Universidad Zaragoza
1542

Trabajo Fin de Grado

SISTEMA DE MONITORIZACIÓN PARA LA DETECCIÓN DE LA COJERA DE UN CABALLO

Monitoring system for the detection of a horse's lameness

Autor

Pedro Gracia Guillermo

Director

Roberto Casas Nebra

Escuela de Ingeniería y Arquitectura

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Pedro Gracia Guillermo ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado (Título del Trabajo)

Sistema de monitorización para la detección de la cojera de un caballo

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 16 de noviembre de 2020

Fdo:

Resumen

El objetivo de este TFG es el desarrollo de un sistema de monitorización de la cojera de un caballo. Para ello se ha diseñado tanto el software como el hardware de un prototipo capaz de obtener información sobre la cojera de un caballo con varios sensores y enviar estos datos vía wifi a un ordenador donde se almacenarán y se mostrarán a través de unas gráficas.

El presente documento comienza con una introducción relatando la problemática a la hora de detectar la cojera de los caballos y de algunas de las posibles soluciones planteadas por algunos de los investigadores de todo el mundo. Se continúa, detallando el principio de funcionamiento de la tecnología existente hoy en día. Por último, se plantea una posible solución en la que se incluye tanto el diseño hardware (que contiene el diseño del esquemático, de la PCB y la elección de los componentes electrónicos) como la realización del software (que incluye la programación realizada en Arduino para enviar datos desde un microcontrolador y la programación realizada en Python para recibir estos datos, de tantos microcontroladores como haya conectados, graficarlos y guardarlos).

Para su realización, ha sido necesario tanto del conocimiento adquirido de las diferentes asignaturas del grado como de la información hallada en algunas fuentes de internet y de algunos libros de texto.

Abstract

The aim of this TFG is to develop a system for monitoring the lameness of a horse. For this purpose, both the software and hardware of a prototype capable of obtaining information about the lameness of a horse with several sensors have been designed and send this data via wifi to a computer where it will be stored and displayed through some graphics.

This document begins with an introduction relating the problem of detecting lameness in horses and some of the possible solutions proposed by some of the researchers from around the world. It goes on to detail the working principle of the technology that exists today. Finally, a possible solution is proposed which includes both the hardware design (which contains the design of the schematic, the PCB and the choice of the electronic components) and the creation of the software (which includes the programming done in Arduino to send data from a microcontroller and the programming done in Python to receive this data from as many microcontrollers as there are connected, graph it and save it).

To do this, it has been necessary both from the knowledge acquired from the different subjects of the degree and from the information found in some internet sources and some textbooks.

Índice

1. Introducción

1.1. Motivación y origen del proyecto

El origen de este proyecto está en la elección y posterior realización de la asignatura optativa de Laboratorio de diseño electrónico.

En esta asignatura se realizan dos pequeños proyectos donde en cada uno de estos se utilizan los conocimientos de las asignaturas relacionadas con la electrónica para realización de un diseño real, con la elección de los componentes y posterior montaje en una placa de circuito impreso.

La satisfacción a la hora de realizar esta asignatura ha llevado a la realización de este trabajo fin de grado.

1.2. Marco del proyecto

Los caballos de carreras son un bien muy preciado, muy valioso, pero estos son especialmente susceptibles a la cojera. Según los expertos existe un porcentaje relativamente elevado de los caballos que se lesionan en algún momento dado, razón por la que son incapaces de alcanzar su máximo potencial. Las pérdidas anuales en la industria de los caballos por estas lesiones pueden alcanzar anualmente los mil millones de dólares, únicamente en Estados Unidos (Allan, 2008).

Un estudio realizado por John B.Kaneene, Whitney A.Ross y RoseAnn Miller para analizar los problemas de salud más frecuentes en los caballos, llegó a la conclusión, de que, la cojera es el problema de salud observado con mayor frecuencia. Para ello, se examinó a 2.469 caballos durante 2 años desde 1992 (Kaneene, 1997).

Para comprender mejor la cojera, el rendimiento en la pista y la locomoción equina es necesario conocer información sobre los tipos y cantidades de fuerzas aplicadas por el caballo en la pista (Ratzlaff, 1987), de ahí nace la necesidad de utilizar sistemas complementarios.

Dado el gran desarrollo que ha tenido la tecnología en las últimas décadas, desde hace unos años se ha estado investigando entre otras cosas, sobre sistemas que evalúan la cojera en la marcha de los caballos. Dichos sistemas harían más fácil el entrenamiento, la coordinación, el desarrollo potencial, en definitiva, el rendimiento de los caballos, así como el tiempo del diagnóstico y la eficacia y exactitud en el tratamiento y la rehabilitación de estos.

Realizar una serie de pruebas para la evaluación de los caballos en una instalación equipada con cintas de correr, cámaras de vídeo, ordenadores, etc. es costoso tanto económicamente como en tiempo. Además, es difícil extrapolar estos resultados al problema real del caballo en el campo. Por tanto, estas pruebas no se suelen realizar en un laboratorio, sino que las realiza un veterinario subjetivamente en la misma cuadra, con el consiguiente problema de inexactitud del resultado.

Una razón por la cual es complicado evaluar y diagnosticar una cojera de manera objetiva, es debido a que los caballos, a distinción de los seres humanos, no son capaces de expresar con facilidad a los veterinarios sus dolencias, ni la situación, ni el lugar ni el momento en el que se produjo el daño (Allan, 2008).

Un estudio realizado en 2010 sobre una muestra reducida de 131 caballos, basado en la subjetividad de los veterinarios, analizó la cojera de estos y demostró lo siguiente: en el caso de que los caballos tuvieran cojeras graves, los veterinarios se ponían de acuerdo en un 93.1%, sin embargo, si la cojera era algo más leve el acuerdo se reducía al 61,9% y si la cojera era muy leve, el acuerdo bajaba al 51,6%.

Como resultado, se concluye en este estudio que ante una cojera leve la evaluación subjetiva no es fiable, y se potencia la búsqueda y el desarrollo de unos métodos objetivos y fiables. (Keegan, 2010)

Una mejora complementaria sería desarrollar sensores inerciales inalámbricos para conseguir una detección más rápida y simple. Esta solución reduciría el nivel de error producido por la subjetividad de los veterinarios y el alto coste de realizar pruebas en un laboratorio. Según un estudio realizado en 2012 estos sensores serían capaces de identificar la lesión a un nivel más bajo de cojera que lo consensuado por 3 veterinarios (McCracken, 2012). Aunque teniendo en cuenta la simpleza e imprecisión de este sensor inercial, seguiría siendo recomendable que la lectura de este sensor fuera complementada con el conocimiento de un veterinario.

Otra posibilidad sería desarrollar sensores de presión que van introducidos mediante adhesivos dentro de las herraduras de los caballos. Estos sensores transmiten una señal de medición a través de un amplificador transmisor, hasta la estación de evaluación. Con esta señal se podrá determinar la distribución de la presión en una sola pezuña o de la comparación de varias pezuñas entre sí. Por último, cabe decir que la precisión de este dispositivo depende especialmente del esfuerzo de la ingeniería mecánica en la fabricación de los elementos y del uso adecuado de los materiales (Ritzinger, 2003).

Otros investigadores han utilizado placas de detección de fuerza instaladas a lo largo de una pista para detectar las fuerzas locomotoras aplicadas. Aunque el resultado no fue muy satisfactorio debido a que la rigidez y dureza relativamente superiores en la placa en relación con la pista hace que la pisada sea más incómoda para los caballos y los resultados sean menos fiables. Incluso se ha llegado a dar el caso en el que los caballos al conocer la posición de estas placas llegaban a esquivarlas, con la consiguiente desviación en los resultados.

Otros han tratado de desarrollar unas herraduras diseñadas específicamente para detectar la fuerza aplicada. Estas herraduras se unían mediante pernos a una placa de fuerza. El inconveniente de usar estas herraduras era su peso y altura, esto hacía que el caballo no trotara con normalidad, alterando el movimiento anatómico de la pata del caballo, y como consecuencia los resultados obtenidos no correspondían fielmente a la realidad (Ratzlaff, 1987).

1.3. Objetivos

El objetivo final de este TFG es el desarrollo de un sistema de monitorización de la cojera de un caballo. Y para conseguir este objetivo ha sido necesario llevar a cabo otros objetivos más pequeños como:

La elección de unos componentes electrónicos donde se ha tenido en cuenta la eficiencia, precisión y el precio. Dando prioridad a la precisión al tratarse de un dispositivo de medida, véase Capitulo 3.2.

El diseño del esquema electrónico (donde se indica la correcta conexión de los diferentes componentes), véase Capitulo 3.3. Y el diseño de la placa de circuito impreso (donde se realizan realmente las conexiones), véase Capítulo 3.4.

La programación en Arduino para que el microcontrolador (ESP32) pueda conectarse a una red wifi en común con un ordenador y enviar constantemente los valores de los diferentes sensores vía wifi a este ordenador, véase Capítulo 4.1.

La programación en Python para recolectar estos datos, guardarlos y graficarlos. Estos datos serán recogidos de tantos microcontroladores como haya conectados, véase Capítulo 4.2.

1.4. Planificación temporal

Para llevar a cabo los objetivos comentados previamente, se ha realizado una serie de tareas más genéricas. Las cuales son: buscar información en fuentes de internet o libros, redactar la memoria, diseñar tanto el esquemático como la placa de circuitos impresos, etc.

Por último, se expondrá la planificación temporal para realizar estas tareas, con un diagrama de Gantt.

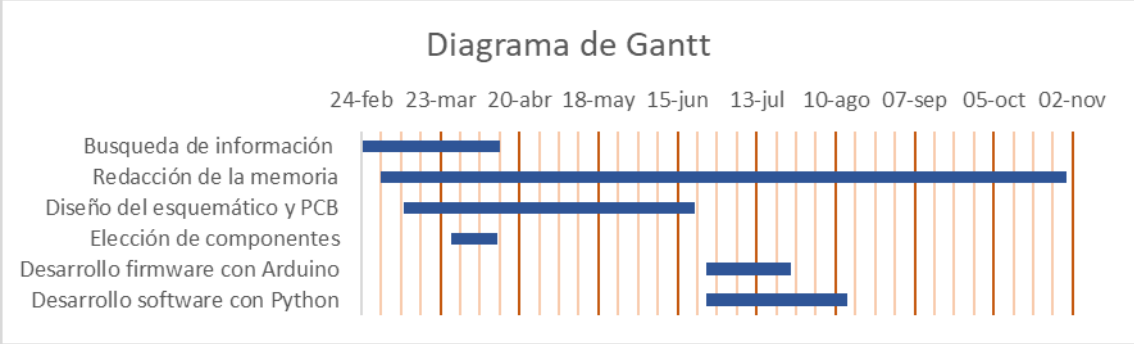


Figura 1.4.1 Diagrama de Gantt (planificación en días)

2. Principio de funcionamiento

En este apartado se exponen los fundamentos básicos de los diferentes sensores que podrían ser utilizados para detectar la cojera de un caballo. Para la explicación de este apartado nos hemos apoyado en los libros (Pérez, 2014), (Pérez, 2006) y (Pallás, 2003), cuyas referencias están indicadas en la bibliografía.

2.1. Galgas extensiométricas

Las galgas extensiométricas son sensores basados en la variación de la resistencia eléctrica al someterse a un esfuerzo mecánico.

El valor de la resistencia eléctrica de un hilo metálico es directamente proporcional a su resistividad y longitud e inversamente proporcional a su sección.

$$R = \rho \frac{l}{A} = \rho \frac{4l}{\pi d^2} \qquad \text{(Ec. 2.1.1)}$$

Tomando logaritmos neperianos y diferenciando se obtiene:

$$\Delta R/R = \frac{\Delta \rho}{\rho} + \frac{\Delta l}{l} - 2\frac{\Delta d}{d} \qquad \text{(Ec. 2.1.2)}$$

Por otra parte, la ley de Poisson dice que al someter a un material a un esfuerzo transversal se produce una variación tanto en la longitud como en la sección. Esta relación de la variación viene dada en la siguiente expresión:

$$\nu = -\frac{\Delta d/d}{\Delta l/l} \qquad \text{(Ec. 2.1.3)}$$

donde, ν es el coeficiente de Poisson.

Por otro lado, para los metales, los cambios porcentuales de sus volúmenes y resistividades son proporcionales, con una constante C, denominada constante de Bridgman, cuyo valor depende del material utilizado.

$$\frac{\Delta \rho}{\rho} = C\frac{\Delta V}{V} \qquad \text{(Ec. 2.1.4)}$$

De la misma manera que hemos procedido anteriormente para la resistencia eléctrica, ahora lo haremos para el volumen (tomando logaritmos neperianos y diferenciando) y se obtiene:

$$\frac{\Delta \rho}{\rho} = C\left(\frac{\Delta l}{l} + 2\frac{\Delta d}{d}\right) \qquad \text{(Ec. 2.1.5)}$$

Por último, si sustituimos la ecuación (2.1.5) y la (2.1.3) en la expresión (2.1.2) queda:

$$\Delta R/_R = [1 + 2\nu + C(1 - 2\nu)]\frac{dl}{l} = K\varepsilon \qquad \text{(Ec. 2.1.6)}$$

donde K es el factor de sensibilidad de la galga y ε es la deformación unitaria.

Esta fórmula también se puede enunciar de la siguiente forma:

$$R = R_0\,(1 + x) \qquad \text{(Ec. 2.1.7)}$$

donde R0 es el valor de la resistencia eléctrica inicial y x = kε.

Actualmente, las galgas están constituidas por un hilo puesto en zig-zag tal como se muestra en la figura (2.1.1), para reducir al máximo su tamaño, además suelen estar sobre una lámina flexible de algún tipo de plástico para estar más protegidas.

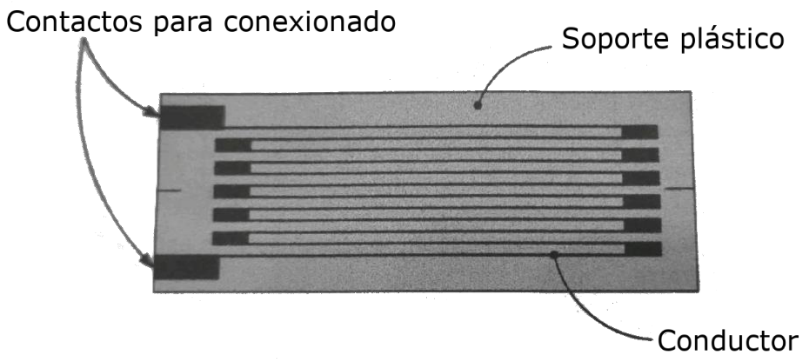


Figura 2.1.1 Galga Extensiométrica, extraída de (Pérez, 2014)

Un problema es que, aunque nos interesa medir los esfuerzos en una única dirección, en realidad también se están teniendo en cuenta los esfuerzos de la dirección perpendicular, y por tanto se está introduciendo un pequeño error. Para reducirlo, se aumenta la sección en los extremos de las galgas, reduciendo así la sensibilidad para los esfuerzos transversales.

Otro de los problemas de las galgas es debido a que también son sensibles a la temperatura, igual que cualquier otro metal, introduciendo así otro pequeño error.

Por último, cabe decir que las galgas se pueden adherir directamente al lugar donde se desea medir la deformación, o se puede utilizar una celda de carga la cuál transmite a la galga la deformación o esfuerzo producido en éste.

2.2. Las celdas de carga

Las celdas de carga son un dispositivo constituido por una pieza mecánica donde se produce una pequeña deformación y ésta es detectada por un puente de galgas con dos o cuatro galgas extensiométricas.

La elección de los parámetros de la celda de carga son clave ya que ésta puede estar sometida a unos esfuerzos importantes y puede llegar a romperse. Entre otros parámetros se debe tener en cuenta el límite máximo de carga, el límite de sobrecarga segura y el valor de carga nominal.

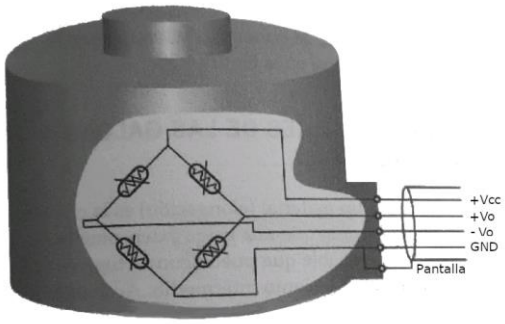


Figura 2.2.1 celda de carga, extraída de (Pérez, 2014)

En el mercado existen diversas realizaciones de células de carga con formas distintas. Un ejemplo se puede ver en la figura (2.2.1), en la que en su interior están adheridas las galgas extensiométricas y, además, tiene los terminales de alimentación y de la señal de la lectura accesibles desde el exterior para poder conectarlos a las etapas posteriores.

2.3. Sensores piezo-resistivos

Este efecto piezo-resistivo trata de la variación de la resistencia eléctrica debido exclusivamente al término de la resistividad $\Delta\rho/\rho$. Este efecto lo poseen tanto los materiales conductores (los metales) como los semiconductores. Sin embargo. el efecto en los conductores es minúsculo, mientras que en los semiconductores es significativo. Por esta razón, los sensores con materiales semiconductores suelen llamarse piezo-resistencias.

Estas galgas piezo-resistivas generan un nivel de señal muy superior al generado por las galgas metálicas, de hecho, en la mayor parte de los casos no es necesaria la utilización de un amplificador. En cambio, tienen la desventaja de tener una linealidad baja y una elevada dependencia con la temperatura.

Hoy en día, las galgas que más se utilizan para medir una fuerza o una deformación son las metálicas, aunque también se utilizan las semiconductoras. Éstas son mayormente utilizadas para tomar medidas de presión, tanto absoluta, como diferencial, manométrica y de vacío de un líquido o un gas.

3. Diseño de hardware

En este apartado se procede a explicar de forma general los diferentes bloques del diseño electrónico realizado. Posteriormente, se comenta la comparativa especifica que se ha llevado a cabo de cada componente para las diferentes alternativas existentes, mostrando finalmente la decisión tomada, en la cual, se ha tenido presente el objetivo de obtener un dispositivo más barato, preciso y eficiente. Ya que se trata de un aparato de medida, se ha hecho un especial hincapié en la precisión. Después, se muestra el esquema electrónico completo del dispositivo. También, se explica un poco el diseño de la PCB. Por último, se comenta el montaje realizado.

3.1. Ideas generales

3.1.1. Los sensores

Los sensores finalmente utilizados para la detección de la cojera han sido las galgas extensiométricas, complementadas por un sensor inercial muy completo como es el BNO055 que contiene un acelerómetro, un giróscopo y un magnetómetro.

