UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo fin de Máster

Máster en Ingeniería de Sistemas y Control

SISTEMA PORTÁTIL DE BAJO COSTE PARA LA MEDICIÓN Y REPRESENTACIÓN DE PARÁMETROS CINEMÁTICOS EN 3D

Juan Francisco Navarro Iribarne

Directores:

Dr. José Sánchez Moreno

Dr. David Moreno Salinas

Curso 2020/2021 – Convocatoria Septiembre

Trabajo fin de Máster

Máster en Ingeniería de Sistemas y Control

SISTEMA PORTÁTIL DE BAJO COSTE PARA LA MEDICIÓN Y REPRESENTACIÓN DE PARÁMETROS CINEMÁTICOS EN 3D

Juan Francisco Navarro Iribarne

Directores:

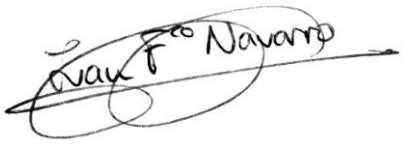
Dr. José Sánchez Moreno

Dr. David Moreno Salinas

HOJA DE CALIFICACIONES

AUTORIZACIÓN

Autorizo a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.



Firmado: Juan Francisco Navarro Iribarne

RESUMEN

El deportista siempre ha buscado, en su afán de competición, el desmarcarse del resto para poder conseguir su objetivo, y es obvio, que el camino para conseguirlo es el entrenamiento. Tradicionalmente, este entrenamiento se realiza empleando procedimientos ya establecidos por la experiencia y conocimiento de los entrenadores, que tratan de perfeccionar la técnica del deportista en base a observaciones directas sobre él. Estas observaciones están limitadas en cierta medida, ya que a simple vista es difícil cuantificar la velocidad de un movimiento o la trayectoria de este, por ejemplo. Esta carencia se puede solventar empleando la monitorización para el análisis deportivo.

El empleo de la monitorización para el análisis deportivo se ha convertido en una herramienta casi indispensable en el entrenamiento profesional de alto nivel. El hecho de poder conocer con exactitud los movimientos realizados por un deportista durante un entrenamiento aportan una gran ventaja con respecto al entrenamiento convencional, ya que si se conoce de forma teórica el movimiento que mejor rendimiento ofrece, el entrenador pretenderá que su deportista aproxime su movimiento real al teórico.

Este trabajo trata sobre el desarrollo de un sistema de bajo coste para la medición y representación de parámetros cinemáticos en 3D. Como ejemplo práctico, se centra en la monitorización del entrenamiento de un deporte, el lanzamiento de disco. Uno de los parámetros más importantes para lanzar el disco a una distancia mayor es la velocidad de salida del disco, así como la trayectoria realizada por el lanzador antes de lanzarlo. Para la captura y procesado de estos datos se ha empleado una unidad de medición inercial (IMU) que va colocada en la muñeca del lanzador y los datos obtenidos son visualizados en un teléfono móvil. Con la ayuda de los datos obtenidos es posible la optimización de lanzamientos futuros con la consiguiente mejora en los resultados obtenidos.

PALABRAS CLAVE

IMU, lanzamiento de disco, Bluetooth LE, Arduino, cuaterniones.

INDICE GENERAL

Contenido

INDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El lanzamiento de disco es posiblemente uno de los iconos más representativos en el mundo del deporte (Figura 1), sobre todo si se tiene en cuenta que se empezó a practicar en el siglo VI a.c. Sin embargo, no fue hasta el comienzo de los Juegos Olímpicos modernos, en el año 1896, cuando se empezó a considerar como un deporte y se estableció un reglamento. Dicho reglamento establecía básicamente que el lanzador se ha de situar dentro de un círculo de 2.5 metros de diámetro y lanzar con una sola mano un disco de 22 cm de diámetro y 2 kg de peso lo más lejos posible [1].

Desde los Juegos Olímpicos de Atenas de 1896 se ha ido sucediendo una mejora continua en cuanto a los récords obtenidos por los lanzadores. El primer registro oficial data de esas olimpiadas y fue de 28.955 metros. Este lanzamiento queda muy lejos del actual récord de 76.89 metros, que se obtuvo en el año 1988. Dicha mejora es fruto del afán de superación que caracteriza a cualquier deportista y sobre todo a la profesionalización de este deporte [2].

Figura 1. Discóbolo de Mirón. Fuente: arte.laguia2000.com

Un deportista realiza su entrenamiento en base a 2 pilares fundamentales, uno es el aspecto físico y el otro es el aspecto técnico. El entrenamiento físico, en el caso concreto del lanzamiento de disco, es aquel destinado a que el lanzador lance el disco con la mayor fuerza posible. Y el entramiento técnico, es aquel que le indicará al lanzador cómo lanzar el disco para que llegue lo más lejos posible. Los primeros lanzadores daban más importancia al aspecto físico, en contrapartida del aspecto técnico. Con el paso de los años, además de mejorar el físico de los lanzadores, mejoró la técnica, y con ello fueron llegando las mejoras en las marcas.

Una de las mejoras en el entrenamiento técnico tiene como aliada a la tecnología. Desde que empezó a formar parte de nuestras vidas se ha empleado en todos los ámbitos, y uno de ellos es el entrenamiento de deportistas. Un ejemplo concreto es la videocámara. Cuando un lanzador de disco quiere mejorar su técnica de lanzamiento, es muy útil conocer qué es lo que hace mal para poder corregirlo [3]. La videocámara permite grabar la secuencia de lanzamiento y posteriormente estudiarla para poder mejorarla. Este sistema tiene el inconveniente de que sólo ofrece información visual del deportista, además de hacerlo solo en un plano de 2 dimensiones. Se han realizado algunos estudios sobre esta materia empleando 2 cámaras para obtener una representación tridimensional del lanzador, pero los resultados no has sido muy favorables.

Pero la tecnología no es un campo que se caracterice por estancarse en el tiempo, si no que avanza día a día. Este avance ha facilitado, por ejemplo, la miniaturización de componentes electrónicos y el desarrollo de nuevas técnicas de captura de variables físicas.

Haciendo uso de estos nuevos avances y con la idea de que es necesario encontrar un sistema de monitorización avanzado del deportista que permita mejorar su técnica, se establece el objetivo de este Trabajo Fin de Máster.

1.1. Objetivo del proyecto.

La grabación en video del lanzador de disco en el momento del lanzamiento y su posterior reproducción puede permitir mejorar ciertos aspectos de su técnica, pero carece de algo muy importante, y es que no existe una cuantificación de los parámetros que intervienen en la mejora de la técnica del lanzador. Estos parámetros son principalmente dos:

- Trayectoria realizada por el lanzador durante el lanzamiento.
- Velocidad de salida del disco.

El objetivo de este TFM es el de captar las variables físicas necesarias durante el lanzamiento del disco para posteriormente visualizar de forma gráfica los parámetros citados.

Esta adquisición se realiza empleando electrónica de bajo coste, y tiene como eje central una placa Arduino. Esta placa incorpora, entre otros dispositivos, un sistema de medición inercial (IMU) que es el encargado de capturar las variables físicas necesarias para el posterior procesamiento del lanzamiento, un módulo de comunicación Bluetooth Low Energy para transferir las mediciones a un teléfono móvil y una batería, que dota al dispositivo de la alimentación necesaria para su funcionamiento. Todos estos componentes han sido ubicados en una carcasa fabricada para tal fin (Figura 2).

Figura 2. Dispositivo.

1.2. Estado del arte

1.2.1. Sistemas de Medición Inercial

- ***¿Qué es una IMU?***

Una unidad de medición inercial o IMU (Inertial Measurement Unit) es un dispositivo electrónico que se emplea para detectar la orientación, localización y movimiento de un equipo a monitorizar. Está compuesto por una serie de sensores (acelerómetros, giroscopios y en algunos casos, magnetómetros), y se basan en la tecnología de sistemas microelectromecánicos (MEMS) [4].

- **Acelerómetros**. El acelerómetro es un sensor que mide la aceleración lineal en cada uno de los ejes. La unidad de medida es la fuerza 'g'. Esta fuerza será de 1 g para cualquier objeto estacionario en la superficie de la Tierra al nivel del mar. Una IMU incorpora 3 acelerómetros, uno por cada eje (X, Y, Z).

- **Giroscopios**. El giroscopio es un sensor capaz de medir la velocidad angular alrededor de un eje. Cada IMU incorpora 3 giroscopios, uno para cada eje (X, Y, Z) y suelen expresar la medida en rad/s.

- **Magnetómetros**. Estos sensores son capaces de medir la fuerza de las líneas magnéticas terrestres que atraviesa cada uno de los ejes. Gracias a estas mediciones, posteriormente se podrá emplear el IMU como una brújula y conocer su orientación determinando la dirección del norte magnético. Una IMU incorpora 3 magnetómetros, uno por cada eje (X, Y, Z).

Atendiendo al número de sensores que incorpore la IMU, se pueden clasificar de 2 formas:

- IMU con 6 grados de libertad (6 DOF). Incorpora 3 acelerómetros y 3 giroscopios.
- IMU con 9 grados de libertad (9 DOF). Además de 3 acelerómetros y 3 giroscopios, incorpora 3 magnetómetros.

Estos dispositivos solo se encargan de captar las variables físicas citadas anteriormente, pero es necesario procesar toda esta información para determinar la orientación y velocidad del dispositivo a monitorizar. Una IMU suele estar ubicado en equipos portátiles, por lo que es usual el empleo de microcontroladores para la gestión de la información que éste aporta.

- ***Errores en las medidas de las variables.***

Cada uno de los sensores que componen una IMU son susceptibles a errores en las medidas y afectarán al resultado final en mayor o menor medida, siendo necesario corregirlos para minimizarlos e incluso anularlos [4].

- **Acelerómetro**. Las fuerzas gravitacionales influyen en las mediciones de las aceleraciones, y han de ser tenidas en cuenta a la hora de interpretarlas para su eliminación.

- **Giroscopio**. Los giroscopios detectan la orientación a través de los cambios de velocidad angular, pero tienden a desviarse con el tiempo porque solo perciben los cambios y no tienen un marco de referencia fijo. La incorporación de datos del acelerómetro a los datos del giroscopio permite minimizar la polarización del giroscopio, lo que resulta en una

estimación de la ubicación más precisa. Los acelerómetros detectan cambios en la dirección con respecto a la gravedad, y esos datos pueden usarse para orientar un giroscopio. El error de medida provocado por la desviación en la medida del giroscopio se denomina “deriva” o “drift” [5].

- **Magnetómetro**. Los magnetómetros son susceptibles a las distorsiones del hierro blando y del hierro duro. Las distorsiones por hierro duro están generadas por fuentes de campos magnéticos que serán sumadas (o restadas) a la lectura del magnetómetro. Ejemplos de este tipo de distorsiones son las generadas por imanes permanentes, fuentes de alimentación o por materiales ferromagnéticos que retienen campos magnéticos residuales tras ser expuestos a grandes campos magnéticos. Las distorsiones por hierro blando representan la magnitud y cambio de dirección que el campo magnético registrado por el magnetómetro experimenta en presencia de objetos ferromagnéticos, por ejemplo, hierro o níquel. Es posible que un magnetómetro se vea sometido a ambas distorsiones de manera simultánea.

Se puede representar de manera gráfica el efecto de estas distorsiones. Si rotamos un magnetómetro 360° en cada eje y se sondean los datos resultantes, los datos formarán un círculo centrado en la ubicación (0, 0) si no hay distorsiones de campo locales causadas por la cercanía de hierro, como se observa en la Figura 3.

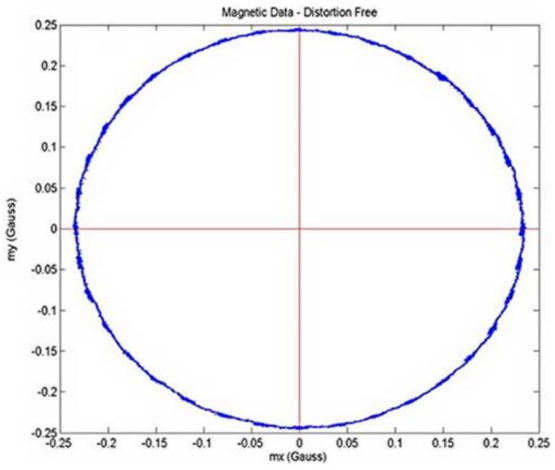


Figura 3. Mediciones sin distorsiones. Fuente: sensorsmag.com

Un efecto de distorsión de hierro duro desplaza los datos magnéticos de la ubicación (0, 0) (Figura 4).

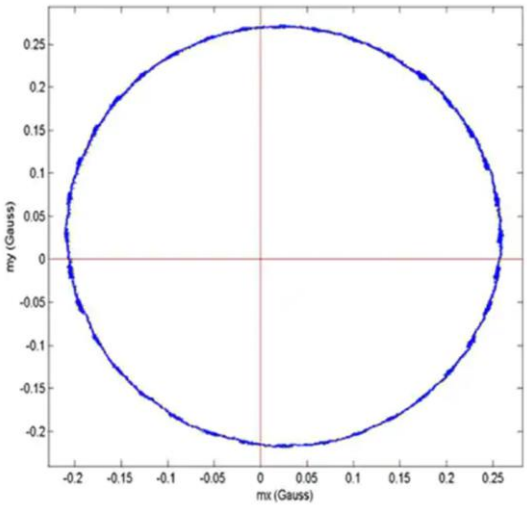


Figura 4. Mediciones con distorsiones por hierro duro. Fuente: sensorsmag.com

Un efecto de distorsión de hierro blando distorsiona los datos magnéticos de 360°, cambiando la forma de círculo a elipse (Figura 5).

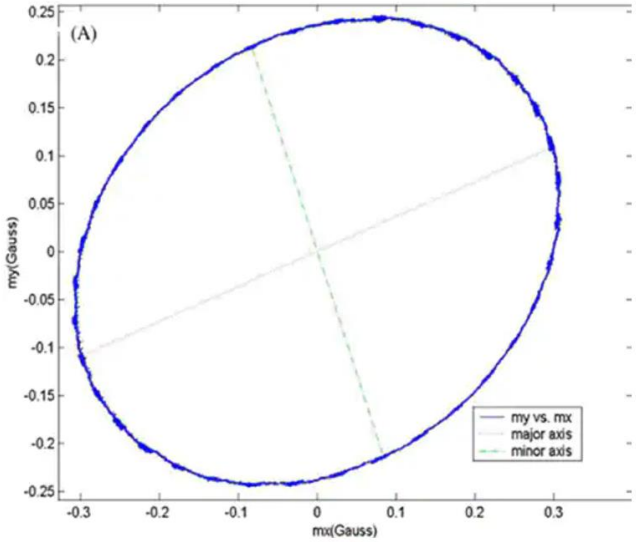


Figura 5. Mediciones con distorsiones por hierro blando. Fuente: sensorsmag.com

- ***Calibración de los sensores.***

El efecto adverso que pueden provocar los errores de los sensores es minimizado mediante la realización de una correcta calibración. Este proceso de calibración y adaptación de lecturas de los sensores se lleva a cabo en el microcontrolador, que lee los valores en bruto y los corrige [4].

- **Calibración del giroscopio.** La forma más sencilla de calibrar el giroscopio de la IMU es calcular el offset de cada eje. Este proceso es sencillo porque la lectura del giroscopio en cada eje estando la IMU en reposo debe ser de 0 grados por segundo (0 dps). Este offset puede ser medido tomando lecturas mientras la IMU no se está moviendo, y será

empleado más tarde para corregir las mediciones del giroscopio cuando esté en movimiento. Este proceso es válido para la mayoría de los usos de una IMU [6].

- **Calibración del acelerómetro.** La calibración del acelerómetro requiere aprovechar la aceleración debida a la gravedad, que podemos utilizar en la orientación positiva y negativa de la IMU. También se dispondrá cada eje del acelerómetro de forma perpendicular a la gravedad terrestre. Esto da como resultado tres valores únicos que se pueden combinar para formular un ajuste lineal entre los tres valores y los valores generados por cada eje del acelerómetro [6].

  Estas tres calibraciones deben realizarse para cada uno de los 3 ejes. Los tres puntos permitirán el ajuste por mínimos cuadrados de un modelo de primer orden al factor de escala y el desplazamiento (pendiente e intersección) de cada eje del acelerómetro.

- **Calibración del magnetómetro**. Durante el proceso de calibración del magnetómetro es muy importante que la IMU se encuentre situada en su ubicación final, es decir, si la IMU irá alojada en una caja con una batería y demás componentes, la calibración hay que realizarla en su disposición final. Esto permitirá que se puedan corregir las posibles distorsiones por hierro blando y hierro duro [7,8].

  Durante la calibración es necesario rotar el dispositivo en todas las direcciones mientras que se toman medidas de las líneas de campo magnético que atraviesan los 3 ejes del magnetómetro. Es necesario conocer la inclinación con la que el campo magnético atraviesa el emplazamiento en el que nos encontramos situados. Con este dato y con las mediciones realizadas, se calcula el offset y la pendiente del ajuste necesario para corregir las distorsiones en cada eje. En la Figura 6 se representan los datos registrados por el magnetómetro antes y después de ser calibrado.

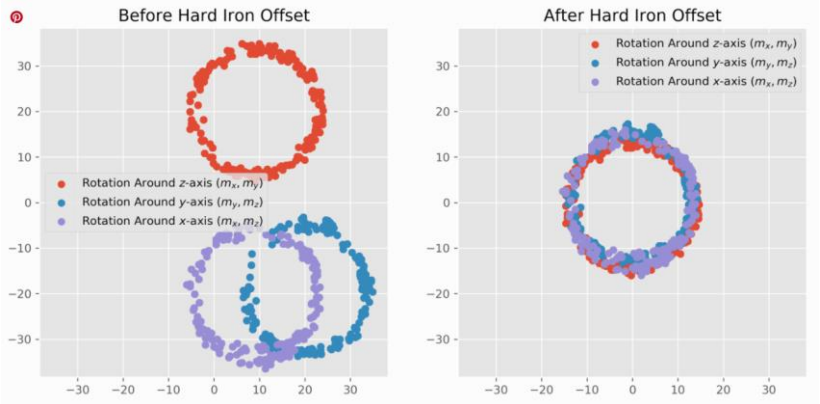


Figura 6. Calibración del magnetómetro. Fuente: makersportal.com

Se pueden dar varias circunstancias por las que sea necesario realizar la calibración de los sensores, ya sea de manera conjunta o de manera individual (solo el magnetómetro).

