ser 0.

```
@Override
public void onStreetViewPanoramaReady(StreetViewPanorama streetViewPanorama) {
    mStreetViewPanorama = streetViewPanorama;
    LatLng location = new LatLng(mSelectedRoute.getLat(), mSelectedRoute.getLng());
    float zoom = mSelectedRoute.getZoom();
    float tilt = mSelectedRoute.getTilt(); // Inclinación de la cámara
    float bearing =  mSelectedRoute.getBearing(); // Orientación de la cámara
    StreetViewPanoramaCamera camera = new StreetViewPanoramaCamera.Builder()
            .zoom(zoom)
            .bearing(bearing)
            .tilt(tilt)
            .build();
    mStreetViewPanorama.setPosition(location);
    mStreetViewPanorama.animateTo(camera, 3000);
    mPosicionAnterior = new Position(location, -1); //posicion1 = posicion inicial de la ruta
    mPosicionActual = new Position(location, -1); //posicion1 = posicion inicial de la ruta
}
```

Ilustración 78: implementación del método onStreetViewPanoramaReady

La Ilustración 78 muestra la implementación del *callaback onStreetViewPanoramaReady(StreetViewPanorama streetViewPanorama)* Cuando se ejecuta este método es porque el panorama está listo para empezar a trabajar con él. El panorama recibido como parámetro se guarda en la variable global *mStreetViewPanorama* para poder acceder a él desde cualquier método de la clase. Llegados a este punto solo queda obtener la ubicación del punto de partida de la ruta, para ello se crea un objeto *LatLng* con la latitud y la longitud guardadas en *mSelectedRoute* y se obtiene la inclinación y el zoom por defecto para dicho punto, es decir, se obtiene la información referente al punto de partida de la ruta seleccionada.

Para indicar la configuración de la cámara a un *StreetViewPanorama* es necesario crear un objeto *StreetViewPanoramaCamera*. En este caso la información del objeto *camera* es la obtenida del punto inicial de la ruta. Seguidamente se pasa la ubicación y el objeto *camera* al *StreetViewPanorama* y él mismo se encarga de crear la imagen en 360º y mostrarla por pantalla. Falta por ver para que sirven los objetos mPosicionAnterior y mPosicionActual. Como se ve en la ilustración son de tipo Position. Un objeto Position solamente alberga dos valores, un objeto *LatLng* con una ubicación y un valor en coma flotante con la elevación de dicha ubicación. En este punto no se sabe la elevación del

punto actual, por tanto, se guarda un -1 para indicar que es el panorama inicial y todavía no está disponible esa información.

Antes de profundizar con la navegación entre imágenes, es importante ver como interactúa la aplicación con la base de datos de firebase

Base de datos en tiempo real de Firebase

Para trabajar con bases de datos de firebase en un proyecto Android basta con seguir los pasos indicados por Android studio para integrar la librería en el proyecto. Una vez realizados estos pasos, ya se puede empezar a trabajar con la base de datos. En el caso concreto de RunsMaps solo se utiliza la base de datos desde el fragmento streetViewActivity. Es necesario acceder a los valores allí almacenados para poder avanzar entre fotogramas según la velocidad del corredor, poder realizar los giros que el corredor desee o incluso actualizar en nivel de resistencia que debe tener la bicicleta. Para todo esto, en el método onCreate, se configuran las referencias a los campos de la base de datos y se llama al método que se encarga de crear los listeners.

```
//Listener para los giros
refDatosRuta.child("giro").addValueEventListener(new ValueEventListener(){

    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        realizarGiro(dataSnapshot.getValue(Integer.class));
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Toast.makeText(getBaseContext(),
                       databaseError.getMessage().toString(),
                       Toast.LENGTH_SHORT).show();
    }
});
```

Ilustración 79: Listener para el campo giro

El funcionamiento del listener de la ilustración 79 y el de los que se muestran a continuación es muy similar, es decir, cuando se detecta un cambio en el campo en cuestión, se activa el listener y se ejecuta el método onDataChange(). A partir del DataSnapshot recibido como parámetro en el método, se puede obtener el valor

modificado y operar con él. En el caso del campo giro, se llama al método para realizar giros, en el caso del campo velocidad se llama al método avanzar para que se avance al siguiente panorama y para el campo movCamara se llama al método para que gire la cámara del panorama tantos grados como se pasen como parámetro.

```
//Listener para camara
refDatosRuta.child("movCamara").addValueEventListener(new ValueEventListener(){

    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        girarCamara(dataSnapshot.getValue(Integer.class));
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Toast.makeText(getBaseContext(),
                databaseError.getMessage().toString(),
                Toast.LENGTH_SHORT).show();
    }
});
```

Ilustración 80: Listener para el campo movCamera

```
//Listener para velocidad
refDatosRuta.child("velocidad").addValueEventListener(new ValueEventListener(){

    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        avanzar(dataSnapshot.getValue(Double.class));
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Toast.makeText(getBaseContext(),
                databaseError.getMessage().toString(),
                Toast.LENGTH_SHORT).show();
    }
});
```

Ilustración 81: Listener para el campo velocidad

Realizar giros:

Cuando el listener del campo “giro” detecta que se ha producido un cambio en la base de datos se ejecuta el método *realizarGiro(direccionGiro)*

```
private void realizarGiro(int direccion) {
    mDireccionGiro = direccion;
}
```

Ilustración 82: Método realizarGiro

Como se observa en la Ilustración 82, este método solamente guarda la dirección hacia la que el corredor quiere girar en la variable global mDireccionGiro, este valor será utilizado posterirome te para obtener el siguiente panoram. El valor de la dirección de giro puede ser:

Tipo de giro	**Valor**
Giro a la izquierda	-1
No desea girar	0
Giro a la derecha	1

Tabla 7: Tipos de giro

Cambio entre panoramas:

Un *StreetViewPanorama* contiene un objeto links el cual es, asimismo una matriz de objetos *StreetViewPanoramaLink*. Cada *StreetViewPanoramaLink* alberga entre otros valores el *bearing* asociado y el identificador del panorama. Para entender el funcionamiento de los panoramas es necesario explicar la función del *beraing*. El *bearing* es la dirección a la cual la cámara está apuntando, se especifica en grados y siempre se empieza a contar desde el norte, es decir, 0 grados quiere decir que la cámara está apuntando al norte, 90º al este, 180º al sur y 270º al oeste. Esto añade dificultad a la hora de obtener el siguiente panorama ya que el corredor nunca indicará el punto cardinal al que quiere ir, sino que deseará girar hacia la derecha o hacia la izquierda.

