industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Diseño y desarrollo de unas plantillas inteligentes para monitorizar la pisada

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Óscar Hernández Castillo**
Director: Juan Antonio López Riquelme
Codirector: Nieves Pavón Pulido

Cartagena, a 10 de octubre de 2019

Universidad Politécnica de Cartagena

Índice

Capítulo 1

Introducción y Objetivos

1.1 Introducción.

En los últimos años el número de personas que practica carrera continua o "running" se ha incrementado considerablemente dado a que los beneficios que tiene este deporte son muchos como la disminución de contraer enfermedades tales como la obesidad, hipertensión o diabetes, además mejora nuestra salud ya que esta práctica hace que nuestro sistema inmunológico este mas activo, acelera nuestro metabolismo, disminuye el riesgo de que se formen coágulos de sangre, ayuda a combatir el estrés y la ansiedad y el marcarnos objetivos y cumplirlos junto con las mejoras físicas que nos aporta este deporte hacen que aumente también nuestra autoestima.

Por otra parte existen alteraciones en los pies o en la pisada las cuales pueden ser la causa de que aparezcan problemas o lesiones debido al tipo de apoyo del pie contra el suelo ya sea caminando o practicando carrera continua, estos problemas o lesiones se podrían detectar mediante los datos que nos puede aportar la baropodometría, que según su definición, es el estudio de la distribución de las cargas que soportan los pies tanto en descanso como durante la marcha [Baropodometría] y se pueden prevenir o curar una vez detectadas mediante métodos ortopédicos.

Con el auge del "running" también han aparecido una serie de inventos o "wereables" cuyo cometido es el de llevarlo puesto como si de una prenda más se tratara, la cual monitorea o cuantifica la actividad física o la actividad cotidiana y muestra los resultados a través de una aplicación de un terminal móvil o aplicación de un ordenador por medio de conectividad inalámbrica tal como el WIFI o el Bluetooth.

La conectividad inalámbrica Bluetooth está presente ya en la mayoría de los aparatos electrónicos, desde un reloj, una televisión, un móvil o incluso un electrodoméstico.

1.2 Objetivos.

Nuestros objetivos a cumplir serán los siguientes:

- Estudio y selección de la tecnología de sensorización más adecuada para monitorizar la pisada en tiempo real
- Diseño y desarrollo de los componentes hardware necesarios que componen la arquitectura
- Diseño y desarrollo de los componentes software que componen la arquitectura
- Integración de todos los componentes hardware y software para conseguir un sistema que permita registrar la presión efectuada en diferentes puntos del pie en tiempo real

A modo de pequeño resumen, en este trabajo se realizará el diseño y desarrollo de un sistema de monitorización de la pisada similar al funcionamiento de la baropodometría a través de la utilización de sensores de detección de fuerza colocados de manera estratégica en una plantilla para que con la presión ejercida por el pie sobre la plantilla un microcontrolador cuantifique todos los datos. Estos sensores estarán conectados a un microcontrolador que recogerá esa información a través de entradas analógicas de manera cableada y la envíe de manera inalámbrica por un sistema de comunicación Bluetooth a un terminal móvil inteligente, el cual tendrá una aplicación en la que se mostrará toda la información recogida por los sensores.

Para la correcta elección de la plantilla se realizarán diferentes pruebas sobre los distintos tipos de plantillas manteniendo la disposición de varios sensores para la obtención de los datos necesarios para realizar un correcto análisis de ellos y posteriormente su comparación para la elección de la plantilla más idónea para colocarlos.

Se realizará un acondicionamiento adecuado para adaptar las impedancias del microcontrolador con los sensores para evitar falseos en las medias tomadas.

El microcontrolador se programará en un lenguaje basado en lenguaje, el cual deberá soporta estructuras de control como bucles, condicionales, saltos, conversiones entre tipos de variables, operaciones matemáticas y trigonometría, interrupciones e interrupciones externas, comunicación por puerto serie e inserción de librerías entre otras funciones.

Capítulo 2

Estado del arte

2.1 Introducción.

Existen diferentes tipos de pisadas ya sea estática y en movimiento con las que se puede caracterizar a una persona. Por ello, en este capítulo abordaremos las pisadas más habituales, así como los problemas relacionados con la pisada y con los pies más comunes. También explicaremos las diferentes soluciones que se pueden encontrar en el mercado relacionadas con los tipos de pisadas y los problemas en la plata de los pies.

En este capítulo también se estudiarán los diferentes sensores que utilizaremos para captar la fuerza ejercida por la planta del pie, así como los diferentes sistemas de procesamiento y comunicación que podemos encontrar para nuestro proyecto y que se podrían utilizar en este trabajo para satisfacer los requisitos necesarios.

Finalmente, también veremos los diferentes sistemas operativos para dispositivos móviles inteligentes y el medio de comunicación que vamos a utilizar para nuestro proyecto.

2.2 Estudio de los tipos de pisada.

2.2.1 Tipos de pisadas no estáticas

Los tres tipos de pisadas más comunes existentes cuando caminamos o corremos son las siguientes:

- **Pie supinador**

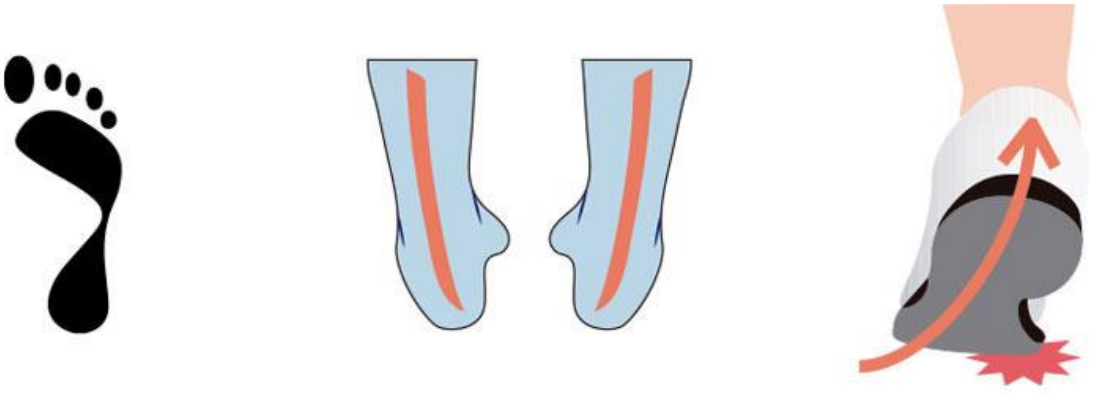


Figura 2.1 Tipo de pisada para un pie supinador [Pie supinador]

Es el tipo de pisada (ver Figura 2.1) menos común en la población. En concreto, se estima que solo un 10% de las personas posee este tipo de pisada, que se caracteriza por el apoyo de la parte más externa de la zona de la planta del pie, por lo que el tobillo se inclina hacia la parte exterior. Un error bastante frecuente es pensar que tenemos este tipo de pisada porque desgastamos la parte exterior de la zona del talón de nuestras zapatillas de deporte o zapatos, pero nada más lejos de la realidad. En realidad, con este tipo de pisada se desgastaría no sólo la parte trasera exterior, sino toda la parte exterior de la suela desde el talón hasta los dedos del pie.

Los problemas más comunes de este tipo de pisada son esguinces de tobillo debido a tener el tobillo inclinado hacia la parte externa, cualquier lesiones por giros o torceduras, fracturas por el exceso de uso de los músculos y a su mínima absorción al impacto, dolores en el lado interno de la tibia debido a que músculos, tendones y ligamentos se inflaman al esforzarse demasiado por la posición del pie, y, por último, puede producir fascitis plantar que ocasionan dolos e inflamación en el talón. Además, en casos aislados este tipo de pisada puede causar también dolores de espalda y dolores de rodilla.

- **Pie pronador**

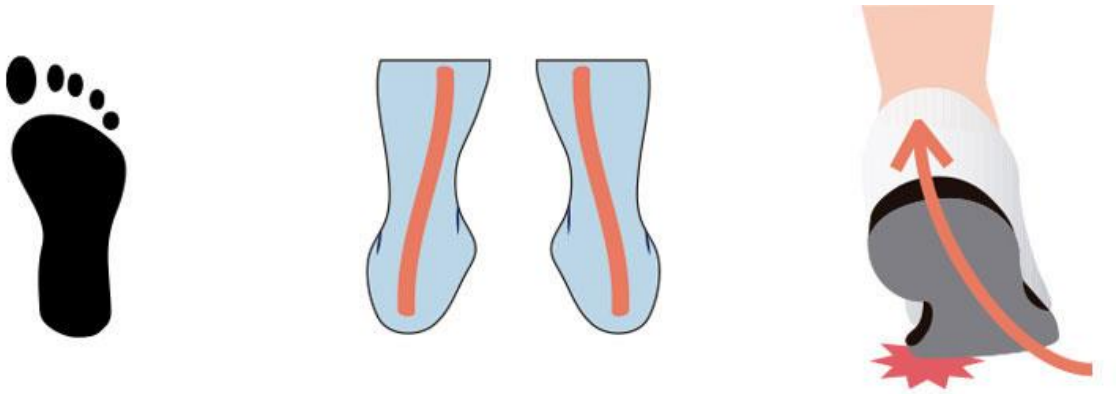


Figura 2.2 Tipo de pisada para un pie pronador [Pie pronador]

Se puede decir que es el tipo de pisada más común, ya que la mayoría de la población posee este tipo de pisada. Al contrario que el pie supinador el tobillo se inclina hacia la zona interna y esto produce que la pisada se incline hacia la parte interior teniendo la mayor zona de apoyo de nuestra planta del pie como se puede observar en la Figura 2.2. Existen casos aislados en los que la inclinación es superior a lo normal produciendo sobrepronación la cual debe ser valorada por un especialista y corregida.

Los problemas más frecuentes de este tipo de pisada son en el arco del pie y en los talones, apareciendo dolores en los mismos y llegando a alcanzar dolores en las pantorrillas, tobillos, rodillas y espalda.

- **Pie neutro**

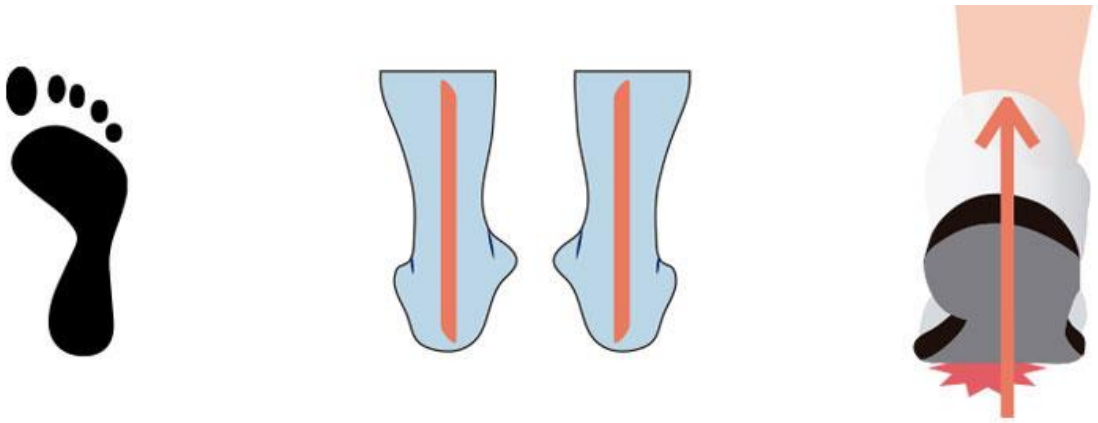


Figura 2.3 Tipo de pisada para un pie neutro [Pie neutro]

Es el conocido como pie universal o pie normal. En este caso, la pisada no ejerce ningún tipo de desplazamiento ni hacia dentro ni hacia fuera, ya que se sigue un desplazamiento lineal. Normalmente no se le acusan lesiones a este tipo de pisada.

2.2.2 Pisada estática

- **Pie Plano**

El pie plano es bastante común sobre todo en niños pequeños que tienen los tejidos de las articulaciones del pie flojos. A partir de los 2 ó 3 años esos tejidos se van tensando y aparece el arco del pie, pero existe la posibilidad de que este arco nunca se forme. Esto es debido a que el talón se desvía hacia fuera y da la sensación de que el pie está totalmente plano ya que la zona donde se encuentra el arco del pie toca el suelo, existen diferentes grados de pie plano (ver Figura 2.4).

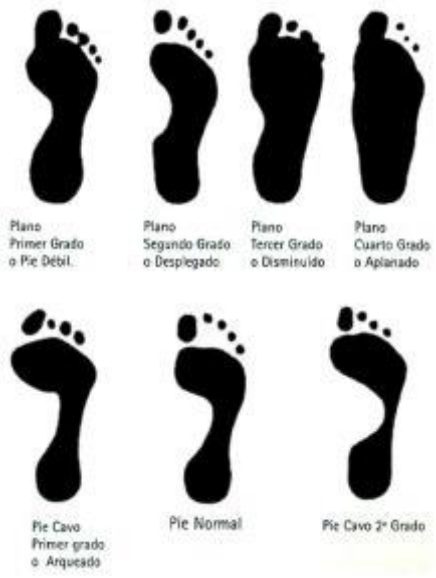


Figura 2.4 Tipos de pisada según puente [Puente]

- **Pie Cavo**

Es justo lo contrario al pie plano, es decir, es cuando el arco es más pronunciado de lo normal. Este tipo de pisada reparte su peso entre la zona del retropié y el antepié y como en el pie plano también existen diferentes grados de este tipo de pisada (véanse los grados en la Figura 2.6).

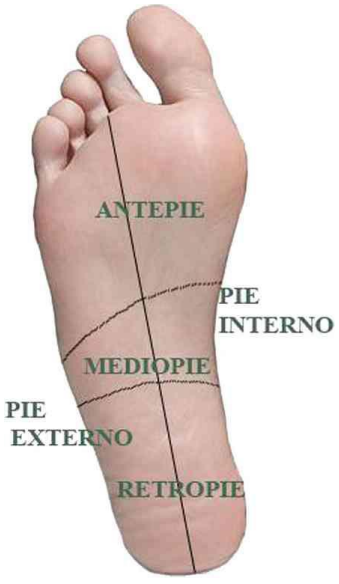


Figura 2.5 Zonas del pie [Zonas del pie]

Figura 2.6 Pisadas de Pie cavo [Pie Cavo]

2.3 Soluciones existentes

2.3.1 Baropodometría electrónica

La baropodometría cuya definición ya hemos dado en la introducción, es el estudio de la distribución de cargas tanto parado como en marcha que soportan los pies. Este método comenzó en los años ochenta y ha avanzado muchísimo debido al gran desarrollo tecnológico. Cada vez existen sensores más pequeños, más baratos y más fiables. Para estos tipos de estudios se han utilizados normalmente dos tipos de sensores: sensores piezoeléctricos y piezorresistivos. Los sensores piezoeléctricos, los cuales tienen la propiedad de polarizarse eléctricamente al ser sometidos a esfuerzos mecánicos de compresión, además de que tienen una gran estabilidad térmica, alta sensibilidad y respuesta lineal. El otro tipo de sensores utilizados son los piezorresistivos, su funcionamiento se basa en que varía su resistencia eléctrica cuando se someten a un esfuerzo mecánico o de compresión como en los piezoeléctricos, con la diferencia de que a estos sensores si les afecta la temperatura.

Normalmente este método de estudio se hace a través de una plataforma cuadrada o rectangular con miles de pequeños sensores repartidos en forma de matriz los cuales al soportar la presión ejercida por el pié varían su resistencia y van mandando datos a un ordenador en el que se pueden observar las mediciones obtenidas por los sensores en tiempo real.

Figura 2.7 Baropodómetro comercial [prtopedianeo]

También existen plantillas con registros de presión en unos pocos puntos y que almacenan toda la información en una memoria sólida, por lo que no te da la posibilidad de ver en tiempo real las mediciones que están dando los sensores.

Existe otro tipo de plantilla llamado PDM 240, la cual se puede observar en la Figura 2.8. Esta plantilla utiliza sensores piezorresistivos en unos puntos concretos y está conectada mediante cables a un dispositivo que se pone en la cintura, capta las variaciones de tensión de los sensores y los transmite mediante radiofrecuencia a un ordenador en tiempo real.

Figura 2.8 PDM 240 [PDM]

2.3.2 Plantillas inteligentes para el estudio de la biomecánica

Son plantillas sensorizadas con toma de datos en tiempo real que, pueden tomar 500 datos por segundo, poseen GPS para registrar la posición en todo momento, permiten medir el número de pasos y la actividad física realizada. Para la fabricación de estas plantillas se hace un escáner 3D de tu pie, por lo que cada plantilla es totalmente personalizada, (ver Figura 2.9.9

Estas plantillas están fabricadas de una lámina de plástico flexible con sensores resistivos y se comunica mediante BLE (Bluetooth Low Energy) con un dispositivo móvil para enviar los datos registrados.

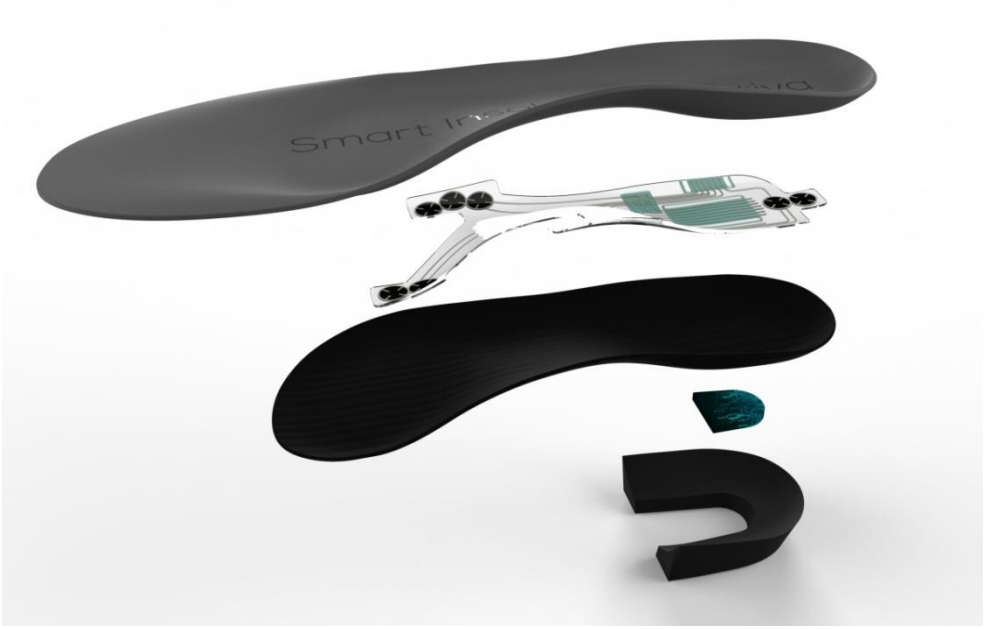


Figura 2.9 Plantillas sensorizadas para el estudio de la biomecánica "Smart Insole Podoactiva" [Smart]

2.3.3 Plantillas inteligentes para ulceras diabéticas

Estas plantillas están diseñadas con tecnología 3D y su función principal es prevenir la aparición de ulceras en los pies de las personas diabéticas, ya que por su mala cicatrización podría conllevar incluso a la amputación del miembro, (ver Figura 2.10).

Su modo de funcionamiento es medir la presión ejercida por el pie al caminar por medio de 21 sensores colocados de manera estratégica. Estas mediciones son mandadas a un sistema de cloud computing y esos datos son evaluados por un profesional de la salud.

Se conecta mediante Bluetooth al teléfono del paciente y cuenta con una aplicación que consulta los datos tres veces al día y avisa al usuario si detecta que existe alguna zona con mayor presión para que corrija su postura y se pueda evitar la aparición de la ulcera.

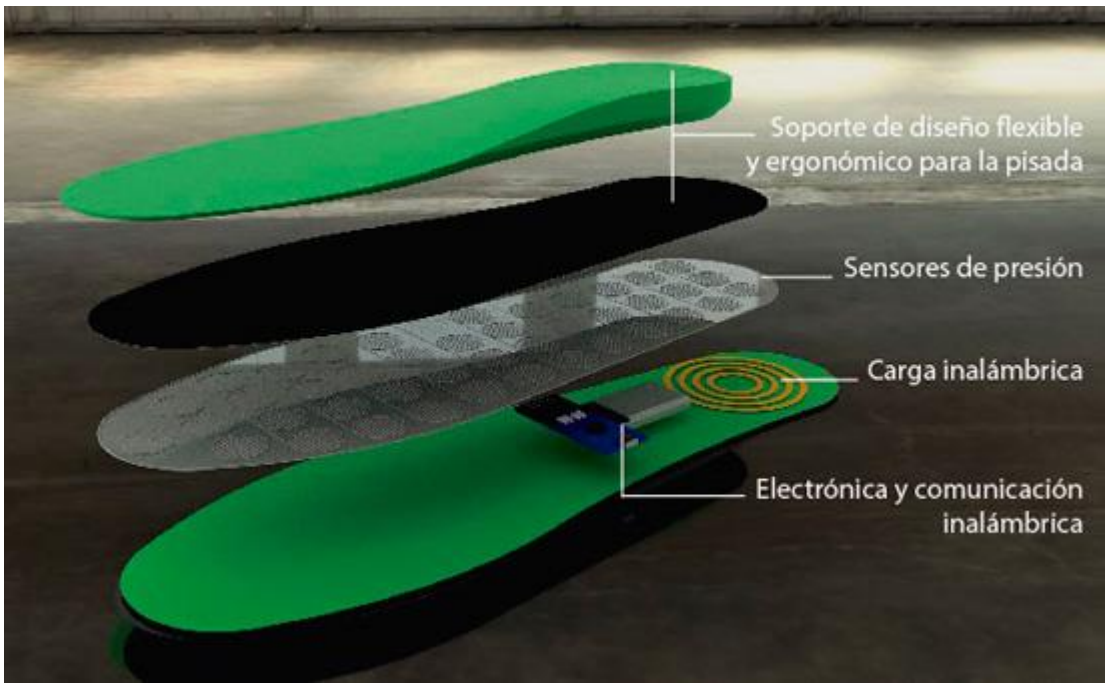


Figura 2.10 Plantillas inteligentes para ulceras diabéticas "Ebers" [Ulceras]

Para terminar el punto 2.3, vamos a realizar un breve resumen con las características más importantes de cada una de las plantillas que hemos analizado anteriormente según el tipo de monitorización de datos, registro de datos, comunicación y visualización de datos.

- **Monitorización de datos:**

En el baropodómetro de la Figura 2.7 la monitorización se realiza por una matriz cuadrada con miles de pequeños sensores piezorresistivos los cuales al posicionarte encima de dicha matriz van recopilando datos.