A partir de la variación resistiva de las galgas extensiométricas se ha necesitado obtener una tensión en un margen útil para el convertidor analógico digital, incluido en el microcontrolador ESP32. Para ello se ha diseñado una etapa acondicionadora de la señal, esta etapa está constituida por un puente de Wheatstone y un amplificador de instrumentación. Tal como se indica en la siguiente figura:

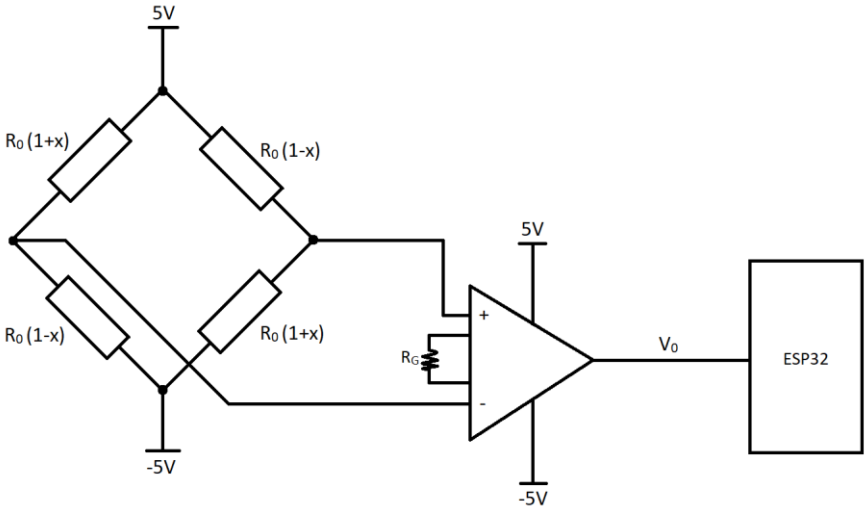


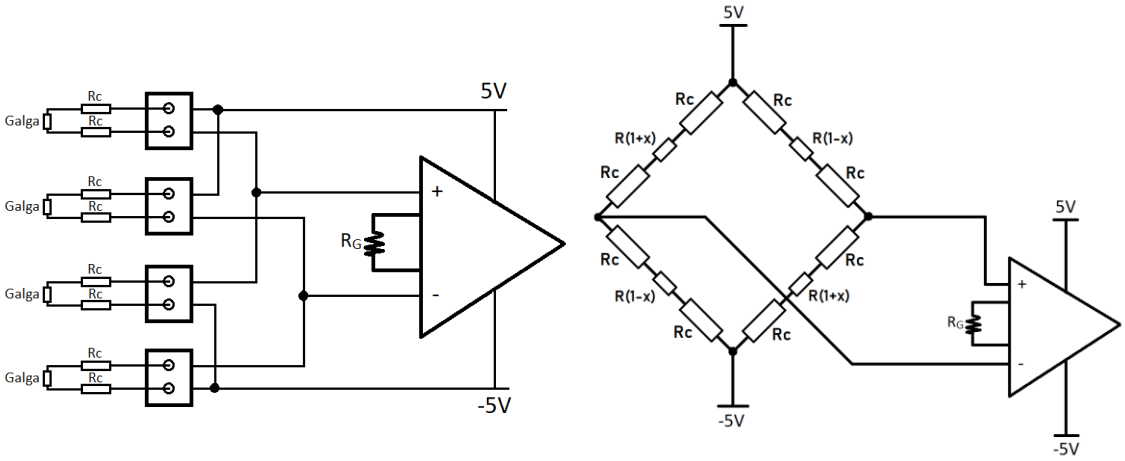
Figura 3.1.1.1 Esquema del acondicionador de la señal de las galgas extensiométricas.

La tensión de salida Vo de esta etapa acondicionadora es:

$$Vo = 10\left(1 + \frac{50K\Omega}{R_G}\right)\left(\frac{Ro(1+x) - Ro(1-x)}{Ro(1+x) + Ro(1-x)}\right) = 10\left(1 + \frac{50K\Omega}{R_G}\right)x \quad \text{(Ec. 3.1.1.1)}$$

Siendo x = K*ε, donde K es una constante que depende del material de construcción de la galga y ε es la deformación unitaria producida en ella.

Para poder conectar los cuatro sensores de galgas de una manera más cómoda se han colocado 4 conectores independientes para cada sensor (figura 3.1.1.2).



(a) (b)

Figura 3.1.1.2 (a) Dibujo real de las galgas (b) Esquema real de las galgas

Al incluir estas resistencias Rc de los cables de conexión, se cambia la ganancia del Puente de Wheatstone y realizando una serie de operaciones matemáticas, se puede observar que realmente se tiene una tensión de salida Vo igual a:

$$Vo = 10\left(1 + \frac{50K\Omega}{Rc}\right)\left(\frac{2R}{4Rc + 2R}\right)x \qquad \text{(Ec. 3.1.1.2)}$$

Como las resistencias de los cables Rc son de un valor mucho más pequeño que la resistencia de las galgas Rgalga, la influencia de estas en el valor de V0 es muy pequeña. Además, este pequeño error es constante y puede ser compensado con un offset en el firmware.

Respecto al sensor BNO055, su conexionado es bastante sencillo. Este se alimenta con una tensión de 3,3V o de 5V. Para este trabajo, se utilizará el protocolo de comunicación I2C. Esta comunicación usa únicamente 2 cables, uno para la señal de reloj (SCL) y otro para la comunicación de datos (SDA).

En el caso de querer utilizar otro protocolo de comunicación habría que utilizar los pines PS0 y PS1. Además, el pin INT se puede configurar para generar una señal de interrupción al detectar con el acelerómetro un evento en particular. Por último, quiero comentar que, si se desean utilizar 2 dispositivos a través del mismo bus I2C, habría que cambiar la dirección de uno de ellos, conectando la patilla ADR a 3,3V.

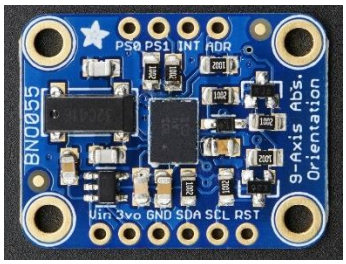


Figura 3.1.1.3 Sensor BNO055

Para este proyecto, únicamente se necesitan conectar los pines: Vin, GND, SDA y SCL del sensor BNO055. Tal como se indica en la figura 3.1.1.4.

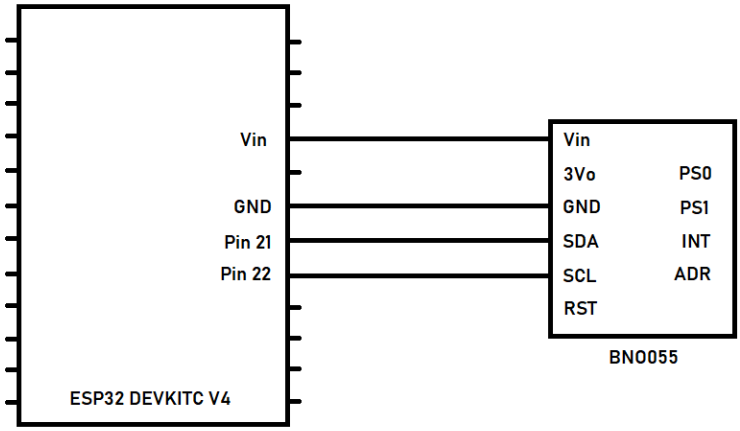


Figura 3.1.1.4 Esquema del conexionado del sensor BNO055

3.1.2. El microcontrolador

El microcontrolador es el componente principal del prototipo de este proyecto. Este recibe y realiza un tratamiento de los datos que provienen de los diferentes sensores, para posteriormente enviarlos vía wifi.

Se ha elegido el microcontrolador ESP32 DEVKITC V4 para el desarrollo de este proyecto (figura 3.1.2.1), esta es una muy buena opción ya que se trata de un microcontrolador barato y versátil. Además, este micro se puede programar de una forma sencilla con Arduino a través de un cable Micro USB.

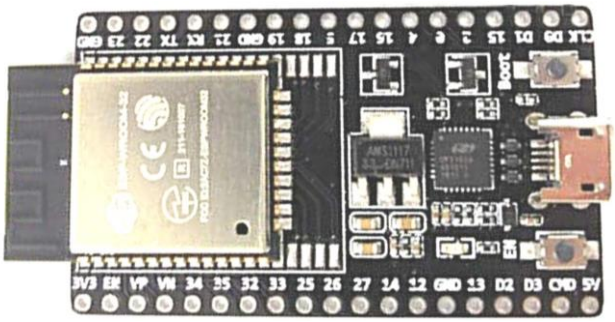


Figura 3.1.2.1 ESP32 DEVKITC V4

Este micro elegido ha sido sacado al mercado por la empresa Espressif. Se trata de una versión mejorada respecto al ESP8266, que ahora además de wifi posee entre otras cosas, Bluetooth BLE (de bajo consumo), un conversor analógico digital con más resolución y un procesador más veloz.

3.1.3. La alimentación

El bloque de la alimentación se puede dividir en 2 etapas: la carga de la batería y la adaptación de tensión desde una batería de ion de litio. Los esquemas para las diferentes etapas de la alimentación se han tomado de las diferentes hojas de características de cada dispositivo.

Para cargar la batería de ion de litio se necesita una etapa que adapte la energía que proviene de un cargador típico de un móvil con conector Micro USB a una corriente y tensión adecuada para la batería. Teniendo en cuenta la comparativa realizada en el siguiente apartado se ha escogido finalmente el chip MCP73811. Este chip posee algoritmos específicos para la carga de una batería de Li-Ion o Li-Polímero.

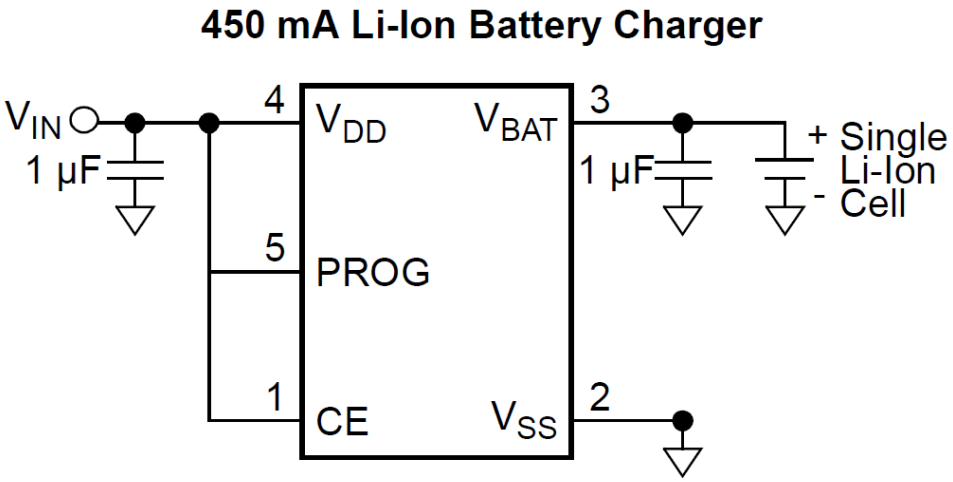


Figura 3.1.3.1 Esquema de la etapa para la carga de la batería

La segunda etapa de la alimentación consta de adaptar la tensión de una batería, que en nuestro caso se encuentra entre los 3 y 4,2V, a una tensión adecuada para el resto de los componentes. Hay dos formas de llevar esto a cabo.

La primera opción es usar un elevador de tensión, mediante una etapa con la misma forma o una forma similar que en la figura 3.1.3.2, para poder alimentar el microcontrolador a 5V, a través del pin 5V. Internamente el micro regularía esta tensión a 3.3V. Esta regulación el módulo ESP32-Devkitc-V4 la realiza con la ayuda del chip AMS1117-3.3. Con la tensión obtenida en el pin 3V3 se alimentaría a los sensores.

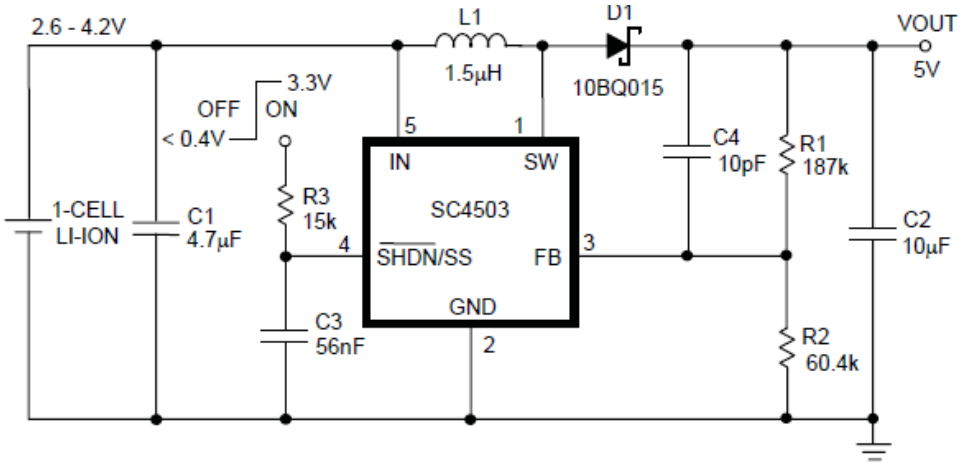
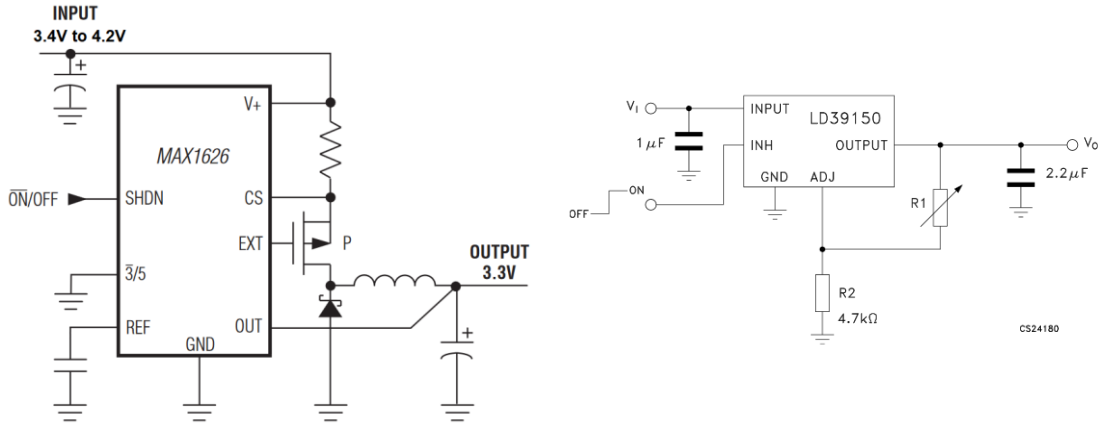


Figura 3.1.3.2 Esquema de la etapa de un elevador de tensión

La segunda opción trata de alimentar tanto el micro como los sensores a 3.3V, utilizando un regulador LDO o un convertidor Buck. Las etapas correspondientes a estos componentes se exponen en las siguientes figuras.



(a) (b)

Figura 3.1.3.3 (a) Etapa de un regulador Buck (b) Etapa de un regulador LDO variable

La alternativa por la que finalmente se ha optado, incluyendo el chip seleccionado, se comenta en el siguiente apartado.

3.2. Elección de componentes

A continuación, se muestran las diferentes tablas que se han ido realizando para comparar y elegir los componentes más adecuados, a continuación de cada tabla se exponen los razonamientos o ideas que se han seguido para tomar cada decisión.

3.2.1. Galgas Extensiométricas

En la tabla 3.2.1.1 se muestran todas las galgas extensiométricas halladas en las páginas Farnell y Rs Components, que son las páginas principales de suministro de material electrónico de la Universidad de Zaragoza.

Galgas Extensiométricas								
Referencia del dispositivo	FSR07BE	FSR05BE	632-180	865-6226	632-168	SS-U-N-S-00001	SS-U-N-S-00009	SS-U-N-S-00015
Distribuidor	farnell	farnell	RS Components	RS Components	RS Components	RS Components	RS Components	RS Components
Fabricante	OHMITE	OHMITE	RS PRO	RS PRO	RS PRO	I.E.E.	I.E.E.	I.E.E.
resistencia eléctrica (Ω)	$(10^7, 1)$	$(10^7, 1)$	120	120	120	$(10^7, 10^3)$	$(10^7, 10^3)$	$(10^7, 10^3)$
tamaño exclusivo de galga (mm)	ø 14,70	ø 5,60	9.5 x 3,5	13 x 4	6 x 2,5	ø 4,05	6,10 x 6,10	ø 9,55
tamaño completo con cable y contorno (mm)	56,34 x 18,00	38,10 x 7,62	-	-	-	36,1 x 7	48.8 x 11,7	51.65 x 15,25
Factor galga ,K	-	-	2,1	2	2	-	-	-
rango de temperatura (°C)	(-20 , 80)	(-20 , 80)	(-30 , 80)	(-30 , 80)	(-30 , 80)	(-30 , 170)	(-30 , 170)	(-30 , 170)
precio (€)	7,01	6,46	9,62	11,81	10,92	19,53	20,7	22
enlace pagina web	https://es.farnell.com/ohmite/fsr07be/force-sensing-resistor-02ah9091/dp/3216605?st=sensor%20de%20fuerza	https://es.farnell.com/ohmite/fsr05be/force-sensing-resistor-02ah9087/dp/3216600?st=sensor%20de%20fuerza	https://es.rs-online.com/web/p/galgas-extensiometricas/0632180/	https://es.rs-online.com/web/p/galgas-extensiometricas/8656226/	https://es.rs-online.com/web/p/galgas-extensiometricas/0632124/	https://es.rs-online.com/web/p/galgas-extensiometricas/1895556/	https://es.rs-online.com/web/p/galgas-extensiometricas/1895562/	https://es.rs-online.com/web/p/galgas-extensiometricas/1895584/

Tabla 3.2.1.1 Comparativa de galgas extensiométricas

De primeras, se han descartado las galgas fabricadas por Ohmite ya que en la página de Farnell se comenta que están pensadas para medir un peso entre 0 y 5 Kg, y el peso de un caballo es muy superior. Indagando un poco más, se ha observado que las fabricadas por I.E.E tienen el rango de medida entre (0.5 , 100) N/cm^2, lo que equivale a (0.05 , 10) Kg/cm^2, y de las galgas seleccionadas ninguna tiene un valor superior a 1 cm^2, por tanto, las galgas fabricadas por I.E.E tampoco valdrían y las únicas galgas que nos serían de utilidad serían las fábricas por RS PRO. Estas últimas, fabricadas por RS, su rango de peso medido depende de la celda de carga utilizado o del lugar donde se coloque.

Respecto al tamaño se ha pensado que por lo menos quepan en la anchura de una herradura (20mm) y si se quisiera colocar en un lateral de la herradura el tamaño de las galgas debería de ser inferior al espesor de la herradura (10mm), todo esto no es muy relevante ya que finalmente se ha decido que las galgas estén colocadas en la pezuña y no en la herradura.

Investigando en internet, se han encontrado otras ideas. En general, las galgas de medición de 3 o 6 mm son una buena solución, excepto para materiales heterogéneos como el hormigón o la madera que se recomiendan galgas más largas. Respecto a la resistencia eléctrica, cuanto menor sea su valor menor será la influencia de las interferencias electromagnéticas, aunque tendrá un mayor consumo y calentamiento.

También se ha tenido en cuenta que por lo menos tuviera integrado un trozo de cable, ya que una unión (soldadura) situada debajo de la pata de un caballo tendría un alto riesgo de desconexión debido al desgaste.

La galga finalmente seleccionada ha sido la 865-6226 de RS PRO, ya que las procedentes de otros fabricantes están pensadas para un rango de peso muy pequeño, y respecto al tamaño se ha escogido la más grande de las tres para poder conseguir una medida más homogénea y así poder conseguir una comparación más realista entre la presión ejercida por las distintas patas.

3.2.2. Amplificador de instrumentación

De todos los amplificadores de instrumentación hallados en Farnell, en la tabla 3.2.2.1 únicamente se han tenido en cuenta los amplificadores que dieran la posibilidad de ser usados con una alimentación unipolar y que tuvieran en su hoja de características un dibujo con una aplicación propia de su uso con el típico puente de Wheatstone.