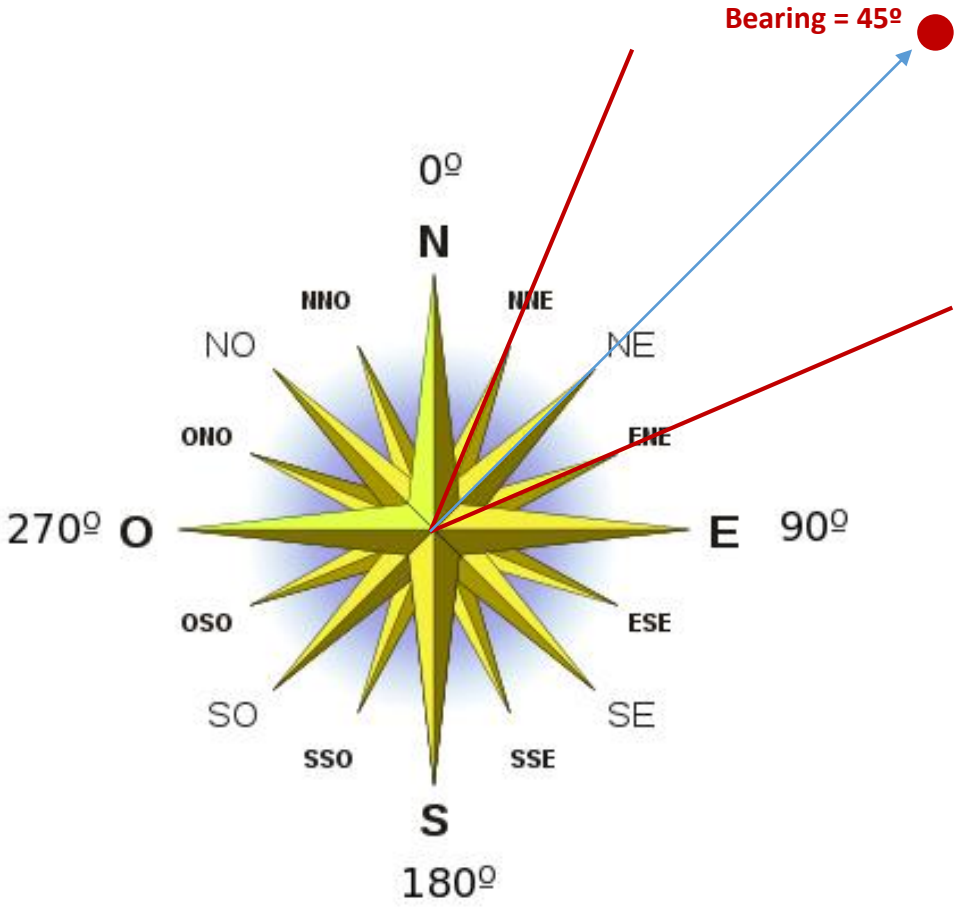


Ilustración 83: Puntos cardinales

Poniendo como ejemplo la Ilustración 83, se supone que el panorama actual tiene un bearing de 45º, y en la lista de panoramas adyacentes al actual, existe uno con un bearing de 20º y otro con un bearing de 80º. Si el usuario desea girar a la derecha entonces se debe de cargar el panorama cuyo bearing es de 80º. En el supuesto caso que los panoramos adyacentes tuvieran un bearing de 20º y de 10º y el corredor deseara girar a la izquierda, se cargaría el de más a la izquierda, es decir, el que cuyo bearing es de 10º. Según se ha desarrollado RunsMaps, existen dos casos a tener en cuenta. El primero de ellos es cuando el usuario desea girar a la izquierda i el bearing del panorama actual es inferiror a 90º, si se da este caso, el bearing del panorama siguiente podría ser

de por ejemplo de 350º y teniendo un bearing más alto, sería el panorama de la izquierda. El otro caso a tener en cuenta es el caso opuesto, es decir, que el panorama actual tenga un bearing superior a 270º y el siguiente tenga un bearing de 5º, aun teniendo un bearing menor, el panorama de 5º de bearing sería el de su derecha.

Para avanzar entre panoramas hay dos aspectos a tener en cuenta, la dirección de giro y la velocidad. Siempre que la velocidad de la base de datos sea mayor a cero el sistema intentará avanzar al siguiente panorama, para ello consulta la variable mDirecionGiro y según su valor obtiene el panorama correspondiente. Esto es posible gracias al listener que guarda la dirección de giro y al que obtiene la velocidad. Cuando se dispara el evento del campo velocidad se llama al método *avanzar()*.

A continuación, se muestran los métodos que implementan dicha funcionalidad.

```
private void avanzar(double velocidad) {
    if(mVelocidad != -1) {
        Toast.makeText(getBaseContext(),
                "Velocidad " + velocidad,
                Toast.LENGTH_SHORT).show();
        StreetViewPanoramaLocation location = mStreetViewPanorama.getLocation();
        //Guardamos la posición Anterior en el objeto mPosicionAnterior
        mPosicionAnterior.setElevation(mPosicionActual.getElevation());
        mPosicionAnterior.setLocation(mPosicionActual.getLocation());
        // Guardamos la posición actual en el objeto mPosicionActual
        mPosicionActual.setLocation(location.position);
        // La tarea asincrona se encarga de obtener la altitud de la posición pasada como
        // parámetro y guardarla en el atributo elevacion del objeto mPosicionActual
        // y de llamar al metodo para calcular la pendiente, la resistencia y actualizar el
        // valor en la bd
        new GetElevationTask().execute(location.position, location.position);

        String idPanorama = obtenerSiguienteImagen(mDireccionGiro);
        if (idPanorama != null && idPanorama != ""){
            mStreetViewPanorama.setPosition(idPanorama);
        }else{
            Toast.makeText(getBaseContext(), "No es posible realizar el giro",
                    Toast.LENGTH_SHORT).show();
        }
    }
    mVelocidad = velocidad;
}
```

Ilustración 84: Método avanzar()

Como se verá en el siguiente apartado, este método es el encargado de actualizar la posición actual y la posición anterior, pero por el momento, lo importante es la llamada al método obtenerSiguienteImagen(mDireccionGiro) y la posterior actualización de

posición del panorama. obtenerSiguienteImagen(mDireccionGiro) es un método sencillo que se encara de llamar al método adecuado según la dirección de giro.

```
private String obtenerSiguienteImagen(int direccionGiro){
    String idPanoram = null;
    switch (direccionGiro){
        case DERECHA:
            idPanoram = obtenerSiguientePanorama_Derecha();
            if (idPanoram == null || idPanoram == "")
                idPanoram = obtenerSiguientePanorama_Recto();
            break;
        case IZQUIERDA:
            idPanoram = obtenerSiguientePanorama_Izquierda();
            if (idPanoram == null || idPanoram == "")
                idPanoram = obtenerSiguientePanorama_Izquierda();
            break;
        default:
            idPanoram = obtenerSiguientePanorama_Recto();
    }
    return idPanoram;
}
```