En el PDM 240 de la Figura 2.8 en la plantilla tan solo tiene sensores piezorresistivos en puntos concretos de la misma plantilla, en el estudio donde he estado buscando y recopilando datos no pone el número exacto de sensores que tiene dicha plantilla [Estudio PDM 240].

Las plantillas "Smart Insole Podoactiva" de la Figura 2.9 poseen 12 sensores de presión FSR (sensor de fuerza resistivo), acelerómetro, giróscopo y GPS. Estos sensores nos dan desde la posición en la que nos encontramos, como el tiempo que esta el pie en el aire durante la pisada, en número de pasos, la presión máxima y la presión instantánea de cada sensor y también la velocidad y aceleración que llevamos en cada momento portando estas plantillas.

En las plantillas "Ebers" de la Figura 2.10 poseen 21 sensores de tipo FSR (sensor de fuerza resistivo) los cuales nos detectan donde existe mayor presión para prevenir la aparición de ulceras en los pies, también poseen un sensor de presión y humedad.

- **Registro de datos**

Los datos del baropodómetro quedan registrados en un ordenador PC convencional para su posterior visualización, en cambio en las plantillas PDM 240 los datos quedan registrados en una tarjeta de memoria física para más tarde introducirla en un ordenador PC y poder visualizar los datos obtenidos. Las dos plantillas que nos quedan, las plantillas "Smart Insole Podoactiva" y las plantillas "Ebers" utilizan un dispositivo móvil para que envíe los datos a un sistema de "nube" o "cloud" en la que se almacenarán los datos obtenidos.

- **Comunicación**

La comunicación usada por el baropodómetro es de radiofrecuencia y se comunica como bien hemos dicho antes con un ordenador PC, al contrario que las PDM 240 que utilizan cables desde la propia plantilla hasta el modulo adaptador de la tarjeta de memoria en el que se guardan todos los datos para posteriormente. Dicha tarjeta hay que introducirla en un ordenador PC para visualizar la información registrada. En las plantillas "Smart Insole Podoactiva" y las plantillas "Ebers" la comunicación es inalámbrica por medio de BLE o Bluetooth de bajo consumo y ambas se comunican con un teléfono móvil inteligente que tiene una aplicación móvil o APK concreta para cada fabricante.

- **Visualización de datos**

Todos los sistemas descritos utilizan una aplicación concreta para su propia plantilla. En el caso del baropodómetro, al igual que las plantillas PDM 240, la aplicación será una aplicación de escritorio válida para un ordenador PC con un sistema operativo "Windows". En las plantillas "Smart Insole Podoactiva" y las "Ebers" la interfaz de usuario esta basada en una app concreta compatible con dispositivos móviles inteligentes.

2.4 Sensores de pisada

Después de analizar el estado del arte se ha considerado que la opción mas adecuada para poder monitorizar la pisada es utilizar sensores de tipo FSR (Sensores de fuerza resistivos), de los que se analizarán en esta sección diferentes alternativas.

2.4.1 Sensores de 26 mm

Los sensores de fuerza resistivos permiten detectar la presión física y la compresión. Estos sensores son resistencias que varían según la presión ejercida. La parte resistiva se compone de una películas delgada (100 µm de grosor) de material piezoresistivo, las capas superior e inferior son de una lámina muy fina de plástico, de manera que cuando no se le aplica ninguna carga el sensor presenta una gran impedancia mayor a 1MΩ. Estas son las medidas de tamaño del sensor, Figura 2.11.

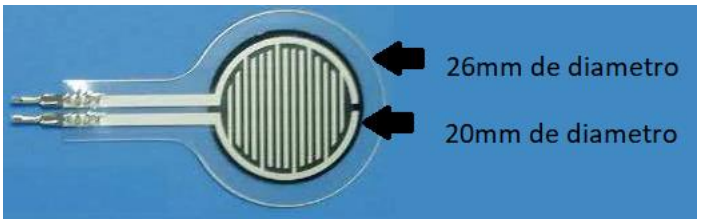


Figura 2.11 Sensor FSR de 26 mm [FSR 26]

En la Figura 2.12se puede observar la curva con la respuesta del sensor anterior al aplicar diferentes kilos de fuerza sobre el mismo

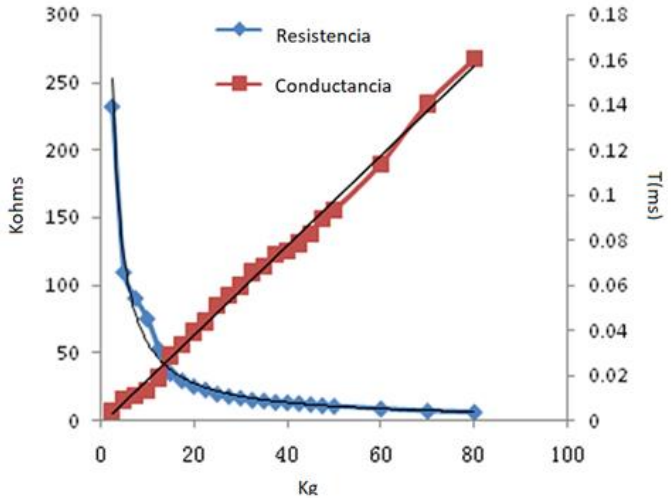


Figura 2.12 Curvas del sensor de 26 mm [Curvas 26 mm]

En la Figura 2.13 podemos ver el circuito y las recomendaciones que da el fabricante para realizar el acondicionamiento del sensor.

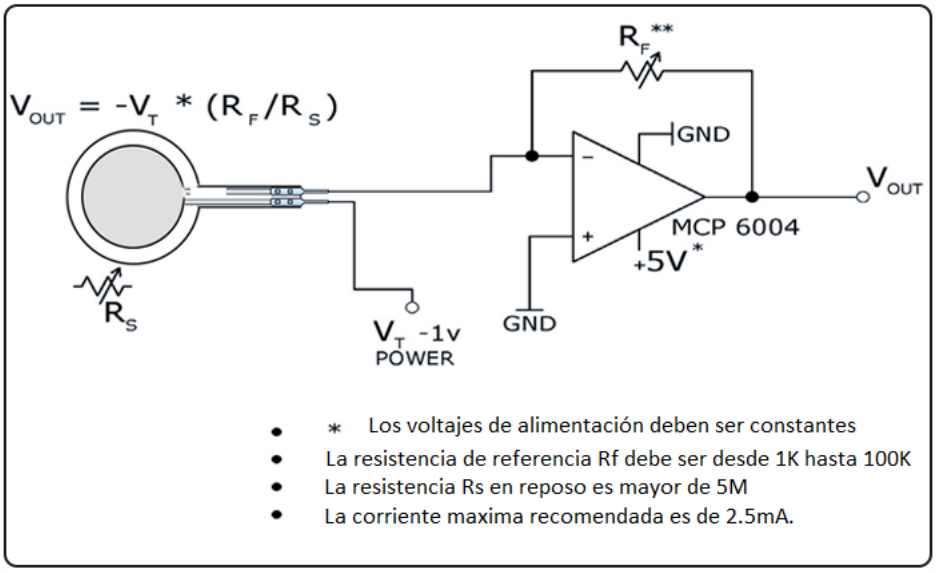


Figura 2.13 Circuito de acondicionamiento [Circuito 26 mm]

2.4.2 Sensores de 9 mm

Estos sensores son de la misma familia que los mencionados anteriormente con la única diferencia de que el diámetro de su zona de detección es menor. El sensor tiene unas medidas exactas de 9mm de diámetro en su parte exterior y de 7,5mm de diámetro en la zona de detección de la fuerza (ver Figura 2.14).

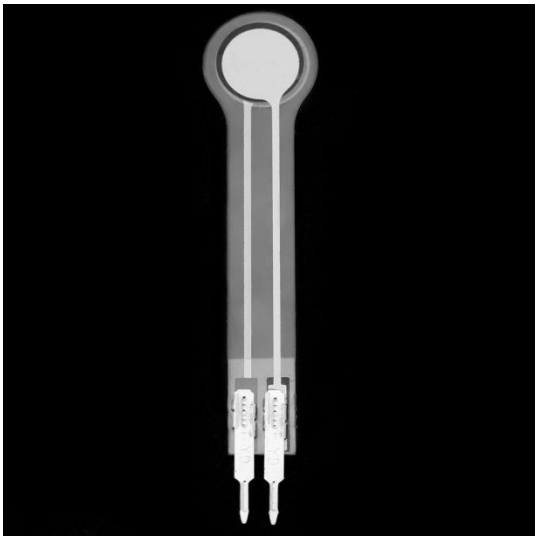


Figura 2.14 Sensor FSR de 9 mm [FSR 9]

Según el fabricante tiene un error de precisión del 2,5%, su vida útil es de un millón de usos, su resistencia inicial sin carga es mayor a 5MΩ sin carga y va disminuyendo cuanta mayor sea la fuerza aplicada hasta llegar a su límite máximo de 200N, su tiempo de

respuesta es de 1ms y su tiempo de restauración típico es de 15ms. Además podemos decir de los datos que nos da el fabricante que su tensión típica de alimentación es de 3,3V en corriente continua [FSR 9].

2.5 Sistemas de procesamiento y comunicación

En este apartado haremos un resumen de los diferentes microprocesadores que poseen las características adecuadas para nuestro proyecto, es decir conectividad, interfaces de entrada y salida y consumo energético.

2.5.1 BLE o Bluetooth Low Energy

Bluetooth Low Energy o BLE es una tecnología de comunicación inalámbrica cuya gran característica es que tiene un consumo de energía bastante reducido manteniendo un alcance muy similar al del Bluetooth convencional, lo que hace que sea un sistema muy usado en aplicaciones como wereables, aplicaciones para el cuidado de la salud y fitness así como en aplicaciones para el hogar y el día a día.

Los principales sistemas operativos, tanto de móviles inteligentes como de computadores PC son compatibles con este tipo de tecnología. BLE se comunica a una frecuencia de 2,4GHz y puede transferir archivos a una velocidad de 1 Mbps. Además al tener un bajo consumo se puede alimentar con pilas de botón y esto hace que el tamaño del dispositivo diseñado con esta tecnología pueda tener reducidas dimensiones.

2.5.2 Tarjeta Nano 2.0

El modelo Nano 2.0 de RedBearLab con chip nRF52832 de Nordic Semiconductor de la Figura 2.15, cuenta con un módulo Bluetooth Low Energy, 11 entradas/salidas y soporta una fuente de alimentación de entrada comprendida entre 3,3V y 13V. Esta tarjeta ofrece un procesador ARM cortex M4F de 32 bits y 64MHz. También es compatible con Bluetooth 4.2 y se puede programar en entorno Arduino gracias a las bibliotecas disponibles. Sus dimensiones son de 18,5 x 21mm y cuenta con la antena incluida dentro del propio circuito

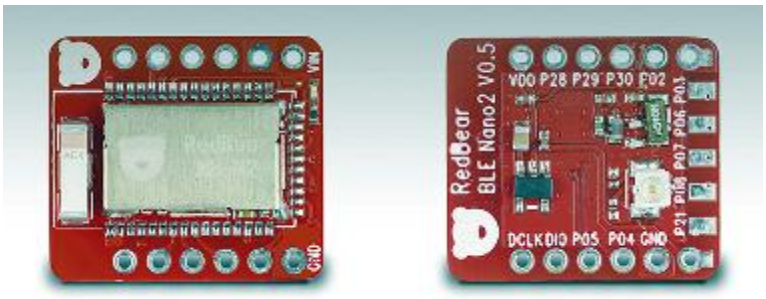


Figura 2.15 Nano 2.0 de Red Bear [Red Bear]

2.5.3 Tarjeta ESP32

El módulo ESP32 de la Figura 2.16 tiene un procesador de 32 bits de doble núcleo que trabaja a 160MHz pudiendo alcanzar los 240 MHz. Posee conectividad WIFI 802.11 b/g/n/e/i/n, WIFI Direct (P2P), P2P Discovery, P2P Group Owner mode y P2P Power Management, Bluetooth v4.2 BR/EDR and BLE y BLE Beacon. Tiene un total de 48 pines, los cuales 2 DAC de 8 bits, que sirven para convertir señales digitales de binario a señales analógicas y 18 de ellos son ADC de 12 bits. También dispone de 16 de esos pines con capacidad PWM, 2 UARTS, 2 buses I^2C , así como 4 buses SPI.

Esta tarjeta es compatible con la plataforma Arduino. Su tensión de alimentación es de 5V, ya que posee un regulador de tensión de 3,3Vde salida que es lo que necesita el microprocesador para funcionar.

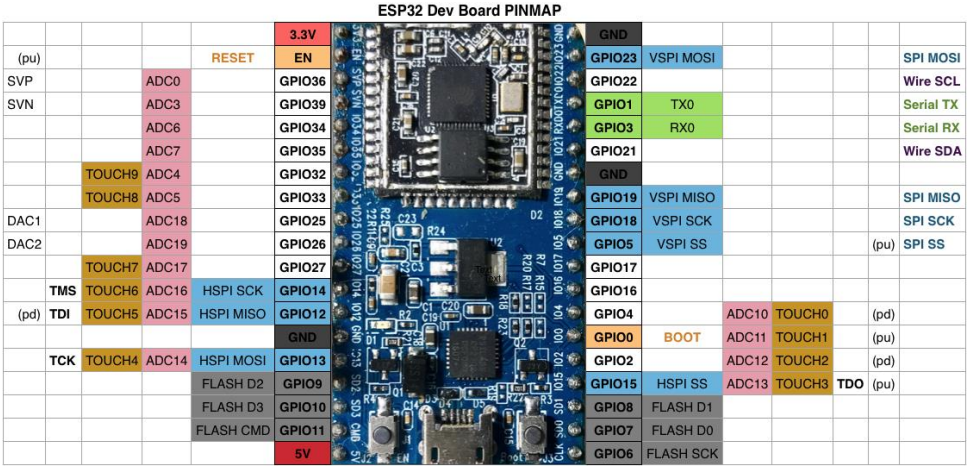


Figura 2.16 ESP32 Pin out [ESP32]

2.5.4 Tarjeta ESP8266

Sus principales características son una CPU de 32 bits a 80 MHz, RAM de instrucción de 64 KB, RAM de datos de 96 KB, capacidad de memoria externa flash QSPI de 512 KB a 4 MB (puede soportar hasta 16 MB), conectividad WIFI IEEE 802.11 b/g/n WIFI, soporta autenticación de redes mediante WEP y WPA/WPA2, 16 pines GPIO (Entradas/Salidas de propósito general), 4 puertos SPI, interfaz I^2C con DMA, pines dedicados a UART, y un conversor ADC de 10-bit de resolución.

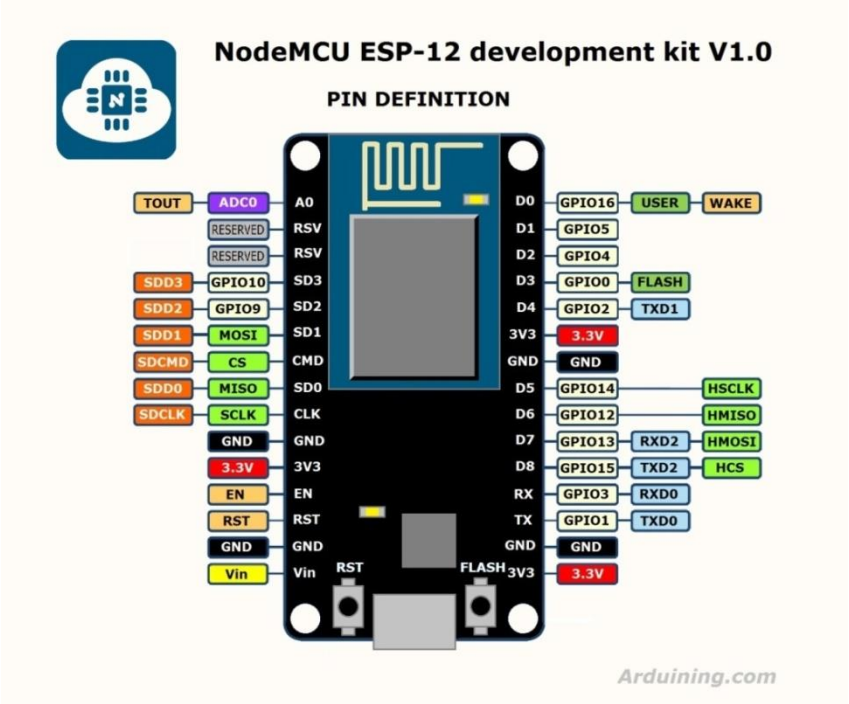


Figura 2.17 ESP8266 Pin out [ESP8266]

2.5.5 MSP 430

Existen diferentes modelos de estos microcontroladores fabricados por Texas Instruments como el de la Figura 2.18. Constan de una CPU de 16 bits, tiene hasta un máximo de 11 puertos que pueden ser entrada/salida de propósito general que pueden configurar con diferentes funcionalidades

La línea MSP430 se caracteriza por proporcionar placas de desarrollo basadas en microcontroladores de ultra bajo consumo y que incluyen diferentes funcionalidades: reloj de tiempo real, tarjeta de memoria microSD, comunicación WiFi y BLE, etc.

Estos microcontroladores vienen con software de desarrollo para su programación basado en C y C++..

Figura 2.18 MSP 430 [MSP430]

2.6 Sistemas operativos para dispositivos móviles inteligentes

Existe una gran variedad de sistemas operativos para dispositivos móviles inteligentes Android e IOS los mas extendidos, y que será los que analicemos en esta sección.

2.6.1 Android

Este sistema operativo está desarrollado por Google y está basado en software de código libre. Es el sistema operativo con mayor número de usuarios con un 74,45% a nivel mundial [smartphone], Este sistema operativo se convirtió en un referente cuando

empezaron a fabricar móviles con pantalla táctil, móviles inteligentes, tabletas y relojes inteligentes.

La primera versión que apareció de este sistema operativo fue en septiembre de 2008 y se llamaba ¨Apple Pie¨, desde entonces se han lanzado 17 versiones en total siendo la última la llamada ¨Q¨, que tiene la principal novedad de ser compatible con redes 5G.

2.6.2 IOS

Es el segundo sistema operativo más utilizado a nivel mundial en teléfonos móviles inteligentes y lo utilizan un 22,85% [smartphone]. Este sistema operativo fue creado en junio de 2007 con IOS 1.0, y hasta la actualidad han sacado 12 versiones, una por año. La última es IOS 12.

Este sistema operativo a diferencia de Android solo es utilizado en terminales de la marca Apple.

2.7 Conclusiones

Después de todo lo visto en este capítulo la conclusión que saco para hacer este proyecto es que vamos a utilizar un terminal móvil inteligente con sistema operativo Android debido a que la gran mayoría de usuarios utiliza este sistema operativo. Además, utilizaré el modulo ESP32 ya que es el microprocesador más potente de todos los analizados y el más versátil a nivel de conectividad.

Después de estudiar los sensores de pisada se llega a la conclusión de que será necesario realizar un estudio más detallado para seleccionar el elemento más optimo para el desarrollo.

Finalmente, en relación a la comunicación se ha seleccionado BLE, ya que gracias a su bajo consumo y al alcance de su señal muy similar al Bluetooth convencional es la mejor opción para este proyecto.

Capítulo 3

Descripción del sistema

3.1 Introducción.

En este capítulo describiremos todo lo relacionado con sensores, su acondicionamiento, pines utilizados en el microcontrolador, así como las tensiones de alimentación utilizadas. También hablaremos sobre la comunicación BLE, tanto del microcontrolador como de la parte del móvil inteligente, los diferentes perfiles y bibliotecas que se utilizarán para desarrollar parte de la arquitectura software del trabajo.

3.2 Sistema de sensorización.

Lo primero que debemos saber es que la tensión de alimentación va a ser de 5V en corriente continua para alimentar todo nuestro circuito, ya que todos los circuitos integrados que vamos a utilizar como el amplificador operacional, multiplexor, ESP32 y también los sensores aceptan esta tensión de alimentación. La configuración por la que he optado para este tipo de sensor FSR ha sido la de un divisor de tensión, en la que la resistencia fija es de 47 KΩ y la variable oscila entre más de 5 MΩ y 50 KΩ. Esto hace que cuando el sensor FSR esté en reposo en el divisor nos dé un valor de tensión de 0V y cuando el sensor tenga el máximo peso que soporta nos dará un máximo de 2,577V.

Como el microprocesador tiene una resolución máxima de 12 bits (4096 valores posibles), el valor máximo de 2,577V se corresponderá con un valor de salida del módulo ADC de aproximadamente 2111.

Para el acondicionamiento de los sensores, utilizaremos un amplificador operacional con una configuración de seguidor de voltaje. En esta configuración el circuito proporciona la misma tensión a la entrada que a la salida y su principal virtud es que se utiliza como adaptador de impedancias, (ver Figura 3.1).

Los amplificadores operacionales se caracterizan por su alta ganancia en lazo abierto, su alta impedancia de entrada, corriente de entrada muy pequeña y rechazo de modo común muy grande.

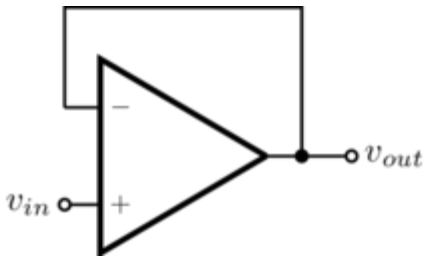


Figura 3.1 operacional en configuración de seguidor de tensión [Amplificador]

La salida de este seguidor de tensión irá directamente al microprocesador a la patilla D33 de nuestro microcontrolador. Este pin será una entrada analógica.

La entrada del amplificador irá conectada a la patilla SIG de nuestro multiplexor.Se utilizará un multiplexor analógico con el fin de conseguir una arquitectura hardware con las mínimas dimensiones posibles.

Llegados a este punto hablaremos sobre las características de un multiplexor. Un multiplexor analógico según su definición es un circuito con varias entradas y una única salida de datos. Estas entradas se seleccionan mediante patillas de selección y funcionan con combinaciones lógicas. Dependiendo de las combinaciones se van seleccionando las diferentes salidas. Por cada entrada existe n patillas de selección siendo el número de combinaciones diferentes 2^n.

El multiplexor que utilizaremos será el 74HC4067 con encapsulado SMD. Está integrado en una placa breakout board ya preparada para poder conectarla sin necesidad de utilizar ningún tipo de acondicionamiento. El nombre completo de la placa que vamos a utilizar es CD74HC4067.

Este multiplexor posee 16 entradas analógicas (C0, C1,..., C15) y una única salida analógica (SIG) con 4 patillas de selección (S0, S1, S2 y S3) y como hemos explicado

anteriormente si utilizamos la fórmula para las diferentes combinaciones sería 2^4 que es 16 (ver Figura 3.2.). Para que el multiplexor funcione la patilla enable debe tener un cero lógico, en caso contrario el multiplexor no estará habilitado.