Amplificador de Instrumentación							
reference	AD623ANZ	INA118P	INA122P	AD8223ARZ	INA155UG4	INA156EA/250	INA321EA/250G4
Case Style	DIP	DIP	DIP	SOIC	SOIC	MSOP	MSOP
price (€)	6,14	9,98	7,36	2,89	4,28	3,35	3,09
Input Bias Current (nA)	17	5	25	25	0,0002	0,001	0,01
Input Offset, VOS (µV)	200	50	250	250	200	2500	200
Common-Mode Rejection (db)	80 - 110	100 -120	96	90	86 -100	66-87	94
Slew Rate (V/µs)	0,3	0,9	0,08	0,2	6.5	6,5	0,4
Single supply high (V)	Vs-0,5	Vs-0,7	Vs–0.05	Vs-0,8	4.99	4.99	Vs-0,05
Single supply low (V)	1	0,035	0.1	1	0.01	0.01	0,05
gain max (V/V)	1000	10000	10000	1000	50	50	1000

Tabla 3.2.2.1 Comparativa de amplificadores de instrumentación

La distancia entre patillas de un DIP es de 2.54 mm, entre las de un SOIC es de 1,27 mm y entre las de un MSOP es de 0,65.

El slew rate es un parámetro para destacar, ya que, el amplificador suele ser el cuello de botella en cuanto a la velocidad de transmisión de datos.

A la hora de elegir, los primeros en ser descartados han sido los chips con encapsulado msop por su dificultad a la hora de soldar, también se ha descartado el Ina155 debido a que su ganancia tiene un valor máximo reducido. También se ha prescindido de los amplificadores AD623ANZ y AD8223 ya que no son capaces de tener un valor de salida inferior a 1 V y esto limita el rango de lectura del conversor ADC, perdiendo precisión.

Una vez descartados los amplificadores anteriores, nos quedan INA118 y el INA122. Se ha escogido el chip INA118 debido a que es el de mayor precisión, basándonos en el conjunto de los parámetros de Ib, Vos y CMRR, y a que la diferencia de unos euros más entre este y el amplificador INA122 no es muy significativa respecto al total del precio del producto. Además, este amplificador no deja de tener un rango de tensión de salida amplio (prácticamente los de la alimentación) y de tener una ganancia máxima suficientemente elevada.

3.2.3. Módulo de ESP32

Realizando una búsqueda exhaustiva en internet, se han encontrado 4 alternativas al módulo ESP32 Devkitc V4 para utilizar en nuestro proyecto. Estas se exponen en la tabla 3.2.3.1.

Modulo Esp32					
Nombre	ESP32 DEVKITC V4	ESP32 Tinypico board (CS-TINYPICO-01)	D1 mini ESP32	TTGO T7 V1.3 MINI32 ESP32	TTGO T7 V1.4 MINI32 ESP32
Distribuidor	Mouser ; Aliexpress	Mouser	Aliexpress	Aliexpress	Aliexpress
Precio (€)	10,89 ; 3,37	28,31	3,64	4,3	7,02
Dimensiones (mm)	27,9 x 54,4	18x32	31x39	31,2x39	31,2x39
Comunicación I2C	√	√	√	√	√
Conversor ADC	√	√	√	√	√
extra	Ya conocido	gestión de la batería LIPO		gestión batería de litio de 3,7 V	gestión batería de litio de 3,7 V

Tabla 3.2.3.1 Comparativa de módulos ESP32

EL módulo CS-TINYPICO-01 es el más pequeño de todos, pero también es con diferencia el más caro. Los módulos más competitivos por precio y espacio son el D1 mini ESP32 y el TTGO T7 V1.3, pero el TTGO T7 V1.3 incluye además la gestión de batería de litio de 3.7V y de esta manera se ahorraría el espacio y el coste de incluir las etapas de carga de la batería. Teniendo en cuenta únicamente el precio el ESP32 Devkitc V4 sería el ganador.

Finalmente, se ha escogido para el desarrollo de este proyecto el módulo ESP32 Devkitc V4 ya que es el oficial del fabricante y reúne todas las características necesarias.

3.2.4. Cargador de baterías

Para analizar los chips de carga, únicamente se han comparado los chips que están específicamente pensados para una batería de Ion de Litio (Li-Ion) y que tengan un número de patillas igual a seis o inferior.

CHIPS DE CARGA					
referencia	MCP73811	MCP73812	MCP73831	STC4054GR	BQ21040DBVT
encapsulado	SOT-23	SOT-23	SOT-23	TSOT-23	SOT-23
fabricante	Microchip	Microchip	Microchip	STMicroelectronics	Texas Instruments
distribuidor	Farnell	Farnell	Farnell	Farnell	Farnell
precio (€)	0,473	0,501	0,519	1,37	1,14
nº componentes extras necesarios	2	3	3	2	3
nº patillas	5	5	5	5	6
I carga máx (mA)	450	500	500	800	800
posibilidad de incluir led	-	-	√	√	√
enlace	https://es.farnell.com/microchip/mcp73811t-420i-ot/ic-batt-mgmt-li-ion-li-poly-sot23/dp/1439477?st=cargador%20de%20bater%C3%ADa	https://es.farnell.com/microchip/mcp73812t-420i-ot/ic-li-ion-li-poly-charger-sot23/dp/1627187?st=cargador%20de%20bater%C3%Ada	https://es.farnell.com/microchip/mcp73831t-2dci-ot/ic-batt-mgmt-li-ion-li-poly-sot23/dp/1840935?st=cargador%20de%20bater%C3%Ada	https://es.farnell.com/stmicroelectronics/stc4054gr/cargador-bater-as-li-ion-0-8a/dp/2762713RL?st=cargador%20de%20bater%C3%Ada	https://es.farnell.com/texas-instruments/bq21040dbvt/batt-charger-li-ion-li-pol-sot/dp/3123767?st=cargador%20de%20bater%C3%ADa

Tabla 3.2.4.1 Comparativa de cargadores de batería

Primero, se pensó en incluir únicamente un chip de carga, aunque posteriormente, analizando los chips que venían integrados en un módulo cargador de baterías como el TP4056, se ha podido observar que algunos tienen incluido un chip de protección. Por eso, se ha planteado la idea de incluir al proyecto un chip de protección.

Los chips de carga tienen incluido una protección frente a la sobrecarga, ya que establecen un límite de tensión de carga. Pero estos no tienen incluida ni una protección frente a la sobredescarga ni frente a una descarga demasiado rápida (un cortocircuito).

Para analizar los chips de protección, aunque existen una gran variedad, se ha escogido una representación de cada fabricante y así no hacer una lista excesivamente extensa.

CHIPS DE PROTECCIÓN			
reference	AP9101CAK-ABTRG1	S-8261ABPMD-G3PT2G	R5478N101CD-TR-FF
Case Style	SOT-25-5	SOT-23-6	SOT-23-6
manufacturer	Diodes Incorporated	Ablic	Ricoh Electronic Devices Company
supplier	mouser	mouser	mouser
price (€)	0,396	0,81	1,51
Overcharge Detection Voltage(V)	4,425	4,2	4,25
Over-Discharge Detection Voltage(V)	2,5	2,8	3
Overcharge hysteresis voltage (V)	0,2	0,1	0,2
Overdischarge hysteresis voltage (V)	0,4	0,1	0,5
enlace	https://www.mouser.es/ProductDetail/Diodes-Incorporated/AP9101CAK-ABTRG1?qs=sGAEpiMZZMsfD%252BbMpEGFJcTCUNTyJQiaKjr7hNWquxo%3D	https://www.mouser.es/ProductDetail/ABLIC/S-8261ABPMD-G3PT2G?qs=sGAEpiMZZMsfD%252BbMpEGFJYkUVygweHO0UmIFN5ncPzA%3D	https://www.mouser.es/ProductDetail/Ricoh-Electronic-Devices-Company/R5478N101CD-TR-FF?qs=sGAEpiMZZMsfD%252BbMpEGFJSnYWe3dSmEhy040E1GO2ok%3D

Tabla 3.2.4.2 Comparativa de protectores de batería

Se ha pensado en 3 opciones:

1º Una opción completa, sería utilizar un chip que tenga una carga rápida y un chip de protección. Como chips de carga rápida pueden servir el chip STC4054GR y el BQ21040DBVT, entre estos dos se prefiere el segundo, ya que tiene un encapsulado más fácil de soldar y su precio es menor. Como chip de protección seleccionamos el S-8261ABPMD-G3PT2G.

2º Una opción más simple, sería poner el chip MCP73811 sin incluir ningún chip de protección.

3º Otra posibilidad sería utilizar los mismos chips que vienen integrados en un módulo ya fabricado como el TP4056, este módulo incluye los siguientes chips:

chip de carga --> TP4056	precio: 0,45 € / 10 unidades
chip de protección --> DW01A	precio: 0,57 € / 20 unidades
2 mosfet tpio N --> FS8205A	precio: 0,37 € / 10 unidades

Tabla 3.2.4.3 Componentes del módulo TP4056

En la siguiente figura se expone una imagen de este módulo con la descripción de los componentes que contiene.

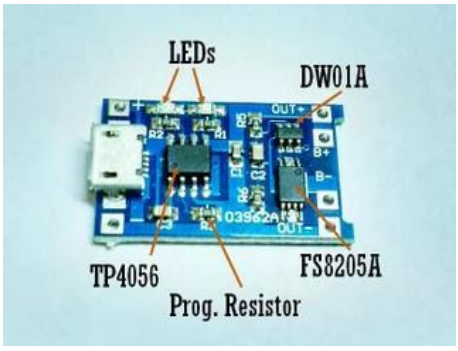


Figura 3.2.4.1 módulo TP4056

Finalmente, se ha escogido la opción 2 debido a que las baterías ya llevan su protección y no es necesario incluir ningún otro chip de protección.

3.2.5. Regulador de tensión

De primeras se había pensado incluir un elevador de tensión y se hizo una tabla de la misma forma que se ha realizado anteriormente, donde únicamente se han analizado los chips que tuvieran menos de 6 pines y en su hoja de características un dibujo con una aplicación típica parecida a la que queremos llevar a cabo.

Elevador de tensión					
referencia	MIC2288YD5-TR	SC4503TSKTRT	MAX1896EUT+T	LT3467AES6#TRMPBF	LTC3426ES6#TRMPBF
encapsulado	SOT-23	TSOT-23	SOT-23	TSOT-23	TSOT-23
fabricante	Microchip	Semtech	Maxim Integrated	Analog Devices	Analog Devices
distribuidor	Farnell	Farnell	Farnell	Rs Components	Rs Components
precio(€)	0,501	1,98	2,96	2,68	3,295 / 2 unidades
nº componentes extra	6	9	9	8	6
rango de tensión de entrada(V)	3 - 4,2	2,6 - 4,2	2,6 - 4,5	2,6 - 4,2	3 - 4,2
tensión de salida (V)	5	5	5	5	5
corriente máxima (mA)	400	750	500	400	750
eficiencia (%)	80-85	85-90	80-83	85-90	90-93
enlace	https://es.farnell.com/microchip/mic2288yd5-tr/voltage-reg-boost-1a-1-2mhz-tsot23/dp/2509929?st=MIC2288	https://es.farnell.com/semtech/sc4503tsktrt/dc-dc-conv-boost-1-3mhz-85deg/dp/3018459?ost=sc4503TSKTRT&ddkey=https%3Aes-ES%2FElement14_Spain%2Fsearch	https://es.farnell.com/maxim-integrated-products/max1896eut-t/conver-dc-dc-boost-1-4mhz-85-c/dp/2909789?st=Max1896	https://es.rs-online.com/web/p/convertidores-dc-dc/7618692/	https://es.rs-online.com/web/p/convertidores-dc-dc/8225957/

Tabla 3.2.5.1 Comparativa de elevadores de tensión

De estos el más destacado es el LTC3426 ya que es de los que menos componentes extra (condensadores, diodo, bobina resistencias, etc.) necesita, y el mejor en cuanto a eficiencia y máxima corriente suministrada. Aunque tanto este como los demás de la tabla podrían funcionar correctamente. Analizando la eficiencia que implica utilizar el regulador que lleva el integrado el ESP32-Devkitc-V4, se puede observar que la eficiencia de este no llega al 66%, es decir, del consumo estimado de nuestro dispositivo, de 1.5W, no se llega al vatio aprovechado.

Siguiendo el consejo de una persona más experimentada en el tema se ha descartado la opción de utilizar un elevador de tensión (boost) y se han buscado otras alternativas como son el uso de reguladores LDO y convertidores Buck.

Aunque se ha encontrado una gran variedad de componentes, solo se han incluido en la comparativa los parecen más adecuados, para así no hacer una lista excesivamente larga.

Reguladores LDO y convertirdores Buck							
referencia	MAX1626ESA+T	LTC1147CS8-3.3#PBF	MCP1825-ADJE/AT	MCP1826T-ADJE/DC	LD39150PT-R	MCP1825	MCP1826
Encapsulado	NSOIC	SOIC	TO-220-5	SOT-223-5	PPACK	SOT-223-3 ; TO-220-3	SOT-223-3 ; TO-220-3
número de patillas	8	8	5	5	5	3	3
tipo de componente	convertidor Buck	convertidor Buck	Regulador LDO ajustable	Regulador LDO ajustable	Regulador LDO ajustable	Regulador LDO fijo	Regulador LDO fijo
precio componente (€)	2,74	4,53	0,75	0,61	0,92	0,47 ; 0,76	0,63 ; 0,91
número de componentes	8	11	4	4	4	2	2
precio total etapa (€)	8,00	8,23	0,96	0,82	1,12	0,56 ; 0,85	0,72 ; 1,00
eficiencia (%)	> 95	>97	(66 , 93)	(69 , 97)	(69 , 97)	(78, 93)	(78, 97)
rango de tension de entrada (V)	3.4 to 16.5	3.5 to 12	3 to 6	3 to 6	3 to 6	3.5 to 6	3.4 to 6
tensión dropout (V) [Isum=400mA]	0,08	0,2	0,2	0,1	0,06	0,2	0,1
tensión de salida (V)	3.3	3.3	2.8	2.9	2.9	3.3	3.3
corriente máxima (A)	0.5	1.25	0.5	1	1.5	0.5	1
enlace	https://es.farnell.com/maxim-integrated-products/max1626esa-t/controlador-dc-dc-buck-soic-8/dp/2516676?st=buck	https://es.farnell.com/linear-technology/ltc1147cs8-3-3-pbf/controlador-de-conmutaci-n-3-3/dp/1273834?st=buck	https://es.farnell.com/microchip/mcp1825-adje-at/ic-ldo-adj-500ma-to-220-5/dp/1578398?st=regulador%20de%20tensi%C3%B3n	https://es.farnell.com/microchip/mcp1826t-adje-dc/ic-ldo-adj-1a-sot-223-5/dp/1578432?st=regulador%20de%20tensi%C3%B3n	https://es.farnell.com/stmicroelectronics/ld39150pt-r/ldo-ajustable-5v-1-22v-1-5a-ppack/dp/2806943RL?st=regulador%20de%20tensi%C3%B3n	https://es.farnell.com/w/search?tipo-de-salida=fija&tension-nominal-de-salida-fija=3.3v&numero-de-pines=3pines&st=MCP1825	https://es.farnell.com/w/search?tipo-de-salida=fija&tension-nominal-de-salida-fija=3.3v&numero-de-pines=3pines&st=MCP1826

Tabla 3.2.5.2 Comparativa de reguladores de tensión

En las siguientes tablas se muestran unos componentes con sus referencias y precios, con el propósito de obtener un valor aproximado del coste total de cada componente, teniendo en cuenta toda la etapa. En la primera tabla se presentan los precios de los componentes extras de las 2 etapas tipo Buck planteadas.

Componentes extra para convertidores Buck	MAX1626ESA+T		LTC1147CS8-3.3#PBF	
	referencia	precio (€)	referencia	precio (€)
bobina	CDRR105NP-680MC	2,12	B82133A5302M	1,3
condensadores	EEUFR1J101	0,81	MC0805N121J500A2.54MM	0,167
	AFK686M16D16T-F	0,549	MC0805B332K500A5.08MM	0,132
	MCGPR50V474M5X11	0,0493	MC0805Y103M500A5.08MM	0,118
	SR215C104KARTR1	0,115	SR215C104KARTR1	0,115
	-		MCGPR25V476M5X11	0,0538
	-		USR1C101MDD1TP	0,055
transistor pMos	MMSF3P02HDR2G	1,16	SI9433BDY-T1-GE3	0,738
Resistencia	RL73K3AR12J	0,401	PCS1206DR0500ET	0,585
diodo	1N5817-E3/54	0,6	MBRS130-M3/5BT	0,44

Tabla 3.2.5.3 Integrantes de las etapas Buck

En la segunda tabla se representan de forma general los extras del resto de los reguladores tipo LDO tanto con tensión de salida fija como variable.

Componentes Extra para los reguladores LDO	valor	referencia	precio (€)
Consdensadores	4,7uF	MCGPR35V475M5X11	0,0459
	2,2uF	MCKSK100M2R2D11S	0,0383
	1uF	MCGPR50V105M5X11	0,0471
Resistencias	20KΩ	MF50 20K	0,0587
	4,7KΩ	MF50 4K7	0,056
	120KΩ	MF50 120K	0,0551
	6,8KΩ	MF50 6K8	0,056

Tabla 3.2.5.4 Integrantes de los reguladores LDO

Por último, se muestra qué componentes son requeridos en las etapas de cada regulador tipo LDO.

Tipo	Valor	MCP1825-ADJE/AT	MCP1826T-ADJE/DC	LD39150PT-R	MCP1825	MCP1826
Consdensadores	4,7uF	√	√		√	√
	2,2uF			√		
	1uF	√	√	√	√	√
Resistencias	20KΩ	√	√			
	4,7KΩ			√		
	120KΩ	√	√			
	6,8KΩ			√		

Tabla 3.2.5.5 Integrantes concretos de las etapas de los reguladores LDO

Cabe comentar, que el encapsulado TO-220 tiene una distancia entre patillas de 2,54 mm , el sot-223-3 una distancia de 2.3mm, el sot-223-5 de 1,27mm y el PPACK de 1.27mm.

Para este proyecto se ha descartado la opción de utilizar un regulador convertidor buck ya que utilizar uno de estos implicaría sacrificar en complejidad, espacio requerido y precio por ganar unicamente de media un 15% en eficiencia.

Una cuestión estaría entre utilizar un regulador fijo o uno variable. El regulador fijo asigna una tensión de salida de 3.3V, pero el variable podría ser capaz de exprimir al máximo la batería, alimentando a 2,8V. Finalmente, se ha optado por uno fijo ya que es suficientemente grande un rango de tension entre 3,4 y 4,2 V para la batería.

Otra cuestion estaría entre elegir un regulador de 0,5A (MCP1825) o uno de 1A (MCP1826). El segundo, además de permitir una corriente más alta, tiene un dropout algo inferior y la desventaja de ser un poco más caro. En la hoja de características del ESP32 se recomienda un suministro de 0,5A o superior. Teniendo en cuenta el consumo añadido de los sensores, se ha pensado que un regulador de 500mA puede ser algo justo, y por esto, se ha preferido la opcion del MCP1826.

También a la hora de elegir entre el encapsulado TO-220 y el sot-223-3, se presenta otra incognita ya que uno es un poco más sencillo de soldar y el otro un poco más barato. En el caso de fabricar una gran cantidad de componentes podría ser mas interesante la segunda opción, pero para fabricar un único prototípo, como es el caso, es preferible que sea más facil de soldar.

Teniendo en cuenta todo lo anterior, se ha elegido como primera opción al MCP1826S-3302E/AB que es el que tiene un encapsulado TO-220.