Ilustración 85: Método obtenerSiguienteImagen()

```
public String obtenerSiguientePanorama_Derecha(){
    Log.d("Giros", "Derecha");
    try {
        double bearingPanoActual = mStreetViewPanorama.getPanoramaCamera().bearing;
        double bearingMax = bearingPanoActual + 90;
        double bearingMin = bearingPanoActual;
        if (bearingMax > 360) { bearingMax =  (45 - (360 - bearingPanoActual)); }
        StreetViewPanoramaLink[] links = mStreetViewPanorama.getLocation().links;
        double bearingEntrePano = 0;
        double bearingMayorEntrePano = 0;
        String idPanorma = "";
        for (int i = 0; i < links.length; i++) {
            bearingEntrePano = Math.abs(bearingPanoActual - links[i].bearing);
            if (bearingMin > bearingMax) {
                if (links[i].bearing >= bearingMin || links[i].bearing <= bearingMax){
                    if (bearingEntrePano > bearingMayorEntrePano) {
                        bearingMayorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }else {
                if ((links[i].bearing >= bearingMin) && (links[i].bearing <= bearingMax)) {
                    if (bearingEntrePano > bearingMayorEntrePano) {
                        bearingMayorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }
        }
        mIdUltimoPanorama = idPanorma;
        return idPanorma;
    }catch (NullPointerException ex){
        return mIdUltimoPanorama;
    }
}
```

Ilustración 86: Método obtenerPanorama_Derecha()

```
public String obtenerSiguientePanorama_Izquierda(){
    Log.d("Giros", "Izquierda");
    try {
        double bearingPanoActual = mStreetViewPanorama.getPanoramaCamera().bearing;
        double bearingMax = bearingPanoActual;
        double bearingMin = bearingPanoActual - 90;
        if (bearingMin < 0) { bearingMin = 360 - Math.abs(bearingMin); }
        StreetViewPanoramaLink[] links = mStreetViewPanorama.getLocation().links;
        double bearingEntrePano = 0;
        double bearingMayorEntrePano = 0;
        String idPanorma = "";
        for (int i = 0; i < links.length; i++) {
            bearingEntrePano = Math.abs(bearingPanoActual - links[i].bearing);
            if (bearingMin > bearingMax) {
                if (links[i].bearing >= bearingMin || links[i].bearing <= bearingMax){
                    if (bearingEntrePano > bearingMayorEntrePano) {
                        bearingMayorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }else {
                if ((links[i].bearing >= bearingMin) && (links[i].bearing <= bearingMax)) {
                    if (bearingEntrePano > bearingMayorEntrePano) {
                        bearingMayorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }
        }
        mIdUltimoPanorama = idPanorma;
        return idPanorma;
    }catch (NullPointerException ex){
        return mIdUltimoPanorama;
    }
}
```

Ilustración 87: Método obtenerPanorama_Izquierda()

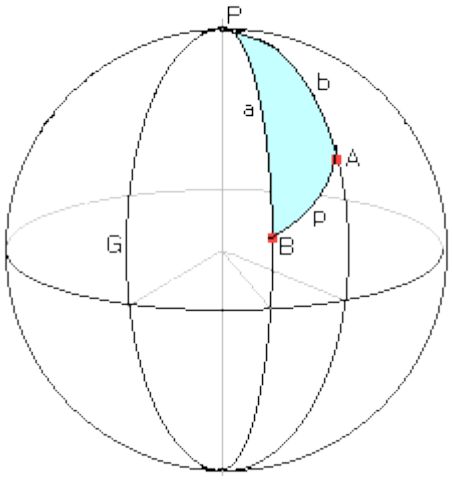
```
public String obtenerSiguientePanorama_Recto(){
    Log.d("Giros", "Recto");
    try {
        double bearing = mStreetViewPanorama.getPanoramaCamera().bearing;
        double bearingMax = bearing + 55;
        double bearingMin = bearing - 55;
        if (bearingMax > 360) { bearingMax =  (45 - (360 - bearing));}
        if (bearingMin < 0) { bearingMin = 360 - Math.abs(bearingMin);}
        StreetViewPanoramaLink[] links = mStreetViewPanorama.getLocation().links;
        double bearingPanoActual = mStreetViewPanorama.getPanoramaCamera().bearing;
        double bearingEntrePano = 0;
        double bearingMenorEntrePano = 361;
        String idPanorma = mIdUltimoPanorama;
        for (int i = 0; i < links.length; i++) {
            bearingEntrePano = Math.abs(bearingPanoActual - links[i].bearing);
            if (bearingMin > bearingMax) {
                if (links[i].bearing >= bearingMin || links[i].bearing <= bearingMax){
                    if (bearingEntrePano < bearingMenorEntrePano) {
                        bearingMenorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }else {
                if ((links[i].bearing >= bearingMin) && (links[i].bearing <= bearingMax)) {
                    if (bearingEntrePano < bearingMenorEntrePano) {
                        bearingMenorEntrePano = bearingEntrePano;
                        idPanorma = links[i].panoId;
                    }
                }
            }
        }
        mIdUltimoPanorama = idPanorma;
        return idPanorma;
    }catch (NullPointerException ex){
        return mIdUltimoPanorama;
    }
}
```

Ilustración 88: Metodo obtenerSiguientePanorama_Recto()

Cálculo del nivel de resistencia:

Como se ha comentado anteriormente se ha dividido la resistencia de la bici en 10 niveles. El nivel de resistencia irá variando según la ruta, para ello es necesario saber la pendiente que tiene el terreno. Google Maps no ofrece ningún servicio para esto, pero si ofrece un servicio para obtener la altitud de un punto en concreto. Con la información que se dispone se puede calcular la pendiente entre dos puntos. Aquí es donde entran en juego los objetos mPosicionActual y mPosicionAnterior, en mPosicionActual se guarda la ubicación y la elevación del panorama actual y en mPosicionAnterior la ubicación y la elevación del panorama anterior. Con esta información ya se puede calcular la pendiente entre dos puntos del mapa. Ahora el primer paso es calcular la

distancia entre los dos puntos. Para calcular dicha distancia se debe de aplicar el teorema del coseno.



Se desea calcular la distancia entre el punto A y el punto B.