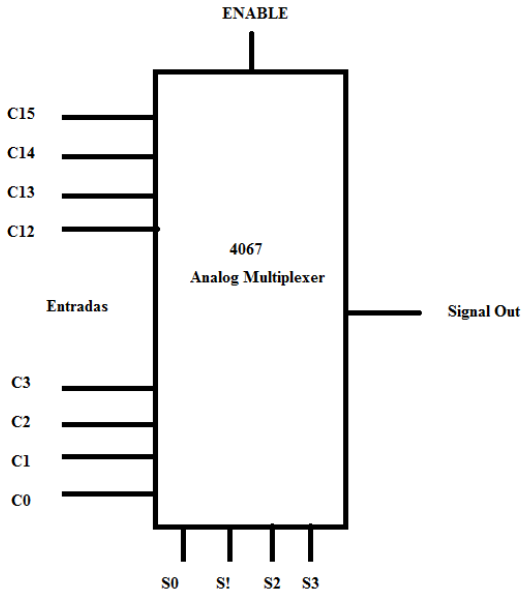


Figura 3.2 Multiplexor 4067 [Multiplexor]

La tensión de funcionamiento va desde 2 a 10 voltios siendo la tensión típica de 5 voltios. El tiempo de propagación es de máximo 9 nanosegundos lo cual nos da un total de $1{,}1111*10^8$ muestras en un segundo.

En nuestro circuito tan sólo utilizaremos las entradas de la C0 a la C11, que irán conectadas a los divisores de tensión. Las patillas de selección irán de la siguiente forma, la patilla S0 que es el bit de menor peso irá conectada a la patilla D14, S1 se conectará a D27, S2 irá conectada a D26 y por último el bit de mayor peso irá conectado a D25. Los pines anteriormente mencionados del microprocesador se utilizarán como salidas digitales.

Los sensores se montarán sobre una lámina de acetato transparente con forma de plantilla y lo fijaremos a la lámina de acetato con cinta adhesiva transparente.

3.3 Sistema de procesamiento y comunicación.

En el punto anterior hemos mencionado los pines del microcontrolador para leer los sensores y para controlar la entrada del multiplexor que queremos leer, además de mencionar el voltaje de alimentación y el voltaje teórico máximo que legará a nuestro microcontrolador.

El siguiente punto que debemos de mencionar es el de la comunicación del módulo ESP32 por medio de BLE. Para ello debemos empezar explicando un poco el protocolo que utiliza este sistema Bluetooth de baja energía.

Existen dos categorías que van incluidas en este protocolo y son el GAP o perfil de acceso genérico y el GATT o perfil de atributo genérico.

Las grandes diferencias existentes entre GAP y GATT son que el GAP se centra en los dispositivos con los que se puede comunicar o que están conectados o habilitados para conectarse y el GATT se dedica más a los paquetes o datos que enviamos entre los dispositivos.

El GAP podemos dividirlo en dos partes, una de difusión y otra de conexión.

En la parte de difusión debe de existir un emisor y un observador. El emisor es el que transmite los paquetes o datos y el observador los recibe.

En la parte de conexión también existen dos tipos de dispositivos, el periférico y el central. El periférico es el que anuncia a un central que se puede establecer una conexión con él. Los periféricos una vez se conectan no transmiten datos a otros dispositivos centrales y se mantienen conectados al dispositivo central. El dispositivo central es el que inicia la conexión con un periférico, y puede conectarse a diferentes dispositivos periféricos. Cuando el Dispositivo central quiere conectarse a un periférico le envía un paquete de datos de solicitud de conexión, de modo que si este dispositivo periférico acepta esta solicitud se establece una conexión.

Para el perfil GATT existen los clientes y los servidores. El rol de cliente está asociado tanto a lectura como a escritura de información, mientras que el rol de servidor se establece para almacenar información

3.4 Sistema de visualización

Para este proyecto he elegido una aplicación móvil ya que el 67% de la población mundial posee un dispositivo móvil. Los dispositivos móviles ya son suficientemente potentes como para poder utilizarlo como sistema de procesamiento de datos, de almacenamiento y de visualización.

Para la programación de la aplicación para dispositivos móviles existen cantidad de cursos online que te enseñan desde lo más básico hasta algo un poco más avanzado para programar en Android.

También existen diferentes entornos de desarrollo para desarrollar apps para Android. A continuación hablaremos mas en detalle del utilizado en este trabajo.

3.4.1 Android Studio

En este punto explicaremos el funcionamiento del programa que vamos a utilizar.

Primero crearemos un nuevo proyecto (ver Figura 3.3):.

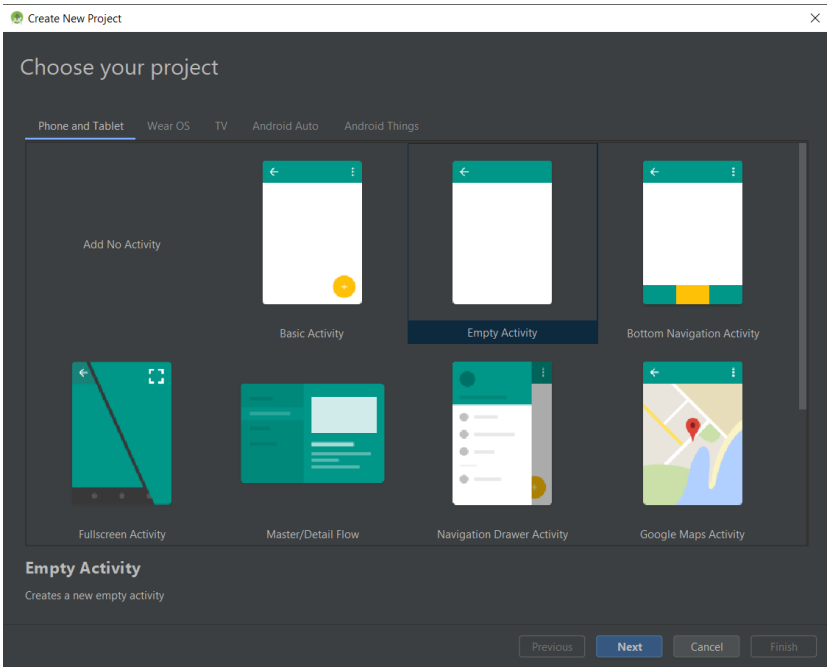


Figura 3.3 Ventana de creación de un nuevo proyecto con Android Studio

Ponemos nombre al proyecto, seleccionamos la ubicación, seleccionamos el lenguaje que vamos a utilizar y la versión mínima en la que se podrá instalar la aplicación. En este caso he seleccionado Android 5.1 que es compatible con el 80% de los dispositivos (ver Figura 3.4).

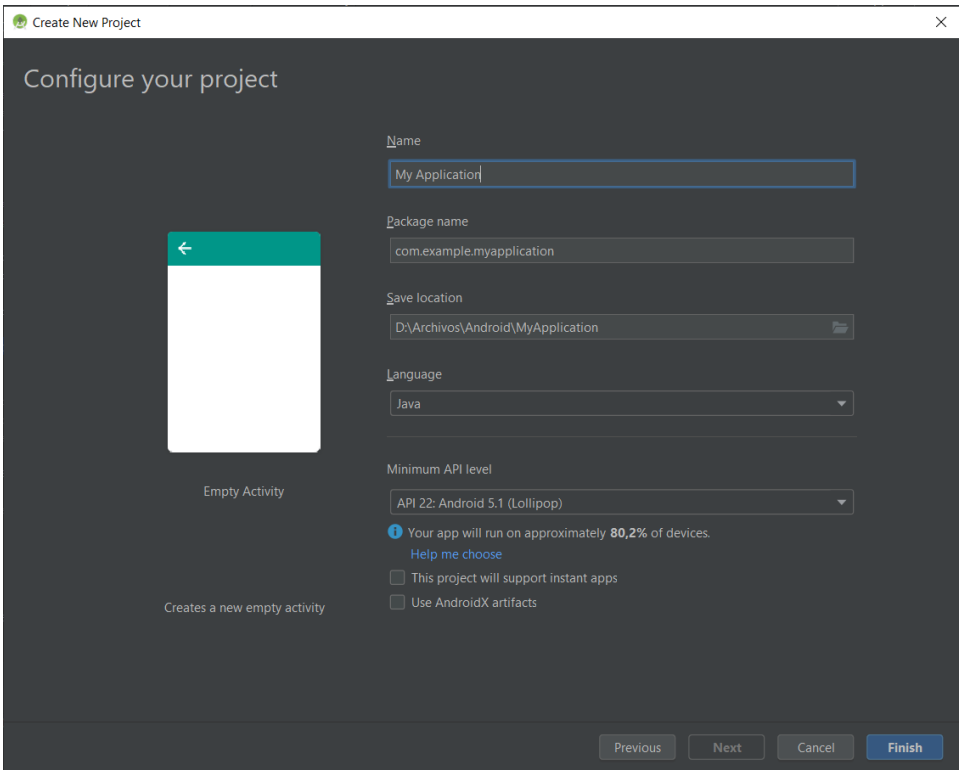


Figura 3.4 Configuraciones del proyecto con Android Studio

Una vez finalizada la configuración inicial nos mostrará una serie de carpetas en las que están todas las clases y directorios de nuestra aplicación (ver Figura 3.5).

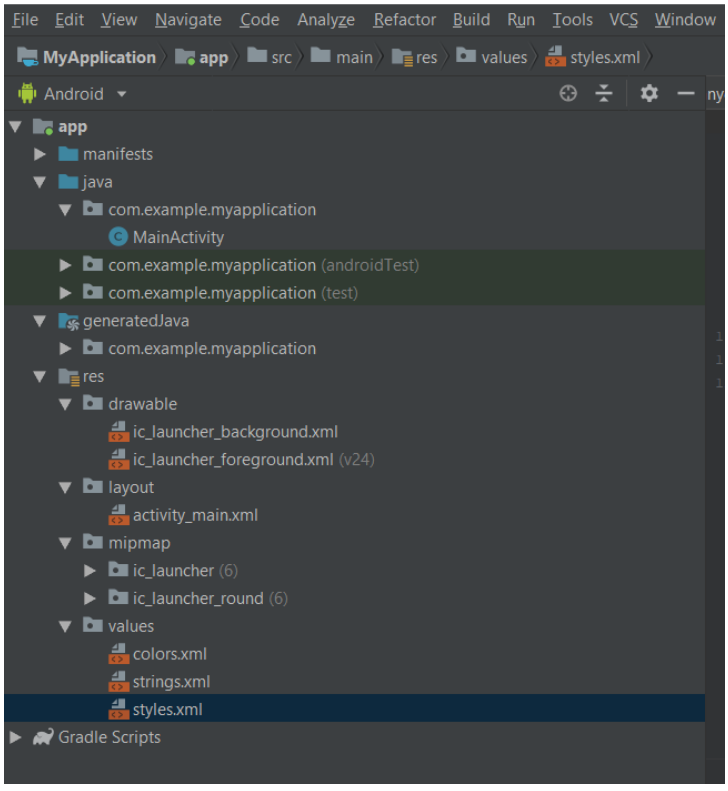


Figura 3.5 Carpetas del nuevo proyecto

Lo más significativo que podemos explicar de este nuevo proyecto es lo siguiente:

En la carpeta res/layout/activity_main.xml se incluye lo que se mostrará en la pantalla principal de nuestra aplicación móvil (ver Figura 3.6).

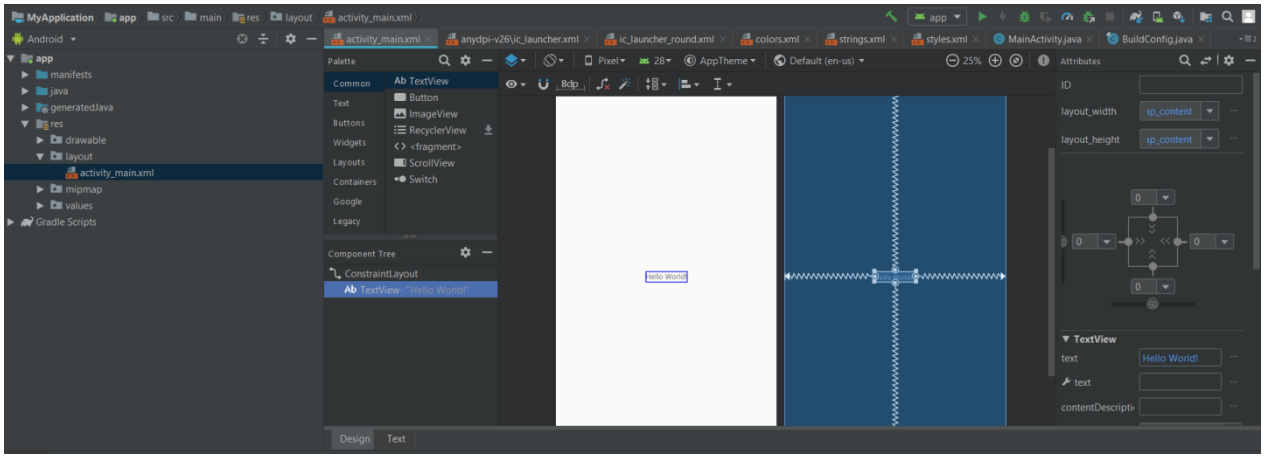


Figura 3.6 Pantalla activity_main manual

Como se puede observar, sólo hemos puesto un texto en el centro de la pantalla en el que pone "Hello World".

Existen dos formas de colocar los diferentes textos, botones y demás componentes que necesitamos utilizar. La primera es arrastrando desde la paleta hasta la pantalla y colocando de forma manual donde queramos y la segunda es programando de manera manual en el archivo activity_main.xml (ver Figura 3.7).

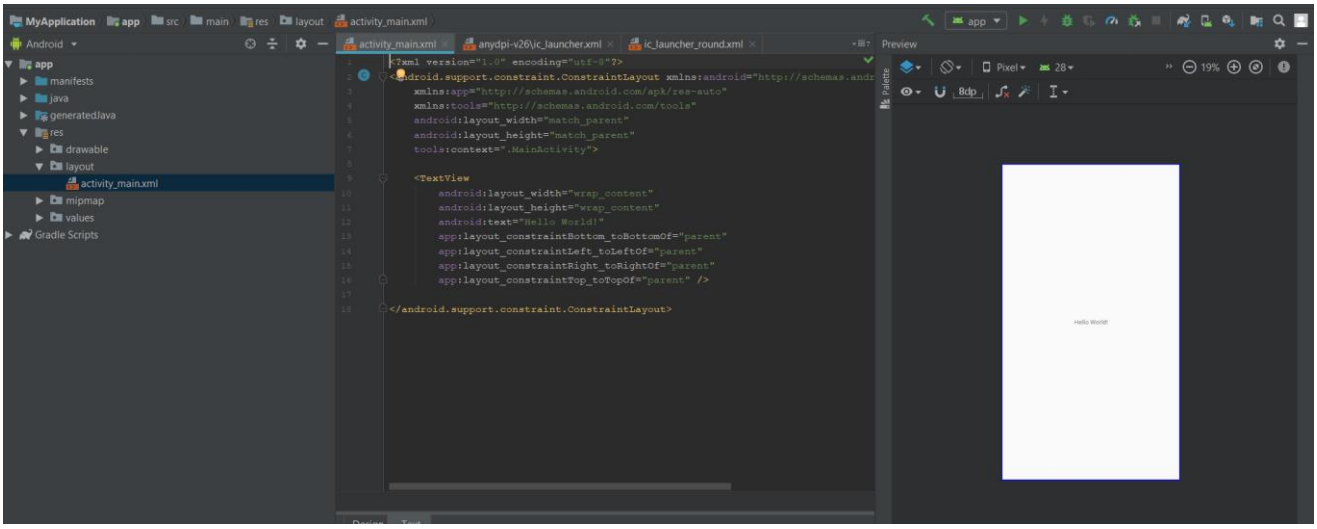


Figura 3.7 Pantalla activity_main.xml

En la paleta podemos encontrar diferentes atributos para utilizar como textos, botones, tablas, etc. (ver Figura 3.8).

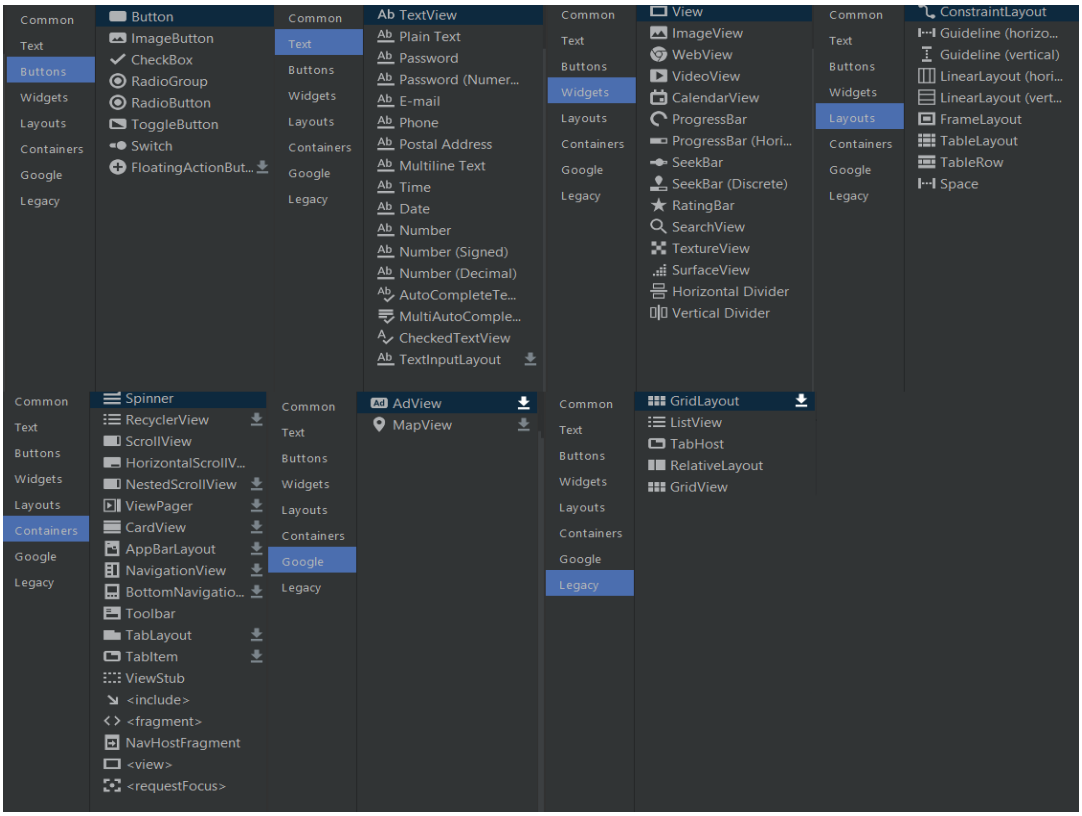


Figura 3.8 Paleta de componentes que ofrece Android Studio

En la misma ubicación, res/… , es donde debemos de crear las diferentes pantallas que componen una aplicación compuesta por varias pantallas.

En esta misma ubicación encontramos, res/mipmap, que es donde estarán las imágenes que utilizaremos, también encontraremos, res/values, que es donde pondremos los diferentes valores varios como colores, strings, etc (ver Figura 3.9).

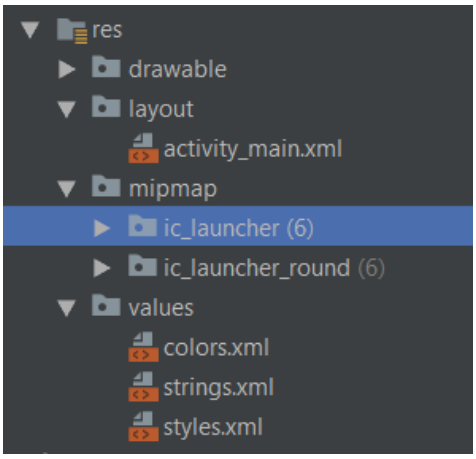


Figura 3.9 Values

En la ubicación, java/com.example.myapplication/MainActivity, podemos decir que es como "un bucle para inicialización". En concreto dentro de él están todas las clases, inicialización de bucles, de textos, etc. (ver Figura 3.10).

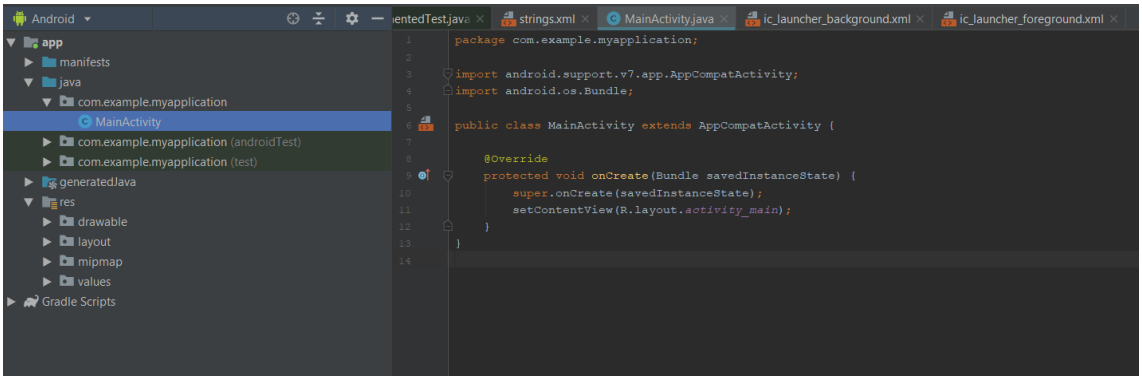


Figura 3.10 MainActivity

En el Capítulo 4 explicaremos la aplicación móvil que he creado para este proyecto, siguiendo los pasos iniciales descritos en esta sección.

3.4.2 Integración BLE

En este punto se explicarán las diferentes características del BLE en la aplicación móvil como perfiles, bibliotecas y métodos.

3.4.2.1 Perfiles

Los perfiles utilizados para BLE en Android son al igual que los utilizados para BLE en nuestro módulo ESP32. El perfil GATT, es el encargado de los datos, y el GAP, es el encargado de los dispositivos. Estos dos perfiles son prácticamente iguales a lo descrito en el punto 3.3 de este capítulo. La novedad en Android es que utiliza un UUID o un ID único universal, que se encarga de reconocer y distinguir un dispositivo u objeto dentro de un sistema o de diferentes contextos y como su propio nombre indica es único y universal.

3.4.2.2 Bibliotecas

En este punto explicaremos las bibliotecas que voy a utilizar en la aplicación móvil para este proyecto. Se pueden dividir en tres tipos, bibliotecas de Android, bibliotecas Java y bibliotecas propias. Comenzaremos con las bibliotecas que hemos utilizado de Android y para qué sirve cada una de ellas.

Primero explicaremos las utilizadas para el Bluetooth.