- **Calibración de todos los sensores.** Esta calibración solo se realiza la primera vez que se monta la placa Arduino dentro de la carcasa.

- **Calibración del magnetómetro**. Esta calibración es recomendable realizarla siempre que se vaya a emplear el dispositivo en una ubicación distinta a la original o si ha habido variaciones importantes en los alrededores de la zona de lanzamiento, por ejemplo,

instalación de postes metálicos para la sujeción de redes. También es recomendable volver a calibrar este sensor si se aprecian medidas muy distintas a las obtenidas originalmente, ya que campos magnéticos cercanos pueden estar variando las mediciones del magnetómetro.

- ***Necesidad de un algoritmo de fusión.***

El principal objetivo de los algoritmos de fusión de datos es el de mejorar la calidad de la información obtenida de procesos con múltiples sensores, conocidos como sinérgicos, ya que actúan conjuntamente para un objetivo común [9].

El utilizar más de un sensor no sólo aumenta la cobertura espacial y temporal de las variables estudiadas, sino que permite aumentar a su vez la supresión del ruido y la precisión del sistema de medición.

Existen varios métodos de filtrado para fusionar los datos del sensor, cada uno con diferentes grados de complejidad.

- **Filtro complementario.** Un filtro complementario es una forma sencilla de combinar datos de varios sensores. Este filtrado se compone de una función lineal que combina un filtro de giroscopio de paso alto y un filtro de acelerómetro de paso bajo. El ruido de alta frecuencia en los datos del acelerómetro, por lo tanto, se filtra a corto plazo y se suaviza con los datos del giroscopio.

- **Filtro Kalman.** Este filtro predice valores utilizando diversas ecuaciones matemáticas basadas en la suposición de que los datos que se filtran toman la forma de una distribución gaussiana, a la que el filtro aplica ecuaciones lineales. El mismo hace uso de las características estocásticas tanto de las señales como del ruido y toma en cuenta la dinámica tanto del proceso, como del proceso de medición.

- **Algoritmo de Madgwich**. Este algoritmo, desarrollado por Sebastián Madgwich, emplea una representación de la orientación por medio de cuaterniones, por lo cual no está sujeto a los problemas de singularidad presentes en las representaciones basadas en matrices de cosenos directores. El algoritmo utiliza el método del gradiente descendente para calcular la dirección del error de la medición del giroscopio. El algoritmo de Madgwich está dividido en cuatro partes principales que son:
    - la integración de las velocidades angulares medidas por el giroscopio.
    - el cálculo de la corrección de estas mediciones con base en los vectores del campo gravitacional medido y del campo magnético.
    - la combinación de las dos estimaciones anteriores.
    - la normalización de los cuaterniones.

- **Algoritmo de Mahony**. Este algoritmo, desarrollado por Robert Mahony, es un filtro complementario, el cual mejora la estimación de la orientación aplicando un filtro pasa bajo a las estimaciones obtenidas de los acelerómetros y magnetómetros. Al mismo tiempo se aplica un filtro pasa alto a las estimaciones obtenidas del giroscopio y por último se fusionan ambas estimaciones. De igual forma que el filtro de Madgwick, el filtro de Mahony se basa en una representación en forma de cuaterniones.

- ***Ángulos Euler vs. Cuaterniones.***

Es posible representar la rotación de un objeto en el espacio de diferentes formas. La más fácil de visualizar es empleando los ángulos de Euler, con sus 3 componentes *yaw*, *pitch*, *roll* (Figura 7).

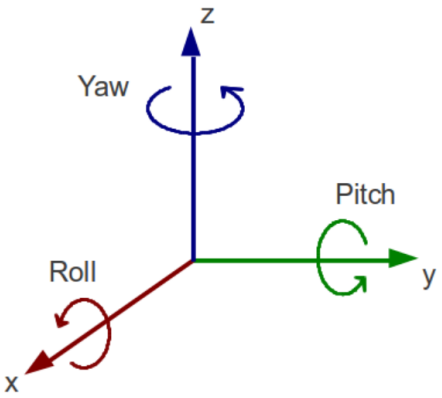


Figura 7. Yaw, pitch, roll

Esta representación propone muchas limitaciones, siendo la principal, y quizás la más importante, el g*imbal lock* [10]. Este fenómeno ocurre cuando 2 ó 3 ejes de rotación están paralelos o muy próximos a ser paralelos, pasando el sistema de tener 3 grados de libertad a 2. En este momento se pierde un grado de libertad y la información referente a este eje desaparece. Cuando aparece el *gimbal lock* es imposible reorientar los ejes sin una referencia externa. En la Figura 8 se muestran diversos ejemplos de *gimbals* con 2 ó 3 ejes paralelos, provocando el *gimbal lock*.

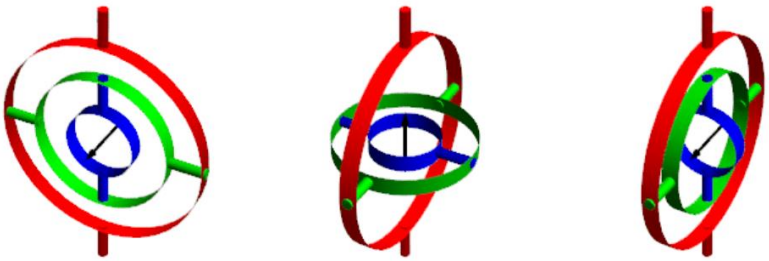


Figura 8. Gimbal lock. Fuente: www.allaboutcircuits.com

Para evitar este problema se emplean los cuaterniones.

- ***Cuaterniones.***

Los cuaterniones pueden ser utilizados como herramienta matemática de gran versatilidad computacional para trabajar con giros y orientaciones [11].

Un cuaternión Q está constituido por cuatro componentes (q_0, q_1, q_2, q_3) que representan las coordenadas del cuaternio en una base {e, i, j, k}. Es frecuente denominar parte escalar del cuaternio a la componente 'e', y '$q_{0\ldots3}$' la parte vectorial. De esta manera, un cuaternio se puede representar como:

$$Q = [q_0, q_1, q_2, q_3] = [s, v]$$

donde ‘s’ representa la parte escalar y ‘v’ la parte vectorial.

Para el empleo de los cuaterniones como metodología de representación de orientaciones se asocia el giro de un ángulo ‘θ’ sobre el vector ‘k’ al cuaternio definido por:

$$Q = Rot(k, \theta) = \left(cos\frac{\theta}{2}, k\ sen\frac{\theta}{2}\right)$$

La composición de cuaterniones es tan sencilla como multiplicar cuaternios entre sí. De tal forma, que el resultado de rotar según el cuaternio Q_1, para posteriormente rotar según Q_2, es el mismo que el de rotar según Q_3, obtenido por la expresión:

$$Q_3 = Q_2 \circ Q_1$$

Es importante tener en cuenta el orden de la multiplicación, ya que el producto de cuaterniones no es conmutativo.

Se observa como el empleo de cuaternios para la composición de rotaciones es un método computacionalmente muy práctico, pues basta con multiplicar cuaternios entre sí, lo que corresponde a una expresión de productos y sumas muy simples.

- ***Matrices de rotación.***

Las matrices de rotación son el método más extendido para la descripción de orientaciones, debido principalmente a la comodidad que proporciona el uso de álgebra matricial. Ésta define la orientación de un sistema móvil con respecto a un sistema fijo, y sirve para transformar las coordenadas de un vector en un sistema a las del otro. También recibe el nombre de matriz de cosenos directores [11].

Supongamos que tenemos 2 sistemas de referencia coincidentes en el origen. Uno es el OXYZ (sistema de referencia fijo) y el OUVW (sistema de referencia móvil). Los vectores unitarios del sistema OXYZ serán $\boldsymbol{i}_x, \boldsymbol{j}_y, \boldsymbol{k}_z$ y los del OUVW $\boldsymbol{i}_u, \boldsymbol{j}_v, \boldsymbol{k}_w$. Un vector **p** del espacio podrá ser referido a cualquiera de los sistemas de la siguiente forma:

$$\boldsymbol{p}_{uvw} = [p_u, p_v, p_w]^T = p_u \cdot \boldsymbol{i}_u + p_v \cdot \boldsymbol{j}_v + p_w \cdot \boldsymbol{k}_w$$

$$\boldsymbol{p}_{xyz} = [p_x, p_y, p_z]^T = p_x \cdot \boldsymbol{i}_x + p_y \cdot \boldsymbol{j}_y + p_z \cdot \boldsymbol{k}_z$$

siendo su equivalencia:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix}$$

donde:

$$R = \begin{bmatrix} \boldsymbol{i}_x\boldsymbol{i}_u & \boldsymbol{i}_x\boldsymbol{j}_v & \boldsymbol{i}_x\boldsymbol{k}_w \\ \boldsymbol{j}_y\boldsymbol{i}_u & \boldsymbol{j}_y\boldsymbol{j}_v & \boldsymbol{j}_y\boldsymbol{k}_w \\ \boldsymbol{k}_z\boldsymbol{i}_u & \boldsymbol{k}_z\boldsymbol{j}_v & \boldsymbol{k}_z\boldsymbol{k}_w \end{bmatrix}$$

es la matriz de rotación que define la orientación del sistema OUVW con respecto al sistema OXYZ.

- ***Matrices de rotación de cuaterniones***

Es posible representar una matriz de rotación empleando cuaterniones mediante la siguiente expresión [12]:

$$R(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

- ***Cálculo de la posición.***

Una IMU es capaz de detectar varias variables físicas, pero entre ellas no se encuentra la velocidad y el desplazamiento realizado, al menos de manera directa [13,14].

El cálculo de la velocidad y posteriormente el desplazamiento, se ha de realizar empleando operaciones matemáticas que tienen como base la aceleración instantánea a la que se somete la IMU.

Cuando se realiza el cálculo de una velocidad y aceleración con respecto al espacio recorrido tenemos que:

$$v(t) = \frac{ds}{dt}$$

$$a(t) = \frac{dv}{dt}$$

Pero en nuestro caso lo que se busca es calcular el espacio recorrido ‘s’ en función de la aceleración ‘a’. Para ello se emplea la operación inversa a la derivada, la integral. La primera integral de la aceleración da lugar a la velocidad, y la segunda integral, al espacio recorrido.

$$v(t) = v(t_{-1}) + \int a\, dt$$

$$s(t) = s(t_{-1}) + \int v\, dt$$

Calcular la posición a partir de la aceleración empleando integrales conlleva un gran inconveniente. Este se debe a que, tanto en el cálculo de la velocidad como en la posición, si existe cualquier error en la medida de la aceleración, sea de la índole que sea, se irá propagando y sumando iteración tras iteración, dando lugar a un valor erróneo, que en ciertas ocasiones puede ser muy grande.

Una de las técnicas empleadas para evitar este problema es el *dead-reckoning*, que consiste en reiniciar la velocidad en determinadas circunstancias para evitar una deriva en la misma [15]. Esta técnica no es aplicable para este TFM, ya que el movimiento es siempre continuo y no hay paradas que permitan dicho reseteo.

Otra de las técnicas que se puede emplear para evitar la deriva de la velocidad es realizar un filtrado paso alto a la señal [16]. Esto permitirá eliminar o reducir la componente continua de la señal.

1.2.2. Bluetooth Low Energy

- ***Introducción.***

Bluetooth Low Energy (BLE), se introdujo como parte de la especificación de Bluetooth 4.0. Aunque existe cierto solapamiento con el Bluetooth clásico, BLE proviene de un proyecto inicialmente desarrollado por Nokia y conocido como 'Wibree' antes de que fuera adoptado por Bluetooth SIG (Special Interest Group) [17]. También es conocido como Bluetooth Smart (Figura 9).

Figura 9. Logo Bluetooth. Fuente: Adafruit.com

Existen diversos protocolos wireless para uso en IOT, pero lo que hace que BLE sea tan interesante es que es el más sencillo para implementar la comunicación entre pequeños dispositivos y una aplicación en cualquier plataforma móvil actual (iOS, Android, Windows phones, etc.), y particularmente en el caso de los dispositivos Apple, es el único método que permite la interacción de periféricos con aplicaciones, sin necesidad de certificaciones MFI y otros requisitos legales que exige iOS.

- ***GAP***

Es el acrónimo para el Generic Access Profile, y se encarga de controlar las conexiones y los anuncios en BLE. GAP es lo que permite que un dispositivo sea público hacia el exterior y determina como dos dispositivos pueden (o no) interactuar entre ellos.

- ***Tipos de dispositivos en función de rol***

El GAP define varios roles para los dispositivos: dispositivos periféricos y dispositivo central.

- Los periféricos son dispositivos pequeños, de baja potencia, de bajos recursos, que pueden conectarse a dispositivos centrales mucho más potentes. Un ejemplo de periférico puede ser un glucómetro, un medidor de pulsaciones, un beacon, etc…

- Un dispositivo central se corresponde normalmente con un teléfono móvil o una tablet que tienen una capacidad de proceso mucho mayor.

- ***GATT***

GATT es el acrónimo de *Generic Attribute Profile*, y define la manera en que dos dispositivos BLE pueden comunicarse usando los Servicios y Características. La comunicación se realiza mediante un protocolo conocido como ATT, que se usa para almacenar los servicios, características y datos relacionados en una tabla usando identificadores de 16-bit para cada entrada en la tabla.

GATT entra en juego una vez se ha establecido una conexión dedicada entre dos dispositivos, lo que significa que ya hemos pasado previamente por el GAP.

Lo más importante a tener en cuenta con el GATT y las conexiones, es que las conexiones son exclusivas. Un periférico BLE sólo puede ser conectado a un dispositivo central (un teléfono móvil, etc.) a la vez. Tan pronto como un periférico se conecta a un dispositivo central, dejará de anunciarse y otros dispositivos ya no podrán verlo o conectarse a él, hasta que se finalice la conexión existente.

Establecer una conexión también es la única forma de permitir la comunicación bidireccional, en la que el dispositivo central puede enviar datos al periférico y viceversa.

- ***Servicios y características***

Las transacciones GATT en BLE se basan en objetos anidados de alto nivel denominados Perfiles, Servicios y Características, y están organizados según el organigrama de la Figura 10.

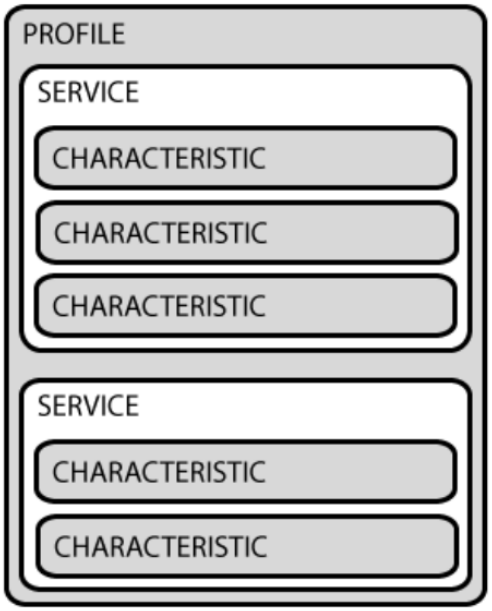


Figura 10. Organización del GATT. Fuente: Adafruit.com

- ***Perfiles***

En realidad, no existe un Perfil en el propio periférico BLE, sino que es una colección predefinida de Servicios que ha sido especificada por el Bluetooth SIG o por el fabricante del periférico. Por ejemplo, el perfil de frecuencia cardiaca combina el servicio de frecuencia cardiaca y el servicio de información de dispositivos.

- ***Servicios***

Los servicios se utilizan para dividir datos en entidades lógicas y contienen trozos específicos de datos llamados características. Un servicio puede tener una o más características y cada servicio se distingue de otros servicios por medio de un ID numérico único denominado UUID, que puede

ser de 16 bits (para servicios BLE adoptados oficialmente) o de 128 bits (para servicios personalizados).

Se puede encontrar una lista completa de los servicios BLE adoptados oficialmente en la página del Portal de desarrolladores de Bluetooth. Por ejemplo, si observamos el Servicio de Frecuencia Cardíaca, podemos ver que este servicio oficialmente adoptado tiene un UUID de 16 bits de 0x180D, y contiene hasta 3 características, aunque sólo es obligatorio la primera: Medición de la frecuencia cardíaca, Sensor corporal Ubicación y Punto de Control de la Frecuencia Cardíaca.

- ***Características***

El concepto de nivel más bajo en las transacciones GATT son las características, que encapsulan un único tipo de dato (aunque puede contener una matriz de datos relacionados, como valores X / Y / Z de un acelerómetro de 3 ejes, etc.).

De forma similar a los Servicios, cada Característica se distingue a través de un UUID predefinido de 16 o 128 bits, y usted es libre de usar las características estándar definidas por el Bluetooth SIG (que asegura la interoperabilidad a través de HW / SW habilitado para BLE) O definir sus propias características personalizadas que sólo su periférico y aplicaciones entienden.

Como ejemplo, la característica de medición de la frecuencia cardíaca es obligatoria para el servicio de frecuencia cardiaca y usa un UUID de 0x2A37. Comienza con un único valor de 8 bits que describe el formato de datos HRM[1] (si los datos son UINT8 o UINT16, etc.), y continúa incluyendo los datos de medición de la frecuencia cardiaca que coinciden con este byte de configuración.

Las características son el elemento principal que vamos a usar para interaccionar con nuestro periférico BLE, por lo que es importante entender el concepto. Pueden ser de solo lectura o de escritura. De esta manera se pueden usar para realizar comunicaciones bidireccionales de una manera muy sencilla. A la hora de implementar el envío de información entre dispositivos BLE, es importante saber que existe una restricción en el tamaño máximo del paquete de datos. Este tamaño es de 20 bytes [18].

[1] Formato de archivo empleado por la empresa Polar para almacenar registros de frecuencia cardiaca. Es el acrónimo de Polar **Heart Rate Monitor**.

1.2.3. Trabajos relacionados

El campo de la monitorización deportiva ofrece un amplio abanico de posibilidades para ser explotado, debido principalmente al fácil acceso y precio reducido de los componentes empleados. A pesar de ello es difícil encontrar un dispositivo que se pueda emplear sin problemas en todas las prácticas deportivas, debido principalmente a la idiosincrasia de los movimientos a medir en cada deporte y a las limitaciones propias de las IMU.