Suponiendo que la imagen anterior es el globo terráqueo, los puntos A y B están definidos por unas coordenadas, es decir una latitud y una longitud. La latitud es la distancia que hay entre un punto dado y el ecuador; y la longitud es la distancia que hay entre un punto dado y el meridiano de Greenwich. Tal y como se puede ver en la imagen, entre los puntos A, B y el polo norte se forma un triángulo esférico. Teniendo esto en cuenta, el teorema del coseno dice lo siguiente:

> *Dado un triángulo ABC cualquiera, siendo α, β, γ, los ángulos, y a, b, c, los lados respectivamente opuestos a estos ángulos entonces:*
>
> $$c^2 = a^2 + b^2 - 2ab\cos\gamma$$

Volviendo al cálculo de la distancia entre el punto A y B del globo terráqueo. Se sabe que la latitud es la distancia de un punto respecto al ecuador y que el valor máximo es 90º para la mitad norte y -90º para la mitad sur, por tanto, se puede obtener la distancia que forma el lado del triángulo, también llamada colatitud, para ello a 90 que es el valor total se le tiene que restar la latitud del punto en cuestión.

Llegados a este punto ya se conocen dos lados del triángulo solo queda por averiguar el valor del ángulo contrario al lado formado por los puntos A y B. Para obtener dicho valor basta con restar las longitudes de los dos puntos. Con todos estos valores, ya se puede

aplicar el teorema del coseno, en la ilustración 89 se muestra la implementación de dicho teorema.

```
private double calcularDistanciaEntrePuntos(LatLng posicion1, LatLng posicion2){
    double lat1 = posicion1.latitude;
    double lon1 = posicion1.longitude;
    double lat2 = posicion2.latitude;
    double lon2 = posicion2.longitude;

    double colat1 = 90 - lat1;
    double colat2 = 90 - lat2;

    double angulo_A = Math.abs(lon1 - lon2);
    double distancia_a2 = 0;

    distancia_a2 = Math.pow(colat2, 2) + Math.pow(colat1, 2) - 2 * colat2 * colat1
            * Math.cos(angulo_A);
    double distancia_a = Math.sqrt(distancia_a2); // distancia en km
    return distancia_a * 1000; //distancia en metros
}
```

Ilustración 89: Método calcularDistanciaEntrePuntos()

Para seguir con el cálculo de la pendiente además de la distancia entre el panorama actual y el panorama anterior es necesario saber la elevación de cada uno. Para obtener dicha elevación se utiliza el servicio Google Maps Elevation. Android no dispone de API para trabajar en el servicio de Google Maps Elevation. Para poder obtener la elevación de un punto, se hace una petición mediante la URL al servidor y este devuelve un objeto json con toda la información. Para poder trabajar cómodamente con este servicio se ha desarrollado una clase que se encara de hacer la petición en segundo plano, y cuando se ha recibido el valor, entonces y solo entonces, se procede al cálculo de la pendiente. Como se ha dicho, el proceso se ejecuta fuera del hilo principal, por tanto, el programa no se detiene esperando la contestación del servidor, sino que, es el segundo hilo el que está esperando para realizar el cálculo de la pendiente y actualizar el valor en firebase.

```java
public static class Elevation{
    private static final String TAG = "Elevation";
    /**
     * Obtiene un array con toda la respuesta de la pagina web
     * @param urlSpec url a la que queremos hacer la peticion
     * @return array con la respuesta de la página web
     * @throws IOException
     */
    private byte[] getUrlBytes(String urlSpec) throws  IOException{
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();
        try{
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();
            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                throw new IOException(connection.getResponseMessage() + ": with " + urlSpec);
            }
            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while((bytesRead = in.read(buffer)) > 0){
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        }finally {
            connection.disconnect();
        }
    }

    private String getUrlString(String UrlSpec) throws IOException{
        return new String(getUrlBytes(UrlSpec));
    }

    public double buscarElevacion(double lat, double lng){
        try{
            String url = Uri.parse("https://maps.googleapis.com/maps/api/elevation/json")
                    .buildUpon()
                    .appendQueryParameter("locations", lat + "," + lng)
                    .appendQueryParameter("key", "AIzaSyDlAcnZEP49RTm-_4iR7nP1IwEjARDVJmI")
                    //.appendQueryParameter("sensor", "true")
                    .build().toString();
            String jsonString = getUrlString(url);
            Log.i(TAG, "Recived JSON: " + jsonString);
            GsonBuilder gsonBuilder = new GsonBuilder();
            gsonBuilder.registerTypeAdapter(Position.class, new PositionDeserializer());
            Gson gson = gsonBuilder.create();
            Position positionElevation = gson.fromJson(jsonString, Position.class);
            return positionElevation.getElevation();
        }catch (IOException ioe){
            Log.e(TAG, "Failed to fetch items", ioe);
            return 0.0;
        }
    }
}
```

Ilustración 90: Clase Elevation

La clase Elevation es la encargada de obtener los datos de la elevación desde el servidor de google. El método buscarElevacion() se encarga de hacer la petición y de decodificar el objeto json que obtiene como respuesta. Para decodificar el objeto json se utiliza la librería Gson. Esta librería funciona muy bien con objetos json básicos, pero no con el objeto complejo que devuelve el API REST del servicio elevation de google Maps. Para solucionar este problema, se tuvo que desarrollar una clase que se encargara de decodificar dicho objeto json.

```
public class PositionDeserializer implements JsonDeserializer<Position>{
    @Override
    public Position deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context)
            throws JsonParseException {

        JsonObject jsonObject = json.getAsJsonObject();
        JsonObject jsonPosition = jsonObject.get("results").getAsJsonArray().get(0).getAsJsonObject();
        return new Position(
          new LatLng(jsonPosition.getAsJsonObject().get("location").getAsJsonObject().get("lat").getAsDouble(),
                  jsonPosition.getAsJsonObject().get("location").getAsJsonObject().get("lng").getAsDouble()),
          jsonPosition.get("elevation").getAsDouble()
        );
    }
}
```