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothSocket;

import android.bluetooth.BluetoothDevice;

La primera de ellas ***"BluetoothAdapter"*** es la encargada de buscar un dispositivo, con ***"BluetoothDevice"*** lo que se hará será devolver los dispositivos vinculados y con ***"BluetoothSocket"*** escucharemos las solicitudes de conexión de los dispositivos.

La siguiente biblioteca será la content, que será la encargada de compartir contenido entre la aplicación.

import android.content.Intent;

import android.content.Context;

En ***"content.Intent"*** entregará mensajes en los diferentes componentes de la aplicación y en ***"content.Context"*** nos servirá para extraer información global de nuestra aplicación.

La biblioteca os es la que transmite mensajes entre procesos del dispositivo.

import android.os.Bundle;

En la librería "***os.Bundle"*** devuelve un paquete vacío.

La biblioteca support como su propio nombre indica es la encargada de dar soporte a nuestro código.

import android.support.v7.app.AppCompatActivity;

import android.support.v7.widget.Toolbar;

La clase ***"AppCompatActivity"*** es la encargada de la pantalla principal y la ***"widget.Toolbar"*** se encarga de la barra de herramientas.

La librería view se encarga de las clases de la pantalla con la interacción con el usuario

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.LayoutInflater;

import android.view.ViewGroup;

La interfaz ***"view.Menu"*** gestiona los elementos de un menú. La interfaz ***"view.MenuItem"*** se encarga de los menús creados previamente, esta clase dentro de un listener en un botón, hace que le pase información cuando se hace click sobre él. La interfaz ***"view.LayoutInflater"*** crea una instancia de un archivo XML de diseño en sus correspondientes objetos de vista. Por último, la instancia ***"view.ViewGroup"*** es una vista especial que puede contener otras vistas.

El paquete de widgets contiene elementos para usar en la pantalla de la aplicación.

import android.widget.Button;

import android.widget.CompoundButton;

import android.widget.SeekBar;

import android.widget.Switch;

import android.widget.TextView;

import android.widget.Toast;

import android.widget.AdapterView;

import android.widget.ListView;

import android.widget.ArrayAdapter;

"widget.AdapterView" muestra elementos cargados en un adaptador, ***"widget.Button"*** un elemento con forma de botón que el usuario puede tocar o hacer clic para realizar una acción. ***"Widget.CompoundButton"*** es un elemento botón que puede estar marcado o sin marcar, ***"widget.SeekBar"*** es una extensión de una barra con un circulo que se puede arrastrar, ***"widget.Switch"*** es un elemento con forma de botón tipo switch, ***"widget.TextView"*** es un elemento de tipo texto, ***"widget.ListView"*** es un elemento tipo lista, ***"widget.Toast"*** son mensajes para mostrar en la pantalla como los de

tipo error, ***"widget.ArrayAdapter"*** es un tipo de adaptador que puede utilizarse en un tipo lista para convertirlo en un array.

La librería com.upct.blurfeet es la biblioteca creada por mí.

import com.upct.bluefeet.adapters.BluetoothDevicesAdapter;

import com.upct.bluefeet.threads.BluetoothThread;

import com.upct.bluefeet.R;

import com.upct.bluefeet.controllers.BluetoothController;

import com.upct.bluefeet.controllers.SensorController;

import com.upct.bluefeet.listeners.BluetoothListener;

import com.upct.bluefeet.listeners.SensorListener;

Podremos dividir en 4 tipos, listeners, controladores, threads y la que llamo R. En concreto, "***listeners.BluetoothListener"*** es un escuchador para Bluetooth, ***"listeners.SensorListener;"*** es un escuchador para los sensores, "***controllers.BluetoothController"*** es el encargado de controlar el escuchador del protocolo Bluetooth, ***"controllers.SensorController"*** es el controlador encargado del escuchador de los sensores. ***"Bluefeet.R"*** es la encargada de asociar elementos visuales con partes del código, y ***"threads.BluetoothThread"*** se encarga de las tareas en segundo plano del Bluetooth.

import java.text.SimpleDateFormat; sirve para dar formato de fecha.

La biblioteca io es específica para entrada-salida.

import java.io.InputStream;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

"io.InputStream" sirve para leer cadenas de caracteres entrantes, ***"io.File"*** sirve para crear archivos, renombrarlos o cambiar la ruta, ***"io.IOException"*** es para las

excepciones con las entradas salidas, e ***"io.FileOutputStream"*** crea un archivo o secuencia de salida.

El paquete util de Java contiene las clases relacionadas con la fecha y la hora, el modelo de eventos, la internacionalización y las clases de utilidades diversas.

import java.util.Calendar;

import java.util.Date;

import java.util.List;

import java.util.UUID;

import java.util.LinkedList;

import java.util.Set;

"util.Calendar" es una clase que se utiliza para convertir un campo de fecha hora y mes, ***"util.Date"*** nos devuelve la fecha y la hora actuales, ***"util.List"*** sirve para importar arrays y listas, ***"util.UUID"*** es la clase para el numero ID único, ***"util.LinkedList"*** sirve para construir una lista que contiene elementos específicos en orden en el que los devuelve el iterador de la colección, ***"util.Set"*** es como ***"util.List"*** con la diferencia de que solo se ejecuta una sola vez.

3.4.3 Almacenamiento

El almacenamiento se realizará en la memoria interna del teléfono móvil y se guardará en un documento de texto tipo .txt. Se crearán 12 columnas separadas por una tabulación y al llegar a la última se realizará un retorno de carro para saltar a la siguiente línea.

A grandes rasgos lo que hace es coger el array que contiene todos los valores, lo recorre con un puntero y los almacena dentro de ese documento .txt. Al crearlo lo nombra con el texto de SENSOR_VALUES y año, mes, día, hora, minuto y segundo para que no coincidan si existen varios archivos.

Capítulo 4

Descripción del hardware y del software desarrollado

4.1 Introducción.

En este capítulo se describirá todo el trabajo realizado tanto de la parte software como de la parte hardware. Para la parte hardware he realizado una serie de pruebas para determinar que sensor elegir. En este capítulo también se incluyen los esquemáticos y el diseño final de la placa. En la parte software se explica el código tanto del microprocesador ESP32 como el código de la aplicación para el terminal móvil inteligente con sistema operativo Android.

4.2 Hardware.

4.2.1 Estudio previo para la elección de sensores y de plantillas.

En este apartado comenzamos con una serie de pruebas que han servido tanto para la elección correcta de los sensores como para la elección del la plantilla que se debe utilizar para el proyecto.

Primero cabe destacar que tenemos dos medidas de sensores para realizar las pruebas y tres tipos de plantillas diferentes. El primero de los sensores y más grande de 26 milímetros de diámetro y el segundo y más pequeño es de 9 milímetros de diámetro. Las plantillas son de diferentes materiales, una de ellas es de cuero, otra es de goma y la última es de gel.

Comenzamos con las pruebas para elegir qué medida de sensores escogeremos y qué tipo de plantilla es la más adecuada para nuestro proyecto.

En el eje X siempre se mostrará el numero de muestras y en el eje Y siempre se mostrará el valor que nos dan los sensores en un rango de valores donde 0 es el mínimo y 4096 el máximo valor que nos podría mostrar el microcontrolador.

Esta primera ronda de pruebas la realizaremos con el sensor grande en el antepié y el sensor pequeño en el mediopié dando un paso recorriendo desde el talón a la punta de los pies y después volviendo otra vez al talón.

La primera prueba la realizamos utilizando la plantilla de gel (ver Tabla 4.1).

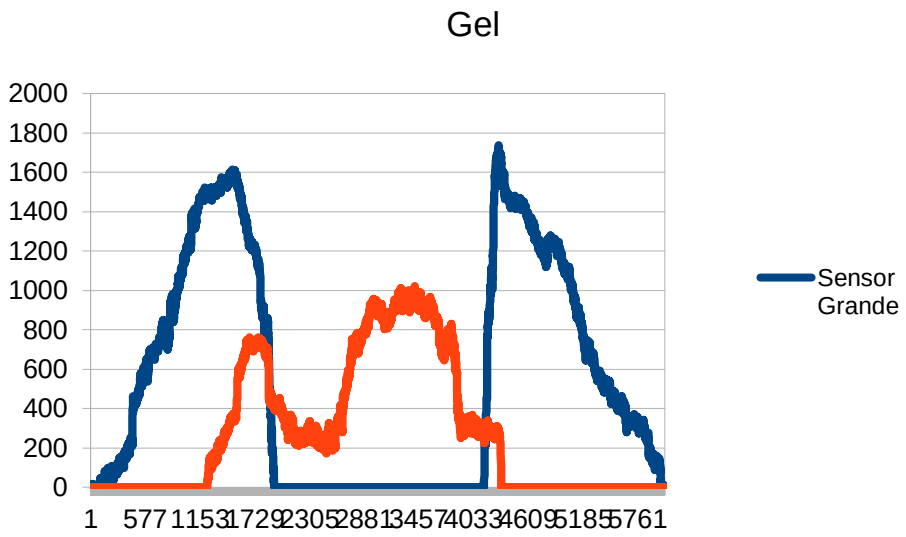


Tabla 4.1 Pruebas con plantillas de gel y dos sensores.

La segunda prueba será con una plantilla de goma (ver Tabla 4.2).

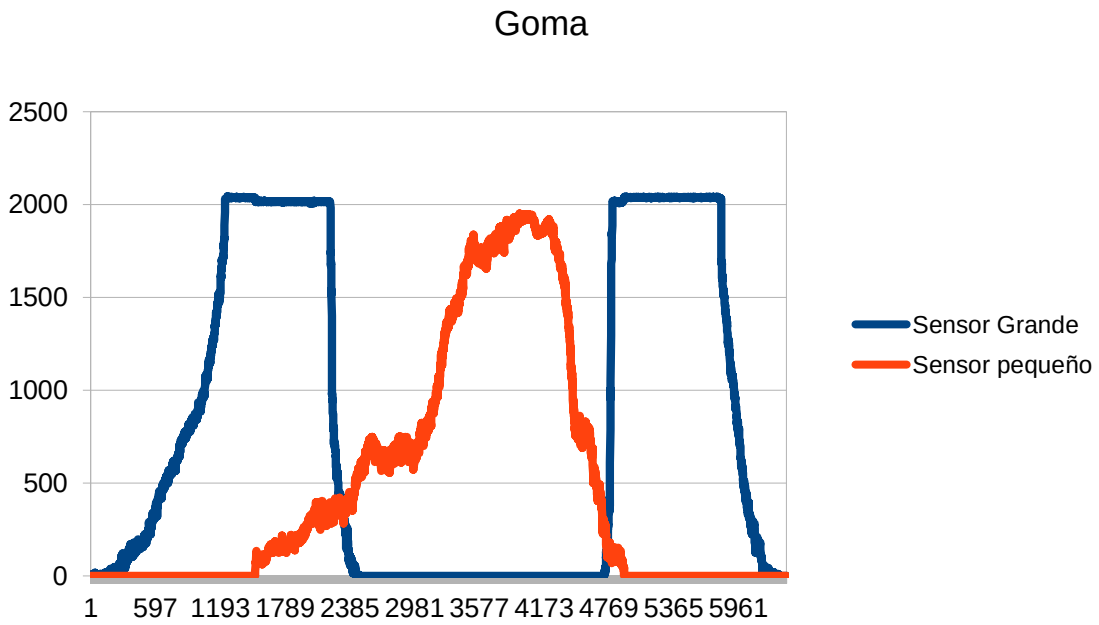


Tabla 4.2 Pruebas con plantillas de goma y dos sensores.

La tercera prueba la realizamos con una plantilla de cuero (ver Tabla 4.3)

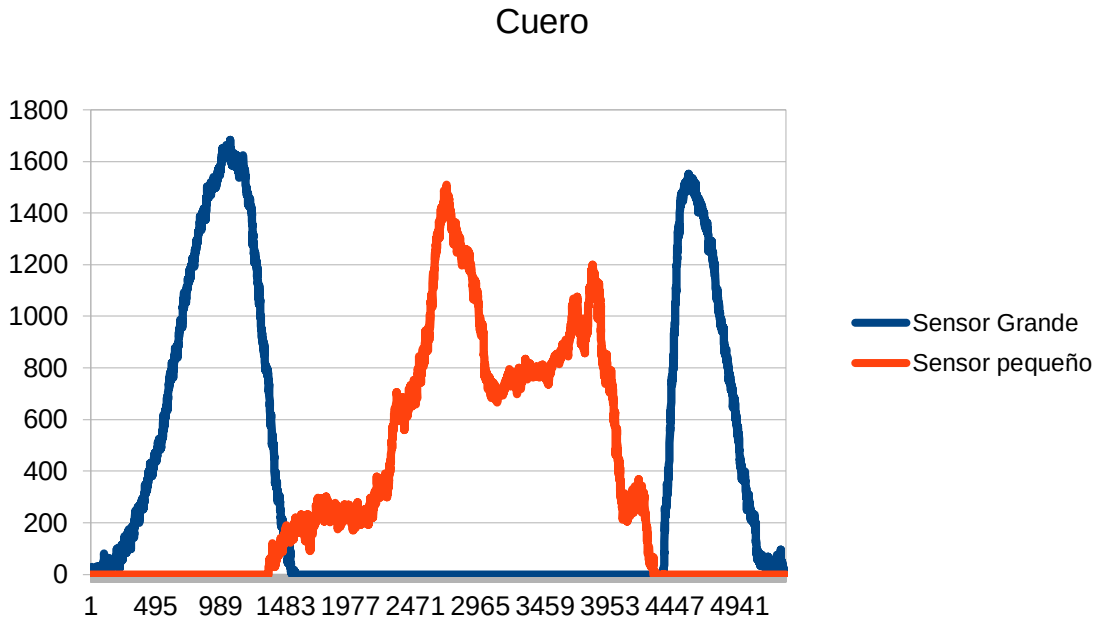


Tabla 4.3 Pruebas con plantillas de cuero y dos sensores.

Con los datos anteriores haré dos gráficas, cada una de ellas con un tipo de sensor, es decir, una gráfica con los valores de los sensores grandes y una gráfica con los valores de los sensores pequeños para ver con mayor facilidad la elección correcta de la plantilla que vamos a utilizar y también del tipo de sensor que vamos a elegir.

Comenzamos con la gráfica de los sensores grandes (ver Tabla 4.4).

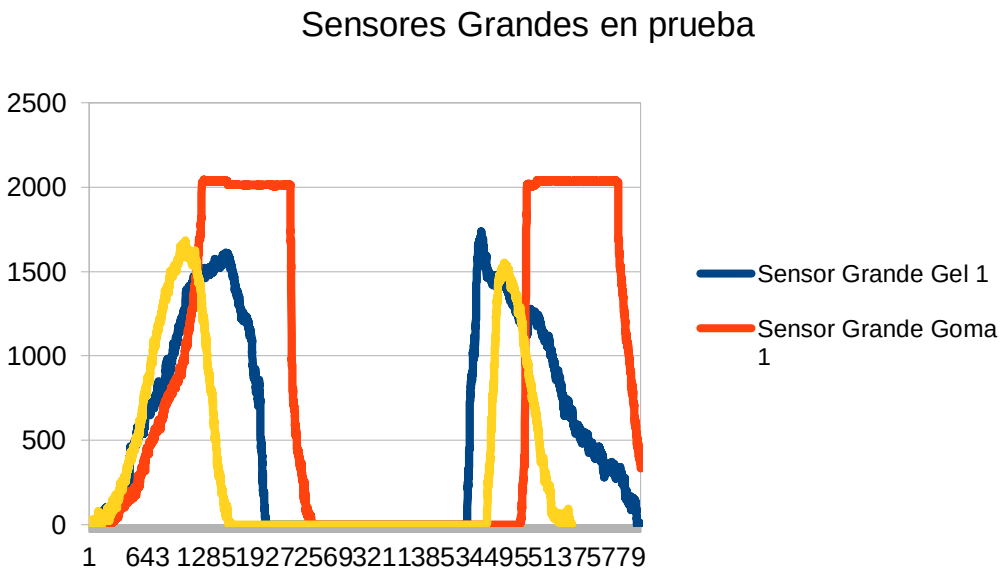


Tabla 4.4 Pruebas realizadas con todos los sensores grandes en las diferentes plantillas.

Como se puede observar la plantilla de goma es la que mejores resultados nos da con este tipo de sensor.

Ahora haremos lo mismo con la grafica de los sensores pequeños (ver Tabla 4.5).

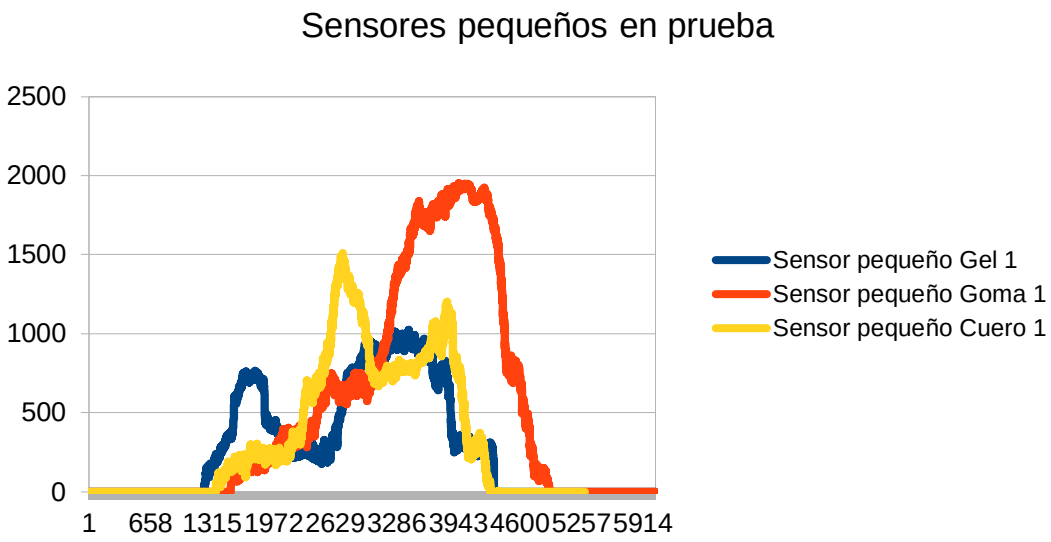


Tabla 4.5 Pruebas realizadas con todos los sensores pequeños en las diferentes plantillas.

En esta gráfica se puede observar que los mejores resultados son los obtenidos con la plantilla de goma al igual que hemos visto con la gráfica anterior. También podemos observar que la señal es menos estable y tiene más ruido.

Con estos datos podemos decir que el funcionamiento de los sensores grandes es mejor que el de los sensores pequeños y que la plantilla de goma es la más adecuada para el proyecto.

Ahora realizaremos una segunda prueba invirtiendo el orden de los sensores, es decir, el sensor pequeño lo colocaremos en el antepié y el sensor grande lo colocaremos en el mediopié. Realizaremos otra vez la prueba con las diferentes plantillas con un movimiento de una pisada que va desde el talón a los dedos y vuelta al talón.

La primera prueba la realizaremos con una plantilla de gel (ver Tabla 4.6).

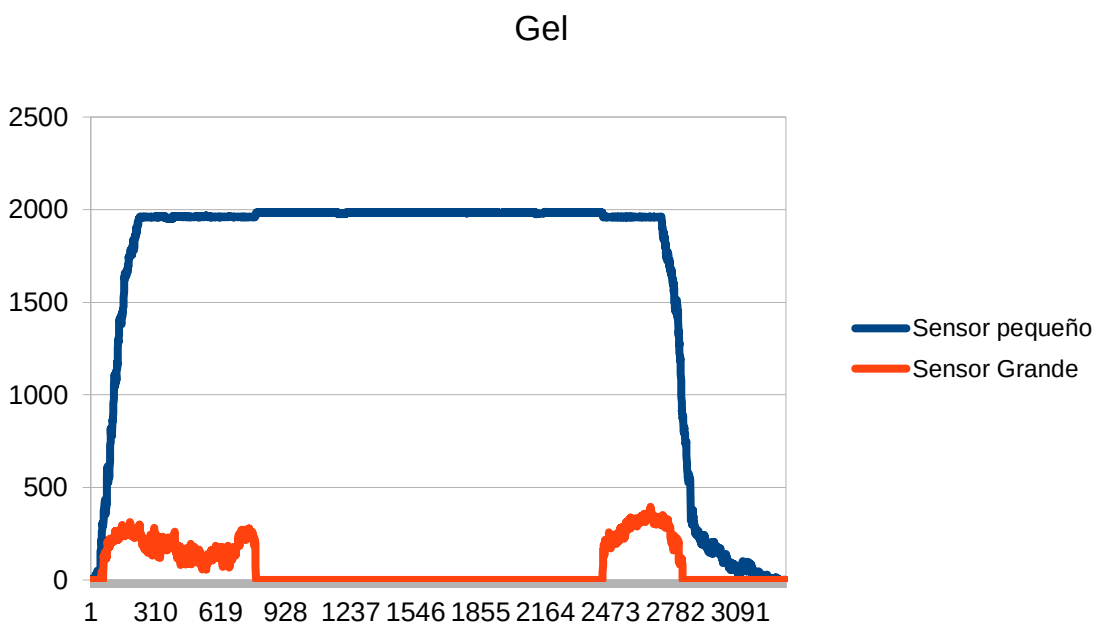


Tabla 4.6 Pruebas con plantillas de gel y dos sensores invertidos de posición.

La segunda prueba la realizaremos con la plantilla de goma (ver Tabla 4.7).

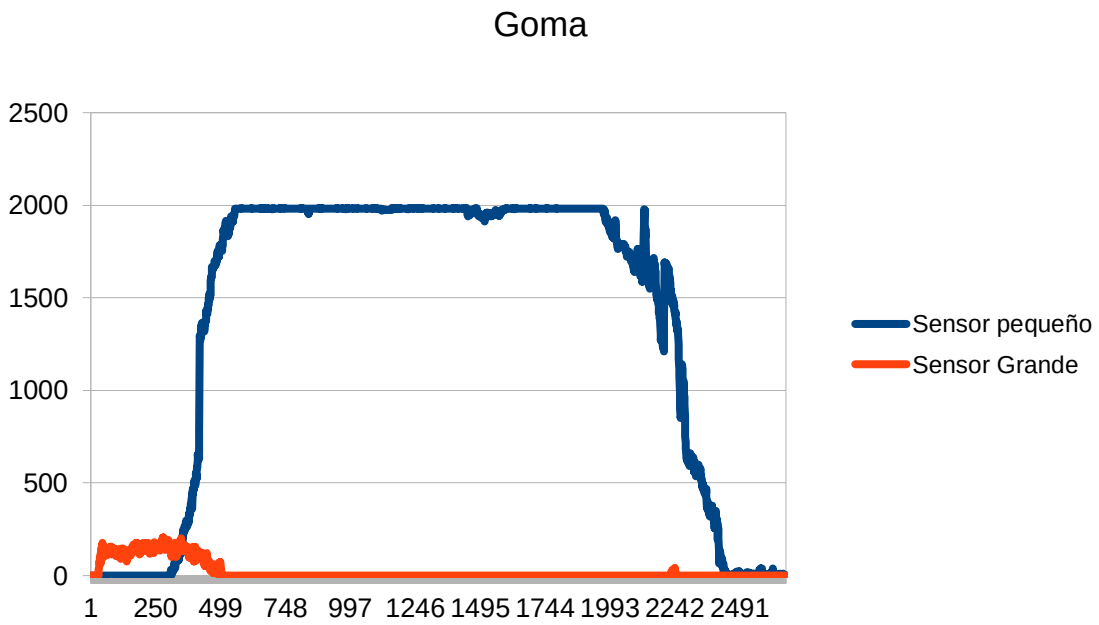


Tabla 4.7 Pruebas con plantillas de goma y dos sensores invertidos de posición.