A continuación, se citan una serie de trabajos en los que se monitoriza una práctica deportiva concreta y se representan los datos obtenidos, ya sea en un ordenador o en un teléfono móvil.

a) Wearable de medida de rendimiento en running. Autor: Diego Sánchez Llorach**.** [19].

El objetivo de este trabajo es analizar el rendimiento de un deportista que practique *running* (Figura 11). Para ello se emplea un dispositivo colocado en el tobillo capaz de medir la calidad de la pisada y enviar los consiguientes datos vía Bluetooth al dispositivo móvil.

Figura 11. Deportistas practicando running. Fuente: runninglife.com.mx

Al tratarse de un trabajo que monitoriza el movimiento de la pisada mientras el deportista está practicando *running*, el dispositivo irá colocado sobre el tobillo del deportista (Figura 12).

Figura 12. Dispositivo en el tobillo. Fuente: trabajo citado.

Para la realización de este trabajo, el autor ha empleado el dispositivo STEVAL-WESU1 [20] de la empresa STMicroelectronics (Figura 13). Este dispositivo está especialmente diseñado para aplicaciones de monitorización de movimientos.

Sus principales componentes son:

- STM32L151VEY6 – 32-bit MCU de bajo consumo
- LSM6DS3 – Acelerómetro y giroscopio.
- LIS3MDL – Magnetómetro.
- LPS25HB – Sensor de presión.
- BlueNRG-MS – Bluetooth LE
- BALF-NRG-01D3 - 50 Ω Filtro anti armónicos
- STNS01 – Cargador de batería Li-Ion

Figura 13. STEVAL-WESU1 . Fuente: st.com

Una vez que se han recopilado los datos de los sensores, éstos son procesados y enviados a un teléfono móvil para su gestión. Para ello se ha creado una aplicación bajo Android (Figura 14) con las siguientes opciones:

- Administrar perfiles de usuario.
- Seleccionar el dispositivo mediante Bluetooth.
- Registrar una nueva actividad.
- Ver el histórico de actividades.
- Exportar los resultados a un archivo CSV.

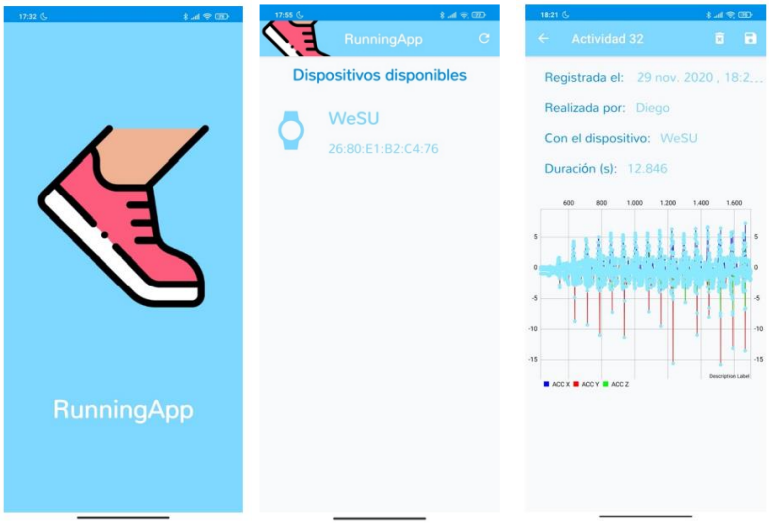


Figura 14. Capturas de la aplicación en Android. Fuente: trabajo citado.

Finalmente, los datos descargados se trasladan a un ordenador para ser analizados e interpretados mediante un programa de análisis matemático. Este envío se realiza mediante Bluetooth. Un esquema del funcionamiento en conjunto se muestra en la Figura 15.

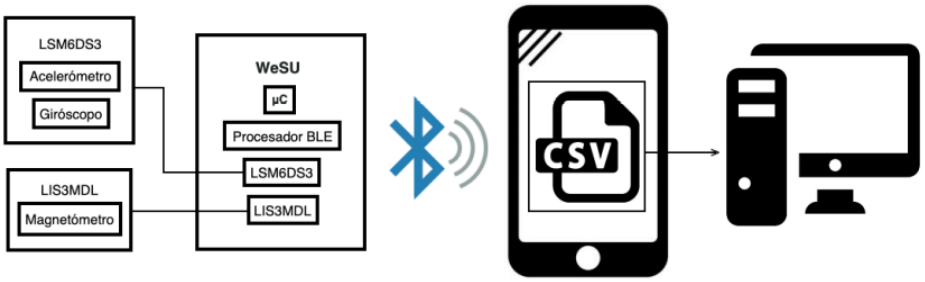


Figura 15. Esquema de funcionamiento. Fuente: trabajo citado.

A pesar de que en la aplicación para el teléfono móvil es posible visualizar las mediciones obtenidas, un análisis más exhaustivo de los datos enviados se llevará a cabo en Matlab, donde se podrán visualizar los valores registrados por el giroscopio, acelerómetro y el magnetómetro (Figura 16).

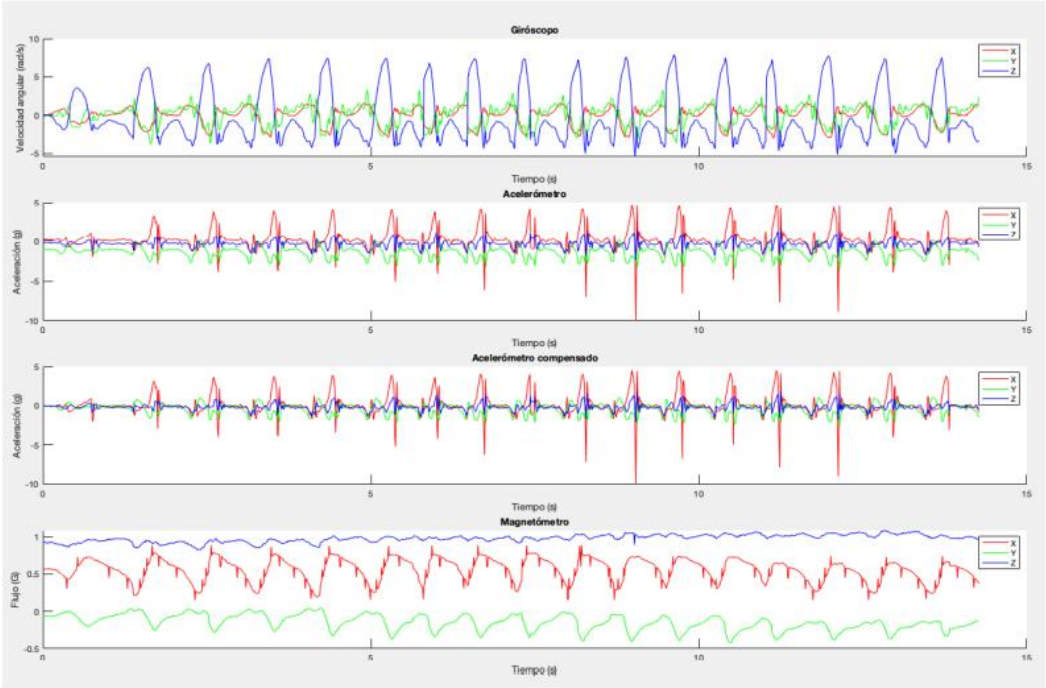


Figura 16. Representación de los datos adquiridos. Fuente: trabajo citado.

b) "Prototipo de dispositivo de medida de rendimiento en deportes de contacto basado en un acelerómetro triaxial y comunicación a dispositivo móvil". Autor: Adrián Luis Arándiga Martínez [21].

El objetivo de este trabajo es la recopilación de datos significativos en los deportes de contacto (en concreto el boxeo) basándose en la información obtenida por una IMU (Figura 17). El hardware es ligero y de tamaño reducido, ya que irá colocado en la muñeca del deportista. Los datos obtenidos son procesados y visualizados en un teléfono móvil.

Figura 17. Distintos tipos de golpeos en boxeo. Fuente: apkpure.com

El desarrollo de este trabajo se realiza empleando el kit de Texas Instruments CC2541 SensorTag (Figura 18) [22]. Se trata de un dispositivo diseñado para el desarrollo de aplicaciones centradas en la adquisición y proceso de datos captados por sensores y con comunicación Bluetooth. Incorpora los siguientes sensores:

- Sensor de temperatura por infrarrojos Texas Instruments TMP006.
- Sensor de humedad Sensirion SHT21.
- Giroscopio Invensense IMU-3000.
- Acelerómetro Kionix KXTJ9.
- Magnetómetro Freescale MAG3110.
- Barómetro Epcos T5400.
- Temperatura interna (incluido en el CC2541).

Figura 18. Texas Instruments CC2541 SensorTag. Fuente: ti.com

Una vez que el dispositivo ha captado los movimientos que se desean analizar, son procesados y enviados a través de Bluetooth a un teléfono móvil Android.

Mediante una aplicación Android creada para este trabajo (Figura 19) se muestra la siguiente información:

- **Datos de aceleración:** se representarán las lecturas del acelerómetro y los máximos alcanzados.

- **Gráficos**: los datos de aceleración se representarán en gráficos dinámicos y estáticos.

- **Detección de guardia:** la interpretación de datos determinará la guardia empleada por el usuario de entre las posiciones de boxeo.

- **Clasificación de golpe**: se compararán los datos con una base de datos para determinar qué tipo de golpe se ha realizado y se contabilizarán.

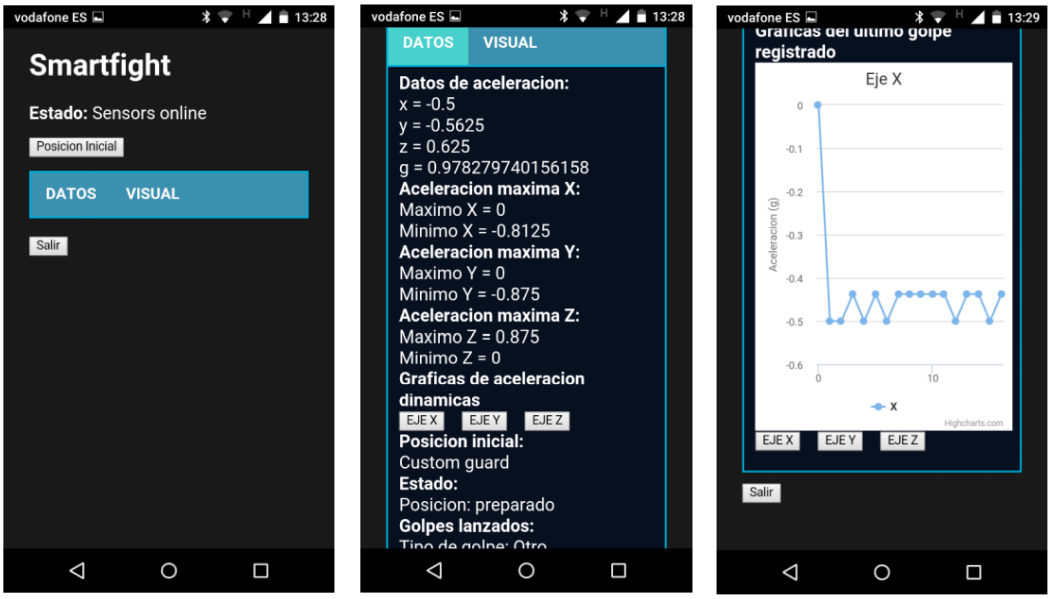


Figura 19. Capturas de pantalla de la app. Fuente: trabajo citado.

c) "Using Wrist-Worn Activity Recognition for Basketball Game Analysis". Autores: Alexander Hölzemann, Kristof Van Laerhoven [23].

En este trabajo se ha desarrollado un dispositivo que permite la monitorización de los movimientos de un jugador de baloncesto (Figura 20), para posteriormente, gracias a la inteligencia artificial, poder predecir con exactitud de qué tipo de movimiento se trata.

Figura 20. Ejemplos de registro. Fuente: trabajo citado.

El dispositivo se coloca en la muñeca de cada deportista (Figura 21). Consta de una pequeña caja que incorpora una IMU (MPU-9250), un microordenador y una batería.

Figura 21. Disposición del wearable. Fuente: trabajo citado

El desarrollo de este trabajo se basa en un potente microordenador, que, a pesar de su potencia, tiene un tamaño y peso ideal para ser usado como wearable. Se trata del EDI2.SPON.AL.S (Figura 22) de *Intel Corporation*. Las principales características de este equipo son:

- Procesador principal Intel Atom Silvermont a 500 MHz.
- Procesador secundario Intel Quark a 100 MHz.
- 1 GB de memoria RAM.
- 4 GB de memoria en tarjeta eMMC.
- Bluetooth LE.
- Ejecución de programas bajo Linux.

Figura 22. Placa EDI2.SPON.AL.S. Fuente: intel.com

Cuando el dispositivo ha capturado los datos de los sensores correspondientes a movimientos del jugador, éstos son empleados para entrenar una red neuronal que posteriormente será capaz de determinar de qué movimiento se trata.

Los movimientos a detectar son:

- Regate corto.
- Regate largo.
- Pase.
- Tiro a canasta.

Para cada uno de estos movimientos se han capturado los datos del acelerómetro y giroscopio. (Figura 23).

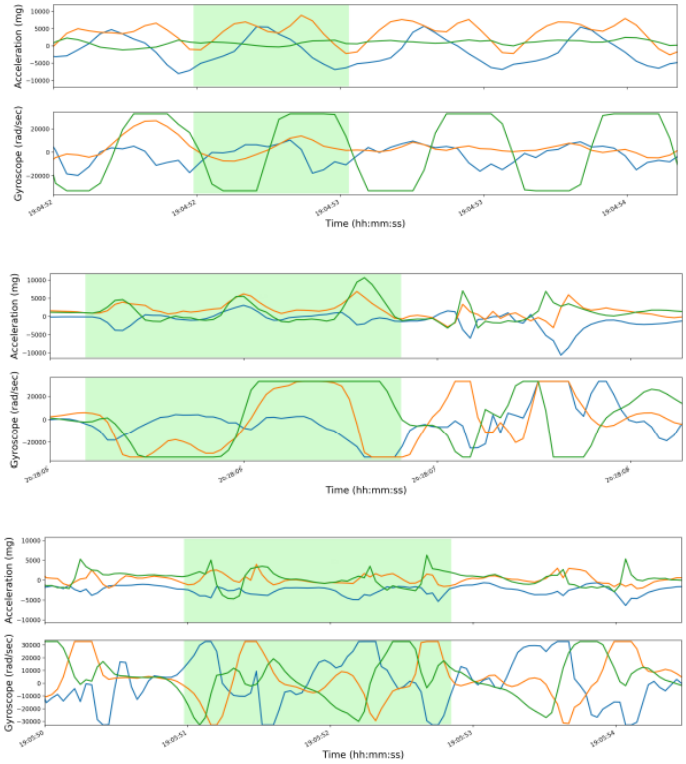


Figura 23. Registros de diversos movimientos del jugador de baloncesto. Fuente: trabajo citado

2. DISEÑO Y FABRICACIÓN

La fabricación y puesta en funcionamiento de este dispositivo se ha llevado a cabo en 2 etapas:

- Diseño y fabricación de la carcasa que alberga los elementos que componen el dispositivo.
- Interconexión de los elementos que componen el dispositivo.

2.1. Diseño de la carcasa

Se ha diseñado una carcasa lo más compacta posible con la idea de que pueda ser llevada cómodamente en la muñeca del lanzador de disco. La sujeción a la misma se realiza con una cinta de velcro que pasa por debajo de la carcasa y permite una firme sujeción. Para la realización del diseño se ha empleado el software de modelado 3D *"Fusion 360"* (Figura 24). Los planos de la carcasa del dispositivo se encuentran en el Anexo I.

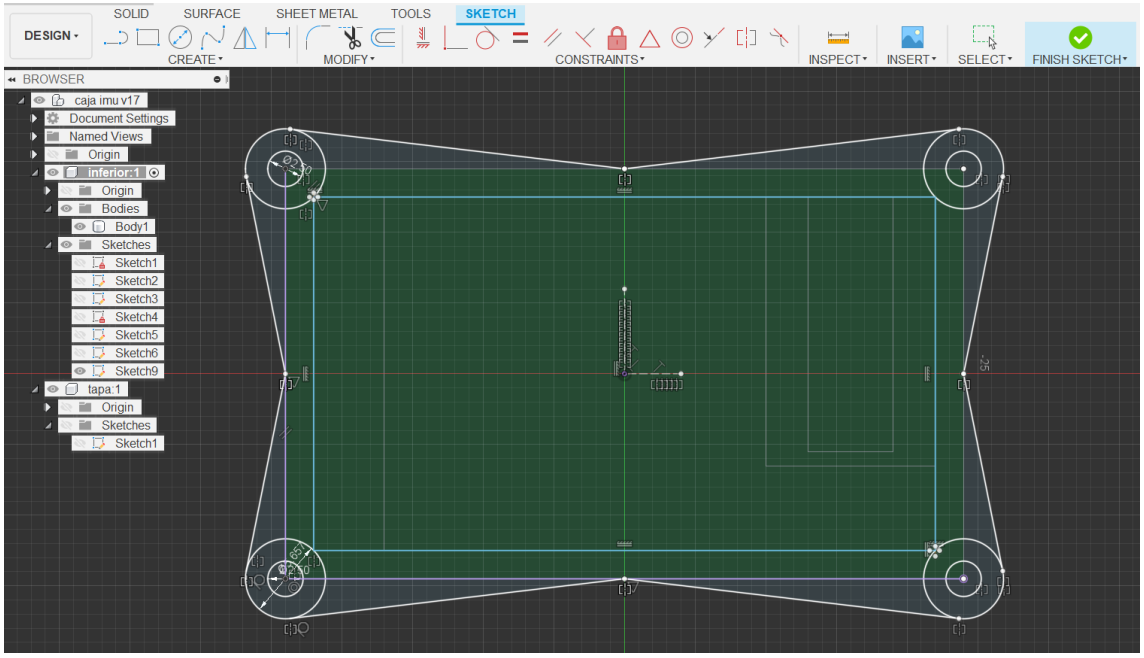


Figura 24. Pantalla de diseño de la carcasa en el software Fusion 360

2.2. Fabricación de la carcasa.

La fabricación de la carcasa se ha realizado mediante una impresora 3D del tipo FDM (deposición de material fundido). El material empleado para la impresión es PLA.