3.3. Esquemático

Una vez agrupadas todas las ideas previas, se ha planteado finalmente el siguiente esquemático:

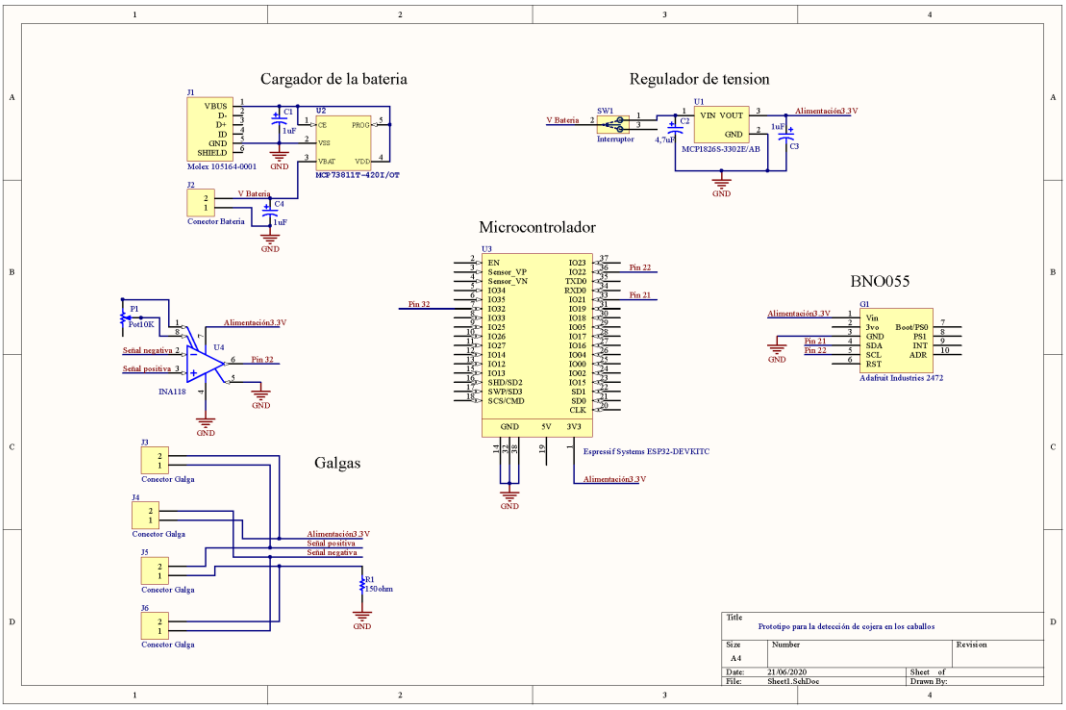


Figura 3.3.1 Esquemático del detector de la cojera de una pata

3.4. Diseño de la PCB

Este proceso de diseño consta de dos partes, una de decidir el lugar donde colocar cada componente en la PCB y la segunda parte trata de decidir el camino de las pistas eléctricas que unen eléctricamente las diferentes patas de los componentes.

En cuanto al posicionamiento de los componentes en la PCB se ha tratado entre otras cosas de:

- Agrupar los componentes por bloques
- No dejar espacios vacíos en la PCB
- Alejar lo máximo posible la alimentación de la antena
- Posicionar los conectores de las galgas en línea y en el extremo de la PCB para un fácil conexionado

En cuanto a las decisiones de ruteo se ha tratado de:

- Hacer las pistas de masa más anchas ya que por estas pasará mucha corriente
- Reducir al máximo el recorrido de las pistas
- Evitar poner vías para una mayor sencillez a la hora de fabricar la PCB
- Poner dos planos de masa tanto en la capa Botton como en la Top
- Poner todos los componentes en la cara Top
- Hacer que la mayoría de los componentes se tengan que soldar únicamente en la parte inferior de la PCB, aunque otros lo harán en la parte superior

A la hora de realizar este diseño ha habido algún imprevisto, por ejemplo, de primeras se pensó incluir un sensor BNO055 en concreto, pero este era diferente al que había en el laboratorio, y al incluir este otro el tamaño y la forma de este sensor era diferente y, por tanto, se ha tenido que realizar un diseño de la PCB completamente nuevo. Por otro lado, la huella de este componente no existía en el repertorio de CircuitMaker y se ha tenido que crear también una huella nueva. Esta huella se muestra en la siguiente figura.

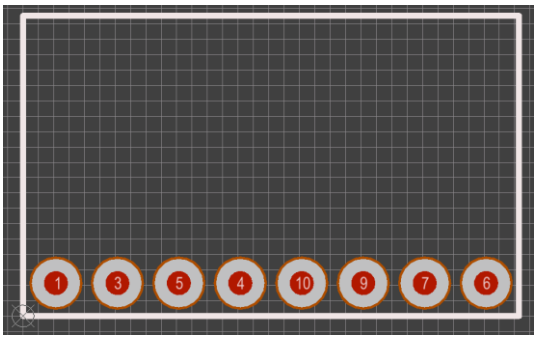


Figura 3.4.1 Huella del BNO055 del laboratorio

Finalmente se ha planteado el siguiente diseño de la PCB, donde las pistas en azul corresponden a las pistas en la cara Botton y las pistas naranjas a las de la cara Top.

Todos los componentes se situarán en la cara Top y se puede ver que los componentes están agrupados por bloques.

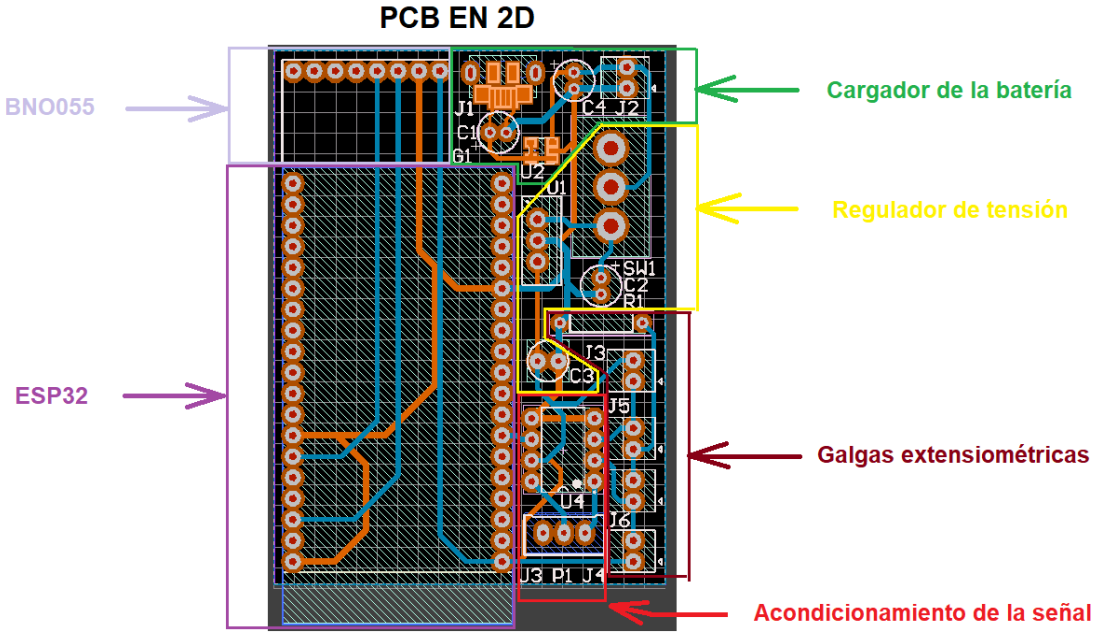


Figura 3.4.2 Diseño PCB en 2D del detector de la cojera de una pata

El diseño en 3D quedaría de la siguiente forma:

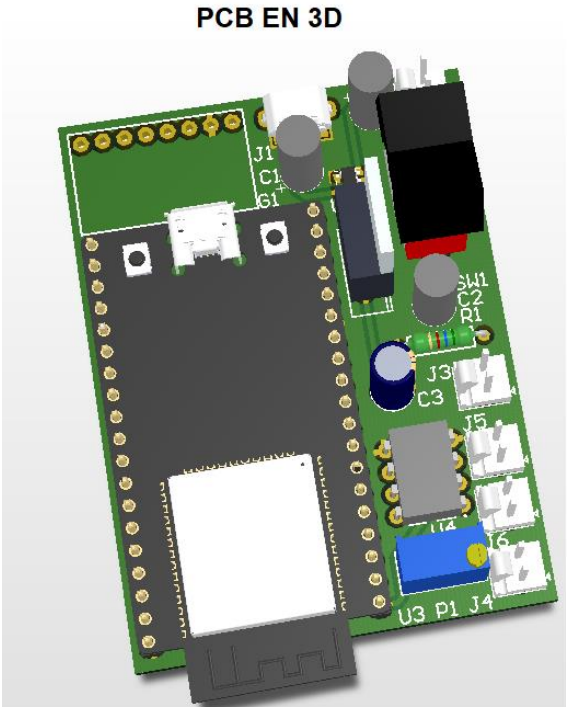


Figura 3.4.3 Diseño PCB en 3D del detector de la cojera de una pata

3.5. Montaje de la PCB

Inicialmente se tenía la idea de realizar el montaje y soldadura de los diferentes componentes en la placa de circuito impreso (PCB) previamente diseñada. Pero debido a que la máquina de la Universidad de Zaragoza, que es capaz de fabricar estas placas de circuito impreso, ha estado estropeada durante estos últimos meses, no se ha llegado a fabricar el diseño realizado.

En consecuencia, se ha llevado a cabo el plan b, donde las conexiones de los diferentes componentes se han realizado con una serie de cables en una protoboard.

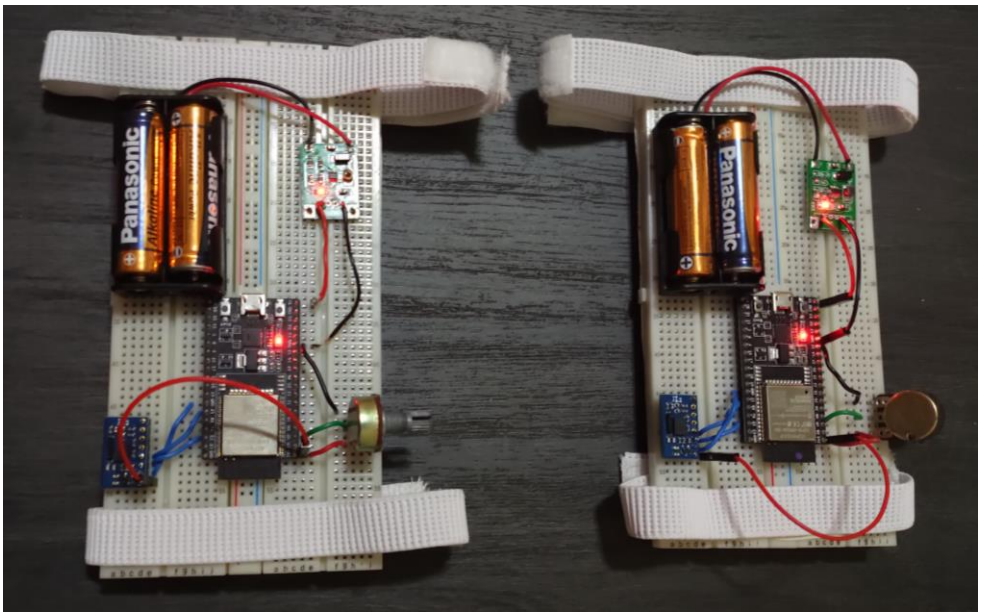


Figura 3.5.1 Montaje realizado en protoboard

De esta manera se han podido realizar sin ningún inconveniente otros objetivos como la puesta a punto con la programación del firmware y del software.

4. Programación

En este apartado se procede a explicar las 2 partes principales de programación que se han llevado a cabo, además de una programación extra realizada y la explicación del tipo de comunicación utilizada.

En la primera parte se comenta el desarrollo del firmware en Arduino que se ha realizado para programar el microcontrolador ESP32, en la segunda parte se muestra la programación realizada en Python para que el ordenador pueda recibir los datos del microcontrolador y utilizarlos tanto para graficarlos como para guardarlos en un fichero.csv.

Para conseguir que la lectura de este TFG sea más amena, en vez de explicar el código de programación línea a línea, se han realizado unos diagramas de flujo para explicar la programación de modo general sin entrar, ni en el detalle ni en el lenguaje propio de cada programa. Para más detalle, véase el código de la programación que se expone en los Anexos.

Respecto a los programas que se han ido realizando, estos a lo largo del TFG han estado evolucionando constantemente, pero por relevancia sólo se mostrará la última versión de cada uno.

4.1. Desarrollo firmware en Arduino

Esta primera parte de la programación consiste en conseguir que el microcontrolador sea capaz de leer los datos de los diferentes sensores con la ayuda de algunas librerías ya existentes y de enviar estos datos vía wifi.

Esta programación se ha estructurado en tres capas: la capa de generación de un punto de acceso, la capa de recepción de los datos y la capa de envío de los datos.

4.1.1. Capa de generación de un punto de acceso

Para comunicarnos desde el microcontrolador con wifi a un ordenador es necesario primero conectarnos a una red wifi en común, para ello es necesario indicarle de alguna manera el usuario y contraseña de la wifi a la que quieres que se conecte. También es necesario indicar la dirección IP del ordenador a donde se desea enviar los datos mediante el protocolo UDP. Una posibilidad es ponerlos directamente por programa, con el inconveniente de que la única manera de cambiar a otro usuario y contraseña wifi o a otra dirección IP sería mediante la modificación del código de programación. Para evitar esto se ha propuesto una solución realizando así una pequeña tarea extra.

En el momento que se desee obtener unos nuevos datos (de usuario y contraseña de la wifi y de dirección IP), el microcontrolador generará un access point (una red wifi propia) con un nombre y contraseña predeterminados para que todo usuario pueda conocerlos.

Realizado este punto de acceso, el microcontrolador espera a que un dispositivo Android se le conecte y a que este le envíe información. Una vez que se comienza a recibir datos del cliente, se empieza a leer bit a bit en búsqueda de las palabras entre "/".

Para empezar a recoger las palabras que le llegan al microcontrolador, se espera a recibir la palabra “LaEntrada”. En el momento que se detecta esta palabra, las siguientes palabras que aparecen se guardan en las variables correspondientes a la dirección IP, al usuario y contraseña de la wifi. De esta forma evitamos que el microcontrolador acepte valores falsos como si fueran válidos. Por último, se apaga el punto de acceso y estas variables se guardan en la memoria EEPROM para su futura utilización.

Antes de continuar, es preciso realizar una breve introducción respecto a las diferencias entre los tres tipos de memoria, la Flash, la EEPROM y la SRAM, que hay en un microcontrolador como el utilizado, el ESP32.

La memoria Flash es una memoria no volátil, es decir, que es capaz de conservar los datos incluso sin energía eléctrica. En esta memoria es donde se guarda el sketch (el programa). La EEPROM también es una memoria no volátil, pero con la diferencia de que sus datos si pueden ser modificados. Por último, la SRAM es donde el sketch crea, manipula y modifica las variables cuando las ejecuta, pero esta no es capaz de conservar sus datos una vez apagado el dispositivo.

4.1.2. Capa de recepción de los datos

Para recibir los datos de los diferentes sensores lo primero que se debe realizar es una pequeña configuración de la comunicación I2C, a través de la cual se comunica el sensor BNO55(que incluye el acelerómetro, magnetómetro y giróscopo).

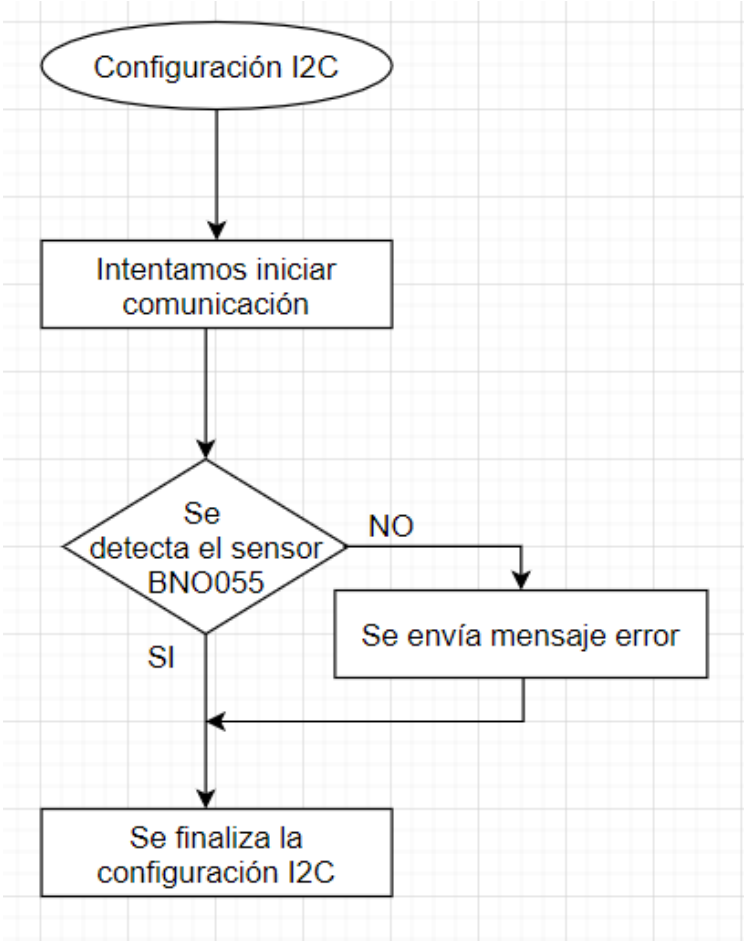


Figura 4.1.2.1 Diagrama de flujo de la configuración de la comunicación I2C

Como se puede observar, la configuración I2C es un proceso sencillo y corto en el que únicamente se trata de iniciar la conexión y en caso de que haya algún problema de cualquier tipo, que el sensor esté estropeado o que en algún punto no haga buen contacto o cualquier otro problema, se envía un mensaje de error por puerto serie al ordenador. Esto le puede ser de utilidad a los talleres de reparación para detectar de dónde procede el fallo.

Después con la ayuda de la librería Adafruit_BNO055 se comienza a leer los datos procedentes de los sensores incluidos en el BNO055. Por otro lado, se lee también la variable analógica del puerto 36, que contiene un valor entre 4095 y 0, pero en vez de utilizar este valor, se divide entre 1240.91 para tener un valor comprendido entre 3.3 y 0, que es el valor real del voltaje recibido por el microcontrolador. Finalmente, estas variables son almacenadas en un buffer para su posterior envío. Para separar una variable de otra, entre las variables se introducen un espacio y un punto y coma.

4.1.3. Capa de envío de los datos

A la hora de enviar los datos, se requiere la inicialización y puesta en marcha de la memoria EEPROM. En esta memoria se guardan los valores de usuario y contraseña de la wifi a donde se intentará conectar el microcontrolador y la dirección IP del ordenador que recibirá los datos.

Después de la inicialización de la EEPROM, se leerán los valores que tiene guardados y se intentará conectar a dicha red wifi. En algunos casos, se ha podido comprobar que la conexión wifi fallaba, pero al desconectar y volver a intentarlo nos conseguíamos conectar, por tanto, se ha tomado la decisión de reiniciar la conexión automáticamente en el caso de que pasen 5 segundos sin poder conectarnos.

En caso de que el microcontrolador no sea capaz de conectarse a la red wifi ya sea por ser la primera vez, o porque queramos conectarnos a una red wifi distinta a la habitual, se abrirá un punto de acceso a los 10 segundos para poder indicar unos nuevos valores de usuario y contraseña de la wifi, donde también se pedirá un nuevo valor de dirección IP. Una vez obtenidos los nuevos valores a través del punto de acceso, se guardarán en la EEPROM. De tal manera que las próximas veces que se encienda el dispositivo, este trate de conectarse a la red previamente guardada y/o utilizada.

Por último, una vez realizada la conexión wifi solo queda enviar los datos almacenados. Para ello, el microcontrolador (el cliente) crea un socket UDP y envía un mensaje al puerto y a la dirección IP predefinidos, sin la necesidad de la conexión previa con el ordenador, lo que conlleva el consiguiente ahorro de tiempo, la principal ventaja de una comunicación UDP.

En Arduino esta comunicación UDP se puede hacer de una manera muy sencilla gracias a la librería WifiUdp, donde lo único que ha que hacer es escribir 3 líneas de código donde se indica: el comienzo del paquete (su apertura), el contenido del buffer y el fin del paquete (su envío).