La tercera prueba que realizaremos será con la plantilla de cuero (ver Tabla 4.8).

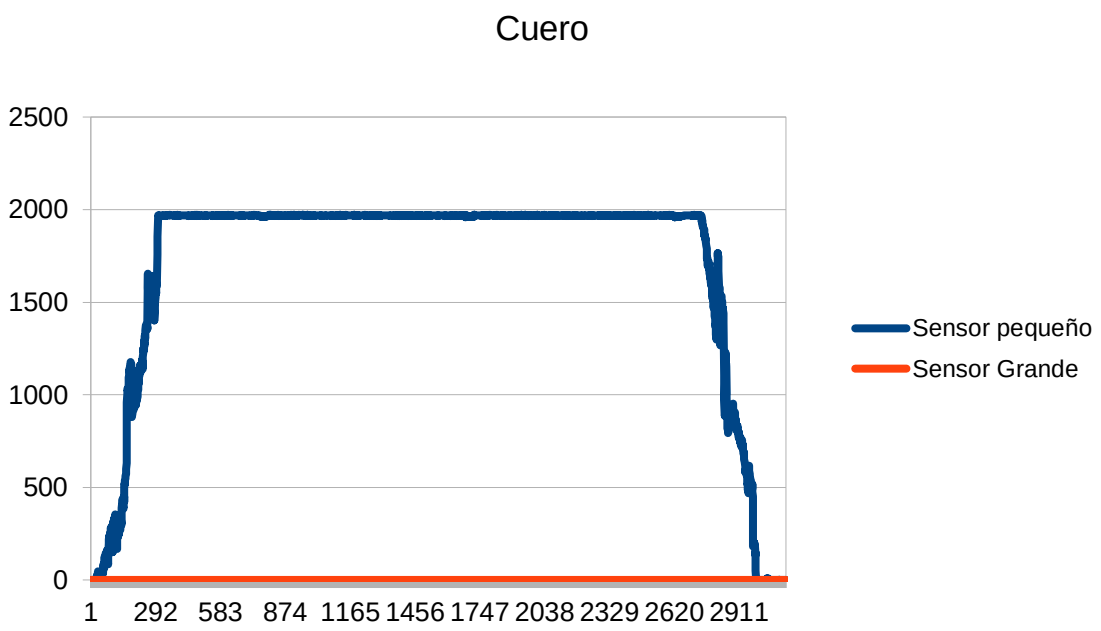


Tabla 4.8 Pruebas con plantillas de goma y dos sensores invertidos de posición.

Ahora volveré a reagrupar los datos por tamaños de los sensores y graficaremos los resultados de las diferentes plantillas.

Empezamos con la grafica de los sensores pequeños (ver Tabla 4.9).

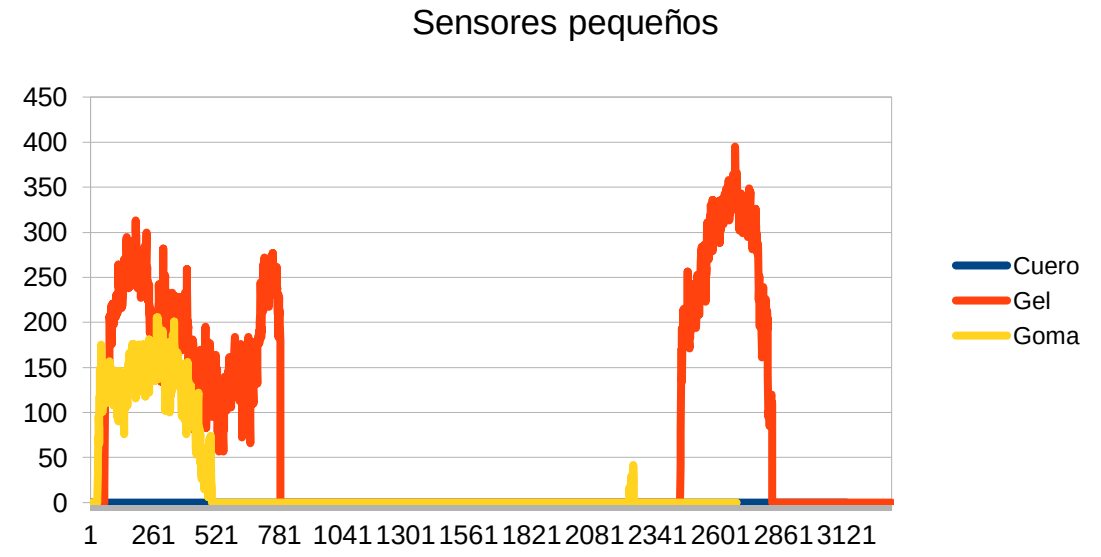


Tabla 4.9 Pruebas realizadas con todos los sensores pequeños en las diferentes plantillas invertidos de posición.

Por último haremos lo mismo con los sensores grandes, Tabla 4.10.

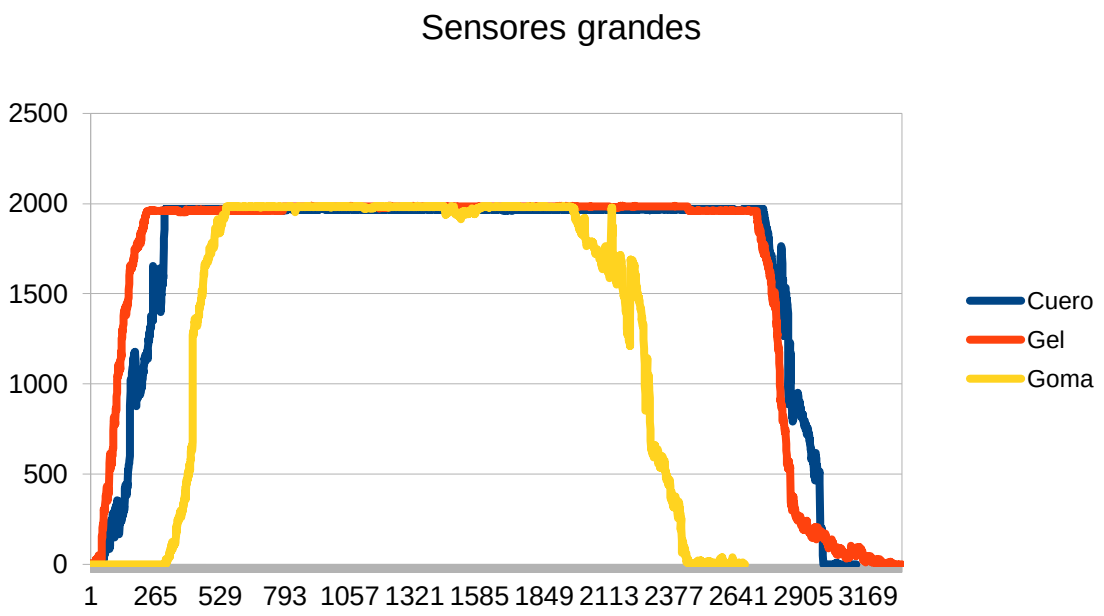


Tabla 4.10 Pruebas realizadas con todos los sensores grandes en las diferentes plantillas invertidos de posición.

Con los datos que nos muestran las dos graficas anteriores se puede observar que los sensores que mejor comportamiento tienen para nuestro proyecto son los sensores grandes. Analizando la grafica de los señores grandes vemos que todas las plantillas utilizadas tienen un resultado muy parecido.

Como conclusión de estas dos pruebas podemos sacar en claro que los mejores sensores para el proyecto son los sensores grandes y que la mejor plantilla que podemos utilizar es la plantilla de goma. Debemos de tener en cuenta que no todas las pisadas que he realizado son 100% iguales. En la última prueba se puede observar como existe un poco de ruido en la pisada con la plantilla de goma al final de la misma, pero en definitiva ha sido la que mejores resultados nos ha dado.

Una vez elegidos los sensores y la plantilla que vamos a utilizar, realizaremos otra prueba con 10 sensores. Su colocación está basada en toda la información sobre los tipos de pisadas y los problemas de puente vistos en el capitulo anterior.

La disposición que se utiliza es la siguiente (ver Figura 4.1).

Figura 4.1 Disposición de sensores 1.

Ahora realizaremos una pisada completa que irá desde el talón hasta la punta de los dedos y después volverá otra vez hasta el talón (ver Tabla 4.11).

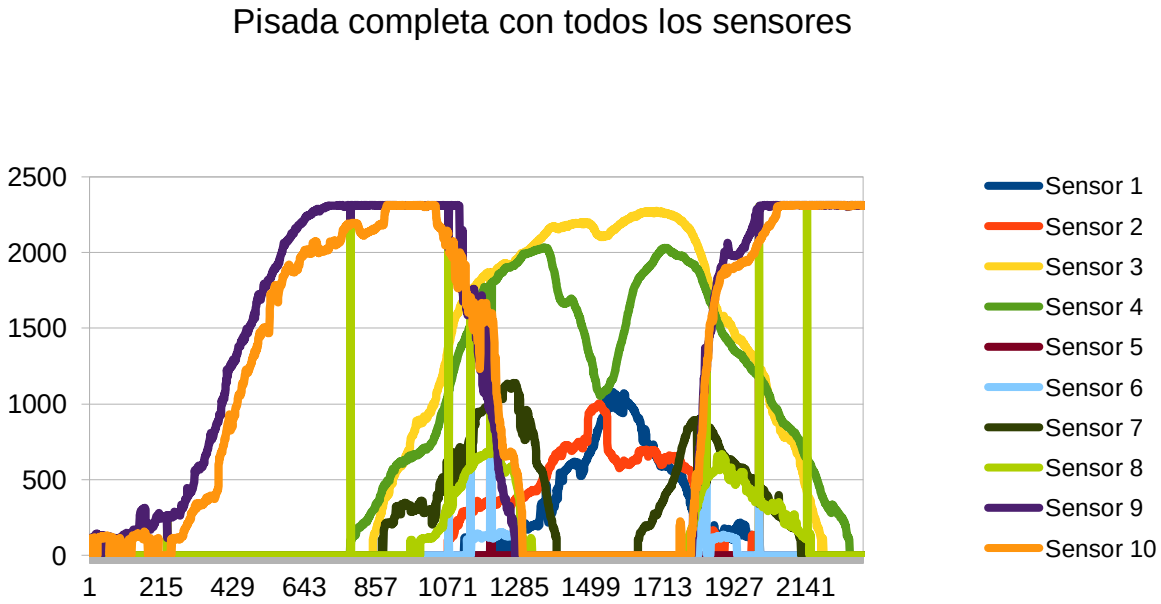


Tabla 4.11 Prueba realizada 10 sensores grandes en la plantilla de goma.

En esta prueba se puede observar que hay sensores que marcan menos seguramente debido a que en ellos el pie ejerce una presión menor como pueden ser los sensores de la parte del puente del pie.

Para el siguiente experimento añadimos 2 sensores más y volvemos a repetir la prueba con la plantilla de goma encima de los sensores.

La nueva disposición será la siguiente (ver Figura 4.2).

Figura 4.2 Disposición de sensores 2.

Y los valores obtenidos son los siguientes (ver Tabla 4.12).

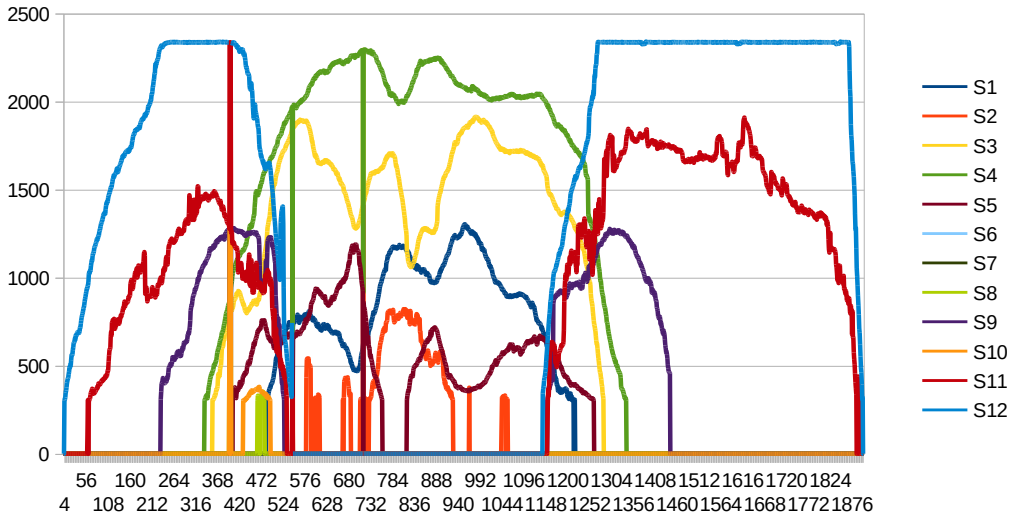


Tabla 4.12 Prueba realizada 12 sensores grandes en la plantilla de goma.

En el experimento se puede observar, al igual que en el anterior, que existe medición en todos los sensores, pero en algunos nos dan unos valores muy bajos. Esto es debido seguramente a que en esos sensores se ejerce menor presión, al igual que en la prueba anterior coinciden con la parte del puente del pie.

El siguiente experimento consistirá en colocar pesos concretos en todos los sensores y medir los valores obtenidos por los sensores. Estos valores lo graficaremos en una tabla cuyo eje Y será el valor entre 0 y 4098 que nos darán los sensores y el eje X será el peso concreto pesado en una bascula de precisión (ver Tabla 4.13).

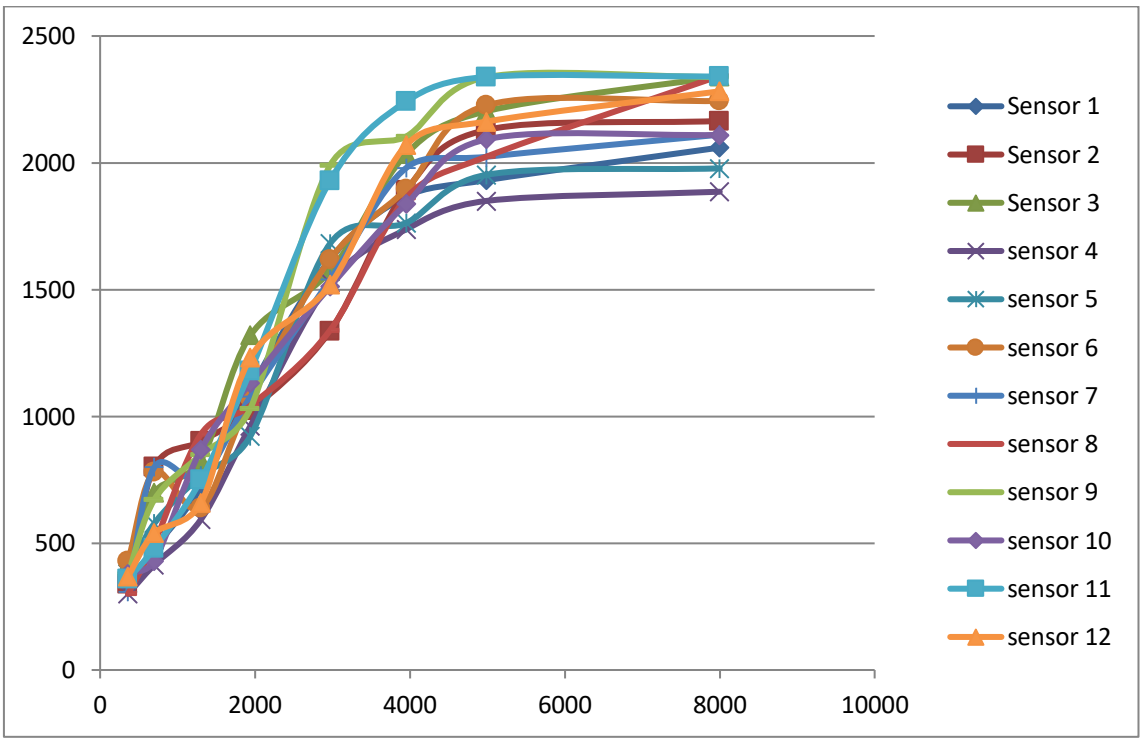


Tabla 4.13 Pruebas realizadas con cada uno de los sensores y con pesos concretos.

Como podemos observar los sensores tienen un comportamiento más o menos lineal hasta que llegamos al peso máximo de alrededor de 5 kg. En ese peso se estanca el crecimiento y se mantiene más o menos el valor dado por lo sensores aun poniendo mayor peso sobre ellos.

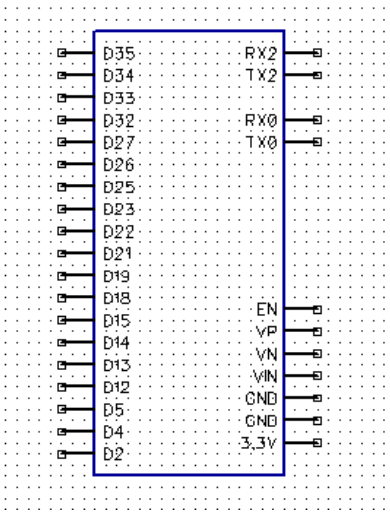
4.2.2 Esquemáticos y diseño de placa

En este apartado mostraremos tanto el diseño de la placa, como se esquemático y la lista de los materiales que vamos a utilizar en el proyecto.

Utilizaremos el software de diseño DipTrace, descargaremos una versión de prueba de 30 días desde su página oficial [DipTrace].

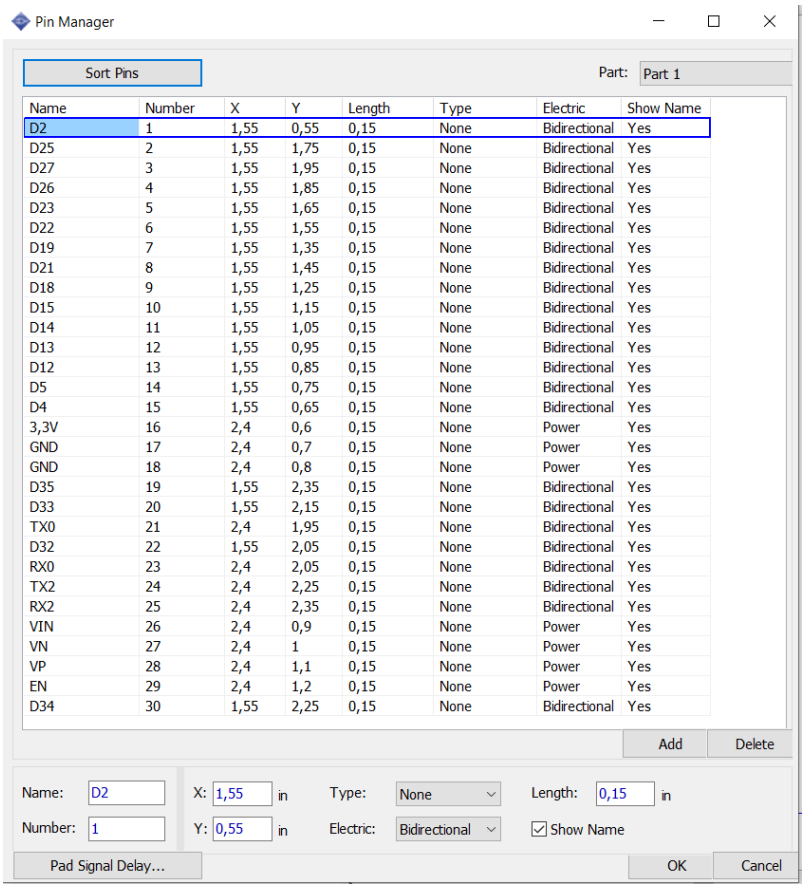
Empezaremos con el esquemático, antes de empezar con él debemos crear dos componentes que no están en la librería de DipTrace.

El primero de ellos es el microcontrolador ESP32, posee 30 pines, de los cuales, solo utilizaremos cuatro pines digitales de salida y un pin analógico para lectura. Este es el diseño que he realizado para el esquemático del circuito (ver Figura. 4.4)



4.3 ESP32 en el editor de componentes para el esquemático.

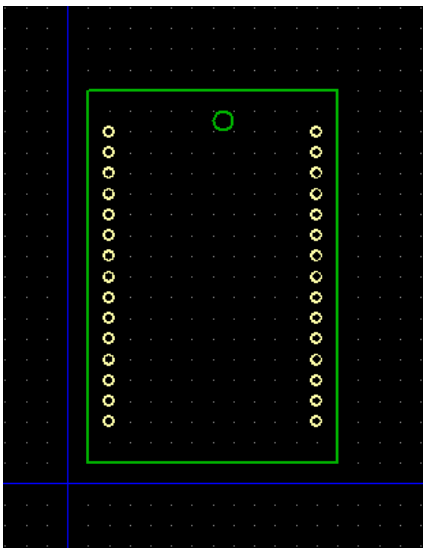
Lo siguiente que realizaremos será la selección de pines (ver Figura 4.5).