La carcasa está dividida en 2 partes: el cuerpo y la tapa. Ambas piezas son ensambladas mediante 4 tornillos situados en las esquinas. Una muestra renderizada de la misma se muestra en las Figuras 25, 26 y 27.

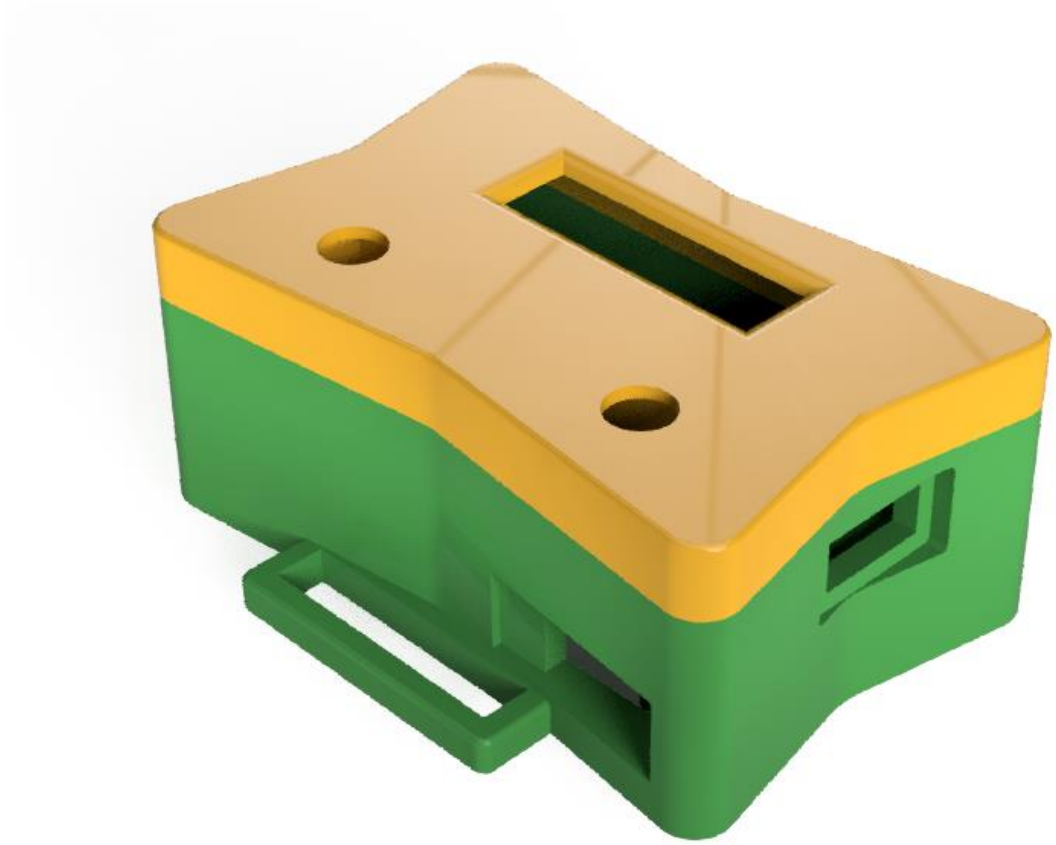


Figura 25. Diseño de la carcasa renderizada

Figura 26. Diferentes vistas de la parte inferior

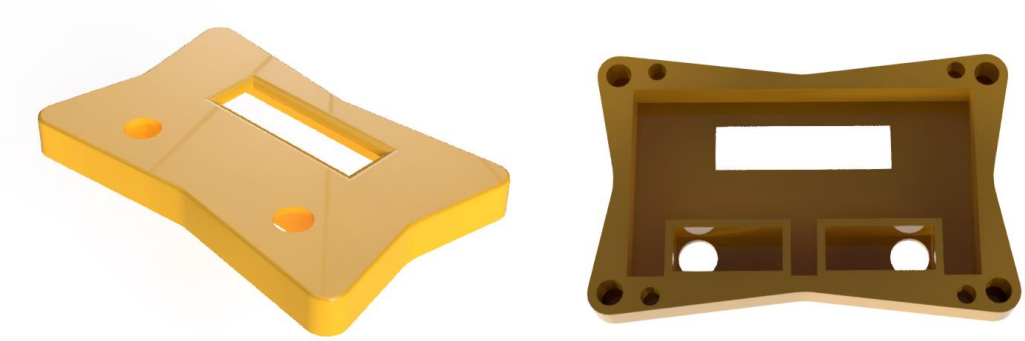


Figura 27. Diferentes vistas de la tapa

2.3. Componentes del sistema

Los elementos que componen este dispositivo son:

2.3.1. Arduino Nano 33 BLE.

Esta placa (Figura 28) es el núcleo central del dispositivo. Incorpora 3 elementos que la hacen ideal para el desarrollo de este trabajo [24]:

Está construida con el **módulo NINA B306**, basado en el procesador nRF 52840 que contiene un potente Cortex M4F. Este microcontrolador (totalmente compatible con Arduino) tiene una velocidad de reloj de 64 MHz y posee una memoria Flash de 1Mb y una memoria RAM de 256 Kb.

Incorpora una **IMU de 9 ejes**. La incorporación del chip LSM9DS1 en la misma placa reduce la complejidad del montaje final, ya que nos evitamos conexiones externas.

Conexión **Bluetooth LE**. El propio módulo NINA B306 es el encargado de administrar esta conexión inalámbrica. Al igual que sucede con la IMU, al estar integrada en la misma placa el emisor/receptor de bluetooth reduce la complejidad y tamaño del montaje.

Figura 28. Placa Arduino Nano 33 BLE. Fuente: Arduino.cc

Es necesario realizar una modificación en esta placa para poder alimentarla externamente a 3.3V. Esta consiste en romper el jumper mostrado en la Figura 29 [25].

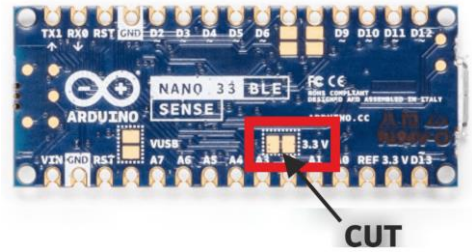


Figura 29. Modificación 3.3V. Fuente: arduino.cc

2.3.2. Batería LiPo de 3.7V.

Esta batería es la encargada de suministrar la alimentación a todo el dispositivo (Figura 30). Se encuentra conectada a la entrada de 3.3V del Arduino Nano 33 BLE, previo paso por un diodo Schotty colocado en serie para reducir la tensión de alimentación de 3.7V a aproximadamente 3.3V, evitando así dañar la placa.

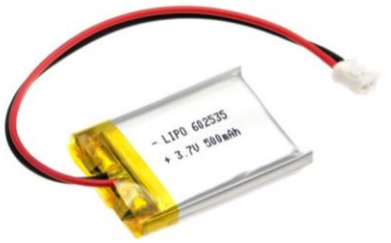


Figura 30. Batería LiPo 3.7V. Fuente: Aliexpress.com

2.3.3. Controlador de carga de batería.

Este pequeño circuito controla la carga de la batería LiPo (Figura 31). Se alimenta con los 5V de la conexión USB de la placa Arduino.

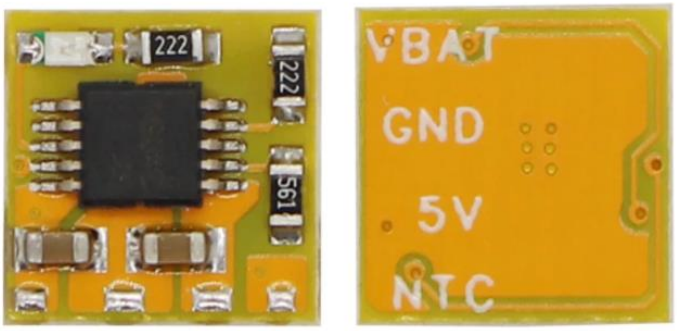


Figura 31. Controlador de carga. Fuente: Aliexpress.com

2.3.4. Motor vibrador.

Para hacer vibrar el dispositivo se ha empleado un vibrador de teléfono móvil que funciona a 3.3V (Figura 32). Este pequeño motor está controlado mediante un transistor NPN configurado en modo corte – saturación.

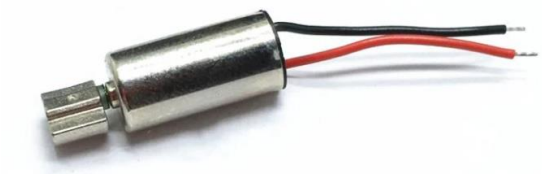


Figura 32. Vibrador. Fuente: Aliexpress.com

2.3.5. Pantalla OLED.

Una pequeña pantalla de 0.91” da información acerca del estado o etapa en la que se encuentra el dispositivo (Figura 33). Las características intrínsecas de una pantalla OLED permite que se pueda ver sin problemas en exterior, aún con luz solar directa.

El chipset que incorpora esta pantalla es el SSD1306 y la conexión con la placa Arduino Nano 33 BLE se realiza mediante el protocolo I2C.

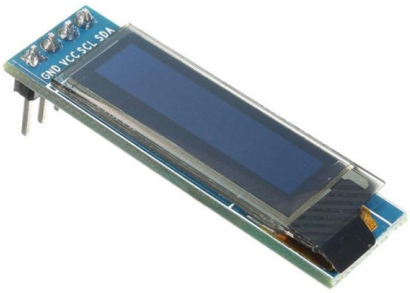


Figura 33. Pantalla OLED 0.91". Fuente: Aliexpress.com

2.3.6. Pulsadores.

Se emplean 2 pulsadores en el dispositivo (Figura 34). Uno de ellos es para comenzar la captura de datos y el otro es para reiniciar el dispositivo y poder realizar otra captura. Estos pulsadores tienen un tamaño bastante reducido, lo que los hace idóneos para este montaje.

Figura 34. Pulsador en miniatura. Fuente: Omron

2.3.7. Interruptor de deslizamiento.

Se emplea para encender y apagar el dispositivo (Figura 35).

Figura 35. Interruptor de deslizamiento. Fuente: Mouser.com

2.3.8. Diodo Schottky.

Este diodo se emplea para reducir la tensión de la batería (3.7V) a la tensión de alimentación de la placa Arduino Nano 33 BLE (3.3V). El modelo usado es el BAT48, con una tensión de codo de 0.4 V (Figura 36) [26].

Figura 36. Diodo Schottky. Fuente: farnell.com

2.3.9. Transistor NPN.

Este transistor, empleado de la forma "Corte – Saturación" se usa para controlar el motor vibrador. El modelo es el BC847 en formato SOT23 (Figura 37) [27].

Figura 37. Transistor BC856B. Fuente: farnell.com

2.4. Esquema eléctrico

En la Figura 38 se muestran las diferentes partes del esquema eléctrico.

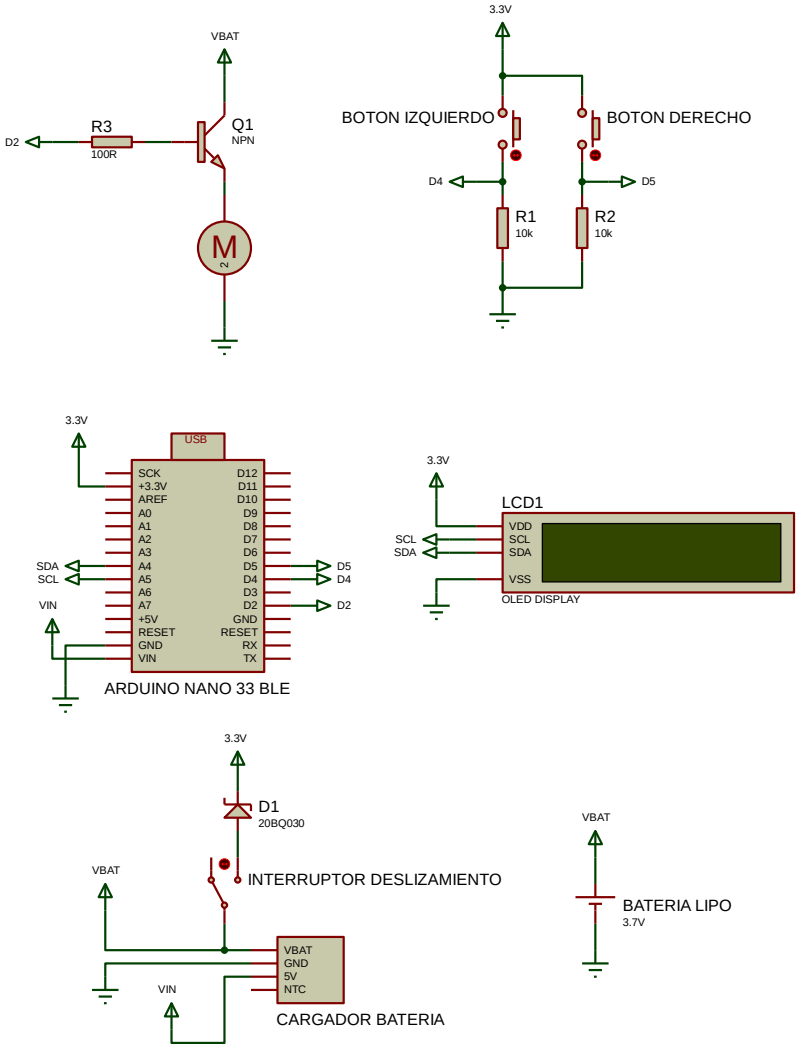


Figura 38. Esquema eléctrico del montaje.

2.5. Imágenes del montaje

En las Figuras 39, 40 y 41 se muestra el montaje final del sistema y diferentes vistas de sus componentes.

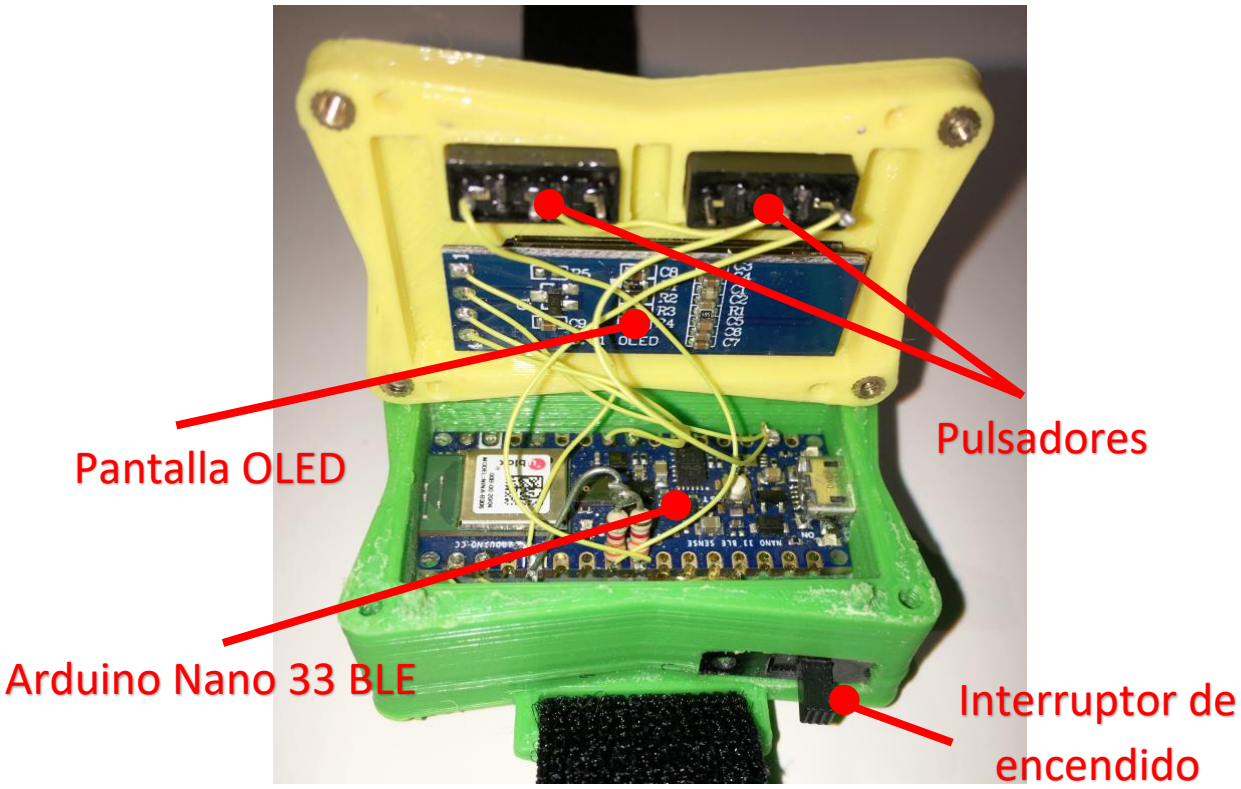


Figura 39. Montaje (1)