En la siguiente página, se muestra de una manera más gráfica el proceso de la configuración wifi, que se ha explicado previamente.

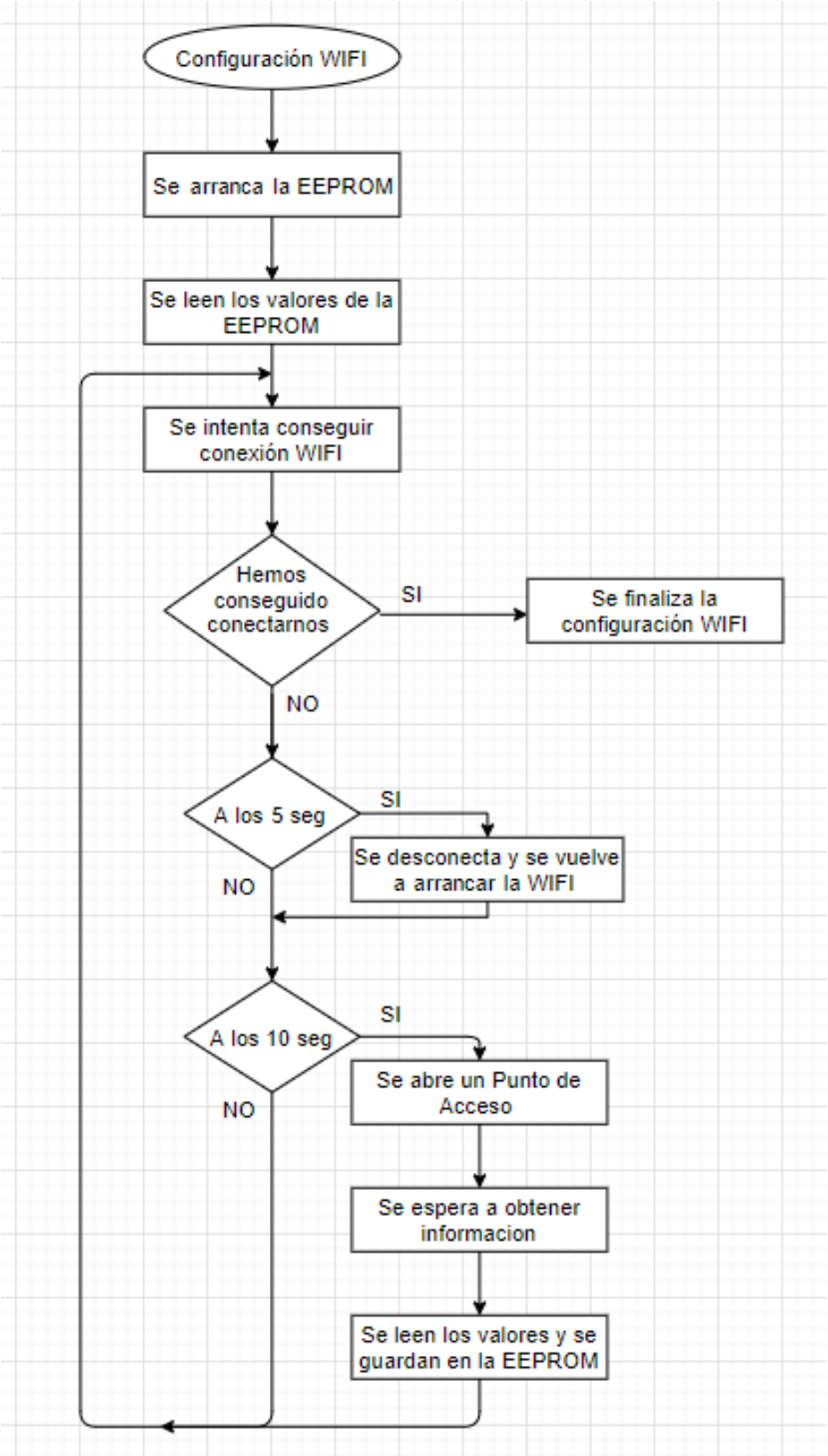


Figura 4.1.3.1 Diagrama de flujo de la configuración Wifi

4.1.4. Resumen del programa Arduino

A modo global, el programa comienza realizando las diferentes configuraciones, conectándose así tanto a los diferentes sensores como a la wifi deseada. Y después, entra en un bucle infinito donde se leen los datos de cada uno de los sensores y se envían vía UDP.

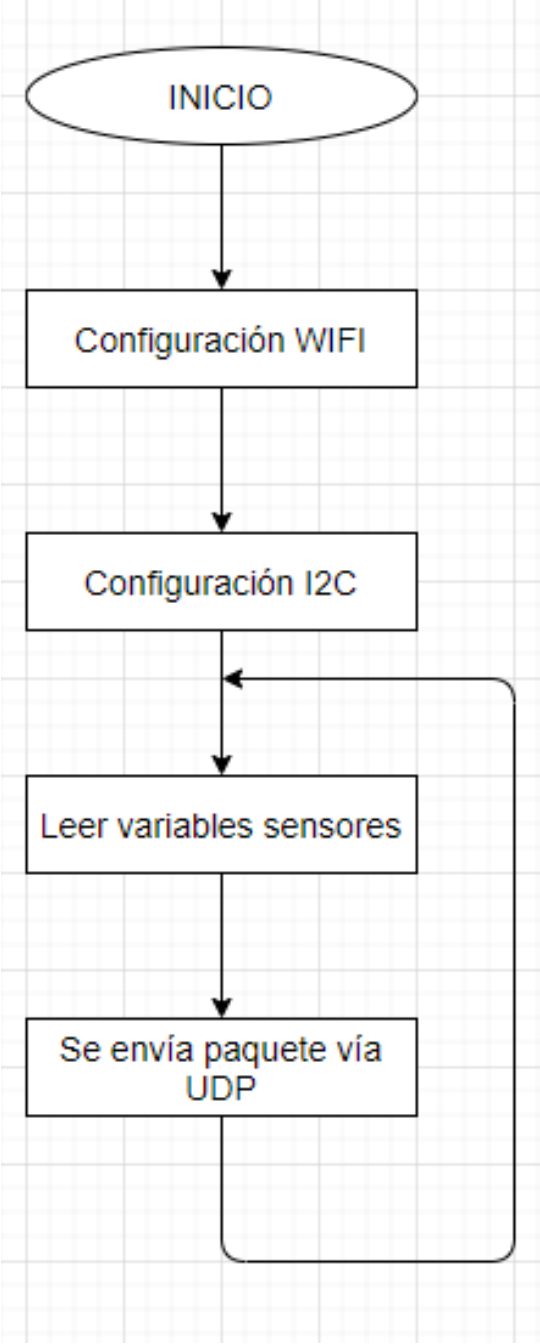


Figura 4.1.4.1 Diagrama de flujo general de Arduino

4.2. Desarrollo software en Python

En esta segunda parte de la programación el objetivo es conseguir que el ordenador, con la ayuda de algunas librerías de Python ya existentes, sea capaz de recibir los datos del microcontrolador, guardarlos en un fichero y pintarlos en las gráficas correspondientes.

Esta programación se puede dividir en tres capas: la capa de recepción y manejo de los datos, la capa de guardar el contenido en un fichero y la capa de la representación de las gráficas.

4.2.1. Capa de recepción y manejo de los datos

Para recibir los datos del microcontrolador, se requiere iniciar algunas variables globales y crear un Socket que se conectará con el puerto correspondiente. Un Socket permite que un cliente (el microcontrolador ESP32) se conecte al servidor (nuestro ordenador).

Cuando se recibe por el puerto predefinido, un paquete con nuevos datos, se trata la información recibida. Estos datos vienen acompañados del puerto y de la IP de donde proceden.

Se empieza eliminando algunos caracteres iniciales y finales que no aportan nada de información. Después se compara la dirección IP de la que proceden dichos datos con las existentes. Si es la primera vez que se ha recibido esta dirección IP se crea un fichero nuevo, se amplía la longitud de los vectores (arrays) donde se almacenan los datos, se genera una ventana nueva para graficar y se guarda el valor de la nueva IP en un array.

Por último, se recorre el buffer, carácter a carácter, hasta que se detecta un nuevo punto y coma, para guardar las medidas de cada sensor, en unas variables dentro de un vector (array).

Una cosa destacable que se ha conseguido es que las variables, vectores se adaptan al número de microcontroladores conectados, de tal forma que no es necesario indicar previamente en el código Python del ordenador cuantos dispositivos se van a conectar. Sino que se puede arrancar directamente el programa, él automáticamente se gestiona para crear tantas variables, ventanas de gráficas y ficheros como sean necesarios.

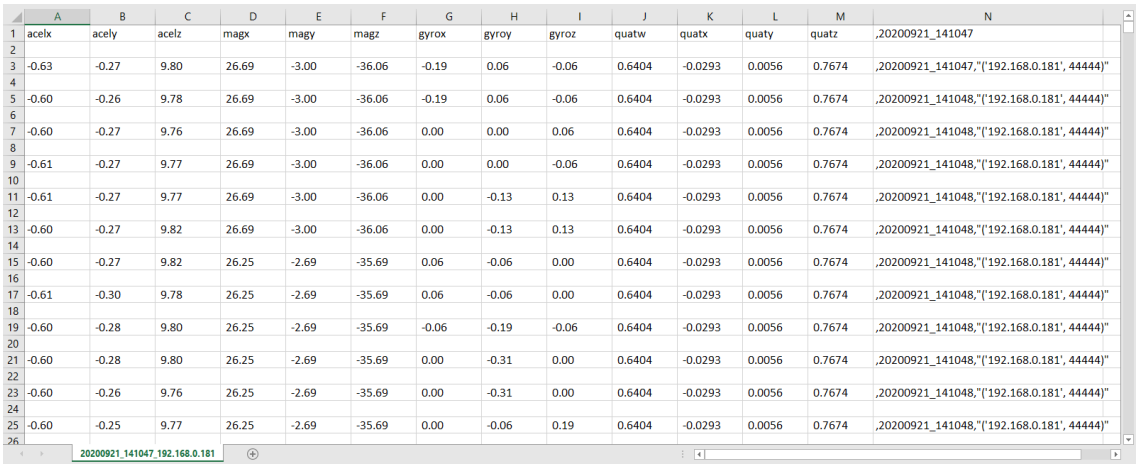
4.2.2. Capa de guardar el contenido en un fichero

Para guardar el contenido en un fichero, cada vez que se reciben los datos de una nueva IP, se crea un fichero .csv nuevo con el siguiente nombre: fecha_hora_dirección IP. Se abre el fichero y se escribe la primera línea (la cabecera) con los nombres de los diferentes sensores.

Una vez hecho esto, el fichero ya está preparado para guardar el contenido del array, que contiene la información de cada paquete que se recibe.

Abriendo una hoja Excel y cargando el fichero .csv generado anteriormente, se pueden ver los diferentes valores de los sensores, además en la última columna se guarda la hora y la fecha en la que estos son recibidos, la dirección IP de la que proceden y el puerto utilizado.

A continuación, se muestra una imagen con las primeras líneas de un ejemplo de los datos que se guardan.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	acelx	acely	acelz	magx	magy	magz	gyrox	gyroy	gyroz	quatw	quatx	quaty	quatz	,20200921_141047
2														
3	-0.63	-0.27	9.80	26.69	-3.00	-36.06	-0.19	0.06	-0.06	0.6404	-0.0293	0.0056	0.7674	,20200921_141047,"('192.168.0.181', 44444)"
4														
5	-0.60	-0.26	9.78	26.69	-3.00	-36.06	-0.19	0.06	-0.06	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
6														
7	-0.60	-0.27	9.76	26.69	-3.00	-36.06	0.00	0.00	0.06	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
8														
9	-0.61	-0.27	9.77	26.69	-3.00	-36.06	0.00	0.00	-0.06	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
10														
11	-0.61	-0.27	9.77	26.69	-3.00	-36.06	0.00	-0.13	0.13	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
12														
13	-0.60	-0.27	9.82	26.69	-3.00	-36.06	0.00	-0.13	0.13	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
14														
15	-0.60	-0.27	9.82	26.25	-2.69	-35.69	0.06	-0.06	0.00	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
16														
17	-0.61	-0.30	9.78	26.25	-2.69	-35.69	0.06	-0.06	0.00	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
18														
19	-0.60	-0.28	9.80	26.25	-2.69	-35.69	-0.06	-0.19	-0.06	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
20														
21	-0.60	-0.28	9.80	26.25	-2.69	-35.69	0.00	-0.31	0.00	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
22														
23	-0.60	-0.26	9.76	26.25	-2.69	-35.69	0.00	-0.31	0.00	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
24														
25	-0.60	-0.25	9.77	26.25	-2.69	-35.69	0.00	-0.06	0.19	0.6404	-0.0293	0.0056	0.7674	,20200921_141048,"('192.168.0.181', 44444)"
26														

20200921_141047_192.168.0.181

Figura 4.2.2.1 Ejemplo de un fichero generado en Python

4.2.3. Capa de la representación de las gráficas

En el momento en el que el programa detecta una nueva IP se crea una nueva ventana con las correspondientes gráficas. Por un lado, se pinta el valor de cada sensor (magnetómetro, acelerómetro, giróscopo) con sus 3 ejes (x, y, z) frente al tiempo y una simulación de los valores de la tensión recogida de la galga extensiométrica. Los ejes x, y, z corresponden con los colores rojo, verde y azul, respectivamente.

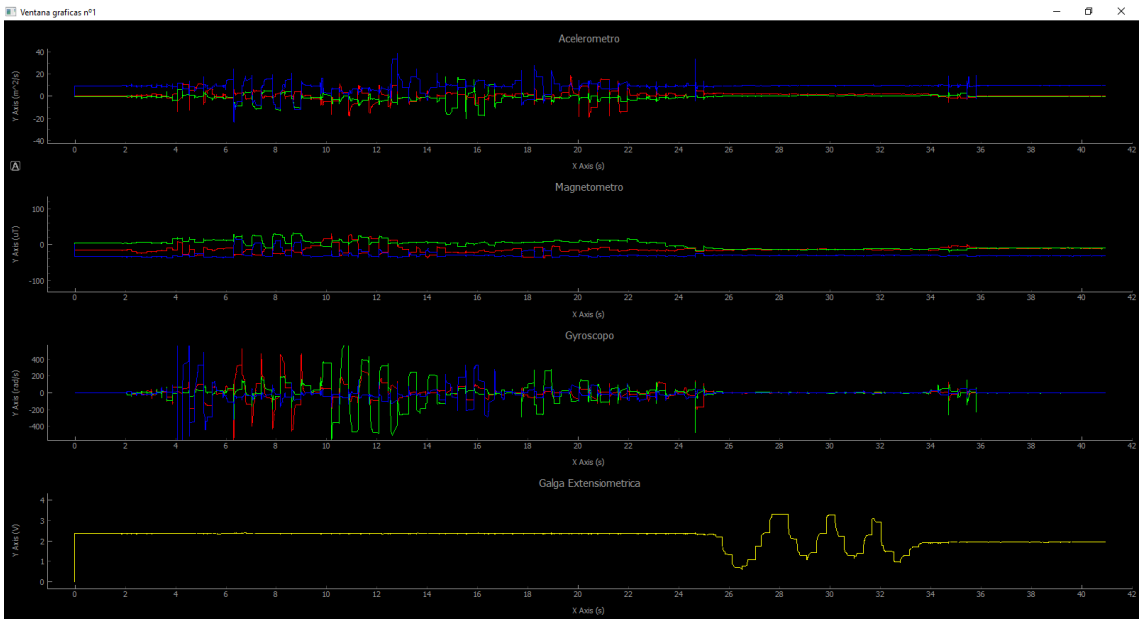


Figura 4.2.3.1 Gráfica de ejemplo de la representación de los valores de los diferentes sensores frente al tiempo

Debido a la gran velocidad a la que se reciben los datos (entre 200 y 300 paquetes por segundo de cada microcontrolador ESP32), estos no pueden ser pintados cada vez que se recibe uno, si no el sistema se quedaría calado. Por eso, se va almacenando cada paquete recibido y se pintan en una gráfica cada 120 paquetes recibidos con un dispositivo (microcontrolador) conectado, cada 240 paquetes con dos dispositivos, cada 360 paquetes con tres dispositivos, y así sucesivamente. Estos valores han sido determinados a prueba y error, es decir se han ido probando diferentes valores y se han escogido los que mejor funcionan.

Además, para estas gráficas se ha usado una librería llamada Pyqtgraph, que genera gráficos más sencillos pero que es capaz de graficarlos más rápidamente. Lo que la hace ideal para realizar unas gráficas lo más próximas al tiempo real.

Por otro lado, se pinta un gráfico con la representación de la orientación en 3D. Estas se pueden abrir y pintar en el momento que se desee sin necesidad de crear ninguna ventana nueva previamente.

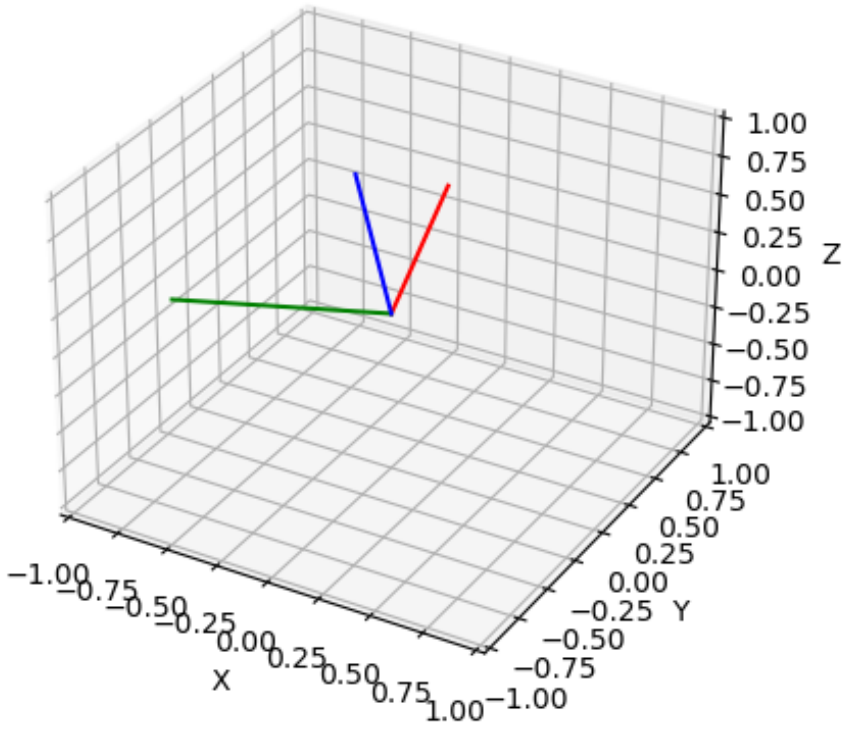


Figura 4.2.3.2 Gráfica de ejemplo de la representación 3D de la orientación del sensor

Esta gráfica se pinta con la misma frecuencia que las gráficas anteriores, pero con la diferencia de que no hace falta ir almacenando cada paquete, sino que únicamente se pinta la orientación en 3D con el último paquete recibido.

En este caso también sería interesante pintar lo más rápidamente posible, aunque en este caso se usa la librería Matplotlib, ya que esta es capaz de pintar gráficos más complejos como es la orientación en 3D, aunque esto suponga graficar, a modo global (todas las gráficas), a una velocidad menor.

4.2.4. Resumen del programa Python

Por último, antes de mostrar un diagrama global de esta parte, cabe comentar que para terminar el programa únicamente hay que mantener pulsada la tecla “s”.