Name	Number	X	Y	Length	Type	Electric	Show Name
D2	1	1,55	0,55	0,15	None	Bidirectional	Yes
D25	2	1,55	1,75	0,15	None	Bidirectional	Yes
D27	3	1,55	1,95	0,15	None	Bidirectional	Yes
D26	4	1,55	1,85	0,15	None	Bidirectional	Yes
D23	5	1,55	1,65	0,15	None	Bidirectional	Yes
D22	6	1,55	1,55	0,15	None	Bidirectional	Yes
D19	7	1,55	1,35	0,15	None	Bidirectional	Yes
D21	8	1,55	1,45	0,15	None	Bidirectional	Yes
D18	9	1,55	1,25	0,15	None	Bidirectional	Yes
D15	10	1,55	1,15	0,15	None	Bidirectional	Yes
D14	11	1,55	1,05	0,15	None	Bidirectional	Yes
D13	12	1,55	0,95	0,15	None	Bidirectional	Yes
D12	13	1,55	0,85	0,15	None	Bidirectional	Yes
D5	14	1,55	0,75	0,15	None	Bidirectional	Yes
D4	15	1,55	0,65	0,15	None	Bidirectional	Yes
3,3V	16	2,4	0,6	0,15	None	Power	Yes
GND	17	2,4	0,7	0,15	None	Power	Yes
GND	18	2,4	0,8	0,15	None	Power	Yes
D35	19	1,55	2,35	0,15	None	Bidirectional	Yes
D33	20	1,55	2,15	0,15	None	Bidirectional	Yes
TX0	21	2,4	1,95	0,15	None	Bidirectional	Yes
D32	22	1,55	2,05	0,15	None	Bidirectional	Yes
RX0	23	2,4	2,05	0,15	None	Bidirectional	Yes
TX2	24	2,4	2,25	0,15	None	Bidirectional	Yes
RX2	25	2,4	2,35	0,15	None	Bidirectional	Yes
VIN	26	2,4	0,9	0,15	None	Power	Yes
VN	27	2,4	1	0,15	None	Power	Yes
VP	28	2,4	1,1	0,15	None	Power	Yes
EN	29	2,4	1,2	0,15	None	Power	Yes
D34	30	1,55	2,25	0,15	None	Bidirectional	Yes

4.4 Selección de número de pin y del tipo de pin para el ESP32.

Después realizaremos el diseño del patrón con la disposición real de todos los pines y las medidas exactas para que a la hora de hacer la placa todas las medidas sean las correctas (ver Figura 4.6).



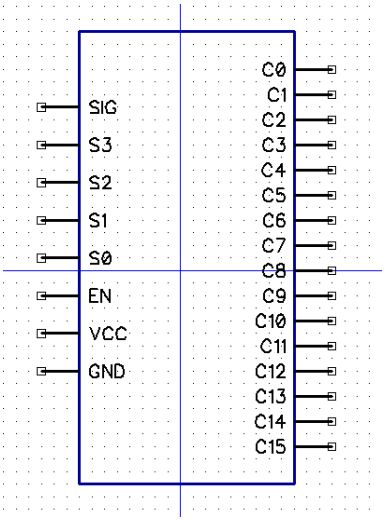
4.5 ESP32 en el editor de componentes para el diseño de la placa.

Una vez realizado el dibujo debemos de seleccionar cada uno de los pines. Los pines seleccionados en el paso anterior los asignemos a un pin concreto del dibujo.

Una vez todos asignados ya podemos buscar el componente en la librería en la que lo hayamos guardado.

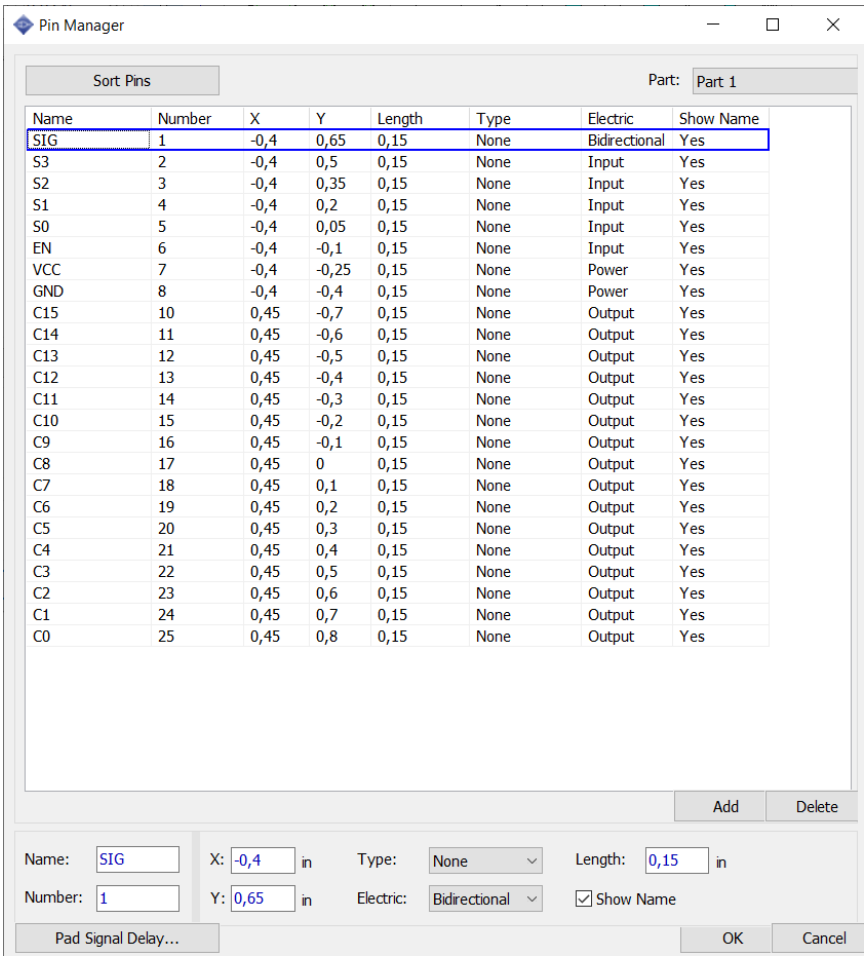
Ahora realizaremos la misma operación con el siguiente componente que no se encuentra en nuestra biblioteca, hablamos del multiplexor.

Comenzamos al igual que para el primer componente haciendo el diseño para el esquemático del circuito. Este circuito tiene 25 pines, de los cuales cuatro de ellos son para la elección de la patilla que vamos a seleccionar, tiene tan solo una sola salida y 16 entradas seleccionables (ver Figura 4.7).



4.6 Multiplexor en el editor de componentes para el esquemático.

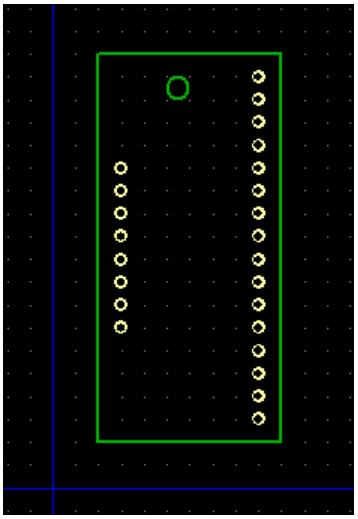
Ahora continuamos con la selección de pines (ver Figura 4.8).



Name	Number	X	Y	Length	Type	Electric	Show Name
SIG	1	-0,4	0,65	0,15	None	Bidirectional	Yes
S3	2	-0,4	0,5	0,15	None	Input	Yes
S2	3	-0,4	0,35	0,15	None	Input	Yes
S1	4	-0,4	0,2	0,15	None	Input	Yes
S0	5	-0,4	0,05	0,15	None	Input	Yes
EN	6	-0,4	-0,1	0,15	None	Input	Yes
VCC	7	-0,4	-0,25	0,15	None	Power	Yes
GND	8	-0,4	-0,4	0,15	None	Power	Yes
C15	10	0,45	-0,7	0,15	None	Output	Yes
C14	11	0,45	-0,6	0,15	None	Output	Yes
C13	12	0,45	-0,5	0,15	None	Output	Yes
C12	13	0,45	-0,4	0,15	None	Output	Yes
C11	14	0,45	-0,3	0,15	None	Output	Yes
C10	15	0,45	-0,2	0,15	None	Output	Yes
C9	16	0,45	-0,1	0,15	None	Output	Yes
C8	17	0,45	0	0,15	None	Output	Yes
C7	18	0,45	0,1	0,15	None	Output	Yes
C6	19	0,45	0,2	0,15	None	Output	Yes
C5	20	0,45	0,3	0,15	None	Output	Yes
C4	21	0,45	0,4	0,15	None	Output	Yes
C3	22	0,45	0,5	0,15	None	Output	Yes
C2	23	0,45	0,6	0,15	None	Output	Yes
C1	24	0,45	0,7	0,15	None	Output	Yes
C0	25	0,45	0,8	0,15	None	Output	Yes

4.7 Selección de número de pin y del tipo de pin para el multiplexor.

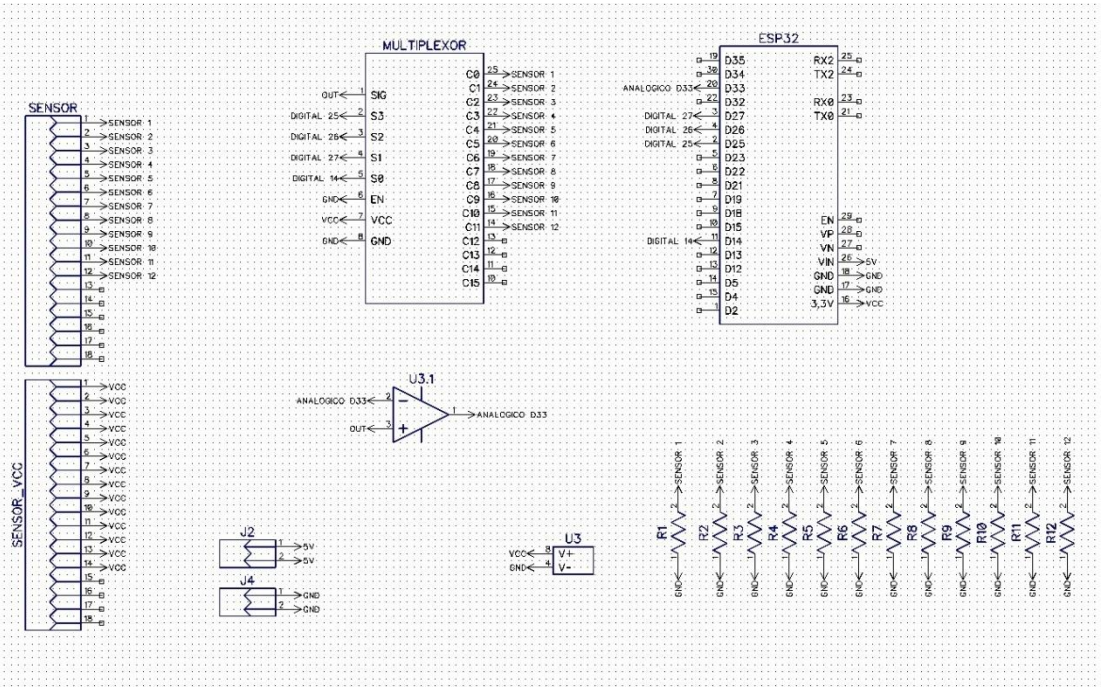
A continuación realizaremos el dibujo con las medidas concretas del componente (ver Figura 4.9).



4.8 Multiplexor en el editor de componentes para el diseño de la placa.

Una vez terminado, asignaremos los pines que anteriormente hemos seleccionado al dibujo y lo guardamos en la librería para su posterior utilización.

Ya tenemos todos los componentes que necesitamos, ahora nos disponemos a realizar el diseño esquemático (ver Figura 4.10).



4.9 Esquemático del proyecto.

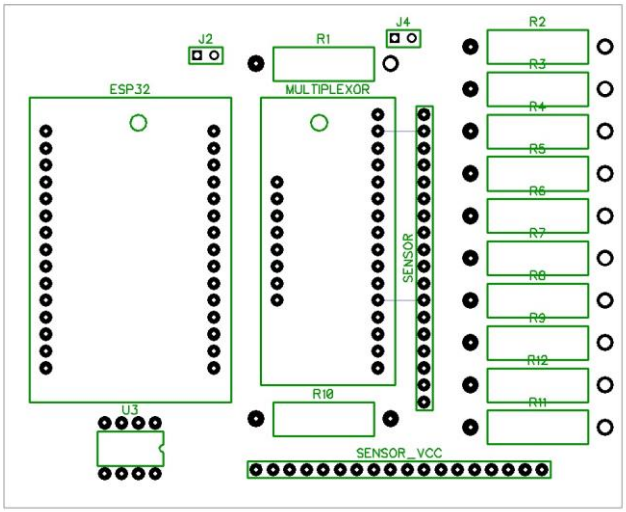
Con el programa podremos sacar una lista de todos los materiales necesarios para nuestro trabajo (ver Figura 4.11).

#	Value	Name	RefDes	Number of Pins
1		ESP32	ESP32	30
2		Pin headed-1x2	J2	2
3		Pin headed-1x2	J4	2
4		Untitled	MULTIPLEXOR	24
5	47K	RES1200	R1	2
6	47K	RES1200	R2	2
7	47K	RES1200	R3	2
8	47K	RES1200	R4	2
9	47K	RES1200	R5	2
10	47K	RES1200	R6	2
11	47K	RES1200	R7	2
12	47K	RES1200	R8	2
13	47K	RES1200	R9	2
14	47K	RES1200	R10	2
15	47K	RES1200	R11	2
16	47K	RES1200	R12	2
17		Pin headed-1x18	SENSOR	18
18		Pin headed-1x18	SENSOR_VCC	18
19		LM358AP	U3.1	3
				121

4.10 Lista de materiales.

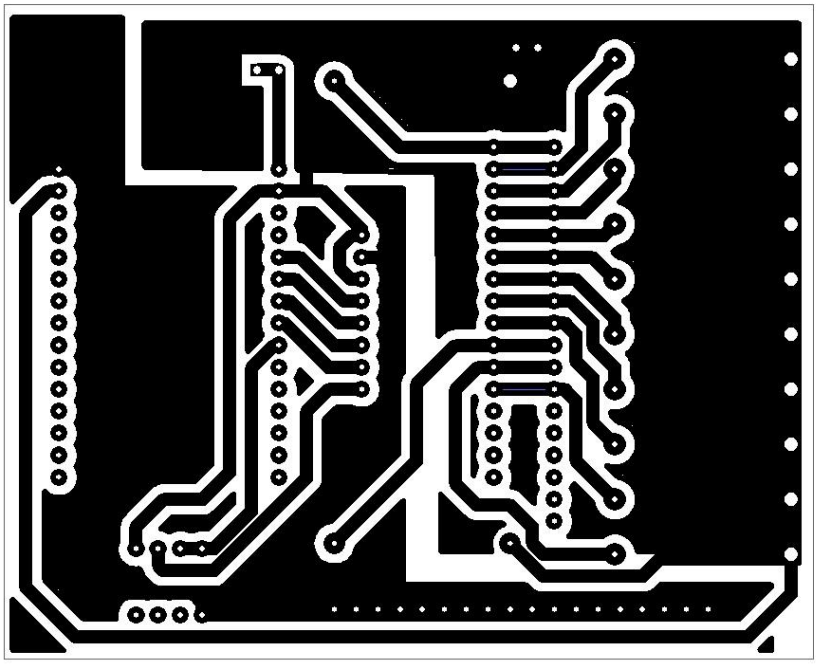
Utilizaremos resistencias de medio de vatio.

Cuando tenemos totalmente terminado el esquemático pasamos a la parte del diseño de la placa. Primero colocaremos los componentes (ver Figura 4.12).



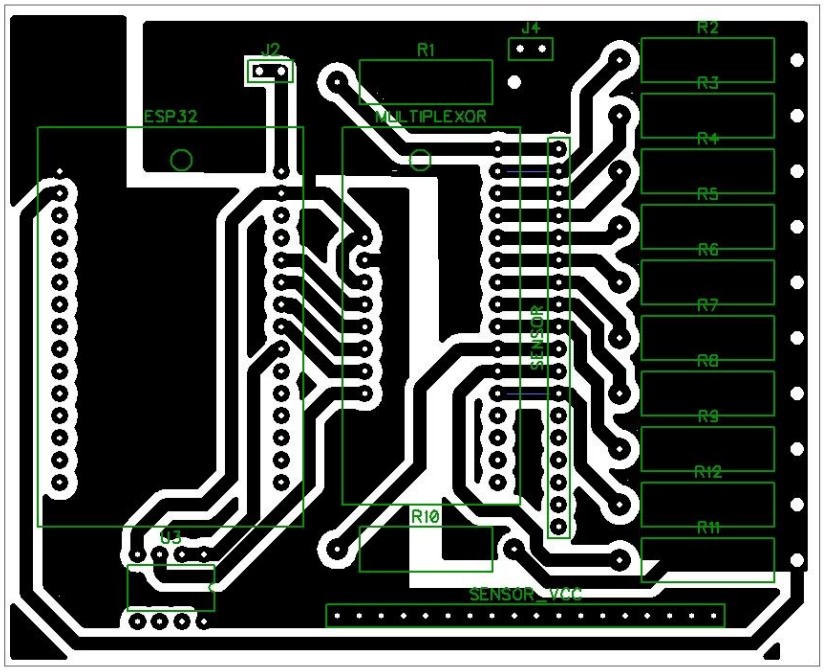
4.11 Distribución de los componentes en la placa del proyecto.

Después rutearemos todas las pistas (ver Figura 4.13).



4.12 Pistas de la placa del proyecto.

Y este sería el resultado final de la placa de nuestro proyecto (ver Figura 4.14).



4.13 Placa completa.

4.3 Software

4.3.1 Programación del ESP32

En este punto explicaremos el programa que introduciremos al ESP32.

Empezamos incluyendo la librería para conectarnos y comunicarnos por Bluetooth y activamos la configuración del Bluetooth.

```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)

#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it

#endif
```

Definimos el pin de entrada que es analógico para leer los sensores y los de salida que son digitales para elegir el sensor que queremos leer en el multiplexor. En este punto solo definiremos cuales son los pines que vamos a utilizar.

```
#define pinSensor 33

int pinMultiplexor3 = 14;

int pinMultiplexor2 = 27;

int pinMultiplexor1 = 26;

int pinMultiplexor0 = 25;
```

Declaramos un array para guardar los valores leídos de los sensores y también declaramos las variables de los sensores.

```
uint arr [12 * 4] ;

unsigned long valorSensor12;

unsigned long valorSensor11;

unsigned long valorSensor10;

unsigned long valorSensor9;

unsigned long valorSensor8;

unsigned long valorSensor7;
```

```
unsigned long valorSensor6;
unsigned long valorSensor5;
unsigned long valorSensor4;
unsigned long valorSensor3;
unsigned long valorSensor2;
unsigned long valorSensor1;
```

Inicializamos el puerto serie Bluetooth

```
BluetoothSerial SerialBT;
```

Creamos una variable booleana, será la encargada de empezar a mandar valores cuando se encuentre en verdadero y dejar de enviarlos cuando se encuentre en falso. Inicializaremos esta variable en falso para que no empiece a enviar valores hasta que el programa no envíe la orden de empezar a mandar.

```
boolean enviarValores = false;
```

Creamos una variable para dar los valores al delay que utilizaremos para leer los sensores. Este delay nos servirá para controlar el tiempo entre una lectura y otra.

```
uint sensorDelay = 10;
```

Inicializamos el bucle Setup, en él renombraremos el ESP32 y aquí es donde inicializamos que los pines que anteriormente hemos escogido los vamos a utilizar como salida.

```
void setup() {
 SerialBT.begin("ESP32test");
 pinMode(pinMultiplexor0, OUTPUT);
 pinMode(pinMultiplexor1, OUTPUT);
 pinMode(pinMultiplexor2, OUTPUT);
```

```
 pinMode(pinMultiplexor3, OUTPUT);
}
```

Inicializamos el bucle loop, que es el que estará ejecutándose sin parar. En este bucle tenemos 2 bucles mas dentro del mismo, que serán los encargados de realizar las acciones de enviar y recibir los datos.

```
void loop() {
```

Creamos un bucle en el que si se manda una petición de enviarnos algo desde la aplicación se meterá en este bucle y nos llevará directamente a otro bucle llamado leerComandos que explicaremos más adelante.

```
 if (SerialBT.available() > 0) {
  leerComando();
 }
```

Creamos un bucle en el que cuando cambie la variable enviarValores de false a true, el cual nos llevará al bucle enviarSensores que explicaremos más adelante.

```
 if (enviarValores) {
  enviarSensores();
 }
}
```

Inicializamos el bucle leerComando

```
void leerComando() {
```

Creamos un bucle while, su cometido es cuando nos hayan enviado un comando desde la aplicación móvil se meterá en este bucle y guardará el comando leído del puerto serie Buetooth en la variable comando.

```
while (SerialBT.available() > 0) {
 uint comando = SerialBT.read();
```

Si el comando leído es un uno pondremos la variable enviarValores a true y empezaremos a mandar valores.

```
 if (comando == 0x01) {
  enviarValores = true;
 }
```

Si el comando leído es un dos pondremos la variable enviarValores a false y pararemos de enviar valores de los sensores.

```
 if (comando == 0x02) {
  enviarValores = false;
 }
```

Si el comando leído es un tres cambiaremos el delay.

```
 if (comando == 0x03) {
  while (SerialBT.available() < 4);
```

Primero inicializamos a cero la variable delay y después vamos leyendo y desplazando para rellenar la variable.

```
  sensorDelay = 0;
  sensorDelay += (SerialBT.read() & 0xFF) << 24;
  sensorDelay += (SerialBT.read() & 0xFF) << 16;
  sensorDelay += (SerialBT.read() & 0xFF) <<  8;
  sensorDelay += (SerialBT.read() & 0xFF) <<  0;
```

Creamos un bucle para que si el valor de la variable es mayor de 1000 la obligamos a que sea 1000.

```
    if (sensorDelay > 1000) sensorDelay = 1000;
  }
 }
}
```