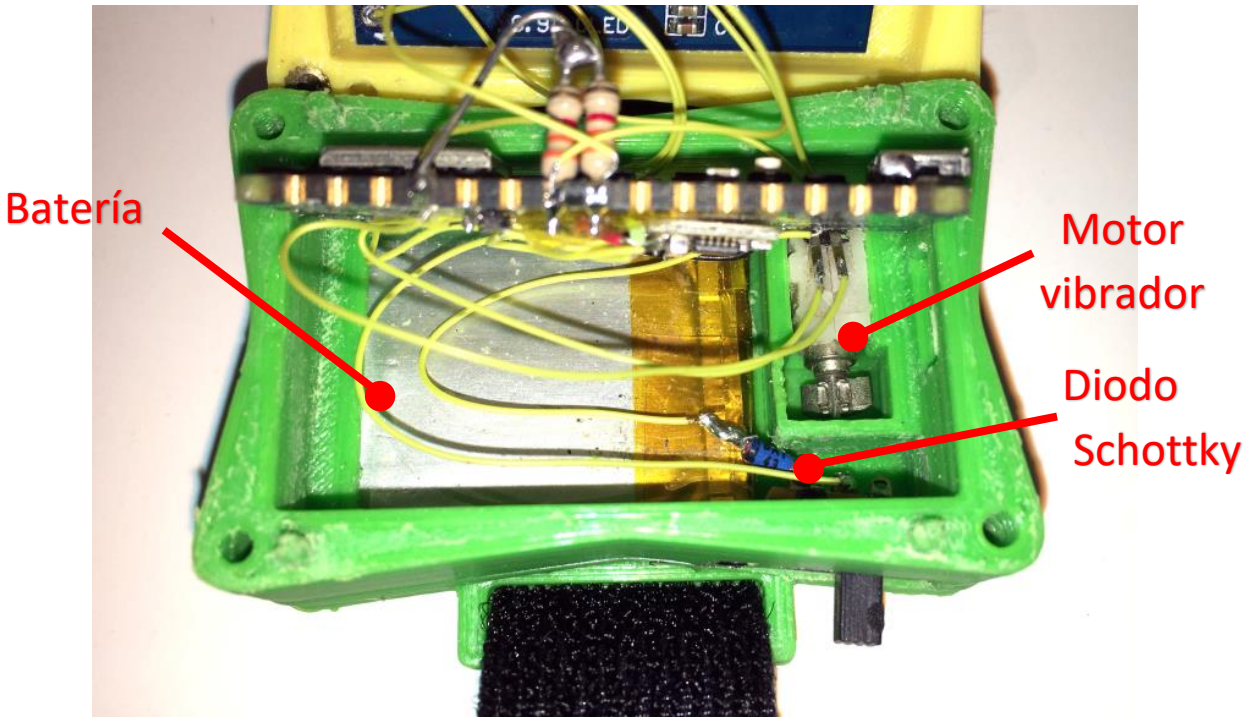


Figura 40. Montaje (2)

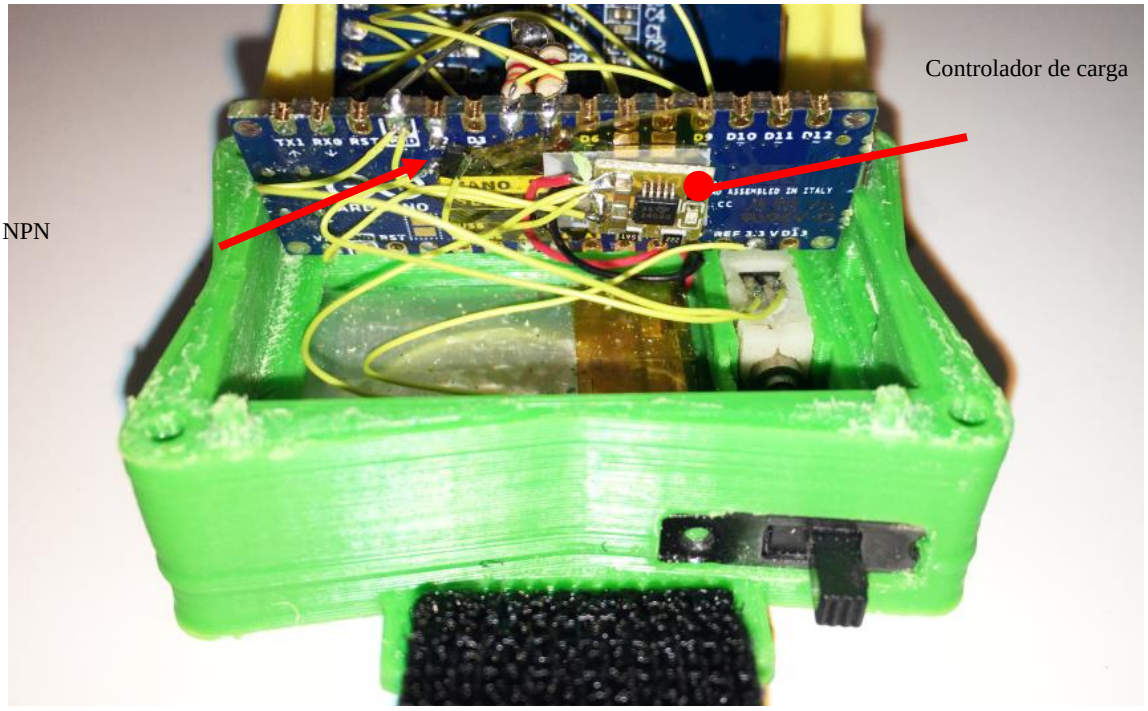


Figura 41. Montaje (3)

3. PROGRAMACIÓN

El desarrollo de la parte lógica se ha realizado en 2 entornos distintos. Uno de ellos es para la programación del microcontrolador (Arduino) que recibirá la información de la IMU, la procesará y la enviará al teléfono móvil, y la otra es la programación en el teléfono móvil, que, tras recibir la información del microcontrolador, la representará en forma de gráficas.

3.1. Programación del microcontrolador (Arduino).

Todo el desarrollo del programa para el microcontrolador ubicado en la placa Arduino Nano 33 BLE se ha escrito en C++ bajo el entorno de programación *"Visual Studio Code"* empleando la extensión *"PlatformIO"* (Figuras 42 y 43). Esta plataforma permite la programación de multitud de microcontroladores, entre el que se encuentra el nRF52840 de la placa Arduino Nano 33 BLE.

Figura 42. Herramientas para programación Arduino

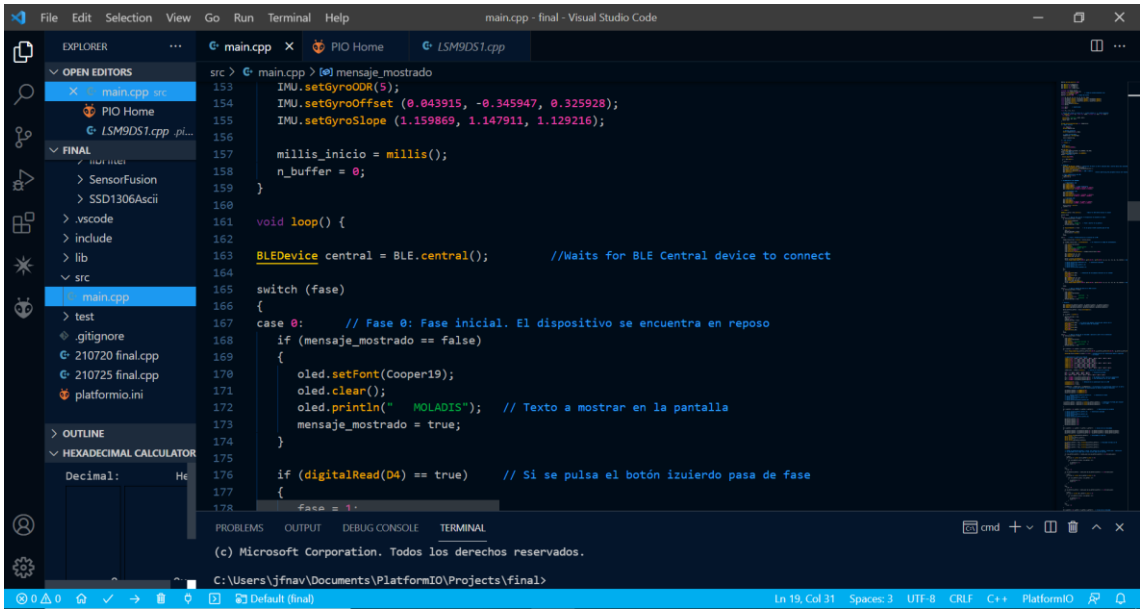


Figura 43. Entorno de programación en Visual Studio Code

3.1.1. Estructura del programa.

La elaboración del programa que recibe, procesa y envía la información procedente de la IMU al teléfono móvil se divide en 2 bloques. Uno es el dedicado a la recepción y procesamiento de la información recibida de la IMU, y el otro es el envío de la información ya procesada a través de Bluetooth hacia el teléfono móvil.

Antes de entrar en detalle con estos 2 bloques, hay un elemento que es común a ambos y que es importante resaltar:

- **Pantalla OLED.** La pantalla emplea el chip SSD1306. Para su uso se ha empleado la librería para Arduino "*SSD1306Ascii.h*". Gracias a esta librería se escribirá en cada fase del programa la información relacionada con la misma, modificando el tamaño, ubicación y tipo de fuente del texto.

3.1.2. Recepción y procesamiento de la información de la IMU.

La IMU empleada en este trabajo es la que hay instalada en la placa Arduino Nano 33 BLE. En concreto se trata del chip LSM9DS1 del fabricante ST. Su principal característica es que dispone de 9 grados de libertad y está conectada al microcontrolador mediante el protocolo I2C.

Para facilitar su uso se ha empleado la librería "*Arduino_LSM9DS1.h*" que incorpora todas las funcionalidades de este chip.

a) **Configuración de sensores**. Antes de poder emplear esta IMU es necesario establecer algunos de los parámetros que permitirán el correcto funcionamiento de los sensores.

b) **Parámetros calibración**. Se indican los valores de offset y pendiente para que cada uno de los 3 sensores quede correctamente calibrado. El proceso de calibración está explicado en el Anexo IV.

c) **Fondo de escala.** Es necesario establecer cuál será la magnitud máxima que podrá leer cada uno de los sensores. Este valor también conlleva un aumento o disminución de la resolución de cada sensor. Para cada sensor se ha determinado el fondo de escala de manera empírica reflejados en la Tabla 1.

SENSOR	FONDO DE ESCALA	RESOLUCIÓN
Acelerómetro	± 8 g	0.244 mg / LSB
Giroscopio	± 2000 dps	70 mdps / LSB
Magnetómetro	± 8 gauss	0.29 mgauss / LSB

Tabla 1. Fondo de escala de sensores

d) **Output data rate (ODR).** Con este parámetro se estable el valor de frecuencia de refresco de cada sensor. Dada la aceleración a la que se somete la IMU y los cambios rápidos de dirección que puede sufrir, es necesario que los sensores tengan una frecuencia de refresco lo más alta posible. Los valores establecidos se muestran en la Tabla 2.

SENSOR	ODR
Acelerómetro	476 Hz
Giroscopio	476 Hz
Magnetómetro	400 Hz

Tabla 2. Frecuencia de refresco de sensores

3.1.3. Estructura de control

El proceso de captura y procesado de la información se controla mediante una estructura de control *"switch...case"*, quedando el programa dividido en 4 fases.

El objetivo de dividir el programa en varias fases es, además del orden y claridad de éste, permitir el mayor tiempo de muestreo posible en la fase de captura de datos del IMU, consiguiendo una mayor calidad en el resultado final. Es importante que en la fase de captura el programa solo se dedique a leer información de la IMU y no gaste tiempo en tareas innecesarias.

a) **Fase I. Modo espera.** Cuando el dispositivo se pone en marcha mediante el interruptor de encendido, éste se queda en modo de espera. La única manera de salir de esta fase y pasar a la siguiente es pulsando el botón de la izquierda.

Durante esta fase en la pantalla se muestra el mensaje "MOLADIS" (Monitorización del Lanzamiento de Disco) (Figura 44).

Figura 44. Fase de espera

b) **Fase II. Acondicionamiento**. Tras pulsar el botón izquierdo aparece en pantalla una cuenta atrás. Es necesario esperar un tiempo para que los cálculos realizados por el algoritmo de fusión se estabilicen (Figura 45). Se ha establecido un valor de 20 segundos para asegurar que los valores calculados sean estables.

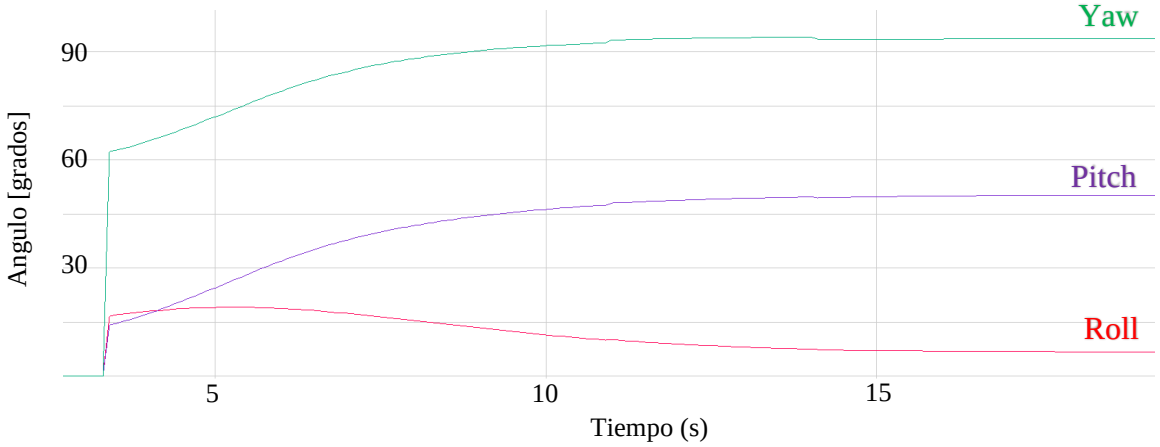


Figura 45. Estabilización de las mediciones

Para representar de manera gráfica la estabilización de los cálculos se han empleado los parámetros *Yaw, Pitch, Roll.* Aunque no se emplean en los cálculos posteriores, ya se trabaja directamente con cuaterniones, son válidos para representar el tiempo necesario por el algoritmo de fusión para su estabilización.

Es conveniente evitar que el programa quede detenido mientras se realiza la espera, por lo que se ha evitado el uso de la instrucción *"delay"*. En su lugar se ha empleado la forma mostrada en la Figura 46.

```
t_calentamiento = 20000;
millis_inicio = millis();
---

if ((millis() - millis_inicio) < t_acondicionamiento)
{
        ---
}
```

Figura 46. Alternativa al delay

De esta manera se puede realizar la temporización sin que el programa tenga que detener su flujo normal de trabajo.

Una vez concluida la temporización, y justo antes de que el programa pase a la siguiente fase, se acciona brevemente el motor vibrador durante 500 ms para indicar que la fase de captura va a comenzar.

c) Fase III. Captura. El proceso de captura se basa en la recogida de una serie de datos procedentes de la IMU. En este caso se realiza una captura de 3000 valores instantáneos, que son almacenados en varios arrays:

- 3 para la aceleración (lectura acelerómetro en ejes x, y, z).
- 3 para el giroscopio (lectura giroscopio en ejes x, y, z).
- 3 para el magnetómetro (lectura magnetómetro en ejes x, y, z).
- 1 para el tiempo entre captura y captura.

Estas 3000 mediciones, teniendo en cuenta que se realiza a intervalos de aproximadamente 2ms, permiten la captura del movimiento durante aproximadamente 6 segundos. Este tiempo es suficiente para monitorizar el movimiento del lanzador de disco.

Al estar trabajando sobre un microcontrolador es necesario tener en cuenta cuáles son sus limitaciones. Una de ellas es la cantidad de memoria RAM de la que dispone. El nRF52840 dispone de una memoria SRAM de 256KB.

Teniendo en cuenta que el tipo de variable empleado para almacenar los datos es *float*, lo que supone reservar un espacio de 4 bytes, el espacio total empleado por los arrays anteriores es de:

$$3 * (3000 * 4 + 3000 * 4 + 3000 * 4) + (3000 * 4) = 120\ KB$$

A pesar de que aún queda suficiente espacio hasta llegar a los 256 KB, hay que tener en cuenta que se emplearán arrays auxiliares para los cálculos de la IMU, empleo de RAM en el resto de los procesos del microcontrolador, y, sobre todo, la carga de las librerías que permitirán la comunicación a través de Bluetooth LE.

Cuando se han recogido las 3000 mediciones, se avanza a la fase de procesado, pero antes se acciona el motor vibrador con una corta secuencia (200 ms ON – 200 ms OFF – 200 ms ON) para indicar que la fase de captura ha finalizado.

d) Fase IV. Procesado. Esta es la fase donde se realizan todos los cálculos necesarios para transformar las mediciones de la IMU obtenidas en la fase de captura (acelerómetro, giroscopio, magnetómetro) en velocidad y posición. Se realiza una operación de procesado por cada una de las mediciones realizadas, es decir, 3000.

Se divide en varias etapas:

I. Aplicación del algoritmo de fusión. Mediante el algoritmo de fusión, en este caso Madgwick, todas las mediciones recogidas en un instante dado se emplean para mejorar la calidad de la información de salida. En la Figura 47 se muestra un esquema de su funcionamiento.

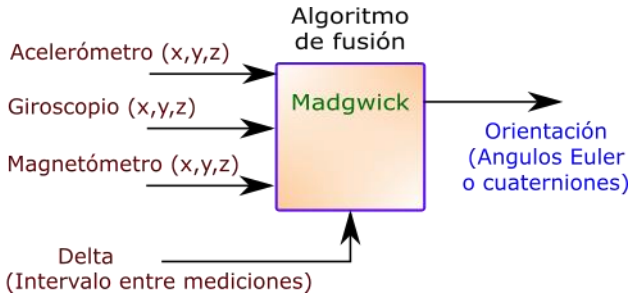


Figura 47. Algoritmo de fusión

Cabe destacar, que, para el correcto funcionamiento de este algoritmo de fusión, se ha de seguir la convención NED (North, East, Down), es decir, los ejes "x, y, z" de los 3 sensores han de estar alineados y los ejes del acelerómetro se deben situar de la siguiente manera:

- Eje x → Norte.
- Eje y → Este.
- Eje z → Hacia abajo.

En la Figura 48, donde se refleja la disposición de los ejes de los sensores de la IMU LSM9DS1, se observa que es necesario realizar algún ajuste para adecuarlo a la convención NED.