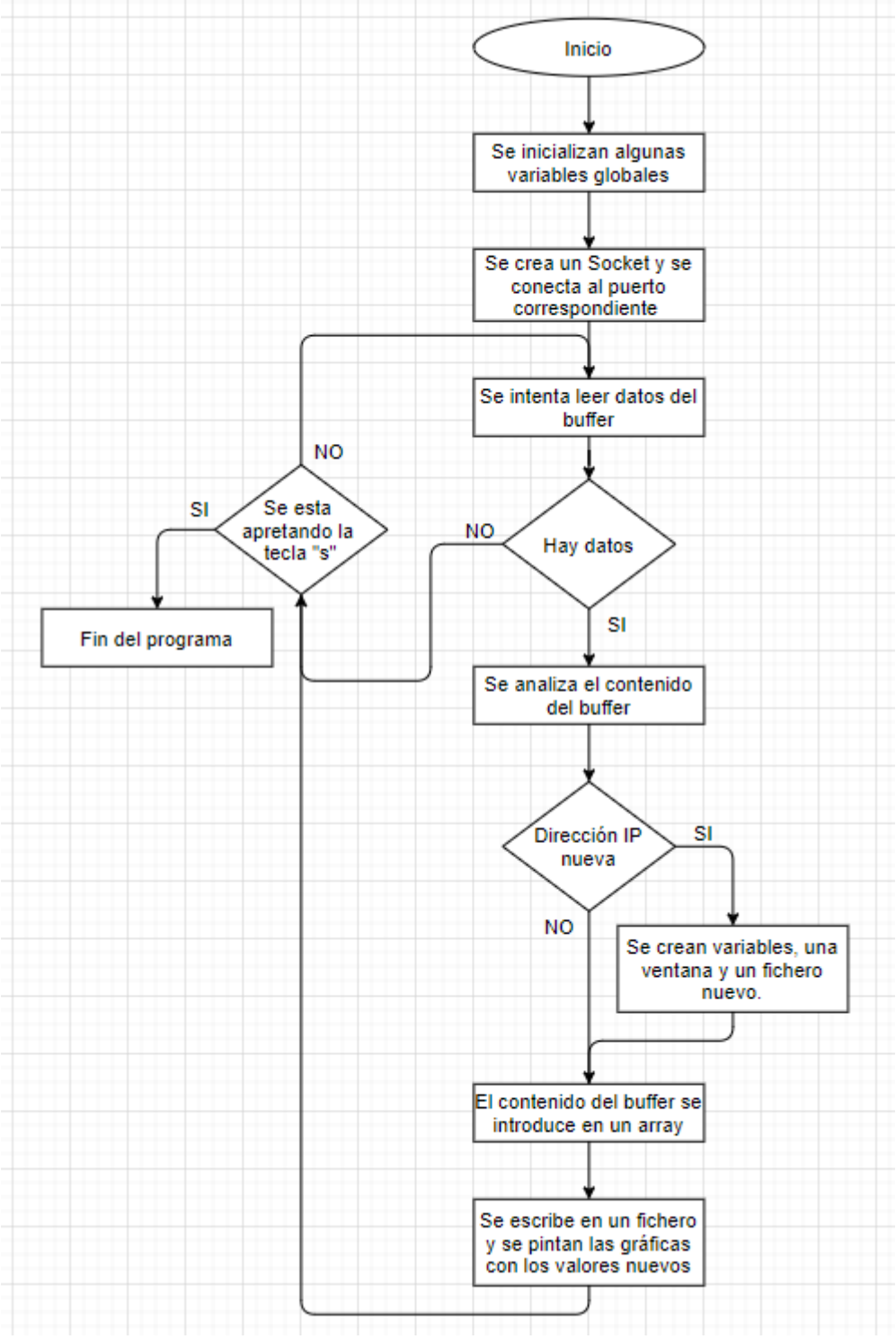


Figura 4.2.4.1 Diagrama de flujo general de Python

4.3. Desarrollo de una aplicación de móvil

Para la creación de una pequeña aplicación móvil capaz de interactuar con el microcontrolador se ha utilizado Appinventor, un entorno de desarrollo de software para la creación de aplicaciones orientadas al sistema operativo Android. Aunque también existen otras alternativas como Kodular, se ha preferido optar por la utilización de Appinventor porque es el entorno que se ha utilizado en la asignatura de Laboratorio de diseño electrónico.

Appinventor es un entorno de desarrollo software muy sencillo de usar, en el que tal y como se muestra en la figura, la programación se realiza mediante la colocación de unos bloques de una manera lógica. Programar de esta manera es muy sencillo y visual, lo que permite que sea apta para cualquier persona no especializada en el tema.

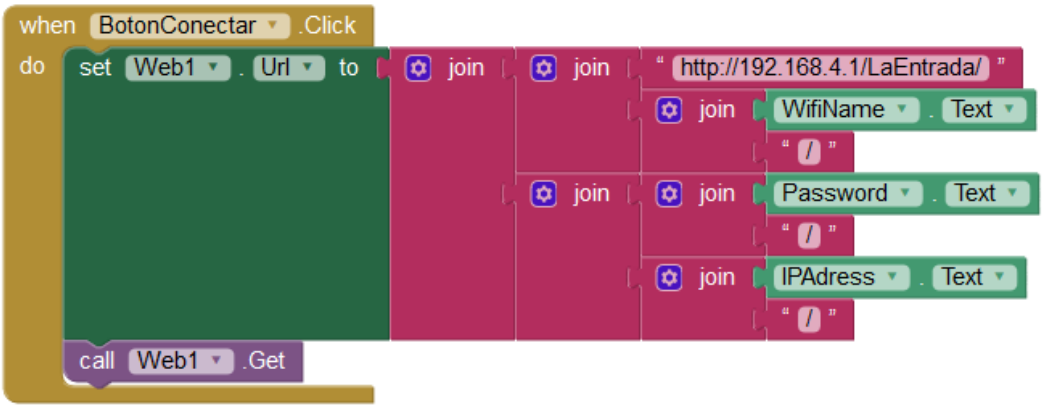


Figura 4.3.1 Programación de la aplicación móvil

Esta aplicación surge de la necesidad de dar el nombre de usuario y la contraseña de la wifi a donde se quiere que se conecte el microcontrolador. Además de indicarle la dirección IP del ordenador a donde se quieren enviar los valores recibidos de los sensores.

Para ello el micro ESP32 se pondrá en modo Access Point, es decir haremos que el microcontrolador genere su propia wifi, para que el dispositivo Android se pueda conectar a él para pasarle la información comentada. Esta wifi generada por el ESP32 tendrá un nombre y una clave predeterminada que el usuario podrá conocer.

En resumen, el microcontrolador generará una wifi, el usuario cogerá su dispositivo Android y lo conectará a dicha red y tendrá que abrir la aplicación creada e introducir los valores requeridos.

En la siguiente figura se muestra la apariencia de la aplicación realizada donde lo único que hay que hacer es entrar en la aplicación rellenar los campos y clicar en conectar.

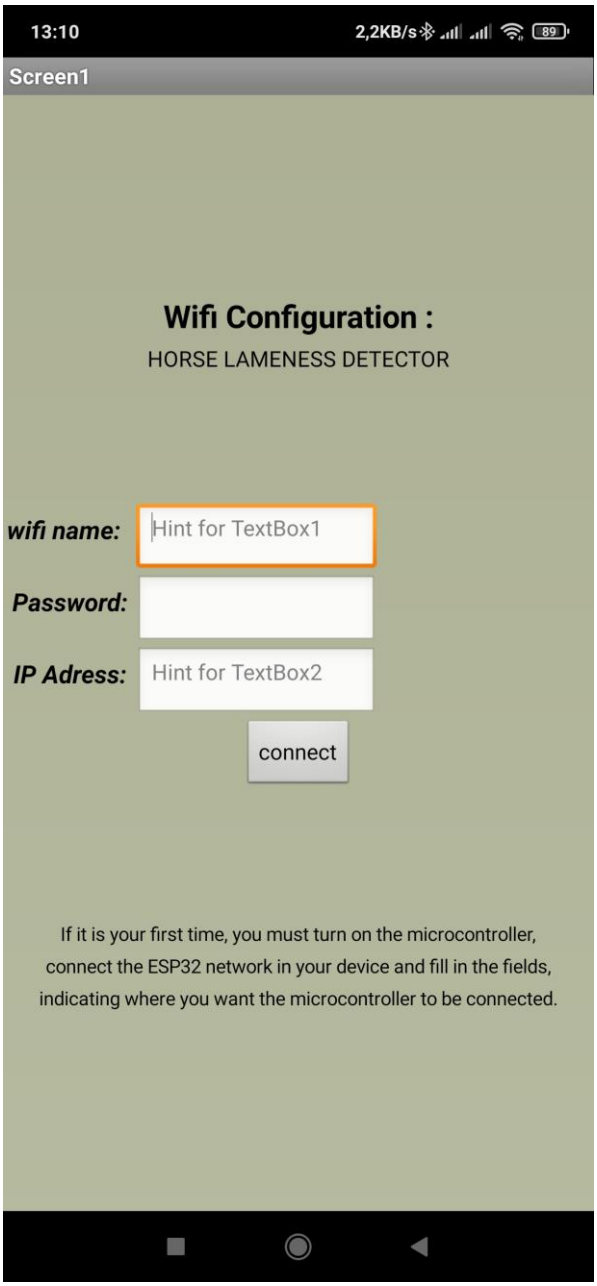


Figura 4.3.2 Aplicación móvil

4.4. Conexión utilizada

Por último, es interesante explicar un poco el protocolo utilizado en la comunicación de los datos entre el ordenador y el microcontrolador.

La organización internacional de normalización (ISO) diseñó el modelo OSI para estructurar las actividades de red. Este modelo es una normativa orientativa dividida en 7 capas, que indican las fases por las que deben pasar los datos para viajar de un dispositivo electrónico a otro. Este modelo se realizó con el objetivo de evitar el problema de incompatibilidad de las redes (Modelo OSI, 2020).

En la tabla se muestran las cuatro primeras capas, donde las tres primeras se encargan del medio físico y la cuarta se encarga más del transporte de los datos.

Capa		Protocolo	
4	De transporte	UDP	TCP
3	De red	IP	
2	De enlace de datos	Ethernet,Token Ring,Wireless,ATM, SNA ...	
1	Física	Bluetooth, USB, DSL...	

Figura 4.4.1 Modelo OSI estructurado en capas

La capa de transporte, donde están los protocolos UDP y TCP, es la que se encarga del flujo de los datos independientemente del medio utilizado.

TCP (protocolo de control de transmisión) establece la conexión entre los dispositivos antes de enviar los datos. Garantiza una fiabilidad a la hora de entregar los datos tanto en el orden como en el contenido.

UDP (protocolo de datagramas de usuario) tiene la ventaja de no necesitar una conexión previa para enviar los datos y de ser más rápido que el protocolo TCP, pero tiene la desventaja de que algunos podrían llegar antes que otros y tampoco tiene garantías de que el paquete haya llegado a su destino.

Como se requiere una transmisión de datos en tiempo real, se ha decidido utilizar en nuestra programación el protocolo UDP porque en nuestro caso sería más perjudicial la presencia de retardos que la pérdida de un paquete de datos ya que se están recibiendo de 200 a 300 por segundo como se ha explicado anteriormente, en el que cada paquete contiene 14 valores de los diferentes sensores conectados.

5. Pruebas con el dispositivo

En este quinto apartado se comentan las diferentes pruebas que se han realizado con el dispositivo. Estas pruebas las he realizado con mis propias piernas, con uno o dos dispositivos dependiendo de la prueba a realizar. En estas pruebas se han utilizado el sensor BNO055 que contiene un acelerómetro, un magnetómetro y un giróscopo, pero no se ha utilizado la galga extensiométrica.

A pesar de que se han realizado varias pruebas en cada caso, solo se mostrará una prueba por cada tipo, para evitar que el documento sea excesivamente extenso.

5.1. Prueba de la detección de la pisada

Lo primero que se ha intentado ver ha sido que el dispositivo es capaz de detectar cuando la "pata del caballo", mi pie, está pisando sobre el suelo. Para conseguir esto, lo primero que se ha tenido que realizar es sujetar el dispositivo con el sensor a la pierna. Para ello se han pegado unas tiras de velcro a la protoboard, tal como se muestra en la siguiente imagen.

Figura 5.1.1 Dispositivo atado a la pierna

Se ha montado una simulación (Figura 5.1.2) con un globo grande y resistente (globo punch ball) lleno de arena y aire, porque, al pisar el pie no se ensancha tanto como la pata de un caballo, y así se consigue este efecto. Para atar este globo al pie se ha utilizado una venda. Por último, se han colocado unos imanes en los extremos del vendaje, de tal manera que el sensor del dispositivo (magnetómetro) pueda detectar el desplazamiento de estos.

Figura 5.1.2 Simulación para imitar el ensanchamiento de la pata de un caballo

A continuación, se muestra la diferencia del ensanchamiento entre el pie apoyado en el suelo y levantado en el aire.

(a) (b)

Figura 5.1.3 (a) pie en el aire (b) pie apoyado en el suelo

Por último, se muestra una gráfica con los valores del sensor recogido únicamente al apoyar y levantar el pie repetidamente.

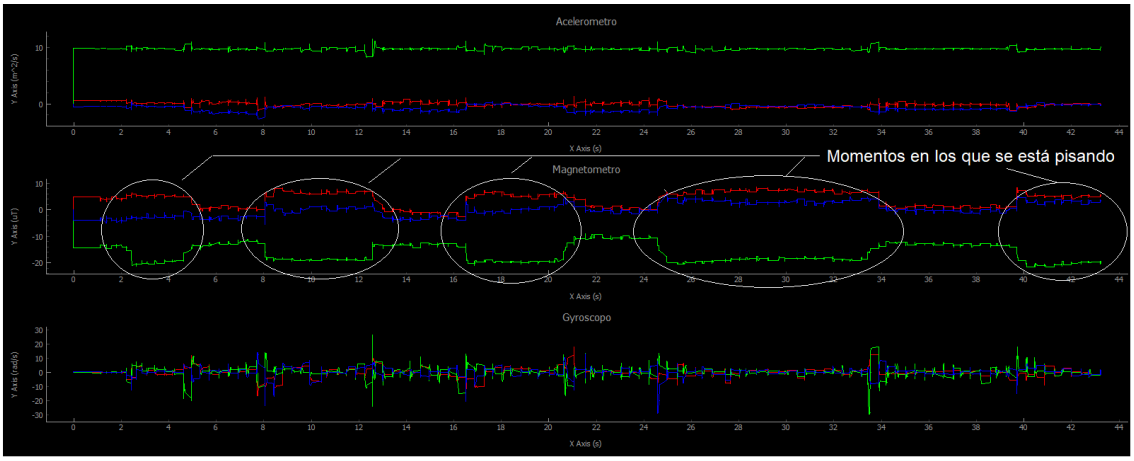


Figura 5.1.4 Gráfica de la prueba de la detección de la pisada

Como se puede ver en la figura, el sensor más adecuado para detectar el momento en el que se está pisando, es el sensor magnético (magnetómetro). En este se aprecian unos escalones debido a la diferencia de separación entre el sensor y el imán.

Por otro lado, también se puede apreciar en la gráfica, tanto con el acelerómetro como con el giróscopo, unos pequeños picos que indican que al tener el pie levantado el pie se mueve un poco (vibra), esto no sucede sí está apoyado en el suelo.

5.2. Prueba caminando con y sin cojera

Para probar la diferencia entre una pisada con cojera y sin cojera se han realizado dos tipos de pruebas diferentes que se desarrollarán a continuación.

5.2.1. Prueba con un dispositivo

Primero se ha probado realizar esta comparativa de caminar con y sin cojera con un único dispositivo, colocado en la pierna derecha, y con el vendaje de la simulación del ensanchamiento de la pata en el pie derecho, tal y como se ha mostrado en el punto anterior.

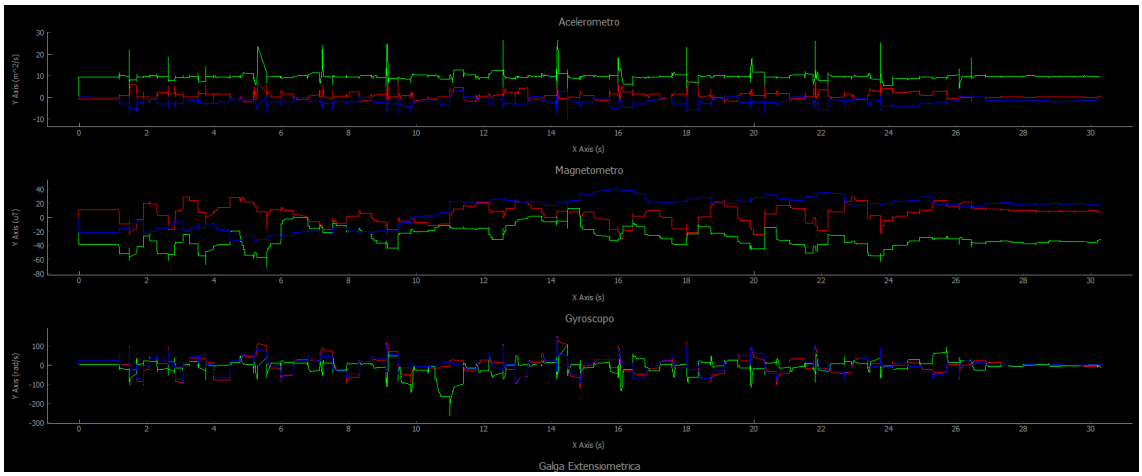


Figura 5.2.1.1 Gráfica de la prueba con un dispositivo y con cojera

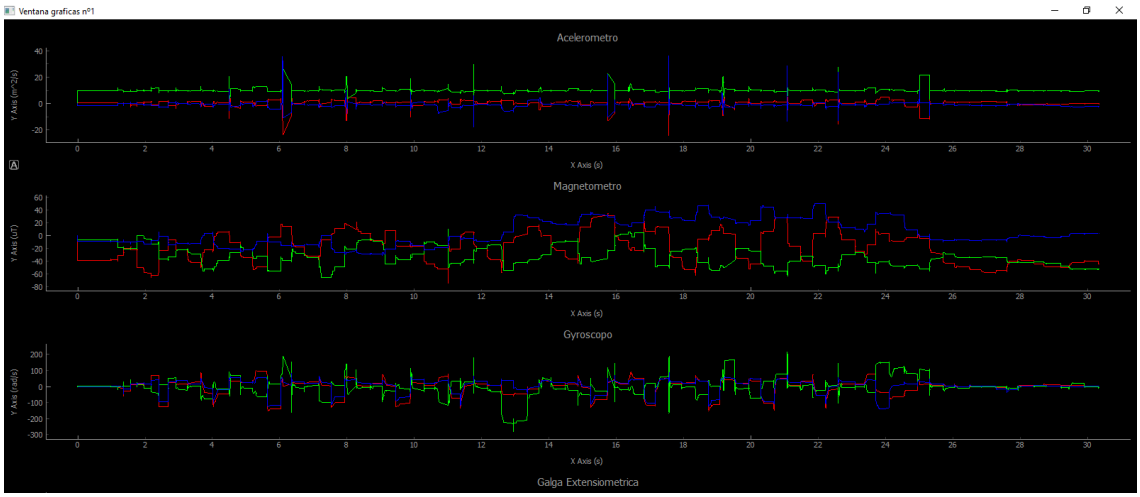


Figura 5.2.1.2 Gráfica de la prueba con un dispositivo y sin cojera

Analizando estas gráficas se puede observar que a pesar de la dificultad que nos encontramos a la hora de diferenciar entre una pisada con cojera y otra sin cojera, se puede ver que al cojear el acelerómetro detecta unos picos mayores en el eje “y” (que corresponde al color verde) ya que, al cojear, se trata de apoyar el mínimo tiempo posible el pie y se realizan unos movimientos más bruscos en la alzada del pie. También al caminar con cojera se puede ver que el magnetómetro muestra unas líneas más planas sin oscilaciones tan marcadas como las mostrada en una pisada sin cojera, ya que la pisada es más suave.