Ilustración 91:Clase positionDeserializer

Solamente falta por ver la tarea asíncrona que instancia la clase Elevation y se encarga de calcular la pendiente, el nivel de la resistencia y lo actualiza en firebase.

```
private class GetElevationTask extends AsyncTask<LatLng,Void,Double>{
    // Primer metodo en ejecutarse al llamar la tarea asincrona
    protected Double doInBackground(LatLng... params) {
        return new Elevation().buscarElevacion(params[0].latitude, params[0].longitude);
    }

    // Este metodo  será llamado cuando finalice la tarea asincrona.
    protected void onPostExecute(Double elevation){
        mPosicionActual.setElevation(elevation);
        double distancia = calcularDistanciaEntrePuntos(mPosicionActual.getLocation(),
                                                        mPosicionAnterior.getLocation());
        // si la elevacion anterior = -1 quiere decir que estamos en el primer panorma,
        // por tanto igualamos las dos altitudes para que no nos de errores
        if (mPosicionAnterior.getElevation() == -1 ) {
            mPosicionAnterior.setElevation(mPosicionActual.getElevation());
        }
        CalcPend calcPend = new CalcPend(mPosicionActual.getElevation(),
                                         mPosicionAnterior.getElevation(),
                                         distancia);
        double pendiente = calcularPendienteEntrePuntos(calcPend);

        // guardamos un entero redondeando a la baja el porcentaje de inclinación del tramo
        double nivelResistencia = pendiente / 10;
        refDatosRuta.child("nivelResistencia").setValue((int)nivelResistencia);
    }
}
```

Ilustración 92: clase asíncrona GetElevationTask()

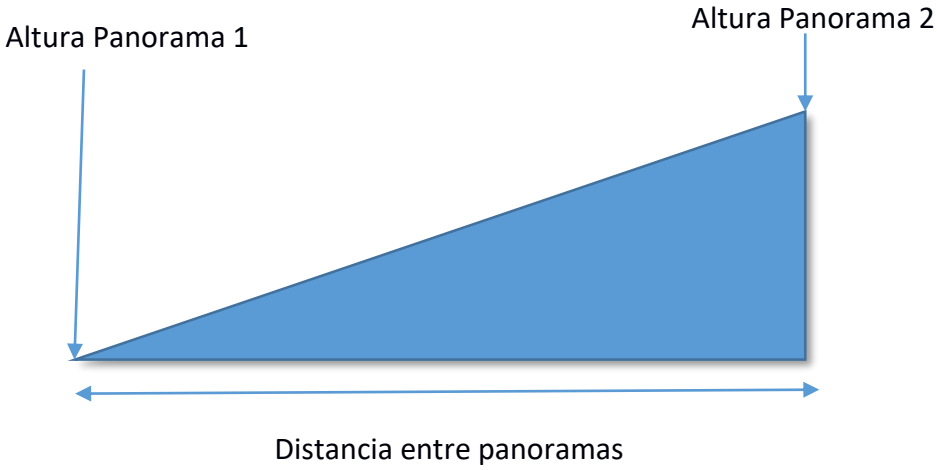
Como se puede observar en la ilustración 92, la clase extiende de AsynTask. El funcionamiento de esta clase es el siguiente, ejecuta el método doInBackground() en segundo plano, es decir, no lo ejecuta en el hilo principal, y cuando termina la ejecución de dicho método, automáticamente ejecuta el método onPostExecute.

Como se puede observar, en el método onPostExecute se calcula la distancia entre los dos panoramas, y con los valores de distancia y elevación se procede a calcular la pendiente. A continuación, se muestra cómo se calcula la pendiente que existe entre los dos panoramas.

```
private double calcularPendienteEntrePuntos(CalcPend calcPend) {

    double pendiente = Math.abs(calcPend.getElevacion1() - calcPend.getElevacion2())
                       / calcPend.getDistancia() * 100;
    return pendiente;
}
```

Ilustración 93: método calculoPendienteEntrePuntos

Como se puede ver, la fórmula de cálculo de la pendiente es muy sencilla.



4. Conclusiones

Este proyecto empezó con la idea de crear una aplicación para televisores Android en la que se pudieran ver imágenes de google Street View al mismo tiempo que se estaba corriendo en una bicicleta elíptica. Después de madurar la idea, se pensó que sería conveniente crear dos dispositivos, uno para gestionar todo lo relacionado con la bicicleta y otro para gestionar los giros dentro de la ruta.

Como ya se ha explicado a lo largo del presente documento, después de algunos problemas con la gestión del motor e inconvenientes al crear el circuito integrado del IMU. Se han podido superar todos los retos encontrados, algunos ocasionados por la inexperiencia y otros por falta de conocimiento en determinadas áreas. Pero dejando los inconvenientes encontrados de lado, el proyecto se ha terminado con éxito cumpliendo las expectativas que se marcaron en un principio, es decir, se ha desarrollado una aplicación Android en la que el corredor puede seleccionar un punto de partida y empezar a correr en el lugar del mundo que quiera, para ello solo se necesita conectar el DCB a la Domyos VE430 y configurar la red wifi doméstica. El siguiente paso sería ponerse la pulsera IMU para gestionar los giros durante la ruta y disfrutar del entrenamiento.

5. Bibliografía

1. 3. Longitud y latitud - La orientación en el planeta tierra [Internet]. [citado 19 de agosto de 2017]. Disponible en: https://sites.google.com/site/laorientacionenelplanetatierra/home/longitud-y-latitud

2. Analystprogrammer. Analista Programador Java: Distancia en Kilometros entre 2 puntos dada su latitud y longitud [Internet]. Analista Programador Java. 2012 [citado 24 de agosto de 2017]. Disponible en: http://analistaprogramador.blogspot.com.es/2012/03/distancia-en-kilometros-entre-2-puntos.html

3. Android Client-Server Using Sockets – Client Implementation [Internet]. [citado 15 de agosto de 2017]. Disponible en: http://androidsrc.net/android-client-server-using-sockets-client-implementation/

4. Android Client-Server Using Sockets - Server Implementation [Internet]. 2016 [citado 15 de agosto de 2017]. Disponible en: http://androidsrc.net/android-client-server-using-sockets-server-implementation/

5. Android SDK: Working with Picasso [Internet]. [citado 12 de agosto de 2017]. Disponible en: https://code.tutsplus.com/tutorials/android-sdk-working-with-picasso--cms-22149

6. Android Studio emulator does not come with Play Store for API 23 - Stack Overflow [Internet]. [citado 9 de agosto de 2017]. Disponible en: https://stackoverflow.com/questions/34291902/android-studio-emulator-does-not-come-with-play-store-for-api-23

7. androidtv-Leanback: :tv: Android TV Leanback Support Library sample app [Internet]. Google Samples; 2017 [citado 26 de julio de 2017]. Disponible en: https://github.com/googlesamples/androidtv-Leanback

8. Blanchon B. ArduinoJson: 🚘 C++ JSON library for IoT. Simple and efficient [Internet]. 2017 [citado 26 de julio de 2017]. Disponible en: https://github.com/bblanchon/ArduinoJson

9. designthemes. Arduino y las interrupciones | Tutoriales Arduino [Internet]. [citado 20 de marzo de 2017]. Disponible en: http://www.prometec.net/interrupciones/

10. designthemes. Arduino y WIFI ESP8266 | Tutoriales Arduino [Internet]. [citado 28 de septiembre de 2016]. Disponible en: http://www.prometec.net/arduino-wifi/