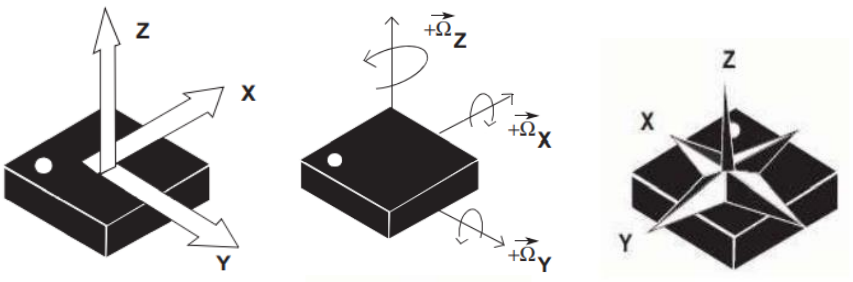


Figura 48. Disposición sensores IMU. Fuente: datasheet LSM9DS1

Esta adecuación se realiza modificando el signo de los componentes de los sensores en la llamada a la función del algoritmo de fusión (Figura 49):

```
fusion.MadgwickUpdate(giro_x, giro_y, -giro_z, acel_x, acel_y, -acel_z, -mag_x, mag_y, -mag_z, delta);
```

Figura 49. Adecuación sensores para NED

II. **Empleo de cuaterniones**. Con el fin de evitar el problema del *gimbal lock* en el dispositivo, se trabajará con cuaterniones en lugar de hacerlo con ángulos de Euler para orientar la IMU (roll, pitch, yaw). El empleo de cuaterniones evita que se utilice trigonometría en los cálculos, lo cual es una ventaja para reducir los tiempos de cálculo.

III. **Eliminación de la gravedad**. Un acelerómetro está sujeto a aceleraciones dinámicas (a las que sometemos al dispositivo de manera voluntaria) y las estáticas (fuerza de la gravedad). Es evidente que para conseguir unas mediciones correctas es necesario eliminar la aceleración estática. Con el cuaternión se calcula la dirección esperada de la gravedad y luego se resta de las lecturas del acelerómetro (Figura 50). El resultado es la aceleración dinámica [28].

```
g[0] = 2 * (q[1] * q[3] - q[0] * q[2]);
g[1] = 2 * (q[0] * q[1] + q[2] * q[3]);
g[2] = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3];

aT_x = 9.80665 * (ax - g[0]);
aT_y = 9.80665 * (ay - g[1]);
aT_z = 9.80665 * (az + g[2]);
```

Figura 50. Eliminación de la gravedad mediante cuaterniones

IV. **Matriz de rotación**. Si la IMU se desplazara con sus ejes paralelos a los ejes de nuestro sistema de referencia, los cálculos para determinar su posición serían bastante sencillos. Pero eso nunca va a ocurrir, porque la IMU siempre tendrá alguno de sus ejes (sistema de referencia de la IMU) rotado con respecto al nuestro (sistema de referencia de la Tierra). En la Figura 51 se muestra un ejemplo de los dos sistemas de referencia.

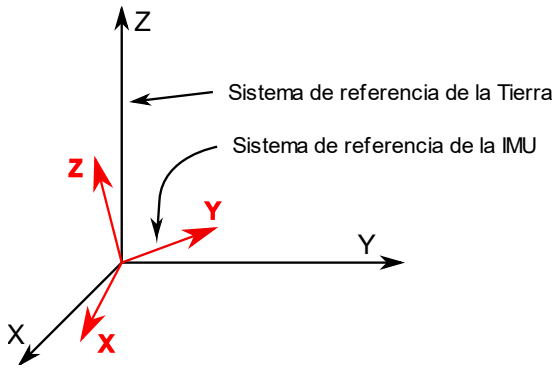


Figura 51. Sistemas de referencia

Para llevar el sistema de referencia de la IMU al nuestro, es necesario rotarlo. Para ello se emplea la matriz de rotación sobre los cuaterniones.

La implementación de la matriz de rotación mediante cuaterniones se implementa de acuerdo con lo mostrado en la Figura 52:

```
// Generación de la matriz de rotación
rotMatrix(0,0) = q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3];
rotMatrix(0,1) = 2 * (q[1] * q[2] + q[0] * q[3]);
rotMatrix(0,2) = 2 * (q[1] * q[3] - q[0] * q[2]);
rotMatrix(1,0) = 2 * (q[1] * q[2] - q[0] * q[3]);
rotMatrix(1,1) = q[0] * q[0] - q[1] * q[1] + q[2] * q[2] - q[3] * q[3];
rotMatrix(1,2) = 2 * (q[2] * q[3] + q[0] * q[1]);
rotMatrix(2,0) = 2 * (q[1] * q[3] + q[0] * q[2]);
rotMatrix(2,1) = 2 * (q[2] * q[3] - q[0] * q[1]);
rotMatrix(2,2) = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3];
```

Figura 52. Implementación de la matriz de rotación con cuaterniones

$$R(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

El valor de las aceleraciones llevadas al sistema de referencia de la Tierra se calcula multiplicando la inversa de la matriz de rotación generada con los cuaterniones por las aceleraciones a las que está sometido el dispositivo, ya que la aceleración gravitacional eliminada.

V. Filtrado de la aceleración. La señal de la aceleración obtenida por la IMU es una señal con bastante ruido y algún *glitch* que puede generar errores en los cálculos posteriores. Para obtener una señal más suavizada se aplica un filtro de reducción de ruido del tipo EWMA (Exponentially Weighted Moving Average) [29]. Este filtro tiene un bajo coste computacional y se basa en tomar N muestras, sumarlas y dividirlas por N. El promedio es "móvil" cuando se recalcula cada vez que se obtiene una nueva muestra tomando las N muestras anteriores. Este filtrado se emplea usando la librería *"ewma.h"* para Arduino [30]. En las Figuras 53 y 55 se muestra la señal en bruto y en las Figuras 54 y 56 filtrada.

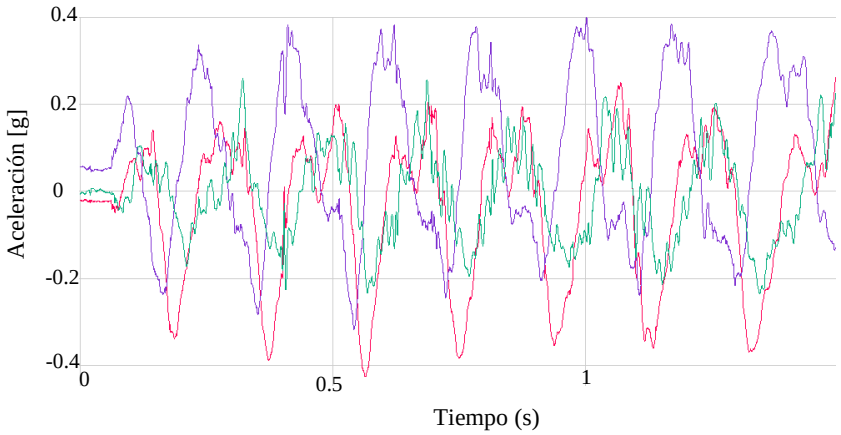


Figura 53. Aceleraciones sin suavizar.

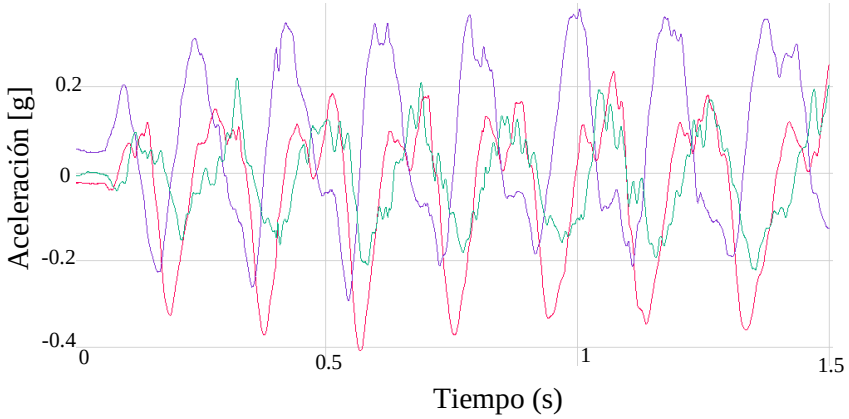


Figura 54. Aceleraciones suavizadas

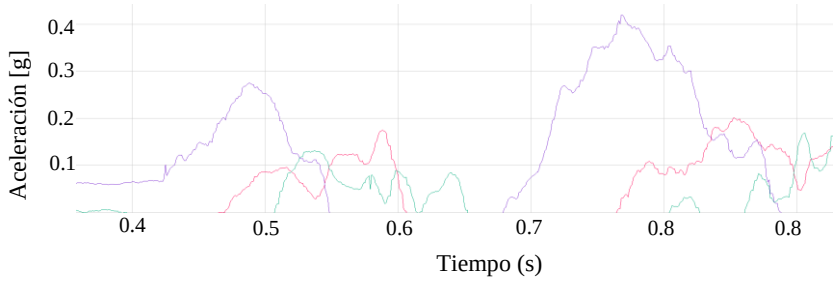


Figura 55. Detalle de una aceleración sin suavizar

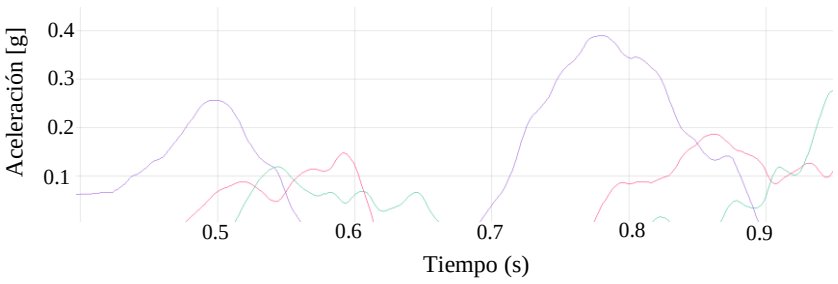


Figura 56. Detalle de una aceleración suavizada

VI. **Cálculo de la velocidad.** Empleando las aceleraciones filtradas de la fase anterior se calcula la velocidad instantánea en cada punto. Como se explica en el apartado 1.2.1, la velocidad se calcula integrando la aceleración.

El cálculo integral se realiza de manera que se divide el área bajo la curva de la aceleración en pequeños rectángulos y se suma el área de todos ellos (Figura 57). El resultado de esta operación es la velocidad. Uno de los lados del rectángulo es la aceleración en ese instante y el otro es el intervalo entre medidas.

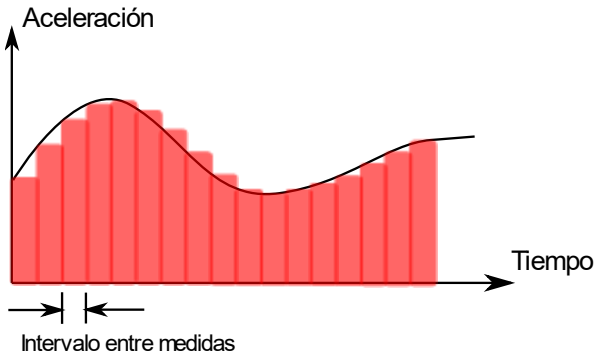


Figura 57. Integración de la aceleración

Esta operación para calcular la velocidad se realiza para cada uno de los ejes (x, y, z)

VII. **Filtrado DC.** El cálculo de la velocidad a partir de la integración de la aceleración conlleva un problema. Dado que el resultado final se calcula como la suma acumulativa de áreas parciales, si en cualquiera de estas áreas hay un error, provocado por ejemplo por ruido en la señal, el error afectará al resultado final. Si el ruido origina muchos errores en diversas áreas, el resultado diferirá bastante del esperado.

Este hecho se aprecia fácilmente cuando se integra la aceleración de un móvil para calcular su velocidad. Si se monitoriza un movimiento del tipo Paro-Marcha-Paro, es posible que la velocidad final calculada no sea exactamente 0 (aunque el móvil realmente esté parado). Si no se corrigen o minimizan estos errores, en los posteriores cálculos de las posiciones, los resultados serían bastantes incoherentes.

Una forma de reducir este problema es eliminar la componente continua de la señal de la velocidad obtenida.

La eliminación de la componente continua se consigue aplicando un filtro paso alto a la señal de la velocidad. [16] El valor de la frecuencia de corte ha sido determinado de forma empírica en virtud de los resultados obtenidos en el cálculo del desplazamiento del dispositivo. Es conveniente que la frecuencia de corte no sea muy alta, ya que esto provoca la eliminación total de la componente continua, anulando cualquier posible representación del desplazamiento del dispositivo. Una frecuencia de corte baja provoca que el filtro apenas tenga efecto en la señal de salida.

Para la aplicación del filtro paso alto se ha empleado la librería *"filters.h"* para Arduino [31].

En las gráficas mostradas en las Figura 58, 59, 60, 61, 62 y 63 se muestra la efectividad de este filtro. Se muestra la velocidad en cada eje, tanto sin filtrar como filtrada. La captura de datos se ha realizado mientras el dispositivo se movía en círculos.

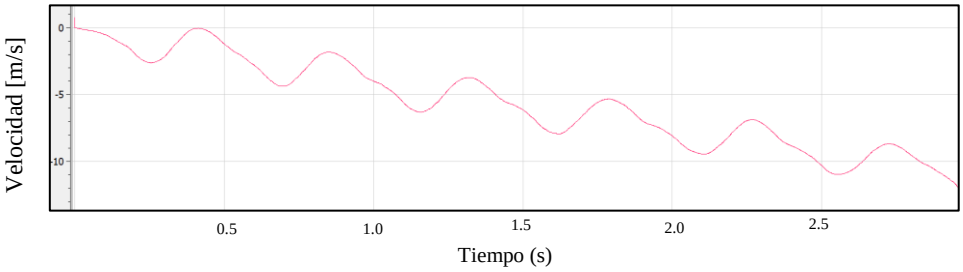


Figura 58. Velocidad en eje X sin filtrar

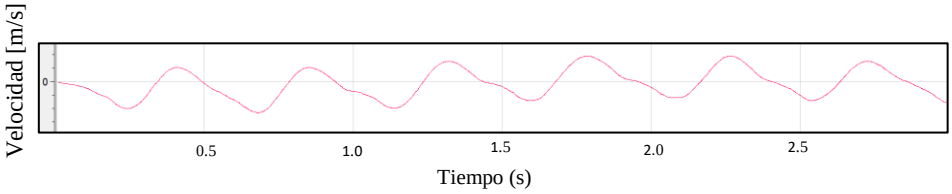


Figura 59. Velocidad en eje X filtrada

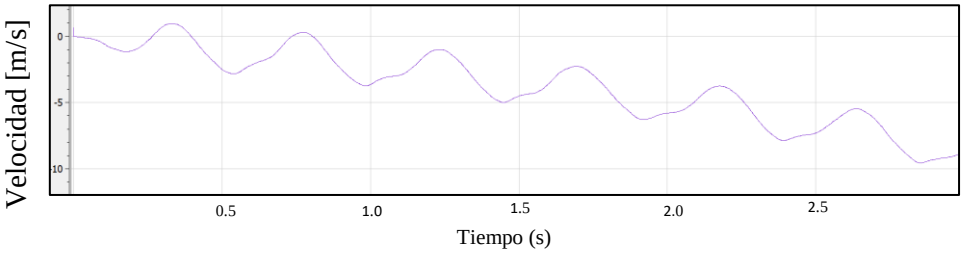


Figura 60. Velocidad en eje Y sin filtrar

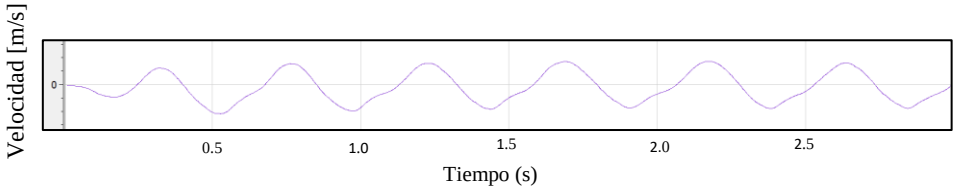


Figura 61. Velocidad en eje Y filtrada

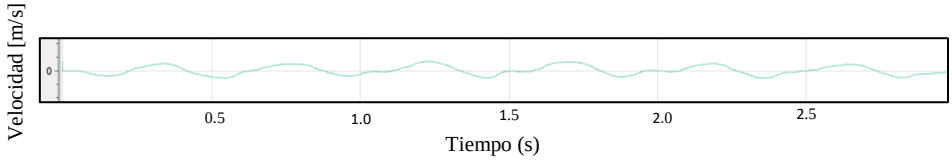


Figura 62. Velocidad en eje Z sin filtrar

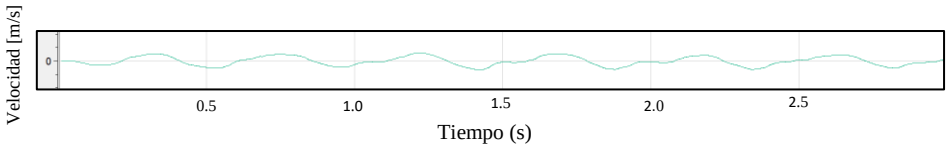


Figura 63. Velocidad en eje Z filtrada

VIII. **Aproximación a 0 de la velocidad**. La aceleración leída por la IMU, aunque éste se encuentre totalmente detenido, nunca será 0.0, sino que tiene un pequeño valor. Este valor, por pequeño que sea, termina afectando a la velocidad calculada.

Con el fin de que la velocidad calculada sea 0.0 cuando el dispositivo se encuentre prácticamente detenido, se aplica un pequeño ajuste. Si la aceleración medida se encuentra dentro de una ventana establecida durante un número de muestras consecutivas también establecido, la velocidad durante todo ese periodo valdrá 0.0 (Figura 64).

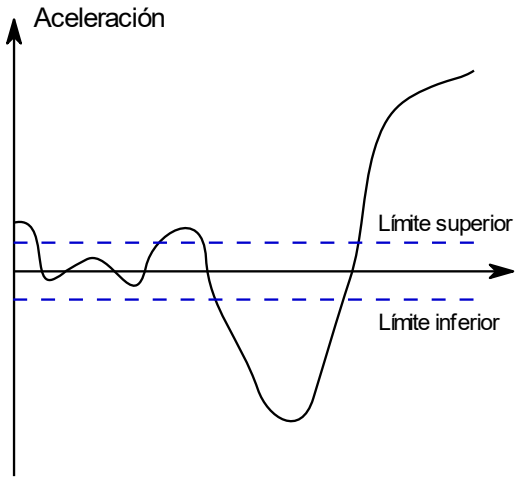


Figura 64. Límites de la aceleración

En el caso de este dispositivo, los límites de la ventana son +1.0 y -1.0, y el número de muestras es de 60. En las Figuras 65 y 66 se muestra el funcionamiento de este algoritmo.