Inicializamos el bucle enviarSensores, este bucle pone los pines del multiplexor en bajo o en alto para ir leyendo los sensores. Leeremos 16 veces los sensores y haremos una media para evitar los posibles ruidos producidos por el circuito.

```
void enviarSensores() {
 for (int i=0; i<16; i++){

  digitalWrite(pinMultiplexor0, LOW);
  digitalWrite(pinMultiplexor1, LOW);
  digitalWrite(pinMultiplexor2, LOW);
  digitalWrite(pinMultiplexor3, LOW);
  valorSensor1 += analogRead(pinSensor);

  digitalWrite(pinMultiplexor0, LOW);
  digitalWrite(pinMultiplexor1, LOW);
  digitalWrite(pinMultiplexor2, LOW);
  digitalWrite(pinMultiplexor3, HIGH);
  valorSensor2 += analogRead(pinSensor);

  digitalWrite(pinMultiplexor0, LOW);
  digitalWrite(pinMultiplexor1, LOW);
  digitalWrite(pinMultiplexor2, HIGH);
```

```
digitalWrite(pinMultiplexor3, LOW);
valorSensor3 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, LOW);
digitalWrite(pinMultiplexor1, LOW);
digitalWrite(pinMultiplexor2, HIGH);
digitalWrite(pinMultiplexor3, HIGH);
valorSensor4 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, LOW);
digitalWrite(pinMultiplexor1, HIGH);
digitalWrite(pinMultiplexor2, LOW);
digitalWrite(pinMultiplexor3, LOW);
valorSensor5 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, LOW);
digitalWrite(pinMultiplexor1, HIGH);
digitalWrite(pinMultiplexor2, LOW);
digitalWrite(pinMultiplexor3, HIGH);
valorSensor6 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, LOW);
digitalWrite(pinMultiplexor1, HIGH);
digitalWrite(pinMultiplexor2, HIGH);
digitalWrite(pinMultiplexor3, LOW);
valorSensor7 += analogRead(pinSensor);
```

```
digitalWrite(pinMultiplexor0, LOW);
digitalWrite(pinMultiplexor1, HIGH);
digitalWrite(pinMultiplexor2, HIGH);
digitalWrite(pinMultiplexor3, HIGH);
valorSensor8 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, HIGH);
digitalWrite(pinMultiplexor1, LOW);
digitalWrite(pinMultiplexor2, LOW);
digitalWrite(pinMultiplexor3, LOW);
valorSensor9 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, HIGH);
digitalWrite(pinMultiplexor1, LOW);
digitalWrite(pinMultiplexor2, LOW);
digitalWrite(pinMultiplexor3, HIGH);
valorSensor10 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, HIGH);
digitalWrite(pinMultiplexor1, LOW);
digitalWrite(pinMultiplexor2, HIGH);
digitalWrite(pinMultiplexor3, LOW);
valorSensor11 += analogRead(pinSensor);

digitalWrite(pinMultiplexor0, HIGH);
digitalWrite(pinMultiplexor1, LOW);
digitalWrite(pinMultiplexor2, HIGH);
```

```
  digitalWrite(pinMultiplexor3, HIGH);
  valorSensor12 += analogRead(pinSensor);
 }

 valorSensor1=valorSensor1/16;
 valorSensor2=valorSensor2/16;
 valorSensor3=valorSensor3/16;
 valorSensor4=valorSensor4/16;
 valorSensor5=valorSensor5/16;
 valorSensor6=valorSensor6/16;
 valorSensor7=valorSensor7/16;
 valorSensor8=valorSensor8/16;
 valorSensor9=valorSensor9/16;
 valorSensor10=valorSensor10/16;
 valorSensor11=valorSensor11/16;
 valorSensor12=valorSensor12/16;
```

En esta parte para evitar ruido cuando el valor del sensor sea menor que 100 lo forzamos a que sea 0 y cuando sea mayor que 100 sea el valor medido. Primero inicializamos a 0 la variable valorSensor, después el valor del sensor lo meteremos en el array.

```
 if(valorSensor1<100){
  valorSensor1=0;
 int a = valorSensor1;
 arr[0 + 0 * 4] = (a >> 24) & 0xFF;
 arr[1 + 0 * 4] = (a >> 16) & 0xFF;
 arr[2 + 0 * 4] = (a >> 8) & 0xFF;
 arr[3 + 0 * 4] = (a >> 0) & 0xFF;
```

```
}
if(valorSensor1>100){
  int a = valorSensor1;
 arr[0 + 0 * 4] = (a >> 24) & 0xFF;
 arr[1 + 0 * 4] = (a >> 16) & 0xFF;
 arr[2 + 0 * 4] = (a >> 8) & 0xFF;
 arr[3 + 0 * 4] = (a >> 0) & 0xFF;
   valorSensor1=0;
}



if(valorSensor2<100){
  valorSensor2=0;
  int a = valorSensor2;
 arr[0 + 1 * 4] = (a >> 24) & 0xFF;
 arr[1 + 1 * 4] = (a >> 16) & 0xFF;
 arr[2 + 1 * 4] = (a >> 8) & 0xFF;
 arr[3 + 1 * 4] = (a >> 0) & 0xFF;
}
if(valorSensor2>100){
  int a = valorSensor2;
 arr[0 + 1 * 4] = (a >> 24) & 0xFF;
 arr[1 + 1 * 4] = (a >> 16) & 0xFF;
 arr[2 + 1 * 4] = (a >> 8) & 0xFF;
 arr[3 + 1 * 4] = (a >> 0) & 0xFF;
  valorSensor2=0;
}
```

```
if(valorSensor3<100){
  valorSensor3=0;
  int a = valorSensor3;
 arr[0 + 2 * 4] = (a >> 24) & 0xFF;
 arr[1 + 2 * 4] = (a >> 16) & 0xFF;
 arr[2 + 2 * 4] = (a >> 8) & 0xFF;
 arr[3 + 2 * 4] = (a >> 0) & 0xFF;
}
if(valorSensor3>100){
 int a = valorSensor3;
 arr[0 + 2 * 4] = (a >> 24) & 0xFF;
 arr[1 + 2 * 4] = (a >> 16) & 0xFF;
 arr[2 + 2 * 4] = (a >> 8) & 0xFF;
 arr[3 + 2 * 4] = (a >> 0) & 0xFF;
 valorSensor3=0;
}

if(valorSensor4<100){
  valorSensor4=0;
  int a = valorSensor4;
 arr[0 + 3 * 4] = (a >> 24) & 0xFF;
 arr[1 + 3 * 4] = (a >> 16) & 0xFF;
 arr[2 + 3 * 4] = (a >> 8) & 0xFF;
 arr[3 + 3 * 4] = (a >> 0) & 0xFF;
  }
```

```
if(valorSensor4>100){
  int a = valorSensor4;
  arr[0 + 3 * 4] = (a >> 24) & 0xFF;
 arr[1 + 3 * 4] = (a >> 16) & 0xFF;
 arr[2 + 3 * 4] = (a >> 8) & 0xFF;
 arr[3 + 3 * 4] = (a >> 0) & 0xFF;
 valorSensor4=0;
}



if(valorSensor5<100){
  valorSensor5=0;
  int a = valorSensor5;
  arr[0 + 4 * 4] = (a >> 24) & 0xFF;
 arr[1 + 4 * 4] = (a >> 16) & 0xFF;
 arr[2 + 4 * 4] = (a >> 8) & 0xFF;
 arr[3 + 4 * 4] = (a >> 0) & 0xFF;
}
if(valorSensor5>100){
 int a = valorSensor5;
 arr[0 + 4 * 4] = (a >> 24) & 0xFF;
 arr[1 + 4 * 4] = (a >> 16) & 0xFF;
 arr[2 + 4 * 4] = (a >> 8) & 0xFF;
 arr[3 + 4 * 4] = (a >> 0) & 0xFF;
 valorSensor5=0;
}
```

```
if(valorSensor6<100){
  valorSensor6=0;
  int a = valorSensor6;
  arr[0 + 5 * 4] = (a >> 24) & 0xFF;
 arr[1 + 5 * 4] = (a >> 16) & 0xFF;
 arr[2 + 5 * 4] = (a >> 8) & 0xFF;
 arr[3 + 5 * 4] = (a >> 0) & 0xFF;
}
if(valorSensor6>100){

  int a = valorSensor6;
  arr[0 + 5 * 4] = (a >> 24) & 0xFF;
 arr[1 + 5 * 4] = (a >> 16) & 0xFF;
 arr[2 + 5 * 4] = (a >> 8) & 0xFF;
 arr[3 + 5 * 4] = (a >> 0) & 0xFF;
 valorSensor6=0;
}

if(valorSensor7<100){
  valorSensor7=0;
  int a = valorSensor7;
  arr[0 + 6 * 4] = (a >> 24) & 0xFF;
 arr[1 + 6 * 4] = (a >> 16) & 0xFF;
 arr[2 + 6 * 4] = (a >> 8) & 0xFF;
 arr[3 + 6 * 4] = (a >> 0) & 0xFF;
}
```

```
if(valorSensor7>100){

  int a = valorSensor7;
  arr[0 + 6 * 4] = (a >> 24) & 0xFF;
 arr[1 + 6 * 4] = (a >> 16) & 0xFF;
 arr[2 + 6 * 4] = (a >> 8) & 0xFF;
 arr[3 + 6 * 4] = (a >> 0) & 0xFF;
 valorSensor7=0;
}


if(valorSensor8<100){
  valorSensor8=0;
 int a = valorSensor8;
  arr[0 + 7 * 4] = (a >> 24) & 0xFF;
 arr[1 + 7 * 4] = (a >> 16) & 0xFF;
 arr[2 + 7 * 4] = (a >> 8) & 0xFF;
 arr[3 + 7 * 4] = (a >> 0) & 0xFF;
}
if(valorSensor8>100){
  int a = valorSensor8;
  arr[0 + 7 * 4] = (a >> 24) & 0xFF;
 arr[1 + 7 * 4] = (a >> 16) & 0xFF;
 arr[2 + 7 * 4] = (a >> 8) & 0xFF;
 arr[3 + 7 * 4] = (a >> 0) & 0xFF;
  Serial.print(valorSensor8);
}
```

```
if(valorSensor9<100){
  valorSensor9=0;
  int a = valorSensor9;
  arr[0 + 8 * 4] = (a >> 24) & 0xFF;
  arr[1 + 8 * 4] = (a >> 16) & 0xFF;
  arr[2 + 8 * 4] = (a >> 8) & 0xFF;
  arr[3 + 8 * 4] = (a >> 0) & 0xFF;
}

if(valorSensor9>100){
   int a = valorSensor9;
  arr[0 + 8 * 4] = (a >> 24) & 0xFF;
  arr[1 + 8 * 4] = (a >> 16) & 0xFF;
  arr[2 + 8 * 4] = (a >> 8) & 0xFF;
  arr[3 + 8 * 4] = (a >> 0) & 0xFF;
  valorSensor9=0;
}

if(valorSensor10<100){
  valorSensor10=0;
  int a = valorSensor10;
  arr[0 + 9 * 4] = (a >> 24) & 0xFF;
  arr[1 + 9 * 4] = (a >> 16) & 0xFF;
  arr[2 + 9 * 4] = (a >> 8) & 0xFF;
  arr[3 + 9 * 4] = (a >> 0) & 0xFF;
}
```

```
if(valorSensor10>100){
  int a = valorSensor10;
  arr[0 + 9 * 4] = (a >> 24) & 0xFF;
  arr[1 + 9 * 4] = (a >> 16) & 0xFF;
  arr[2 + 9 * 4] = (a >> 8) & 0xFF;
  arr[3 + 9 * 4] = (a >> 0) & 0xFF;
  valorSensor10=0;
}

if(valorSensor11<100){
  valorSensor11=0;
  int a = valorSensor11;
  arr[0 + 10 * 4] = (a >> 24) & 0xFF;
  arr[1 + 10 * 4] = (a >> 16) & 0xFF;
  arr[2 + 10 * 4] = (a >> 8) & 0xFF;
  arr[3 + 10 * 4] = (a >> 0) & 0xFF;
}

if(valorSensor11>100){
  int a = valorSensor11;
  arr[0 + 10 * 4] = (a >> 24) & 0xFF;
  arr[1 + 10 * 4] = (a >> 16) & 0xFF;
  arr[2 + 10 * 4] = (a >> 8) & 0xFF;
  arr[3 + 10 * 4] = (a >> 0) & 0xFF;
  valorSensor11=0;
}
```

```
 if(valorSensor12<100){
  valorSensor12=0;
  int a = valorSensor12;
  arr[0 + 11 * 4] = (a >> 24) & 0xFF;
  arr[1 + 11 * 4] = (a >> 16) & 0xFF;
  arr[2 + 11 * 4] = (a >> 8) & 0xFF;
  arr[3 + 11 * 4] = (a >> 0) & 0xFF;
 }

 if(valorSensor12>100){
  int a = valorSensor12;
  arr[0 + 11 * 4] = (a >> 24) & 0xFF;
  arr[1 + 11 * 4] = (a >> 16) & 0xFF;
  arr[2 + 11 * 4] = (a >> 8) & 0xFF;
  arr[3 + 11 * 4] = (a >> 0) & 0xFF;
  valorSensor12=0;
 }
```

Una vez terminada la lectura de todos los sensores y una vez guardados en el array lo enviamos por el puerto serie Bluetooth.

```
 for (int i = 0; i < 12 * 4; i++) {
  SerialBT.write(arr[i]);
 }
```

Este comando es el delay que tenemos que esperar hasta hacer la próxima lectura de los sensores y depende del comando que nos haya enviado la aplicación móvil.

```
 delay(sensorDelay);
```

```
}
```

4.3.2 Programación de la aplicación móvil

En este apartado nos centraremos en la programación de la aplicación móvil.

La primera parte del código que vamos a explicar es la relacionada con la pantalla principal, activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
```

Primero introducimos un Lineal Layout para poder colocar todos los elementos en una línea o columna.

```
<android.widget.LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
```

El siguiente bloque es para la parte de la barra de menú o barra de herramientas.

```
   <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:theme="@style/AppTheme.AppBarOverlay"
      android:background="?attr/colorPrimary"
```

```
app:popupTheme="@style/AppTheme.PopupOverlay" />
```

Este bloque se encarga del ***"ConstraintLayout"***, se encarga de posicionar un widget en relación con otro widget según la posición del anterior.

```
<android.support.constraint.ConstraintLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```

Este bloque hace referencia a la imagen que tenemos de fondo, la cual tiene un pie dibujado.

```
<ImageView
  android:id="@+id/imageView"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_marginHorizontal="20dp"
  android:layout_marginTop="90dp"
  android:layout_marginBottom="20dp"
  app:srcCompat="@mipmap/main_background" />
```

Introducimos otro ***"Lineal Layout"*** dentro del que teníamos previamente para poner los valores de los sensores en su posición.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_margin="20dp"
```

```
android:orientation="vertical">
```

Ahora introducimos un ***"Table Layout"*** para posicionar los tres botones

```
<TableLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content">

  <TableRow>

    <Switch
      android:id="@+id/switchTestMode"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/switch_test_mode" />

    <TextView
      android:layout_width="fill_parent"
      android:layout_weight="1" />

    <Button
      android:id="@+id/buttonStart"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_start" />
```

```
    <Button
        android:id="@+id/buttonStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_stop" />
</TableRow>

</TableLayout>
```

Volvemos a introducir un ***"Table Layout"*** para posicionar el ***"Text View"*** que nos muestra el valor exacto de delay que elegiremos con la barra de selección y también la ***"Seek Bar"*** que es la barra de selección con la que cambiamos este delay.

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TableRow>

        <TextView
            android:id="@+id/textDelay"
            android:layout_width="40dp"
            android:layout_height="wrap_content"
            android:text="0" />
```

```
        <SeekBar
            android:id="@+id/seekBarDelay"
            android:layout_width="fill_parent"
            android:layout_weight="1" />
    </TableRow>

</TableLayout>
</LinearLayout>
```

El siguiente bloque lo utilizamos para poner todos los ***"Text View"*** de los valores de los sensores con posiciones relativas con ***"Relative Layout"***

```
<RelativeLayout
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toTopOf="@id/imageView"
    app:layout_constraintBottom_toBottomOf="@id/imageView"
    app:layout_constraintRight_toRightOf="@id/imageView"
    app:layout_constraintLeft_toLeftOf="@id/imageView">

    <TextView
        android:id="@+id/sensorText0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="90dp"
```

```
    android:layout_marginLeft="90dp"
    android:layout_marginTop="50dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="205dp"
    android:layout_marginLeft="205dp"
    android:layout_marginTop="98dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="80dp"
    android:layout_marginLeft="80dp"
    android:layout_marginTop="180dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText3"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="140dp"
    android:layout_marginLeft="140dp"
    android:layout_marginTop="170dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="200dp"
    android:layout_marginLeft="200dp"
    android:layout_marginTop="200dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="140dp"
    android:layout_marginLeft="140dp"
    android:layout_marginTop="285dp"
    android:text="0" />
```

```
<TextView
   android:id="@+id/sensorText6"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginStart="100dp"
   android:layout_marginLeft="100dp"
   android:layout_marginTop="300dp"
   android:text="0" />

<TextView
   android:id="@+id/sensorText7"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginStart="185dp"
   android:layout_marginLeft="185dp"
   android:layout_marginTop="320dp"
   android:text="0" />

<TextView
   android:id="@+id/sensorText8"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginStart="105dp"
```

```
    android:layout_marginLeft="105dp"
    android:layout_marginTop="400dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText9"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="160dp"
    android:layout_marginLeft="160dp"
    android:layout_marginTop="400dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText10"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="160dp"
    android:layout_marginLeft="160dp"
    android:layout_marginTop="500dp"
    android:text="0" />

<TextView
    android:id="@+id/sensorText11"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="100dp"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="500dp"
        android:text="0" />
    </RelativeLayout>

</android.support.constraint.ConstraintLayout>
```

En este punto empieza el código de ***menú_main.xml***, que es el encargado de hacer el menú desplegable. En este menú eliges si quieres ver los diferentes dispositivos que tienes sincronizados para poderte conectar a ellos o si quieres exportar los valores que se han guardado.

```
<menu
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/bluetooth_devices"
        android:title="@string/bluetooth_devices"
        app:showAsAction="never" />

    <item
        android:id="@+id/export_values"
```

```
    android:title="@string/export_values"
    app:showAsAction="never" />

</menu>
```

Este código corresponde a ***"activity_bluetooth_devices.xml"*** este código es el encargado de listar los dispositivos Bluetooth que tienes sincronizados previamente, en el código utilizamos un ***"Lineal Layout"*** para que aparezcan ordenados en forma de columna y el elemento ***"List View"*** es el encargado de de crear la lista con los dispositivos.

```
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay"
        android:background="?attr/colorPrimary"
        app:popupTheme="@style/AppTheme.PopupOverlay" />
```

```
    <android.widget.ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</android.widget.LinearLayout>
```

El siguiente código corresponde a ***"MainActivity.java"***, este fichero es el principal y el código va asociado a la pantalla principal, comenzamos inicializando el pack ***"bluefeet.activities"*** y seguidamente importamos todas las librerías que necesitamos.

```
package com.upct.bluefeet.activities;


import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.SeekBar;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;
```

```
import com.upct.bluefeet.R;

import com.upct.bluefeet.controllers.BluetoothController;

import com.upct.bluefeet.controllers.SensorController;

import com.upct.bluefeet.listeners.BluetoothListener;

import com.upct.bluefeet.listeners.SensorListener;


import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;
```

Inicializamos una clase ***"MainActivity"*** y creamos todas las variables o widgets que necesitamos y creamos ***"BluetoothListener"*** y ***"SensorListener"*** que nos servirá para los sensores y para las acciones que ocurran con el Bluetooth.

```
public class MainActivity extends AppCompatActivity implements
BluetoothListener, SensorListener {


    private Toolbar toolbar;


    private TextView sensorText0;

    private TextView sensorText1;

    private TextView sensorText2;

    private TextView sensorText3;
```

```
private TextView sensorText4;

private TextView sensorText5;

private TextView sensorText6;

private TextView sensorText7;

private TextView sensorText8;

private TextView sensorText9;

private TextView sensorText10;

private TextView sensorText11;


private Button buttonStop;

private Button buttonStart;

private Switch switchTestMode;

private SeekBar seekBarDelay;

private TextView textDelay;
```

Inicializamos los textos de los sensores y el estado inicial de los botones de la pantalla inicial

```
@Override

protected void onCreate(Bundle savedInstanceState) {

   super.onCreate(savedInstanceState);


   BluetoothController.getUniqueStance().addBluetoothListener(this);

   SensorController.getUniqueStance().addSensorListener(this);
```

```
setContentView(R.layout.activity_main);


toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);


sensorText0 = findViewById(R.id.sensorText0);
sensorText0.setText("0");


sensorText1 = findViewById(R.id.sensorText1);
sensorText1.setText("0");


sensorText2 = findViewById(R.id.sensorText2);
sensorText2.setText("0");


sensorText3 = findViewById(R.id.sensorText3);
sensorText3.setText("0");


sensorText4 = findViewById(R.id.sensorText4);
sensorText4.setText("0");


sensorText5 = findViewById(R.id.sensorText5);
sensorText5.setText("0");


sensorText6 = findViewById(R.id.sensorText6);
```

```
sensorText6.setText("0");

sensorText7 = findViewById(R.id.sensorText7);
sensorText7.setText("0");

sensorText8 = findViewById(R.id.sensorText8);
sensorText8.setText("0");

sensorText9 = findViewById(R.id.sensorText9);
sensorText9.setText("0");

sensorText10 = findViewById(R.id.sensorText10);
sensorText10.setText("0");

sensorText11 = findViewById(R.id.sensorText11);
sensorText11.setText("0");

buttonStart = findViewById(R.id.buttonStart);

buttonStart.setClickable(!SensorController.getUniqueStance().isTestMode());
```

La línea anterior pregunta el estado en el que esta y si no está habilitado el modo test se inhabilita el botón.

La siguiente línea es la encargada del escuchador de los botones.

```
buttonStart.setOnClickListener(new View.OnClickListener() {
```

```
@Override
public void onClick(View v) {
    BluetoothController.getUniqueStance().sendStartCommand();
```

En la línea anterior cuando se hace click envía el comando start a la plantilla.

```
}


});
```

En la siguiente línea hacemos lo mismo que en la parte anterior del código con el botón start pero con el botón stop.

```
buttonStop = findViewById(R.id.buttonStop);

buttonStop.setClickable(!SensorController.getUniqueStance().isTestMode());
buttonStop.setOnClickListener(new View.OnClickListener() {


    @Override
    public void onClick(View v) {
        BluetoothController.getUniqueStance().sendStopCommand();
    }


});
```

En la siguiente línea hacemos lo mismo que en la parte anterior del código con el botón start y el stop pero con el botón test.

```
switchTestMode = findViewById(R.id.switchTestMode);

switchTestMode.setChecked(SensorController.getUniqueStance().isTestMode());

switchTestMode.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {

    @Override

    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {

        if (isChecked) {

            BluetoothController.getUniqueStance().sendStartCommand();

            SensorController.getUniqueStance().enableTestMode();

        } else {

            BluetoothController.getUniqueStance().sendStopCommand();

            SensorController.getUniqueStance().disableTestMode();

        }
```

Las líneas siguientes actualizan el estado de los botones.

```
buttonStart.setClickable(!SensorController.getUniqueStance().isTestMode());

buttonStop.setClickable(!SensorController.getUniqueStance().isTestMode());

    }
```

```
});
```

Las siguientes líneas fijan los valores máximos y mínimos de la barra del delay e inicializan los escuchadores de dicha barra

```
seekBarDelay = findViewById(R.id.seekBarDelay);

seekBarDelay.setProgress(10);

seekBarDelay.setMax(300);

seekBarDelay.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {


    @Override

    public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {


BluetoothController.getUniqueStance().sendDelayCommand(progress);


       textDelay.setText(Integer.toString(progress));

    }


    @Override

    public void onStartTrackingTouch(SeekBar seekBar) {


    }


    @Override
```

```
        public void onStopTrackingTouch(SeekBar seekBar) {


        }
    });
```

La siguiente línea es la encargada del valor de la barra.

```
    textDelay = findViewById(R.id.textDelay);
    textDelay.setText("10");
}
```

Las siguientes líneas se encargan de que los escuchadores paren.