11. corochann. AsyncTask usage summary [Internet]. [citado 13 de agosto de 2017]. Disponible en: http://corochann.com/asynctask-usage-summary-341.html

12. Building an Android Project - Travis CI [Internet]. [citado 1 de agosto de 2017]. Disponible en: https://docs.travis-ci.com/user/languages/android/

13. Magnum Topografía, Ingeniería y Construcción. Calculo de Pendientes y Calculo de Cotas a partir de una Pendiente [Internet]. [citado 20 de agosto de 2017]. Disponible en: https://www.youtube.com/watch?v=0VI05oJOhUY

14. Cámara y vista | Google Maps Android API [Internet]. [citado 15 de agosto de 2017]. Disponible en: https://developers.google.com/maps/documentation/android-api/views?hl=es

15. Class Documentation — firebase-arduino 1.0 documentation [Internet]. [citado 16 de agosto de 2017]. Disponible en: http://firebase-arduino.readthedocs.io/en/latest/#firebase-arduinoclass_firebase_arduino_1a2ffac88673914b1127500fcaf4ad9187

16. Manzana ML. Como configurar MPU6050 Arduino | Listo para usar | • • • Muerde L Manzana • • • [Internet]. Como configurar MPU6050 Arduino | Listo para usar | • • • Muerde L Manzana • • •. 2015 [citado 13 de marzo de 2017]. Disponible en: http://muerdelapple.blogspot.com.es/2015/11/como-configurar-mpu-6050-arduino-listo.html

17. Cómo instalar MicroPython en un módulo ESP8266-12 [Internet]. [citado 16 de noviembre de 2016]. Disponible en: http://www.akirasan.net/como-instalar-micropython-esp8266-12/

18. Configurar apps con métodos de más de 64K | Android Studio [Internet]. [citado 17 de agosto de 2017]. Disponible en: https://developer.android.com/studio/build/multidex.html

19. Construction of BrowseFragment - Android TV app Tutorial 2 [Internet]. [citado 12 de agosto de 2017]. Disponible en: http://corochann.com/android-tv-application-hands-on-tutorial-2-85.html

20. Construya un sensor de temperatura listo para la nube con Arduino Uno y la Fundación IBM IoT, Parte 1: Construya el circuito y configure el ambiente [Internet]. 2014 [citado 5 de abril de 2017]. Disponible en: http://www.ibm.com/developerworks/ssa/cloud/library/cl-bluemix-arduino-iot1/index.html

21. Construya un sensor de temperatura listo para la nube con Arduino Uno y la Fundación IBM IoT, Parte 2 : Escriba el sketch y conéctese a la Fundación IBM IoT Quickstart [Internet]. 2014 [citado 5 de abril de 2017]. Disponible en: http://www.ibm.com/developerworks/ssa/cloud/library/cl-bluemix-arduino-iot2/index.html

22. Construye una app con Material Design utilizando la biblioteca Android Design Support [Internet]. [citado 30 de julio de 2016]. Disponible en: https://codelabs.developers.google.com/codelabs/material-design-style-

sp/index.html?index=..%2F..%2Findex#1

23. Create a 6DOF IMU with a gyro and accelerometer for (Arduino) multicopters. [Internet]. [citado 26 de abril de 2017]. Disponible en: http://www.brokking.net/imu.html

24. DetailsOverviewRowPresenter & FullWidthDetailsOverviewRowPresenter - Android TV app Tutorial 5 [Internet]. [citado 13 de agosto de 2017]. Disponible en: http://corochann.com/android-tv-application-hands-on-tutorial-5-138.html

25. Displaying a Location Address | Android Developers [Internet]. [citado 14 de agosto de 2017]. Disponible en: https://developer.android.com/training/location/display-address.html

26. eslint-config-google: ESLint shareable config for the Google JavaScript style guide [Internet]. Google; 2017 [citado 16 de agosto de 2017]. Disponible en: https://github.com/google/eslint-config-google

27. ESP8266. En: Wikipedia [Internet]. 2017 [citado 29 de agosto de 2017]. Disponible en: https://en.wikipedia.org/w/index.php?title=ESP8266&oldid=797646725

28. esp8266/Arduino [Internet]. GitHub. [citado 1 de mayo de 2017]. Disponible en: https://github.com/esp8266/Arduino

29. ESP8266 core for Arduino [Internet]. ESP8266 Community Forum; 2017 [citado 30 de agosto de 2017]. Disponible en: https://github.com/esp8266/Arduino

30. ESP8266 Network Configuration [Internet]. The Garage Lab. 2016 [citado 4 de abril de 2017]. Disponible en: http://blog.thegaragelab.com/esp8266-network-configuration/

31. ESP8266 - NURDspace [Internet]. [citado 28 de agosto de 2017]. Disponible en: https://nurdspace.nl/ESP8266

32. ESP8266 softAP + station channel configuration - ESP8266 Developer Zone [Internet]. [citado 1 de mayo de 2017]. Disponible en: http://bbs.espressif.com/viewtopic.php?f=10&t=324

33. ESP8266 Thing Hookup Guide - learn.sparkfun.com [Internet]. [citado 27 de marzo de 2017]. Disponible en: https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/using-the-arduino-addon

34. firebase-arduino: Arduino samples for Firebase [Internet]. Firebase; 2017 [citado 16 de agosto de 2017]. Disponible en: https://github.com/firebase/firebase-arduino

35. firebase-arduino: Arduino samples for Firebase [Internet]. Firebase; 2017 [citado 16 de agosto de 2017]. Disponible en: https://github.com/firebase/firebase-arduino

36. Fragmentos | Android Developers [Internet]. [citado 5 de agosto de 2017]. Disponible en: https://developer.android.com/guide/components/fragments.html

37. Fragmentos | Android Developers [Internet]. [citado 5 de agosto de 2017]. Disponible en: https://developer.android.com/guide/components/fragments.html

38. Funciones de la biblioteca de compatibilidad | Android Developers [Internet]. [citado 17 de agosto de 2017]. Disponible en: https://developer.android.com/topic/libraries/support-library/features.html#multidex

39. FW5GA8T.png (Imagen PNG, 562 × 295 píxeles) [Internet]. [citado 23 de mayo de 2017]. Disponible en: http://i.imgur.com/FW5GA8T.png

40. Getting Started with TV Apps | Android Developers [Internet]. [citado 1 de agosto de 2017]. Disponible en: https://developer.android.com/training/tv/start/start.html#media

41. Getting the Last Known Location | Android Developers [Internet]. [citado 14 de agosto de 2017]. Disponible en: https://developer.android.com/training/location/retrieve-current.html#play-services