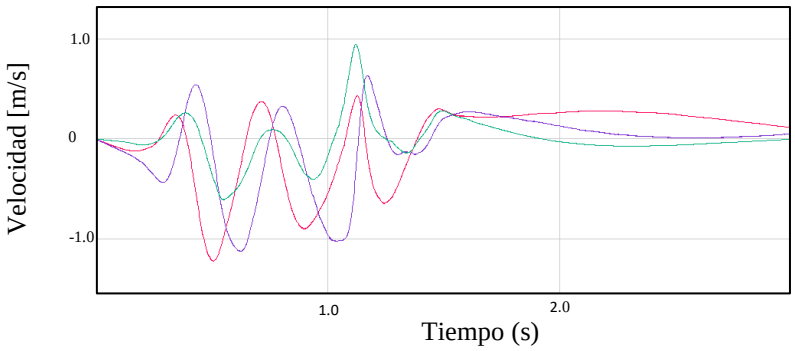


Figura 65. Velocidad sin ajuste

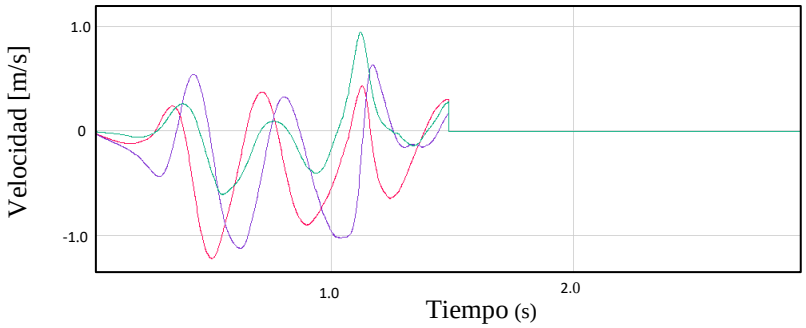


Figura 66. Velocidad ajustada

IX. **Cálculo del módulo de la velocidad**. Uno de los objetivos de este trabajo es el cálculo de la velocidad con la que se mueve la mano del lanzador de disco. En el paso previo se ha calculado la velocidad de cada uno de los componentes, así que el valor del módulo de la velocidad para cada instante es:

$$Velocidad_{(t)} = \sqrt{Vx_{(t)}{}^2 + Vy_{(t)}{}^2 + Vz_{(t)}{}^2}$$

X. **Cálculo de las posiciones**. Al igual que se hizo para el cálculo de las velocidades a partir de las aceleraciones, las posiciones se calculan mediante la integración de la velocidad. Este proceso se realiza en cada instante y para cada eje.

A diferencia del cálculo de la velocidad, los resultados obtenidos para la posición no deben ser manipulados, ya que afectaría de manera directa al resultado. Si por ejemplo se realizase un filtrado para eliminar la componente continua, es posible que estemos eliminando un posible desplazamiento que sí se estaba realizando.

e) **Fase V. A la espera de conexión Bluetooth**. La siguiente fase tras el proceso de cálculo del desplazamiento es el envío mediante Bluetooth. Pero antes de realizar el envío es necesario configurar la comunicación.

3.1.4. Configuración BLE

a) Esperando la conexión de la central. El programa quedará detenido en este punto hasta que el teléfono móvil no establezca comunicación con el dispositivo.

b) Empaquetado de datos. Una de las limitaciones a tener en cuenta con el Bluetooth Low Energy es que, a diferencia de la transmisión serie del Bluetooth convencional, no se puede enviar una trama grande de caracteres. El BLE solo permite la transmisión de 20 bytes en cada trama [18].

Con el fin de aprovechar al máximo cada trama y que la transmisión de toda la información dure el menor tiempo posible, se ha realizado el empaquetado mostrado en la Tabla 3.

Dato	Posición X					Posición Y					Posición Z					Velocidad				
Bytes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Formato	x	x	.	x	x	x	x	.	x	x	x	x	.	x	x	x	x	.	x	x

Tabla 3. Empaquetado de datos

La trama se prepara concatenando los valores de las posiciones en los 3 ejes y la velocidad para cada una de las mediciones. Una vez que se recibe cada trama en el teléfono móvil se decodificará siguiendo el esquema de empaquetado anterior.

c) Envío de datos al teléfono móvil. Las tramas son enviadas al teléfono móvil según se van generando. El total de tramas enviadas corresponde con el número de muestras tomadas por la IMU, en este caso 3000.

- **Diagrama de funcionamiento del programa**

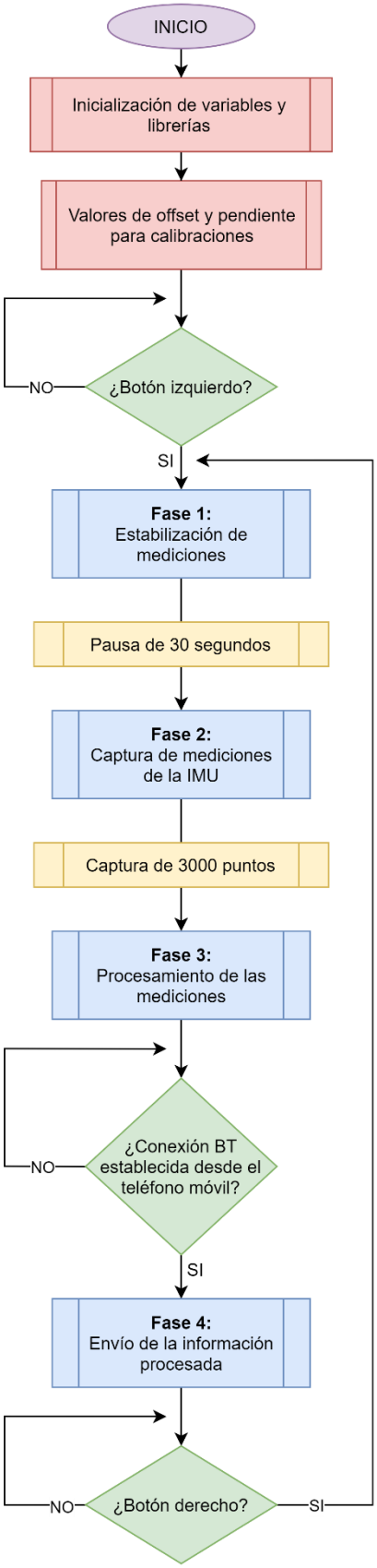


Figura 67. Diagrama del funcionamiento del programa

3.2. Programación en el teléfono móvil.

El programa encargado de recibir la información enviada por el dispositivo vía Bluetooth LE y representar de manera gráfica el desplazamiento y velocidad de la mano del lanzador de disco, ha sido escrito usando el lenguaje de programación Python.

Para los dispositivos iOS existe un entorno de programación llamado *Pythonista* (Figura 68), que permite la edición y ejecución de cualquier programa escrito en Python. En concreto, este programa ha sido escrito y ejecutado en un iPhone 6S (Figura 69).

Figura 68. Python y Pythonista logos

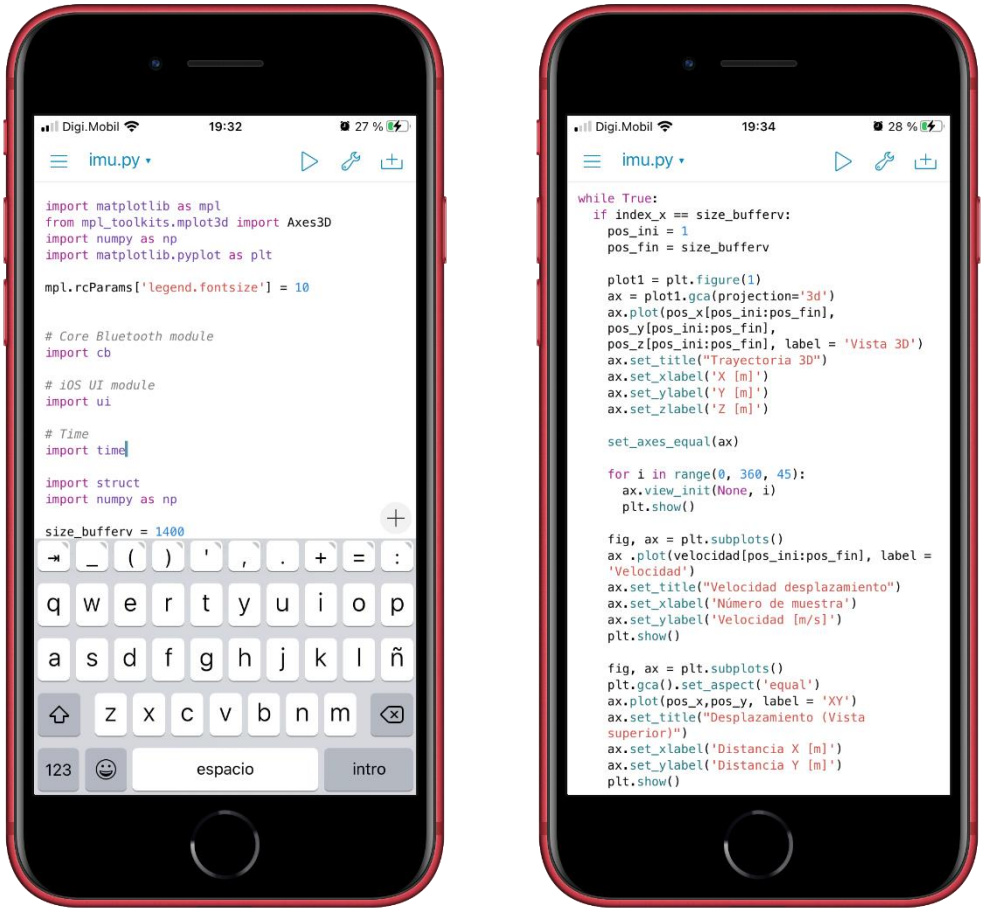


Figura 69. Entorno de programación en Pythonista

El programa se divide en 2 bloques:

- Recopilación de datos
- Representación gráfica

3.2.1. Recopilación de datos.

Todas las funciones necesarias para la recepción de datos mediante Bluetooth Low Energy se obtienen gracias al módulo *"cb"* (Core Bluetooth Module) cargado al inicio del programa [33].

El teléfono móvil hará de central en la transmisión BLE. Por este motivo ha de hacer un escaneo de los dispositivos visibles (periféricos) que están a su alcance y emparejarse con uno en concreto, nuestro dispositivo, que lo conocerá mediante el nombre que le fue asignado, *"Arduino"*. Posteriormente, la central ha de buscar el servicio con el UUID asignado en nuestro dispositivo y una vez encontrado, buscará la característica cuyo UUID también fue establecido en el dispositivo [32].

Cada vez que se recibe un paquete de datos desde el dispositivo, se está recibiendo la trama de datos de 20 bytes que formamos previamente. Mediante un simple acotado de ésta, se separan y almacenan en arrays independientes cada una de las 4 variables recibidas. El código que realiza esta operación se muestra en la Figura 70.

```
# La trama recibida de 20 bytes se divide en las variables de posición y velocidad
    pos_x[index_x] = c.value[:5]            # Parseo de la posición en X
    float(pos_x[index_x])                   # Se convierte a float la cadena recibida
    pos_y[index_x]= c.value[5:10]           # Parseo de la posición en X
    float(pos_y[index_x])                   # Se convierte a float la cadena recibida
    pos_z[index_x] = c.value[10:15]         # Parseo de la posición en X
    float(pos_z[index_x])                   # Se convierte a float la cadena recibida
    velocidad[index_x] = c.value[15:20]     # Parseo de la posición en X
    float(velocidad[index_x])               # Se convierte a float la cadena recibida
```

Figura 70. Descomposición de la trama de 20 bytes

3.2.2. Representación gráfica

Para la representación gráfica se ha empleado el módulo denominado *"matplotlib"* [34]. Este módulo permite la representación de valores almacenados en arrays tanto en 2D como en 3D.

- **Representación de la trayectoria**. Para representar la trayectoria seguida por la mano del lanzador se utiliza una gráfica en 3D. El programa muestra la misma gráfica desde distintos puntos de vista, con el fin de poder seleccionar y guardar la que ofrezca una mejor visión de la trayectoria. El comienzo del movimiento está en el punto (0,0,0). En la Figura 71 se muestran una serie de representaciones del mismo lanzamiento.

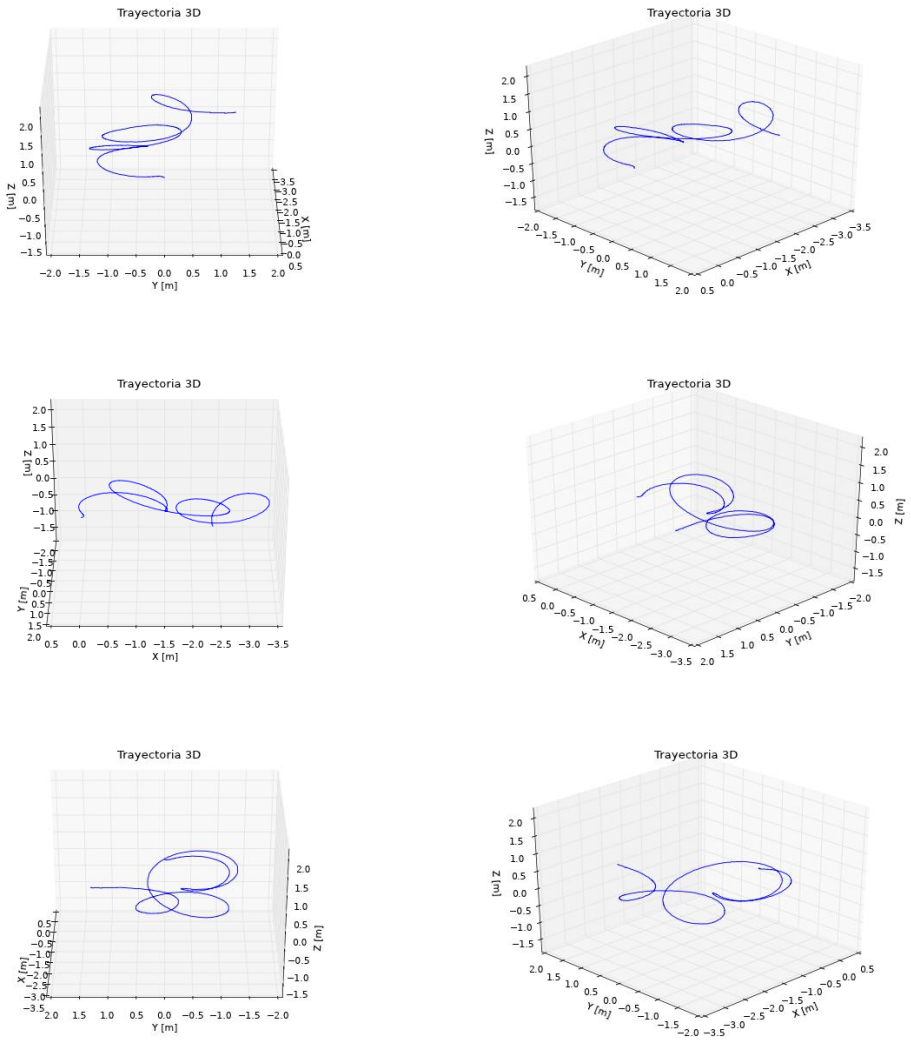


Figura 71. Diversos ángulos de la representación de la trayectoria.

También se representa la trayectoria del movimiento en 2 dimensiones, para que sea más fácil su visualización. En concreto se visualiza en una vista superior (plano XY) (Figura 72) y en una vista lateral (plano YZ) (Figura 73).

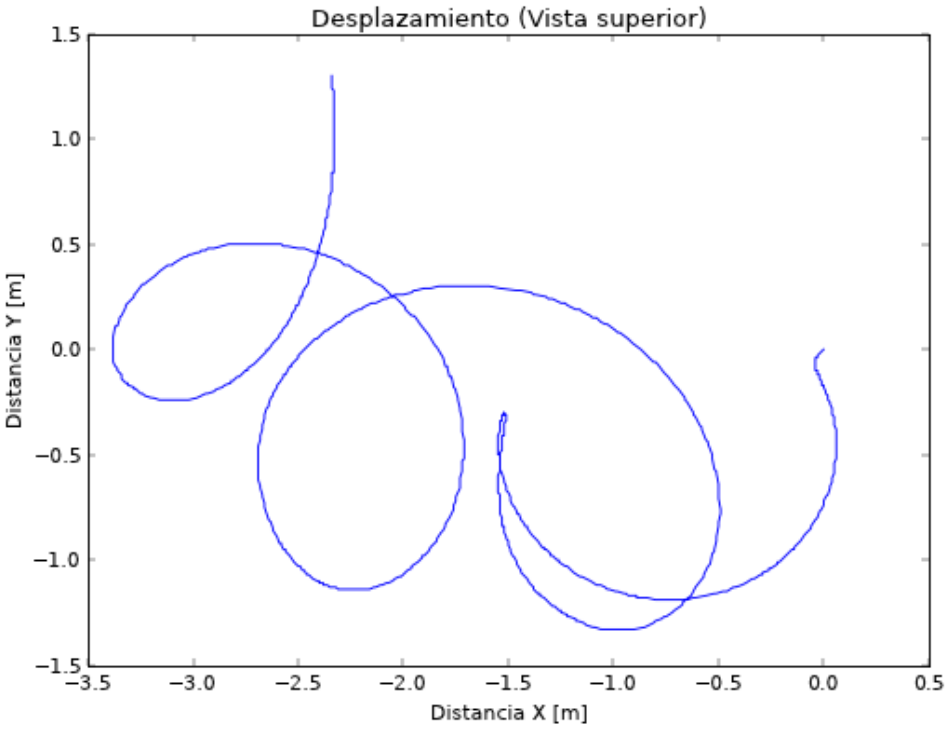


Figura 72. Vista superior del movimiento. Plano XY

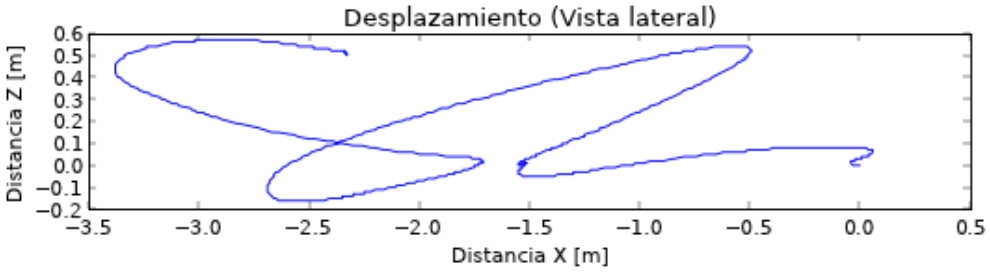


Figura 73. Vista lateral del movimiento. Plano YZ.