5.2.2. Prueba con dos dispositivos

A continuación, probamos con dos dispositivos, uno en cada pierna, y volvemos a comparar entre caminar con y sin cojera, pero esta vez sin ningún vendaje. Tal y como se muestra en la siguiente imagen.

Figura 5.2.2.1 Dos dispositivos conectados

A continuación, se muestran las gráficas obtenidas.

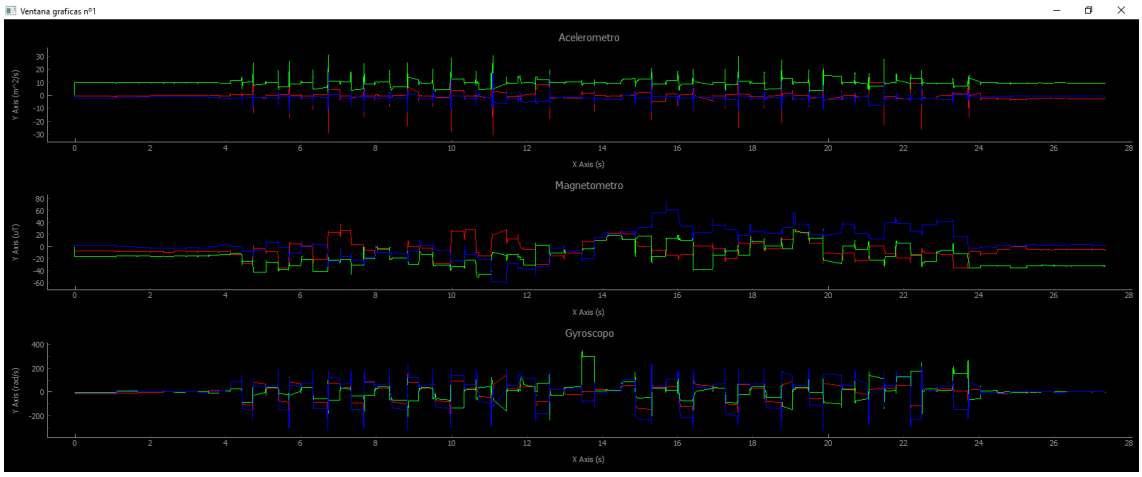


Figura 5.2.2.2 Grafica del pie izquierdo sin cojera

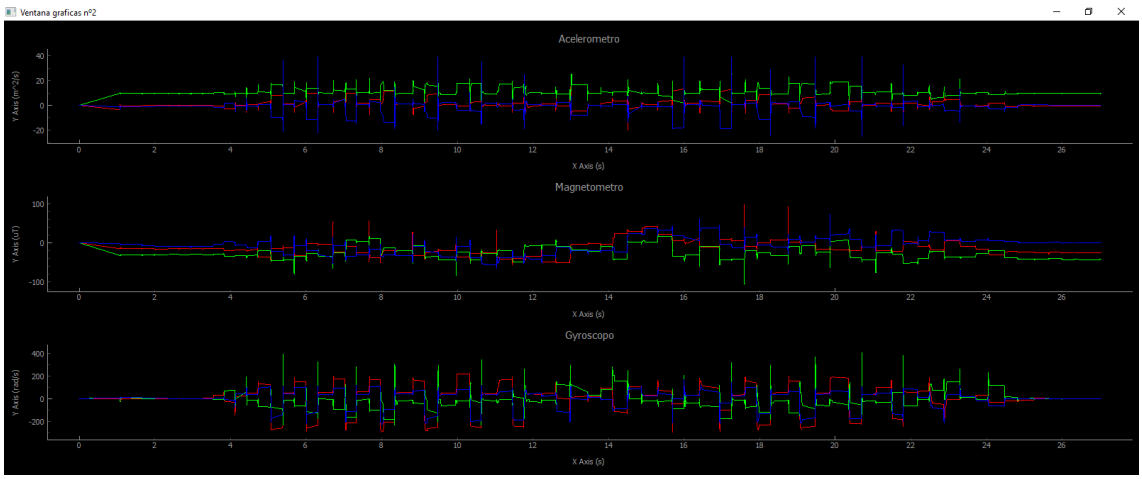


Figura 5.2.2.3 Grafica del pie derecho sin cojera

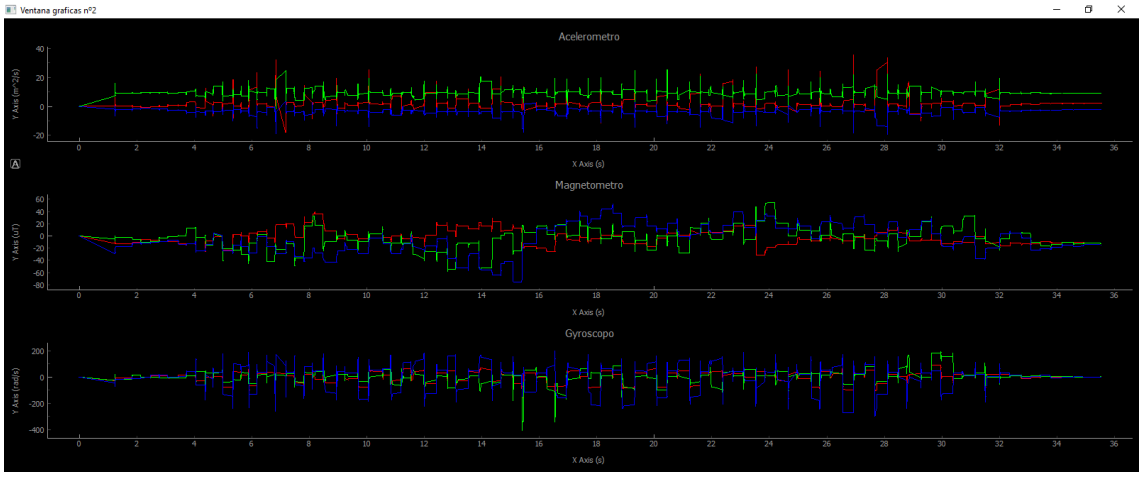


Figura 5.2.2.4 Grafica del pie izquierdo con cojera en el derecho

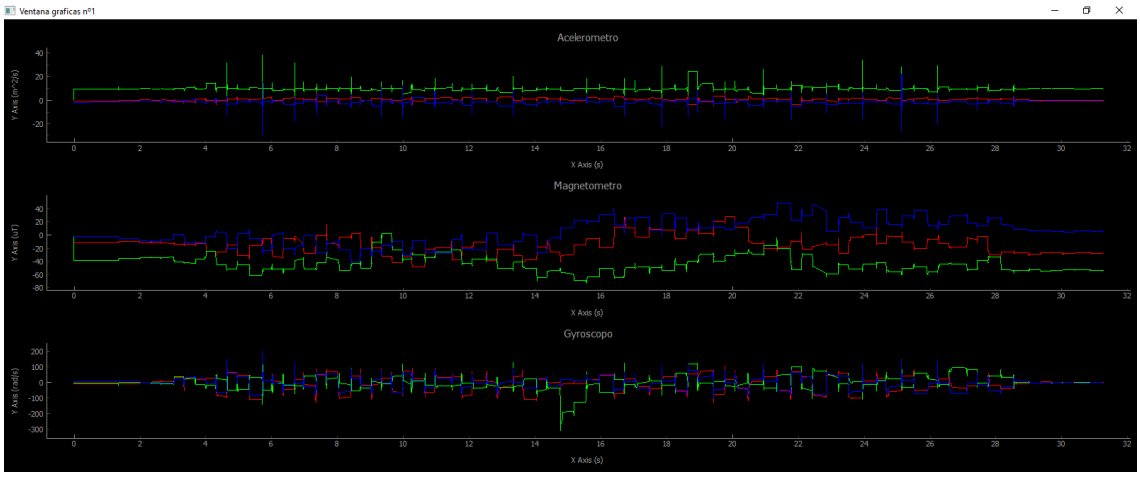


Figura 5.2.2.5 Grafica del pie derecho con cojera en el derecho

En las gráficas se puede observar con los datos del acelerómetro que el pie con la cojera, el derecho, presenta unos picos más acentuados ya que realiza unos movimientos mucho más bruscos que el otro pie y también mayores que cualquiera de los pies al caminar normal. Estos pueden presentar unas oscilaciones con forma escalonada casi cuadrada.

A la vista de los ejemplos, igual que en el apartado anterior se puede observar la dificultad, pero no imposibilidad de diferenciar entre un caminar con cojera y un caminar normal.

6. Conclusión

6.1. Personales

En este trabajo se han aprendido nuevos conocimientos y reforzados otros, tanto en el campo de la electrónica como en el de la programación. También he podido mejorar y coger un poco más de experiencia de cara al futuro en el mundo del diseño electrónico (diseñar esquemáticos y PCBs).

En cuanto a inconvenientes que han surgido, se puede comentar que, inicialmente se tenía la idea de además de realizar el diseño y de programar, hacer también el montaje y la soldadura de los diferentes componentes en la placa de circuito impreso (PCB) diseñada. Pero no ha sido posible por la razón explicada en el capítulo 3.5.

A pesar de este pequeño inconveniente, puedo decir que ha sido agradable y satisfactoria la realización de este proyecto.

También agradezco la colaboración y atención prestada del director y colaboradores del equipo de investigación para la realización del trabajo fin de grado.

6.2. Objetivos alcanzados

Para cada componente seleccionado, se ha realizado una comparativa entre los diferentes componentes que reunían unos requisitos mínimos para poder ser utilizados. Escogiendo finalmente el componente más adecuado en cada caso teniendo en cuenta la eficiencia, precisión y el precio. Dando prioridad a la precisión al tratarse de un dispositivo de medida. Por ejemplo, en el caso de la selección del amplificador de instrumentación.

Se ha realizado el diseño del esquema electrónico (donde se indica la correcta conexión de los diferentes componentes previamente seleccionados). Y el diseño de la placa de circuito impreso (donde se realizan las conexiones físicamente).

Se ha conseguido obtener un programa en Arduino capaz de manejar el microcontrolador (ESP32) para que pueda conectarse a una red wifi en común con un ordenador y para enviarle constantemente vía wifi, los valores tanto de los sensores incluidos en el BNO055 (acelerómetro, magnetómetro y giróscopo) como el valor de la galga extensiométrica.

También se ha logrado obtener un código de programación en Python capaz de recolectar los datos recibidos de los microcontroladores que hayan sido conectados, uno o varios, guardarlos en un fichero con formato .csv, graficar la orientación del BNO055 en 3D y graficar también el valor de los diferentes sensores frente al tiempo con la ayuda de la librería Pyqtgraph que permite una mayor velocidad de muestreo.

Aunque no era un objetivo inicial también se ha conseguido desarrollar una aplicación móvil (Android) capaz de enviar al microcontrolador ESP32 los datos que necesita para conectarse a la wifi.

Para finalizar se puede concluir, que se han podido alcanzar satisfactoriamente los objetivos propuestos al principio de esta memoria.

6.3. Posibles mejoras

En cuanto a opciones de mejora, existe margen de mejora en la fluidez a la hora de graficar en un ordenador con Python, a pesar de que se ha conseguido mejorar bastante incluyendo una librería capaz de graficar más rápido, (donde se ha tenido que modificar bastante parte del código para adaptarlo a esta nueva librería)

Otra parte que se puede llegar a mejorar es la aplicación móvil para enviar el nombre y contraseña de la wifi y la dirección IP donde se quiere que se conecte el dispositivo. Se ha realizado una versión funcional, pero cabe una gran parte de mejora en el diseño de esta aplicación, ya que se ha considerado que al ser una parte opcional (no mandada por ningún profesor) no sería inteligente dedicar un exceso de horas en esta parte.

7. Bibliografía

Allan Davies, M. (2008). Animal instrumentation. Recuperado de https://patents.google.com/patent/EP1956981A1/en?q=pressure+sensor+system+evaluate+horse+lameness+horseshoe&oq=pressure+sensor+system+to+evaluate+horse+lameness+horseshoe+

Kaneene, J. B., Ross, W. A., & Miller, R. (1997). The Michigan equine monitoring system. II. Frequencies and impact of selected health problems. Preventive veterinary medicine, 29(4), 277-292. Recuperado de https://www.sciencedirect.com/science/article/pii/S016758779601080X

Ratzlaff, M. H., Frame, J. M., & Grant, B. D. (1987). Electronic animal hoof force detection systems. Recuperado de https://patents.google.com/patent/US4703217A/en?q=gauge+sensor+system+evaluate+horse+lameness+horseshoe&oq=gauge+sensor+system+to+evaluate+horse+lameness+horseshoe

Keegan, K. G., Dent, E. V., Wilson, D. A., Janicek, J., Kramer, J., Lacarrubba, A., ... & Frees, K. E. (2010). Repeatability of subjective evaluation of lameness in horses. Equine veterinary journal, 42(2), 92-97. Recuperado de https://beva.onlinelibrary.wiley.com/doi/abs/10.2746/042516409X479568

McCracken, M. J., Kramer, J., Keegan, K. G., Lopes, M., Wilson, D. A., Reed, S. K., ... & Rasch, M. (2012). Comparison of an inertial sensor system of lameness quantification with subjective lameness evaluation. Equine veterinary journal, 44(6), 652-656. Recuperado de https://beva.onlinelibrary.wiley.com/doi/full/10.1111/j.2042-3306.2012.00571.x

Ritzinger, R. (2003). Device and method for determining the compression force ratios in a hoof, in particular for horses. Recuperado de https://patents.google.com/patent/EP1369036A2/en?q=pressure+sensor+system+evaluate+horse+lameness+horseshoe&oq=pressure+sensor+system+to+evaluate+horse+lameness+horseshoe+

Pérez García, M. A. (2014). Instrumentación electrónica. Madrid: Paraninfo.

Pérez García, M. A., Álvarez Antón, J. C., Campo Rodríguez, J. C., Ferrero Martín, F. J., Grillo Ortega, G. J. (2006). Instrumentación electrónica. (2ªed.) Madrid: Paraninfo.

Pallás Areny, R. (2003). Sensores y acondicionadores de señal. (4ªed.) Barcelona: Marcombo.

Modelo OSI. (2020,26 de septiembre). Wikipedia, La enciclopedia libre. Fecha de consulta: 00:30, septiembre 27, 2020 desde https://es.wikipedia.org/wiki/Modelo_OSI

Anexos

Anexo A.1. Checklist de la PCB

En las siguientes imágenes podemos ver unas listas con los requisitos seguidos a la hora de hacer el diseño de la PCB. Cabe comentar que en algunos cuadraditos no se ha puesto un tic (√) porque hay cosas que en esta placa en concreto no se necesita llevar a cabo.

1. COMPONENTES		
Asegúrate de que **todos** los componentes de tu BOM (lista de materiales) tienen el encapsulado que has usado en tu PCB	√	
NO utilizar **autoposicionamiento** de componentes. Haz un posicionamiento razonado, para que las pistas queden lo más cortas, directas, ordenadas y sin cruces posibles y agrupando elementos según su función. (ayuda visualizar mentalmente el recorrido que tendrá que hacer la señal de componente a componente. Ver si es un recorrido muy amplio, si pasa por componentes ruidosos, etc)	√	
Posicionar con el esquemático a la vista para ver los componentes y conexiones.	√	
Posicionar pensado en ROOMS agrupando todos los componentes que estén en ellas cerca (luego podremos aplicar reglas especiales a ellas: tamaño de pistas, etc.).	√	
Posicionar en orden de prioridades para evitar mezclar componentes: Señales más agresivas (Alimentacion, señales con fuertes cambios de corriente o de tensión (Clocks, conmutación de transistores, relés,..), señales críticas (sensores analógicos, pines de enable, reset), las demás.	√	
Nombres de componentes basada en números Rxx, Cxx, Uxx (habrán sido adjudicados en el esquemático con el comando annotate)	√	
Nombres de componentes posicionados junto a los componentes y en un mismo sentido para que te ayuden en la soldadura	√	
Elegir encapsulados posibles y lógicos: -Nivel de integración acorde con el tamaño de la PCB. Por lo general no suele ser razonable usar pasivos SMD más pequeños de 1206 -Separación de pads mayor de 0,3 mm. (Excepcionalmente 0,2 mm) Por ejemplo TSOP, SSOP, QFP, LQFP.	√	
La ficha de entrada de 230V ponla con paso mínimo de 5 mm.		
Proveer de la máxima superficie de cobre en la conexión de los pads para que aguante el calentamiento y de un buen soporte al estaño. Se recomienda que el tamaño del pad sea el doble que el taladro que soportan, si no al taladrar pierden mucha superficie de cobre y son difíciles de soldar.	√	
No utilizar componentes cuyos pads sean octogonales ni agujeros cuadrados.	√	
Utilizar las herramientas de alineación de componentes para mejorar el ruteo.	√	

Figura 1 Checklist de los componentes de la PCB

2. PISTAS		
Define las reglas de diseño (anchura de pistas, separación, etc.)	√	
La anchura de pista mínima recomendada es 0,5 mm. Éste es un valor mínimo a utilizar en casos excepcionales. En general es preferible que sean más anchas sobre todo la de masa, siempre guardando coherencia con el tamaño de los pads. En caso de tener un integrado de SMD, la anchura de la pista estará limitada a la anchura del pad correspondiente.	√	
Evitar quiebros en el ruteado, ángulos rectos, cambios de sentido. Intentar que las pistas no salgan de los pads a 45 grados; mejor salir a rectos y luego hacer el giro a 45º.	√	
Por lo general, las pistas sensibles (señales pequeñas, antenas, etc.) que salen de integrados deben de ser de la misma anchura que sus pads para evitar cambios de impedancia.	√	
Evitar pista muy pegadas a los pads del integrado.	√	
La mayoría de las pistas tienen que estar en la cara Bottom, SOBRE TODO en componentes que no se pueden soldar por ambas caras (conectores 2,54mm, fichas, conectores alimentación, header ICs,...).	√	
Asegúrate de que no tienes pistas sin conexión en ningún componente ni partes sueltas de pistas.	√	
Evitar pistas largas, aquellas que no lo cumplan incluir guarda a ambos lados (pista conectada a GND en ambos extremos con una via).	√	
Evitar pistas con muchos cambios de cara (pensar en reposicionarlo de otra manera).	√	

Figura 2 Checklist de las pistas de la PCB

4. PLANOS		
Necesitas planos en ambas caras (TOP y BOTTOM), lo normal es que estén conectados a GND. Al generar los planos de masa seleccionar la opción "Pour Over All Same Net Objects" no "Pour Over Same Net Polygons Only".	√	
Aumentar la distancia de separación entre los planos a cualquier punto (pista, pad, vía,...), el mínimo para que soldéis sin muchos problemas sería 0,5 mm.	√	
En las zonas donde vaya a pasar mucha corriente, aumentar el ancho de las conexiones con el plano (conectores de entradas de alimentación y salida de actuadores).	√	
Para zonas que no tengan conexión con nada y no corte el plano de masa colocar "Polygon Pour Cutout" para eliminar el cobre.		
Evitar islas de cobre, hacer una vía para conectarlo si hay algo conectado, sino eliminarlo.		
En caso de 230V, necesitas cambiar la distancia del plano de masa con el resto de pistas, se recomienda no poner nada a 5 mm de esas pistas (agrandar pistas de alimentación 230V, generar los planos y reducirlos después sin regenerar).		