42. Google [Internet]. [citado 16 de agosto de 2017]. Disponible en: https://www.google.es/?gws_rd=ssl

43. iltaen. HOMEDEVICE » The basic example for ESP8266-DEV with MOD-MPU6050 [Internet]. [citado 27 de marzo de 2017]. Disponible en: http://homedevice.pro/the-basic-example-for-esp8266-dev-with-mod-mpu6050/

44. How to Directly Program an Inexpensive ESP8266 WiFi Module | Hackaday [Internet]. [citado 22 de noviembre de 2016]. Disponible en: http://hackaday.com/2015/03/18/how-to-directly-program-an-inexpensive-esp8266-wifi-module/

45. How to get image in infoWindow on Google Maps with picasso - Android - Stack Overflow [Internet]. [citado 12 de agosto de 2017]. Disponible en: https://stackoverflow.com/questions/36335004/how-to-get-image-in-infowindow-on-google-maps-with-picasso-android

46. How to load Souliss on ESP8266. [Internet]. souliss. [citado 27 de marzo de 2017]. Disponible en: http://souliss.net/media/how-to-load-a-sketch-on-ESP/

47. Nilanchala. How to Use Picasso Image Loader Library in Android [Internet]. 2014 [citado 12 de agosto de 2017]. Disponible en: http://stacktips.com/tutorials/android/how-to-use-picasso-library-in-android

48. How to use Presenter and ViewHolder? - Android TV app Tutorial 3 [Internet]. [citado 12 de agosto de 2017]. Disponible en: http://corochann.com/android-tv-application-hands-on-tutorial-3-105.html

49. Indoors workout stepper with Raspberry Pi & Arduino IoT [Internet]. [citado 27 de julio de 2016]. Disponible en: http://www.instructables.com/id/Indoors-workout-stepper-

WITH-RASPBERRY-PI-ARDUINO-/

50. KATO D. INSTALLING GOOGLE PLAY SERVICES ON AN ANDROID STUDIO EMULATOR [INTERNET]. 2017 [CITADO 9 DE AGOSTO DE 2017]. DISPONIBLE EN: HTTPS://MEDIUM.COM/@DAI_SHI/INSTALLING-GOOGLE-PLAY-SERVICES-ON-AN-ANDROID-STUDIO-EMULATOR-FFFCEB2C28A1

51. INTRODUCTION | USER MANUAL FOR ESP-12E DEVKIT [INTERNET]. [CITADO 21 DE MAYO DE 2017]. DISPONIBLE EN: HTTPS://SMARTARDUINO.GITBOOKS.IO/USER-MANUAL-FOR-ESP-12E-DEVKIT/CONTENT/INDEX.HTML

52. INTRODUCTION - ANDROID TV APPLICATION HANDS ON TUTORIAL 1 [INTERNET]. [CITADO 12 DE AGOSTO DE 2017]. DISPONIBLE EN: HTTP://COROCHANN.COM/ANDROID-TV-APPLICATION-HANDS-ON-TUTORIAL-1-45.HTML

53. VADUVA L. IOT PROJECT - COMMUNICATION BETWEEN TWO ESP8266 - TALK WITH EACH OTHER [INTERNET]. GEEKSTIPS.COM. 2016 [CITADO 5 DE ABRIL DE 2017]. DISPONIBLE EN: HTTP://WWW.GEEKSTIPS.COM/TWO-ESP8266-COMMUNICATION-TALK-EACH-OTHER/

54. VADUVA L. IOT PROJECT - COMMUNICATION BETWEEN TWO ESP8266 - TALK WITH EACH OTHER [INTERNET]. GEEKSTIPS.COM. 2016 [CITADO 5 DE ABRIL DE 2017]. DISPONIBLE EN: HTTP://WWW.GEEKSTIPS.COM/TWO-ESP8266-COMMUNICATION-TALK-EACH-OTHER/

55. LAB09.PDF [INTERNET]. [CITADO 29 DE AGOSTO DE 2017]. DISPONIBLE EN: HTTP://WWW0.UNSL.EDU.AR/~INTERFASES/LABS/LAB09.PDF

56. G R. LIBRERÍA PARA LECTURA DE ÁNGULOS DE INCLINACIÓN DE UN MPU-6050 CON DMP [INTERNET]. NOTAS SOBRE ROBÓTICA, SISTEMAS OPERATIVOS Y PROGRAMACIÓN. 2014 [CITADO 14 DE ABRIL DE 2017]. DISPONIBLE EN: HTTPS://MINIBOTS.WORDPRESS.COM/2014/12/16/LIBRERIA-PARA-LECTURA-DE-ANGULOS-DE-INCLINACION-DE-UN-MPU-6050-CON-DMP/

57. J T. MAILBAG !! MPU6050 MODULE I2C DRIVER, INIT AND CONFIG [INTERNET]. [CITADO 27 DE MARZO DE 2017]. DISPONIBLE EN: HTTP://WWW.ESP8266-PROJECTS.COM/2015/12/MAILBAG-MPU6050-MODULE-I2C-DRIVER-INIT.HTML

58. MAPANET - CALCULO DE DISTANCIAS ENTRE DOS COORDENADAS GEOGRÁFICAS [INTERNET]. [CITADO 20 DE AGOSTO DE 2017]. DISPONIBLE EN: HTTPS://WWW.MAPANET.EU/RESOURCES/SCRIPT-DISTANCE.HTM

59. DESIGNTHEMES. MODELOS ESP8266 | TUTORIALES ARDUINO [INTERNET]. [CITADO 29 DE SEPTIEMBRE DE 2016]. DISPONIBLE EN: HTTP://WWW.PROMETEC.NET/MODELOS-ESP8266/

60. MPU-6000-DATASHEET1.PDF [INTERNET]. [CITADO 13 DE MARZO DE 2017]. DISPONIBLE EN: HTTPS://WWW.INVENSENSE.COM/WP-CONTENT/UPLOADS/2015/02/MPU-6000-DATASHEET1.PDF

61. MQTT V3.1 PROTOCOL SPECIFICATION [INTERNET]. [CITADO 5 DE ABRIL DE 2017]. DISPONIBLE EN: HTTP://PUBLIC.DHE.IBM.COM/SOFTWARE/DW/WEBSERVICES/WS-MQTT/MQTT-V3R1.HTML

62. Firebase. Node.js apps on Firebase Hosting Crash Course - Firecasts [Internet]. [citado 16 de agosto de 2017]. Disponible en: https://www.youtube.com/watch?v=LOeioOKUKI8