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


    BluetoothController.getUniqueStance().removeBluetoothListener(this);
    SensorController.getUniqueStance().removeSensorListener(this);
}
```

Las siguientes líneas de código son las encargadas del menú.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

Estas líneas siguientes se ejecutan cuando se marcan las opciones del menú.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
```

En las siguientes líneas de código mostramos ventana de dispositivos Bluetooth.

```
    if (id == R.id.bluetooth_devices) {
        startActivity(new Intent(getBaseContext(), BluetoothDevicesActivity.class));
        return true;
    }
```

Exportamos los valores a fichero

```
    if (id == R.id.export_values) {
```

Pasa del formato año, mes, día y la siguiente linea llama al objeto date (fecha) y lo pasa a un stream.

```
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd_HHmmss");
        Date actualDate = Calendar.getInstance().getTime();
```

Lo pasa a texto.

```
        String strActualDate = dateFormat.format(actualDate);
```

Busca la carpeta donde va a escribir.

```
File exportDirectory = getBaseContext().getExternalFilesDir("Bluefeet");
```

Si no existe lo crea.

```
if (!exportDirectory.exists()) {
   exportDirectory.mkdirs();
}
```

Crea el fichero en el directorio anterior.

```
File exportFile = new File(exportDirectory, "SENSOR_VALUES_" + strActualDate + ".txt");
```

Recupera el listado de los sensores (buffer)

```
String buffer = SensorController.getUniqueStance().exportValues();
```

Crea un objeto stream que sirve para escribir los btyes, con un puntero que recorre todo en el fichero

```
try {
```

Abre el stream.

```
FileOutputStream outputStream = new FileOutputStream(exportFile);
```

La siguiente línea de texto escribe en el buffer.

```
outputStream.write(buffer.getBytes());
```

Cierra y guarda el buffer

```
outputStream.close();
```

Si hay error manda un mensaje de error o si no lo hay manda un mensaje en el que exporta en la ruta especificada.

```
Toast.makeText(getBaseContext(), "Exportado en '" + exportFile.getAbsolutePath() + "'", Toast.LENGTH_LONG).show();

} catch (IOException ex) {

Toast.makeText(getBaseContext(), "Error en la exportación", Toast.LENGTH_LONG).show();

}


return true;

}


return super.onOptionsItemSelected(item);

}
```

Si hay un mensaje lo envía.

```
@Override

public void showBluetoothMessage(final String message) {

runOnUiThread(new Runnable() {
```

Recibe los valores de los sensores.

```
@Override
```

```
            public void run() {
                Toast.makeText(getBaseContext(), message,
Toast.LENGTH_LONG).show();
            }

        });
    }

    @Override
    public void updateSensorValues(final int[] value) {
        runOnUiThread(new Runnable() {

            @Override
            public void run() {
                sensorText0.setText(String.valueOf(value[0]));
                sensorText1.setText(String.valueOf(value[1]));
                sensorText2.setText(String.valueOf(value[2]));
                sensorText3.setText(String.valueOf(value[3]));
                sensorText4.setText(String.valueOf(value[4]));
                sensorText5.setText(String.valueOf(value[5]));
                sensorText6.setText(String.valueOf(value[6]));
                sensorText7.setText(String.valueOf(value[7]));
                sensorText8.setText(String.valueOf(value[8]));
                sensorText9.setText(String.valueOf(value[9]));
```

```
                sensorText10.setText(String.valueOf(value[10]));
                sensorText11.setText(String.valueOf(value[11]));
            }

        });
    }

}
```

Ahora se explicará el código de ***"BluetoothDevicesActivity.java"***, como siempre empezamos incluyendo las librerías y paquetes que vamos a utilizar.

```
package com.upct.bluefeet.activities;

import android.bluetooth.BluetoothDevice;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;

import com.upct.bluefeet.R;
import com.upct.bluefeet.adapters.BluetoothDevicesAdapter;
```

```
import com.upct.bluefeet.controllers.BluetoothController;

import com.upct.bluefeet.listeners.BluetoothListener;
```

Se crea una clase ***"BluetoothDevicesActivity"*** y se implementa un escuchador para el Bluetooth.

```
public class BluetoothDevicesActivity extends AppCompatActivity implements BluetoothListener {


    private Toolbar toolbar;


    private ListView listView;
```

En las siguientes líneas se crea una lista con los dispositivos Bluetooth que el teléfono tiene vinculados utilizando escuchadores Bluetooth.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);


        BluetoothController.getUniqueStance().addBluetoothListener(this);


        setContentView(R.layout.activity_bluetooth_devices);


        toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);

getSupportActionBar().setDisplayShowHomeEnabled(true);


listView = findViewById(R.id.listView);

listView.setAdapter(new BluetoothDevicesAdapter(getBaseContext(),
BluetoothController.getUniqueStance().getBondedDevices()));

listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
```

Cuando se hace click en uno de los dispositivos y se selecciona de la lista.

```
@Override

public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {

BluetoothDevice device = (BluetoothDevice)
parent.getItemAtPosition(position);


BluetoothController.getUniqueStance().selectBluetoothDevice(device);

}


});

}
```

Se procede a parar el escuchador.

```
@Override

protected void onDestroy() {
```

```
        super.onDestroy();


        BluetoothController.getUniqueStance().removeBluetoothListener(this);
    }
```

Si presionamos el botón back volveremos al menú anterior.

```
    @Override
    public boolean onSupportNavigateUp() {
        onBackPressed();
        return true;
    }
```

Si existe algún error con el Bluetooth nos lo mostrará en la pantalla.

```
    @Override
    public void showBluetoothMessage(final String message) {
        runOnUiThread(new Runnable() {


            @Override
            public void run() {
                Toast.makeText(getBaseContext(), message,
Toast.LENGTH_LONG).show();
            }


        });
```

```
    }

}
```

A continuación explicaremos el código de ***"BluetoothDevicesAdapter.java"***, como en los códigos anteriores comenzamos importando librerías y paquetes.

```
package com.upct.bluefeet.adapters;

import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import java.util.LinkedList;
import java.util.Set;
```

Creamos la clase ***"BluetoothDevicesAdapter"***, se crea un listado genérico y como mostrar cada ítem.

```
public class BluetoothDevicesAdapter extends ArrayAdapter<BluetoothDevice> {
```

```
    public BluetoothDevicesAdapter(Context context, Set<BluetoothDevice> devices) {
        super(context, android.R.layout.simple_list_item_2, new LinkedList<BluetoothDevice>(devices));
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view;

        if (convertView == null) {
            LayoutInflater inflater = (LayoutInflater) getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            view = inflater.inflate(android.R.layout.simple_list_item_2, null);
        } else {
            view = convertView;
        }

        TextView text1 = view.findViewById(android.R.id.text1);
        TextView text2 = view.findViewById(android.R.id.text2);

        BluetoothDevice device = super.getItem(position);
        text1.setText(device.getName());
        text2.setText(device.getAddress());
```

```
        return  view;
    }

}
```

Seguimos con el código ***"BluetoothController"***, primero importamos librerías y paquetes.

```
package com.upct.bluefeet.controllers;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;

import com.upct.bluefeet.listeners.BluetoothListener;
import com.upct.bluefeet.threads.BluetoothThread;

import java.util.LinkedList;
import java.util.List;
import java.util.Set;
import java.util.UUID;
```

Creamos la clase ***"BluetoothController"***, creamos las variables.

```
public class BluetoothController {

    private static BluetoothController uniqueStance;
```

El listener avisa cuando cambia algo de la lista Bluetooth.

```
private List<BluetoothListener> bluetoothListeners;
```

Crea una lista vacía.

```
private BluetoothThread bluetoothThread;


private BluetoothController() {
    bluetoothListeners = new LinkedList<>();
}
```

Llama al adaptador Bluetooth y te devuelve el número de dispositivos ya vinculados

```
public Set<BluetoothDevice> getBondedDevices() {
    return BluetoothAdapter.getDefaultAdapter().getBondedDevices();
}
```

Selecciona el dispositivo

```
public void selectBluetoothDevice(BluetoothDevice device) {
```

Si no existe el hilo del dispositivo al que me quiero conectar es porque me quiero conectar a la plantilla.

```
    if (bluetoothThread == null) {
        connectBluetoothDevice(device);
    } else {
```

Si existe un hilo lo que quiero es desconectarme.

```
            disconnectBluetoothDevice();
        }
    }

    public void connectBluetoothDevice(BluetoothDevice device) {
        UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
```

Si no estoy conectado al dispositivo creo un hilo y es universal y unico ID (nombre de conexión)

```
        if (bluetoothThread == null) {
            bluetoothThread = new BluetoothThread(device, uuid);
            bluetoothThread.start();
        }
    }

    public void disconnectBluetoothDevice() {
        if (bluetoothThread != null) {
            bluetoothThread.close();
            bluetoothThread = null;
        }
    }
```

Función a la que se llama cada vez que se cambia el switch

```
public void switchTestMode() {
    sendStopCommand();

    if (SensorController.getUniqueStance().isTestMode()) {
        SensorController.getUniqueStance().disableTestMode();
    } else {
        SensorController.getUniqueStance().enableTestMode();
    }
}
```

Manda el comando stop

```
public void sendStopCommand() {
    if (bluetoothThread != null) {
        byte[] buffer = { 0x02 };
        bluetoothThread.send(buffer);
    }
}
```

Manda comando start

```
public void sendStartCommand() {
    if (bluetoothThread != null) {
        byte[] buffer = { 0x01 };

        bluetoothThread.send(buffer);
```

```
    }
  }
```

Manda comando delay

```
  public void sendDelayCommand(int delay) {
    if (bluetoothThread != null) {
      byte b0 = (byte) ((delay >> 24) & 0xFF);
      byte b1 = (byte) ((delay >> 16) & 0xFF);
      byte b2 = (byte) ((delay >>  8) & 0xFF);
      byte b3 = (byte) ((delay >>  0) & 0xFF);

      byte[] buffer = { 0x03, b0, b1, b2, b3 };

      bluetoothThread.send(buffer);
    }
  }

  public void addBluetoothListener(BluetoothListener bluetoothListener) {
    bluetoothListeners.add(bluetoothListener);
  }
```

Permitir que una ventana se suscriba a los cambios.

```
  public void removeBluetoothListener(BluetoothListener
bluetoothListener) {
```

```
        bluetoothListeners.remove(bluetoothListener);
    }
```

Eliminar lo anterior.

```
    public void showBluetoothMessage(String message) {
        for (BluetoothListener bluetoothListener : bluetoothListeners) {
            bluetoothListener.showBluetoothMessage(message);
        }
    }
```

Recorre la lista de suscritos y le manda la notificación instanciación perezosa, solo llama una sola vez, una vez que se crea ***"BluetoothControler"*** ya no se va a definir más y pasará a return. Siempre llamará a la instancia creada.

```
    public static BluetoothController getUniqueStance() {
        if (uniqueStance == null) {
            uniqueStance = new BluetoothController();
        }

        return uniqueStance;
    }

}
```

El siguiente código corresponde a ***"SensorController.java"***, como siempre comenzamos importando los paquetes y las librerías.

```
package com.upct.bluefeet.controllers;

import com.upct.bluefeet.listeners.SensorListener;

import java.util.LinkedList;
import java.util.List;
```

Creamos una instancia única.

```
public class SensorController {

    private static SensorController uniqueStance;
```

Creamos una lista escuchadora única.

```
    private List<SensorListener> sensorListeners;
```

Guarda el array y lee como valor entero.

```
    private boolean testMode;
```

Creamos una variable de on/off /modo test.

```
    private List<int[]> valueList;
```

Inicializamos el controlador del sensor.

```
    private SensorController() {
```

Inicializamos la lista de valores.

```
    sensorListeners = new LinkedList<>();
    valueList = new LinkedList<>();
}
```

Esta función nos sirve para saber en qué estado estará el switch (test o normal)

```
public boolean isTestMode() {
    return testMode;
}
```

Iniciamos el modo test.

```
public void enableTestMode() {
    this.testMode = true;
    valueList.clear();
}
```

Desactivamos el modo test.

```
public void disableTestMode() {
    this.testMode = false;
    valueList.clear();
}
```

Una vez conectado, parsevalues recibe el array y lo cambia de byte a entero. Cuando termina de llenar new int buffer lo mete en valuelist que es como el histórico updatesensorvalues avisa a los afiliados a esa lista.

```
public void parseValues(byte[] buffer) {
    int[] value = new int[buffer.length / 4];

    for (int i = 0; i < value.length; i++) {
        value[i] = 0;
        value[i] += (buffer[i * 4 + 0] & 0xFF) << 24;
        value[i] += (buffer[i * 4 + 1] & 0xFF) << 16;
        value[i] += (buffer[i * 4 + 2] & 0xFF) << 8;
        value[i] += (buffer[i * 4 + 3] & 0xFF) << 0;
    }
```

Cuando no esté en modo test guardará.

```
    if (!testMode) {
        valueList.add(value);
    }

    updateSensorValues(value);
}
```

Es la función encargada de exportar los valores a un fichero.

```
public String exportValues() {
    StringBuffer buffer = new StringBuffer();

    for (int[] values : valueList) {
```

```
        for (int value : values) {
```

Lee un valor y pone un punto y coma " ; ".

```
            buffer.append(Integer.toString(value) + "\t");
        }
```

Cuando termina salto de línea y retorno de carro

```
        buffer.append("\n");
    }

    return buffer.toString();
}
```

Crea suscriptores a esta lista.

```
public void addSensorListener(SensorListener sensorListener) {
    sensorListeners.add(sensorListener);
}
```

Elimina suscriptores.

```
public void removeSensorListener(SensorListener sensorListener) {
    sensorListeners.remove(sensorListener);
}
```

Envía los mensajes a los listener.

```
    public void updateSensorValues(int[] value) {
        for (SensorListener sensorListener : sensorListeners) {
            sensorListener.updateSensorValues(value);
        }
    }
```

Si no está creada la instancia la creamos y si esta creada pasa a ser perezosa.

```
    public static SensorController getUniqueStance() {
        if (uniqueStance == null) {
            uniqueStance = new SensorController();
        }

        return uniqueStance;
    }

}
```

El siguiente código pertenece a ***"BluetoothListener.java"***, es el escuchador del Bluetooth.

```
package com.upct.bluefeet.listeners;

public interface BluetoothListener {
```

La clase que quiera implementar suscribirse a los mensajes de Bluetooth deberán de llamar a esta función.

```
    void showBluetoothMessage(String message);
```

}

Para el siguiente código para el escuchador es el ***"SensorListener.java"***, este es para los sensores.

package com.upct.bluefeet.listeners;

public interface SensorListener {

La clase que quiera implementar suscribirse a los mensajes de sensor deberán de llamar a esta función

void updateSensorValues(int[] value);

}

Y por último vamos a explicar el ***"BluetoothTread.java"***, primero importamos los paquetes y librerías.

package com.upct.bluefeet.threads;

El hilo que se ejecuta en segundo plano y se encarga de escuchar al Bluetooth.

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothSocket;

import com.upct.bluefeet.controllers.BluetoothController;

import com.upct.bluefeet.controllers.SensorController;

import java.io.IOException;

import java.io.InputStream;

```
import java.io.OutputStream;

import java.util.UUID;


public class BluetoothThread extends Thread {


   private BluetoothDevice device;
```

El socket es el que se encarga de conectar un punto con otro.

```
   private BluetoothSocket socket;
```

Esta línea es el constructor del hilo, dispositivo y el UUID

```
   private UUID uuid;


   public BluetoothThread(BluetoothDevice device, UUID uuid) {
      this.device = device;
      this.uuid = uuid;
   }
```

Opción de enviar (para los comandos del Arduino).

```
   public void send(byte[] buffer) {
      try {
         OutputStream outputStream = socket.getOutputStream();
         outputStream.write(buffer);
      } catch (IOException ex) {
         BluetoothController.getUniqueStance().disconnectBluetoothDevice();
         BluetoothController.getUniqueStance().showBluetoothMessage("Error en
la comunicacion");
      }
```

```
}
```

El siguiente código es para que no exista un desbordamiento.

```
public void run() {
    try {
```

Abre una conexión por el número único.

```
        socket = device.createInsecureRfcommSocketToServiceRecord(uuid);
```

Busca el dispositivo.

```
        BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
```

Se conecta.

```
        socket.connect();
```

Si todo ha ido bien mostrará un mensaje de "Dispositivo conectado".

```
BluetoothController.getUniqueStance().showBluetoothMessage("Dispositivo
conectado");
```

El buffer, longitud del buffer y el offser es lo que yo he leído, 12 sensores por 4 Bytes son igual a 48.

```
        int offset = 0;
        int length = 48;
        byte[] buffer = new byte[48];
```

En la siguiente línea lee el ***"inputstream"***.

```
        InputStream inputStream = socket.getInputStream();
```

Lee todo lo que manda la plantilla.

```
while (socket.isConnected()) {
```

Si existe cualquier problema o se aleja un dispositivo de otro se desconecta y manda un mensaje de error.

```
try {
```

El cable de lectura lee y lo manda al buffer, desde que el buffer está en la posición cero hasta que llega a la posición 48.

```
int size = inputStream.read(buffer, offset, length);
```

Si se ha conseguido leer se escribe en la posición en la que se ha quedado.

```
if (size != -1) {
   offset += size;
   length -= size;
}
```

Una vez termine de leer el buffer llama al sensor y manda el buffer y reinicia todo.

```
if (length == 0) {
   SensorController.getUniqueStance().parseValues(buffer);

   offset = 0;
   length = 48;
}
} catch (IOException ex) {
   BluetoothController.getUniqueStance().disconnectBluetoothDevice();
   BluetoothController.getUniqueStance().showBluetoothMessage("Error en la comunicacion");
}
}
```

Si se desconecta de manera correcta no saldrá por el catch y no mostrará un mensaje de error.

```
    } catch (IOException ex) {
        BluetoothController.getUniqueStance().disconnectBluetoothDevice();
        BluetoothController.getUniqueStance().showBluetoothMessage("Error en la conexion con el dispositivo");
    }
  }

  public void close() {
    BluetoothController.getUniqueStance().showBluetoothMessage("Dispositivo desconectado");

    if (socket != null) {
        try {
            socket.close();
        } catch (IOException ex) {
            BluetoothController.getUniqueStance().showBluetoothMessage("Error en la desconexion");
        }
    }
  }

}
```

Capítulo 5

Resultados, conclusiones y trabajo futuro

5.1 Resultados.

Los resultados obtenidos en la aplicación son bastante prometedores. Viendo los resultados de las graficas obtenidas en las pruebas del capítulo 4, los sensores dan buenos resultados con mi pisada y sabiendo que zonas de mi pie son las que ejercen mayor presión sobre la plantilla comparando con plantillas de zapatillas ya usadas. Por otro lado se debe destacar que estos sensores deberían de tener un rango máximo de 20kg por sensor y en las pruebas realizadas en el capítulo 4 se puede observar que a partir de los 5kg el sensor siempre da una tensión constante e igual aunque ejerzamos en él mayor presión.

5.2 Conclusiones

En conclusión se podría decir que el funcionamiento de esta plantilla es el esperado y todos los valores que se obtienen de la aplicación móvil con la supervisión y el análisis de un experto en la pisada podrían hacer que esta herramienta tuviera una aceptación bastante buena, ya que en el mercado no existe ningún producto que pueda tenga una aplicación como la que se ha creado, de manera inalámbrica y con una interfaz sobre un terminal móvil inteligente.

5.3 Trabajo futuro

En una futura modificación se podría crear una visualización previa en el propio dispositivo móvil que graficara los resultados obtenidos en ese mismo instante y los guardados en el histórico.

Otra modificación que se podría considerar sería la de entrenar al sistema con inteligencia artificial y que la propia aplicación te dijera el tipo de pisada o si en tu pisada existe algún tipo de problema.

Bibliografía

[Baropodometría] https://www.google.com/search?q=definici%C3%B3n+de+baropodometr%C3%ADa&spell=1&sa=X&ved=0ahUKEwj_mJuAleHjAhUHcBQKHdwADOcQBQgtKAA&biw=780&bih=786.

[Pie supinador] https://www.webconsultas.com/ejercicio-y-deporte/medicina-deportiva/tipos-de-pisada.

[Pie pronador] https://www.webconsultas.com/ejercicio-y-deporte/medicina-deportiva/tipos-de-pisada.

[Pie neutro] https://www.webconsultas.com/ejercicio-y-deporte/medicina-deportiva/tipos-de-pisada.

[Puente] https://residenciasalcalamahora.wordpress.com/2013/07/20/deformidades-de-los-pies-pies-planos/.

[Zonas del pie] https://www.partesdel.com/pie_humano.html.

[Pie Cavo] https://www.elsevier.es/es-revista-revista-espanola-reumatologia-29-articulo-alteraciones-boveda-plantar-13055069.

[Ortopedianeo] https://www.ortopedianeo.com/baropodometro-computarizado/index.html.

[PDM] https://www.researchgate.net/figure/Figura-1-Sistema-de-baropodometria-electronica-PDM-240-componentes_fig1_39426889.

[Smart] https://www.podoactiva.com/es/blog/la-baropodometria-no-engana?utm_referrer=https%3A%2F%2Fwww.google.com%2F.

[Ulceras] http://www.unciencia.unc.edu.ar/2017/marzo/idean-una-plantilla-para-evitar-la-formacion-de-ulceras-en-los-pies-de-personas-diabeticas.

[Estudio PDM 240] https://fondoscience.com/sites/default/files/articles/pdf/rpt.1101.fs970607-sistema-electronico-portatil-pdm-240.pdf

[FSR 26] https://es.aliexpress.com/item/32960696900.html?spm=a2g0o.cart.0.0.707d3c00tuiWOx.

[Curvas 26 mm] https://es.aliexpress.com/item/32960696900.html?spm=a2g0o.cart.0.0.707d3c00tuiWOx.

[Circuito 26 mm] https://es.aliexpress.com/item/32960696900.html?spm=a2g0o.cart.0.0.707d3c00tuiWOx.

[FSR 9] https://www.amazon.es/Resistencia-Detecci%C3%B3N-Altamente-Precisa-Pel%C3%ADCula/dp/B07D6GFQ5Y/ref=sr_1_1?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=force%2Bresistive%2Bsensor&qid=1564733164&s=gateway&sr=8-1&th=1.

[Red Bear] https://www.comunicacionesinalambricashoy.com/tarjeta-desarrollo-ble-iot/.

[ESP32] https://www.prometec.net/instalando-esp32/.

[ESP8266] https://github.com/jaimelaborda/Planta-Twittera/wiki/1.-Introducci%C3%B3n-al-ESP8266-y-NodeMCU.

[MSP430] https://es.wikipedia.org/wiki/MSP430.

[Smartphone] https://www.pcworld.es/articulos/smartphones/iphone-vs-android-cuota-de-mercado-3692825/.

[Amplificador] https://es.wikipedia.org/wiki/Amplificador_operacional#Amplificador_operacional_real.

[Multiplexor] https://www.prometec.net/multiplexor-74hc4067/.

[DipTrace] https://diptrace.com/es/.