Figura 3 Checklist de los planos de la PCB

5. MECANIZADO		
Pensad si necesitáis tornillos de sujeción e incluirlos si fuesen necesarios.		
No posicionar ningún componente (excepto conectores y fichas) cerca del borde de la placa dejar un margen.	√	
Necesitas un contorno de placa; hay que dibujar una línea de corte en la Keep-Out Layer, para que la fresadora sepa por donde tiene que cortar la placa.	√	

Figura 4 Checklist de mecanizado de la PCB

Anexo A.2. Código de programación Arduino

```
#include <WiFi.h>
#include <WiFiUdp.h>
#include <EEPROM.h>
#include <Wire.h>
#include <Adafruit_BNO055.h>

#ifdef _ESP32_HAL_I2C_H_   //variables comunicacion I2C
#define SDA_PIN 21
#define SCL_PIN 22
#endif


// EN ESTE PROGRAMA NO HACE FALTA INTRODUCIR NOMBRE Y CONTRASEÑA DE LA
// RED DE CASA O MOVIL. PARA LA PRIMERA VEZ, HAY QUE INSTALAR UNA
// APLICACION EN EL MOVIL E INTRODUCIR AHI LOS DATOS
const char* ssid = ""; //nombre de wifi
const char* pwd = ""; //contraseña

// TAMPOCO HACE FALTA INTRODUCIR LA DIRECCION IP DEL ORDENADOR PORQUE
// TAMBIEN LE PASAREMOS ESTE DATO A TRAVES DE LA APLICACION MOVIL
const char* udpAddress = ""; //direccion ip del ordenador
const int udpPort = 44444;   //sirve "44444" aunque tambien funciona con otros puertos

// Nombre de la Red que abre el microcontrolador para poder introducir los datos
const char* ssidESP32     = "ESP32-Access-Point";
// Se establece el puerto 80 para abrir el servidor web
WiFiServer server(80);
// Variables globales donde recibe los datos de la aplicacion movil
String header;
String ssidRecivido;
String pswRecivido;
String IpAdressRecivido;

// Esta linea es para el BNO055
// Comprueba la dirección del dispositivo I2C (por defecto la dirección es 0x29 o 0x28)
// Se pasa como parametros (id, dirección)
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x29);

// Se crea una instancia UDP
WiFiUDP udp;

//----------------Inicio del programa-------------------
void setup() {
  Serial.begin(9600);          // Configura el puerto serie para pintar en la pantalla
  EEPROM.begin(271);           // Inicializa la eeprom para poder utilizar los 271 primeros valores
  leerCampos();               // Lee las variables de la eeprom(ssid,pwd,udpAddress)
```

```
  configuracionWifi();        // Configura la comunicacion wifi
  configuracionI2C();         // Configura la comunicacion I2C
}

//----------------Bucle Infinito-------------------
void loop() {

  //Actualizamos variables de los sensores y las guardamos en unas variables
  imu::Vector<3> accelerometer = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
  String ax = String(accelerometer.x());
  String ay = String(accelerometer.y());
  String az = String(accelerometer.z());
  String acelerometro = ax + " ; " + ay + " ; " + az + " ; " ;

  imu::Vector<3> magnetometer = bno.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
  String mx = String(magnetometer.x());
  String my = String(magnetometer.y());
  String mz = String(magnetometer.z());
  String magnetometro = mx + " ; " + my + " ; " + mz + " ; " ;

  imu::Vector<3> gyroscope = bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  String gx = String(gyroscope.x());
  String gy = String(gyroscope.y());
  String gz = String(gyroscope.z());
  String giroscopo = gx + " ; " + gy + " ; " + gz + " ; " ;

  String galgaExtensiometrica = String (analogRead(36) / 1240.91) + " ; ";

  imu::Quaternion quat = bno.getQuat();
  String Qw = String(quat.w(), 4);
  String Qx = String(quat.x(), 4);
  String Qy = String(quat.y(), 4);
  String Qz = String(quat.z(), 4);
  String quaternio =  Qw + " ; " + Qx + " ; " + Qy + " ; " + Qz + " ;" ;


  // Juntamos todas las variables en un unico string
  String mensaje = acelerometro + magnetometro + giroscopo + galgaExtensiometrica + quaternio;

  // Inicializamos el buffer
  uint8_t buffer[150];

  // Convertimos los caracteres del string en bytes (numeros ascii)
  int i = 0;
  while (i < 127) {
    buffer[i] = int(mensaje[i]);
    i++;
  }

  //ENVIAMOS EL PAQUETE
  udp.beginPacket(udpAddress, udpPort);
  udp.write(buffer, 127);              // EL paquete que se envia ocupa 84 Bytes
  udp.endPacket();
  //delay(5);
  //memset(buffer, 0, 150); // Introduce el caracter 0 al principio de cada buffer


  /* CON ESTA SENTENCIA PODRIAMOS RECIBIR DATOS DEL ORDENADOR
```

```
    EN NUESTRO CASO NO ES NECESARIO
    AL INTRODUCIR ESTA SENTENCIA SE MANDAN LOS DATOS MUCHO MÄS LENTO
   //receive response from server,
   //processing incoming packet, must be called before reading the buffer
   //udp.parsePacket();
   if(udp.read(buffer, 150) > 0){
   Serial.print("Server to client: ");
   Serial.println((char *)buffer);
   }
  */
}
//----------------Configuracion de la comunicacion wifi--------------------
void configuracionWifi() {
  //Aqui nos intentamos conectar a la wifi a traves de la cual enviamos los datos
  WiFi.begin(ssid, pwd);
  Serial.println("");
  int i = 0;
  //Bucle while hasta que se conecta
  while (WiFi.status() != WL_CONNECTED) {
    delay(1);
    Serial.print(".");
    i += 1;
    if (i % 5000 == 0) {          //si no consigues conectarte en 5s apaga y vuelve a arrancar
      WiFi.disconnect();
      delay(5000);
      WiFi.begin(ssid, pwd);
      Serial.println("");
    }
    if (i % 9999 == 0) {          //si en 10 segundos no consigues conectarte
      WiFi.disconnect();          //se apaga
      inicioAP ();              //abrimos la red propia del microcontrolador(Access point)
      obtenerValorAP();          //aqui se obtienen nuevos valores para conectarte a la wifi
    }
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  udp.begin(udpPort);              //Esto inicializa el udp y el buffer
}

//----------------Inicio Access point-----------------------------------

void inicioAP () {

  // Abrir la red Wi-Fi con el SSID indicado
  Serial.print("Setting AP (Access Point)…");
  WiFi.softAP(ssidESP32);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);
  server.begin();
}
//----------------comunicacion Access point-----------------------------------
//Metodo para comunicarse a traves de Access point y obtener los valores escritos desde la aplicacion
void obtenerValorAP() {
  boolean listo = false;
```

```
 while (listo != true) {
  WiFiClient client = server.available();   // Se escucha a los clientes entrantes
  if (client) {                             // Si se conecta un nuevo cliente...
    Serial.println("New Client.");
    ssidRecivido = "";
    pswRecivido = "";
    IpAdressRecivido = "";
    header = "";
    int contP = 0;
    int contContest = 0;
    while (client.connected()) {            // Bucle mientras el cliente esta conectado
      if (client.available()) {             // Si hay bytes para leer del cliente,
        char c = client.read();             // se lee un byte
        header += c;
        Serial.print(c);
        if (c == '/') {                     // Se guardan las variables que se reciben entre "/"
          contP++;                          // En la aplicacion se escribe:  ip/LaEntrada/ssid/pwd/IpAdress/
        } else {
          if (contP == 2) {
            if (header.indexOf("LaEntrada") == -1) { // Se espera a recibir la palabra LaEntrada,
              contP = 1;                    // para empezar a guardar las letras en variables
            } else {
              ssidRecivido += c;
            }
          } else if (contP == 3) {
            pswRecivido += c;
          } else if (contP == 4) {
            IpAdressRecivido += c;
          } else if (contP == 5) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            listo = true;
            break;
          }
        }
      }
    }
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
 }
 ssid = ssidRecivido.c_str();             // Se introducen las variables recibidas de la aplicación movil en
 pwd = pswRecivido.c_str();               // otras utilizables para abrir la wifi
 udpAddress = IpAdressRecivido.c_str();
 grabar(0, ssidRecivido);                 // Se graban las variables recibidas en la memoria eeprom
 grabar(100, pswRecivido);
 grabar(200, IpAdressRecivido);
}

//----------------Configuracion de la Comunicacion I2C y Sensor BNO055-------------------
void configuracionI2C() {
// Aqui se configura la comunicacion I2C para poder leer los valores
// de los sensores conectados a traves de esta comunicacion.
#ifdef _ESP32_HAL_I2C_H_
```

```
  Wire.begin(SDA_PIN, SCL_PIN);
#else
  Wire.begin();
#endif
  if (!bno.begin()) {
    //Si hay un problema detectando el BNO055 ... comprueba tus conexiones
    Serial.print("Ooops, no se ha detectado el BNO055 ... Revisa tu cableado o I2C ADDR!");
    while (1);
  }
}

//----------------Función para grabar en la EEPROM-------------------
void grabar(int addr, String a) {
  int tamano = a.length();
  char inchar[70];
  a.toCharArray(inchar, tamano + 1);
  for (int i = 0; i < tamano; i++) {       // Se escribe a partir de la direccion asignadada,
    EEPROM.write(addr + i, inchar[i]);    // los caracteres de las variables (ssid,pwd,IpAdress)
  }
  for (int i = tamano; i < 70; i++) {     // Desde el ultimo caracter de cada variable
    EEPROM.write(addr + i, 255);          // se rellena el resto de bytes con un valor de 255
  }
  EEPROM.commit();
}

//-----------------Función para leer la EEPROM------------------------
String leer(int addr) {
  byte lectura;
  String strlectura;
  for (int i = addr; i < addr + 70; i++) {     // Se leen los bytes a partir de la direccion indicada
    lectura = EEPROM.read(i);
    if (lectura != 255) {                      // Los bytes con un valor de 255 no se leen
      strlectura += (char)lectura;
    }
  }
  return strlectura;
}

//----------------Funcion que leer los campos guardados----------
void leerCampos() {
  ssidRecivido = leer(0);                 // Se introduce el valor leido en unas variables utilizables
  ssid = ssidRecivido.c_str();            // para poder conectarse a la red wifi
  pswRecivido = leer(100);
  pwd = pswRecivido.c_str();
  IpAdressRecivido = leer(200);
  udpAddress = IpAdressRecivido.c_str();
}
```

Anexo A.3. Código de programación Python

```
import socket
import time
from timeit import default_timer as timer
import csv
import numpy as np
import math
import keyboard
import matplotlib.pyplot as plt
from pytransform3d.rotations import *
from pyqtgraph.Qt import QtGui, QtCore
import pyqtgraph as pg

def crearFichero(nombreIP,nVentana):
    global writer
    #Se nombra y se abre el fichero
    name=time.strftime("%Y%m%d_%H%M%S", time.localtime())+"_"+nombreIP
    f=open(name+'.csv', 'wt');
    writer[nVentana-1] = csv.writer(f)
    cabecera="acelx;acely;acelz;magx;magy;magz;gyrox;gyroy;gyroz;vGalgExt;quatw;quatx;quaty;quatz;"
    #Se escribe aqui la primera linea del fichero
    writer[nVentana-1].writerow((cabecera,time.strftime("%Y%m%d_%H%M%S", time.localtime())))

def crearGrafica(nVentana,nGrafica,nGrafTitle,numYRange,nGrafLabel):
    global p,win,ejex,ejey,ejez,vAnalog
    numVector = 3*(nVentana-1)+(nGrafica-1)   # Cada array utiliza una posicion para cada grafica
    # Lo primero que hacemos es crear una ventana pero solo con la primera grafica
    if nGrafica==1:
        #Se crea la ventana
        win[nVentana-1] = pg.GraphicsWindow(title="Ventana graficas nº"+ str(nVentana) )
        #Se modifica su tamaño
        win[nVentana-1].resize(1200,800)
    else:
        #nextRow() Pinta la grafica en la siguiente fila y así se pinta una grafica debajo de otra
        win[nVentana-1].nextRow()
    p = win[nVentana-1].addPlot(title=nGrafTitle)   #Se itroduce el titulo de la grafica

    if nGrafica <= 3:        # Solo pintamos en las 3 primeras gráficas el valor de los sensores en los 3 ejes
        ejex[numVector] = p.plot(pen='r')            # El valor del ejex de cada sensor lo pintamos en rojo
        ejey[numVector] = p.plot(pen='g')            # El valor del ejey.....verde
        ejez[numVector] = p.plot(pen='b')            # El valor del ejey.....azul
        p.setRange(yRange=[-1*numYRange, numYRange])   # Limitamos el valor maximo de cada grafica
    else :                                  # En la cuarta grafica pintamos un único valor frente al tiempo
        vAnalog[nVentana-1] = p.plot(pen='y')        # El valor del ejex de cada sensor lo pintamos en negro
        p.setRange(yRange=[0, numYRange])            # Limitamos el valor maximo de cada grafica


    p.setLabel('left', "Y Axis", units=nGrafLabel)  # Etiquetamos los ejes de la grafica
    p.setLabel('bottom', "X Axis", units='s')

def crearVentana(nVentana):
    #Con cada Ventana creamos tres graficas con el acelerometro, magnetometro y giroscopo
    crearGrafica(nVentana,1,'Acelerometro',40,'m^2/s')
    crearGrafica(nVentana,2,'Magnetometro',120,'uT')
    crearGrafica(nVentana,3,'Gyroscopo',500,'rad/s')
    crearGrafica(nVentana,4,'Galga Extensiometrica',4,'V')
```

```
def pintarGraficas(nVentana):
    global gData,ejex,ejey,ejez,vAnalog
    #Actualizamos los valores de toda la ventana
    #Para cada una de las 3 primeras gráficas, pinto los nuevos valores del ejex, ejey, ejez de cada sensor
    for i in range(3):
        ejex[i+3*(nVentana)].setData(gData[nVentana][10], gData[nVentana][3*i])
        ejey[i+3*(nVentana)].setData(gData[nVentana][10], gData[nVentana][1+3*i])
        ejez[i+3*(nVentana)].setData(gData[nVentana][10], gData[nVentana][2+3*i])

    #Para la gráfica numero 4 pintamos el nuevo valor analogico de la galga extensiometrica
    vAnalog[nVentana].setData(gData[nVentana][10], gData[nVentana][9])

    QtGui.QApplication.processEvents()   #Se procesan los nuevos datos introducidos

def pintarejex3D(nVentana):
    global ultiData
    plt.figure("Visualización nº"+str(nVentana+1))     #Le ponemos nombre a la ventana
    plt.title('Orientacion 3D "quaternios"')          #Le ponemos nombre a la grafica
    quat = ultiData[nVentana][10:14]
    rot = matrix_from_quaternion(quat)                #Convertimos los cuaternios en una matriz rotacion
    ax = plot_basis(R=rot, ax_s=1)                    #Pintamos los valores de la nueva matriz de rotacion
    plt.draw()                                    #Forzamos a actualizar la grafica
    plt.pause(.005)                                 #Es necesario hacer una pausa despues del draw(),
                                                #Aunque no noto la diferencia entre poner un valor de tiempo u otro

#------------------COMIENZO DEL PROGRAMA--------------------------
#Inicializamos las varibles de conexion
#Aqui se puede poner la direccion IP del orednador que usa python, o la ip '0.0.0.0'  tambien funciona
HOST = '0.0.0.0'
#Puerto de escucha, debe tener el mismo valor que en arduino, el puerto '44444' funciona, otros tambien
PORT = 44444
#Tamaño del buffer
BUFFER_SIZE = 1024


#Creamos un socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Enlazamos el socket al host y al puerto
s.bind((HOST, PORT))

#Creamos algunas variables
gData = []
ultiData =  []
win = []
ejex = []
ejey = []
ejez = []
vAnalog = []
writer = []
arrayIpESP = []
contadorprint=0
contVentanas = 0
numVentana = 0
existeIp= False

#Bucle infinito hasta que pulsas la tecla 's'
while True:
```

```
#Leemos el contenido del buffer
data = s.recvfrom(BUFFER_SIZE)
if data:
    #Introduzco en datalimpio el contenido del mensaje sin complementos
    datalimpio = str(data[0])
    datalimpio = datalimpio [2:len(datalimpio)-1]

    #Guardo en una variable el valor de la ip
    #y elimino todos los caracteres que no sean de la direccion ip
    ipESP = str(data[1])
    ipESP = ipESP[2:]
    indice= ipESP.find("'")
    ipESP = ipESP[:indice]

    # Inicializo la variable a falso y si ya habiamos recibido datos de esa Ip la ponemos en true
    existeIp= False
    for x in range(contVentanas):
        #Comparamos el valor de la nueva Ip con los que hay guardados
        if arrayIpESP[x]==ipESP:
            #Actualizamos el numero de la ventana donde pintamos los datos recibidos
            numVentana=x
            existeIp= True
            break
    # Si no existe la ip recibida, generamos todo lo necesario para la nueva IP
    if existeIp==False:

        #Añadimos un vector vacio a cada variable para intrudicir los datos de esta nueva ventana
        ultiData.append(np.zeros(14))
        gData.append([[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0]])
        #Iniciamos variables para crear Graficas
        p = None
        win.append([None])
        for i in range (3):
            ejex.append([None])
            ejey.append([None])
            ejez.append([None])
        vAnalog.append([None])
        #Iniciamos más variables para crear un nuevo fichero CSV para guardar los datos
        for i in range (3):
            writer.append([None])
        #Aumentamos el tamaño en una unidad del array para guardar la nueva Ip del ESP32
        arrayIpESP.append(0)
        #Creamos las Ventanas
        crearVentana(contVentanas+1)
        #Creamos el fichero donde guardar los datos de esta nueva ventana
        crearFichero(ipESP,contVentanas+1)

        #El valor de la ventana actual es igual a el valor de la nueva ventana
        numVentana=contVentanas
        #Guardamos en un array el valor de la ip de la nueva ventana
        arrayIpESP[contVentanas]=ipESP
        #Incrementamos el valor del contador de ventanas
        contVentanas +=1

    #inicializamos variables para interpretar el buffer recibido
    contador = 0
    lectura="
```

```
        #Tratamos el buffer caracter a caracter para obtener las diferentes variables y guardarlas en un array
        for x in range(len(datalimpio)):
            #Cada variable esta separada por un ;
            if (datalimpio[x]==';'):
                #Guardamos en un array el valor recibido hasta un ;
                ultiData[numVentana][contador] = float(lectura)
                lectura=''
                contador+=1
            else:
                lectura = lectura + datalimpio[x]

        #Pintamos los valores en la ventana python
        print('Client to Server: ' , datalimpio, data[1])
        fecha="%Y%m%d_%H%M%S"
        #Guardamos lo que se recibe en el fichero
        writer[numVentana].writerow((datalimpio,time.strftime(fecha,time.localtime()),data[1]))

        #Pintamos los valores nuevos
        if(contadorprint==0):
            #Ponemos el tiempo a 0 en el momento de pintar el primer valor en las primeras graficas
            start_time = timer()

        for x in range (10):
            #Introducimos los ultimos valores recividos en un vector
            gData[numVentana][x].append(ultiData[numVentana][x])
        #Guardamos en una variable el valor del tiempo en el que se recibe los datos
        ploting_time = timer() - start_time
        #Esta variable se guarda en un array
        gData[numVentana][10].append(ploting_time)

        #Pintamos 1 valor de cada ("60" * el numero de ventanas que hay)
        muestreoGraf=60*contVentanas
        if(contadorprint%muestreoGraf==0):
            #Alternamos el numero de la ventana que se actualiza
            pintarGraficas(math.floor(contadorprint/muestreoGraf)%contVentanas)
            pintarejex3D(math.floor(contadorprint/muestreoGraf)%contVentanas)
        contadorprint+=1

    #Cerramos el programa si detecta que has pulsado la tecla s
    if keyboard.is_pressed('s'):
        print('se presionó [s] de salir!')
        break
pg.QtGui.QApplication.exec_()
```