63. NODEMCU_DEVKIT_SCH.png (Imagen PNG, 2339 × 1653 píxeles) [Internet]. [citado 14 de mayo de 2017]. Disponible en: https://raw.githubusercontent.com/nodemcu/nodemcu-devkit/master/Documents/NODEMCU_DEVKIT_SCH.png

64. NodeMcu ESP8266 [Internet]. [citado 28 de agosto de 2017]. Disponible en: http://www.naylampmechatronics.com/inalambrico/153-nodemcu-esp8266.html

65. Node MCU Motor Shield | Post | Hackaday.io [Internet]. [citado 23 de agosto de 2017]. Disponible en: https://hackaday.io/project/8856-incubator-controller/log/29291-node-mcu-motor-shield

66. Picasso [Internet]. [citado 12 de agosto de 2017]. Disponible en: http://square.github.io/picasso/

67. PicassoBackgroundManager - Android TV app Tutorial 4 [Internet]. [citado 12 de agosto de 2017]. Disponible en: http://corochann.com/android-tv-application-hands-on-tutorial-4-122.html

68. Kravets I. PlatformIO: An open source ecosystem for IoT development [Internet]. PlatformIO. [citado 6 de mayo de 2017]. Disponible en: http://platformio.org

69. Programming GPIO on the ESP8266 with NodeMCU - Falafel Software Blog [Internet]. [citado 29 de noviembre de 2016]. Disponible en: http://blog.falafel.com/programming-gpio-on-the-esp8266-with-nodemcu/

70. Providing a Card View | Android Developers [Internet]. [citado 11 de agosto de 2017]. Disponible en: https://developer.android.com/training/tv/playback/card.html

71. ¿Qué Resistencia Poner a un LED? [Internet]. EducaChip. 2014 [citado 23 de mayo de 2017]. Disponible en: http://www.educachip.com/resistencia-led/

72. Recorrido · Valencia Ciudad del Running [Internet]. Valencia Ciudad del Running. [citado 29 de julio de 2016]. Disponible en: http://www.valenciaciudaddelrunning.com/medio/recorrido-medio-maraton/

73. roll and pitch angles ranges [Internet]. I2Cdevlib Forums. [citado 26 de abril de 2017]. Disponible en: https://www.i2cdevlib.com/forums/topic/24-roll-and-pitch-angles-ranges/

74. RS-MPU-6000A-00 - MPU-6000-Register-Map1.pdf [Internet]. [citado 25 de marzo de 2017]. Disponible en: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

75. Schema | ESP8266 RGB Strip Control [Internet]. [citado 23 de mayo de 2017].

Disponible en: http://www.esp8266color.com/schema/

76. Socket.IO — Native Socket.IO and Android [Internet]. [citado 15 de agosto de 2017]. Disponible en: https://socket.io/blog/native-socket-io-and-android/#

77. Start Integrating Google Sign-In into Your Android App [Internet]. Google Developers. [citado 30 de julio de 2016]. Disponible en: https://developers.google.com/identity/sign-in/android/start-integrating

78. Street View | Google Maps Android API [Internet]. Google Developers. [citado 4 de mayo de 2017]. Disponible en: https://developers.google.com/maps/documentation/android-api/streetview?hl=es

79. unicoos. Trigonometria- Teorema del seno BACHILLERATO [Internet]. [citado 25 de agosto de 2017]. Disponible en: https://www.youtube.com/watch?v=GY3wiZ8HUn0&t=314s

80. troubleshooting_translator_issues [Zotero Documentation] [Internet]. [citado 16 de noviembre de 2016]. Disponible en: https://www.zotero.org/support/troubleshooting_translator_issues

81. Tutorial Arduino: Acelerómetro - Giróscopo MPU6050 [Internet]. [citado 25 de marzo de 2017]. Disponible en: http://www.leantec.es/blog/48_Tutorial-Arduino--Aceler%C3%B3metro---Gir%C3%B3scopo-MP.html

82. Tutorial Arduino: Acelerómetro - Giróscopo MPU6050 [Internet]. [citado 25 de marzo de 2017]. Disponible en: http://www.leantec.es/blog/48_Tutorial-Arduino--Aceler%C3%B3metro---Gir%C3%B3scopo-MP.html

83. Tutorial de Arduino y MPU-6050 – robologs [Internet]. [citado 13 de marzo de 2017]. Disponible en: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

84. txb0108.pdf [Internet]. [citado 27 de julio de 2016]. Disponible en: http://www.ti.com.cn/cn/lit/ds/symlink/txb0108.pdf

85. Unidad de medición inercial. En: Wikipedia, la enciclopedia libre [Internet]. 2015 [citado 20 de abril de 2017]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_medici%C3%B3n_inercial&oldid=87408018

6. Anexos

6.1. Anexo 1. Enlaces para comprar los componentes.

NodeMCU:

http://www.ebay.es/itm/NodeMcu-V3-Lua-ESP8266-ESP12E-CH340-WiFi-Wireless-Module-Development-Board-W013-/201601344798?hash=item2ef060651e:g:y5wAAOSwSlBY3lu~

Assembled Feather HUZZAH w/ ESP8266 WiFi With Stacking Headers:

https://www.adafruit.com/product/3213

Espruino WiFi:

https://www.adafruit.com/product/3514

MicroPython pyboard v1.1 with headers:

https://store.micropython.org/#/products/PYBv1_1H

ESP8266:

http://www.ebay.es/itm/ESP8266-ESP-12E-MODULO-WIFI-ARDUINO-SENSOR-Enhanced-version-Serial-WIFI-W0011-/201703263001?hash=item2ef6738b19:g:pEYAAOSwqBJXWaSz

MPU-6050:

http://www.ebay.es/itm/MPU-6050-Modulo-Acelerometro-Giroscopio-3-Arduino-GYROSCOPE-ACCELEROMETR-M0017-/201712922501?hash=item2ef706ef85:g:r4IAAOSwe7BWvwoh

Resistencia de 10K:

https://lcsc.com/product-detail/Chip-Resistor-Surface-Mount_YAGEO_RC0603JR-0710KL_10K-103-5_C99198.html

Conensador 100nF:

https://lcsc.com/product-detail/Multilayer-Ceramic-Capacitors-MLCC-SMD-SMT_FH_1206CG104J500NT_100nF-104-5-50V_C46348.html

Led amarillo:

https://lcsc.com/product-detail/Light-Emitting-Diodes-LED_EVERLIGHT_67-21S-KK3C-H5050R2R52835Z15-2T_0-5W_C73543.html

Pulsador:

https://lcsc.com/product-detail/Tactile-Switches_ReliaPro_665mm-RHOS_6x6x5-Plastic-head_C21456.html