- **Representación de la velocidad.** Para representar la velocidad se utiliza una gráfica en 2D (velocidad – nº de muestra). La velocidad está expresada en m/s. (Figura 74)

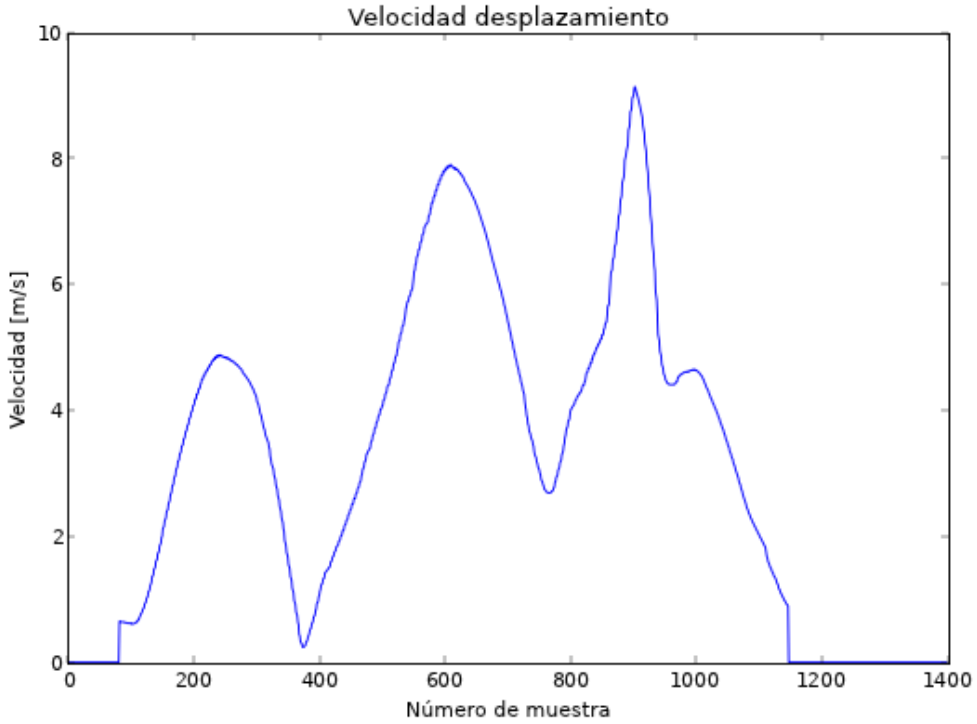


Figura 74. Velocidad de la mano durante el lanzamiento.

4. PRINCIPIO DE FUNCIONAMIENTO

El objetivo final de este dispositivo es la representación gráfica en un teléfono móvil del recorrido realizado por la mano de un lanzador de disco, así como su velocidad. El proceso, de principio a fin, se divide en varias fases:

- **Colocación del dispositivo**. El lanzador de disco se ha de sujetar con firmeza el dispositivo a la muñeca empleando la cinta de velcro que incorpora (Figuras 75, 76). Si no queda bien sujeto se pueden originar vibraciones que originen lecturas incorrectas en la IMU.

Figura 75. Dispositivo en la muñeca

Figura 76 Dispositivo en la muñeca II

- **Encendido**. Para poner en marcha el dispositivo es necesario mover el interruptor de deslizamiento hacia la derecha. Al encenderlo aparece en la pantalla el texto MOLADIS (Monitorización de Lanzamiento de Disco) (Figura 77).

Figura 77. Pantalla de inicio

- **Inicio monitorización.** Para que la IMU comience la captura de datos es necesario pulsar el botón de la izquierda (Figura 78).

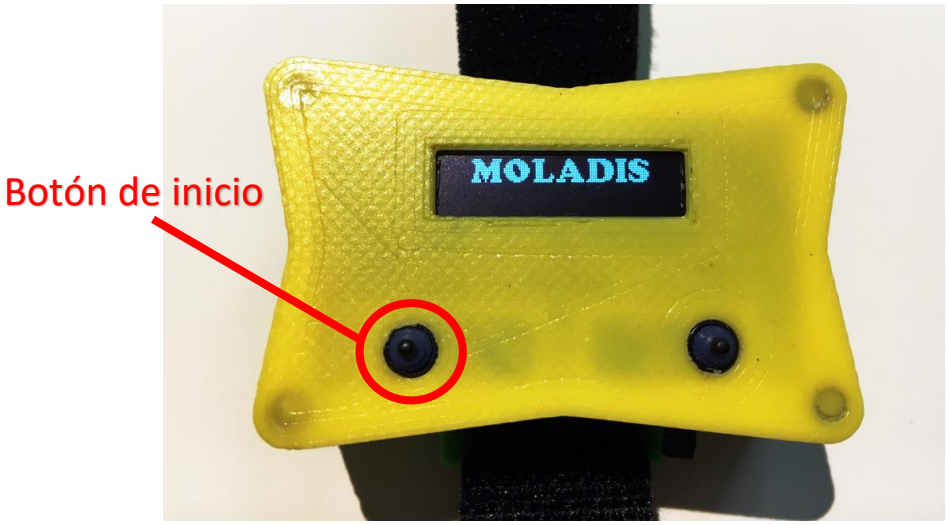


Figura 78. Botón de inicio.

En la pantalla aparece el texto "*Estabilización*" junto a una cuenta atrás (Figura 79). Este es el tiempo de estabilización de los cálculos realizados por el algoritmo de fusión, y también le sirve al lanzador para que se vaya preparando.

Figura 79. Estabilización de las medidas

Una vez terminada la cuenta atrás, el dispositivo realizará una pequeña vibración para indicar que comienza la adquisición de datos por la IMU, además de aparecer en la pantalla el texto *"Capturando movimiento"* (Figura 80).

Figura 80. Capturando movimiento

En este momento, el lanzador de disco debe comenzar a realizar el movimiento para lanzar el disco.

El proceso de captura de datos tiene una duración de 6 segundos y en ese transcurso recoge 3000 mediciones. Cuando termina este proceso, el dispositivo realiza otra vibración con un patrón distinto al anterior para indicar que ha finalizado el proceso de captura de datos.

Una vez finalizada la captura, el microcontrolador procesa toda la información capturada. Mientras dure este proceso en la pantalla aparece el texto *"Calculando velocidad y posiciones"* (Figura 81).

Figura 81. Realizando cálculos

La fase siguiente al procesado es el envío de datos desde el Arduino al teléfono móvil mediante Bluetooth. Esta fase de envío es indicada en la pantalla con el texto *"Esperando conexión Bluetooth"* (Figura 82). Cuando aparece este mensaje es el momento de ejecutar el programa de recepción y visualización de datos en el teléfono móvil. Mientras se esté enviando información se muestra el mensaje *"Enviando datos"* (Figura 83).

Figura 82. Esperando conexión Bluetooth

Figura 83. Enviando datos

En el teléfono móvil es necesario ejecutar el programa creado para recibir la información. Para ello se pulsa el botón ejecutar dentro del programa Pythonista (Figura 84).

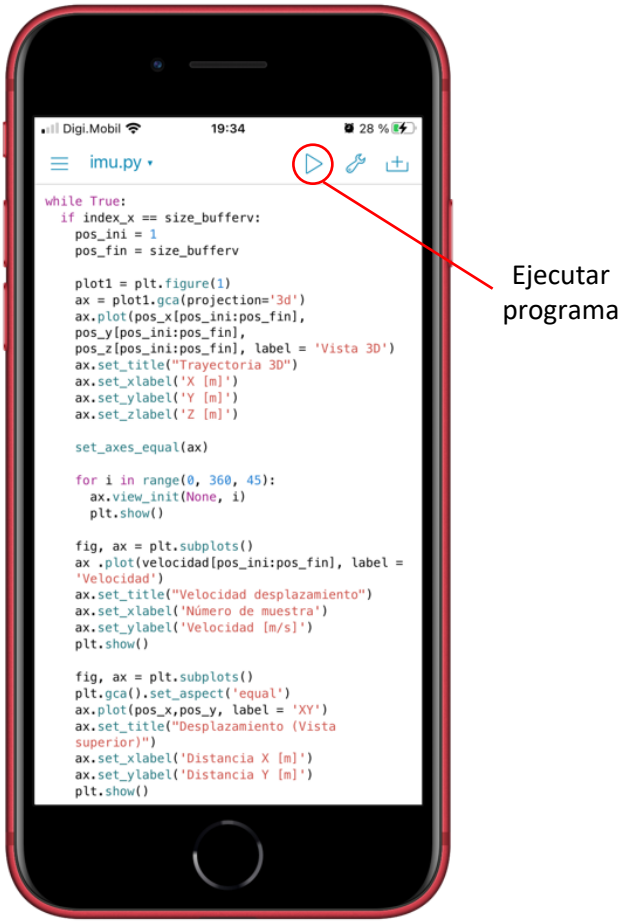


Figura 84. Ejecutar programa.

El programa se ejecuta en modo consola. Si la conexión con el dispositivo se realiza de manera correcta, primero se conectará a éste, después buscará un servicio y posteriormente una característica del mismo (Figura 85). Si encuentra la característica de manera correcta, el dispositivo comienza la transmisión de los datos que son recibidos por el teléfono móvil.

Figura 85. Conexión con el dispositivo.

Cuando la información es recibida e interpretada por el programa, van apareciendo las gráficas generadas, según se muestra en las Figuras 86 y 87.

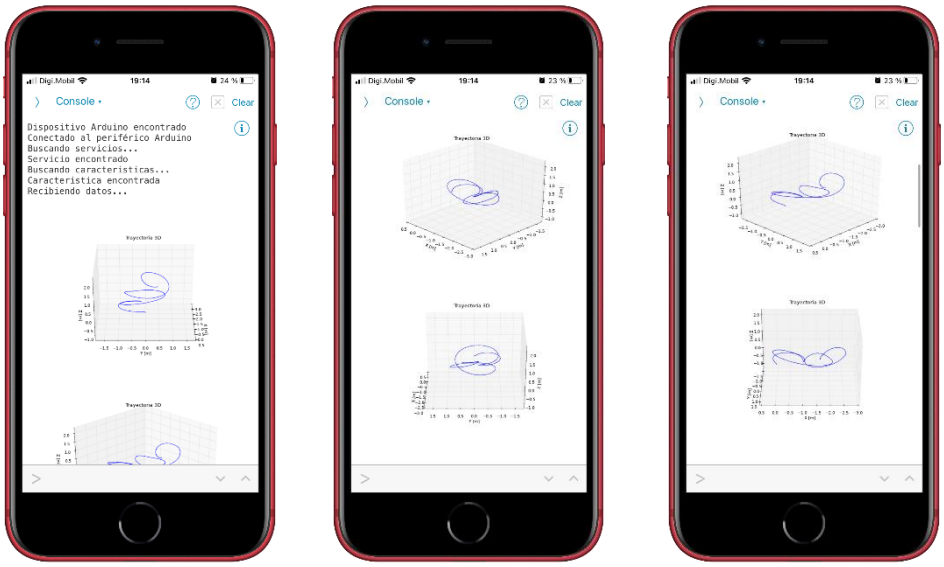


Figura 86. Recibiendo información del dispositivo.

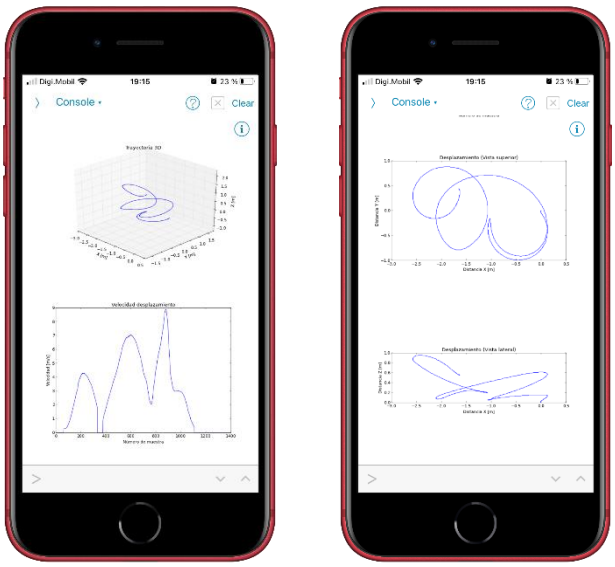


Figura 87. Recibiendo información del dispositivo II

Para una mejor visualización es posible hacer un zoom sobre cualquier gráfica, o bien, descargarla a la galería de fotos para su posterior estudio (Figura 88).

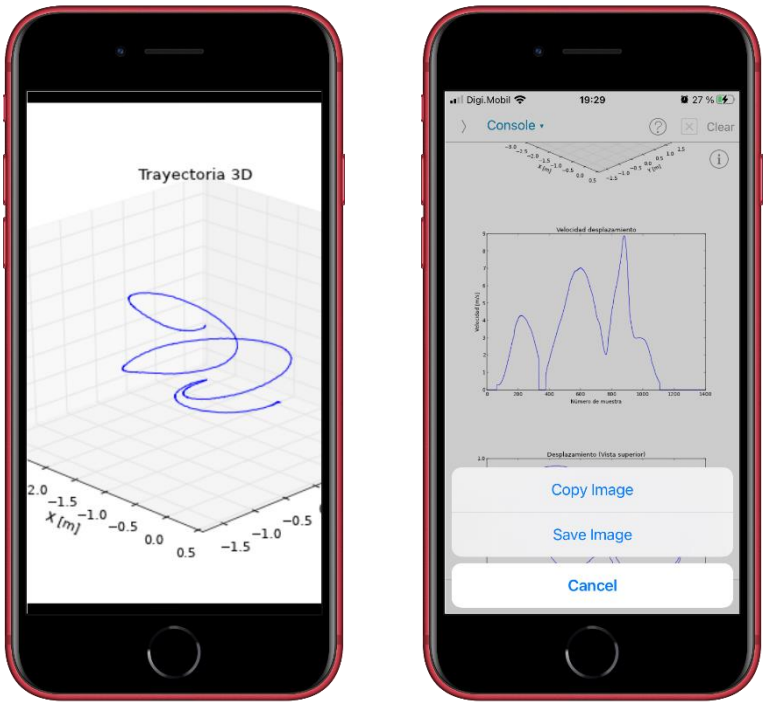


Figura 88. Operación de zoom y descarga

Estas gráficas son de 3 tipos:

- **Movimiento en 3D de la mano del lanzador**. Se representa el movimiento desde perspectivas diferentes (Figura 89).

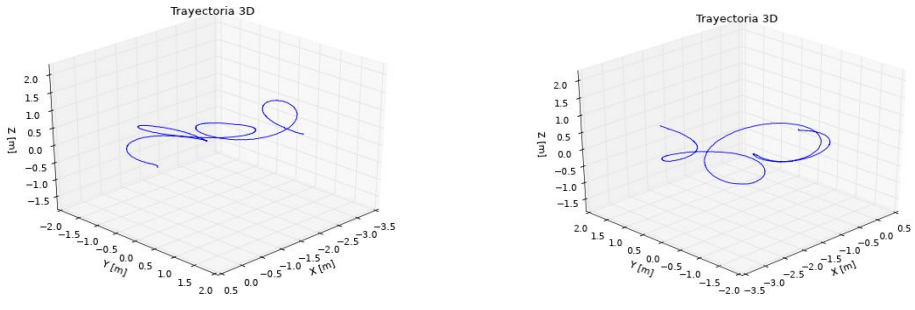


Figura 89. Trayectoria del lanzador en 3D

- **Vista superior y lateral del movimiento de la mano del lanzador.** Estas representaciones ofrecen una visión complementaria a las de 3 dimensiones (Figuras 90 y 91)

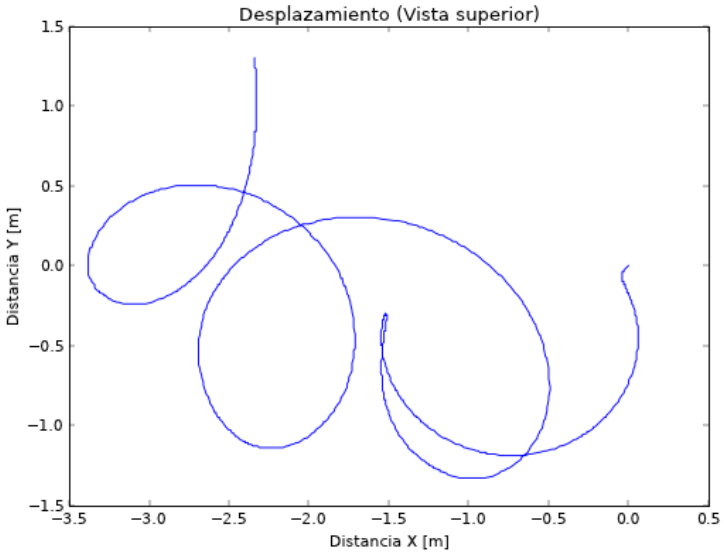


Figura 90. Vista superior de la trayectoria

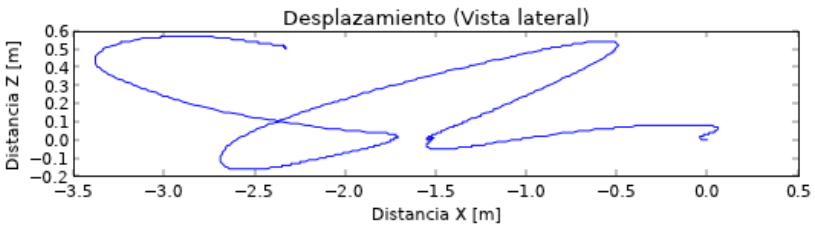


Figura 91. Vista lateral de la trayectoria

- **Velocidad de la mano del lanzador.** Esta gráfica representa la velocidad instantánea de la mano del lanzador durante el lanzamiento (Figura 92). Está expresada en m/s.

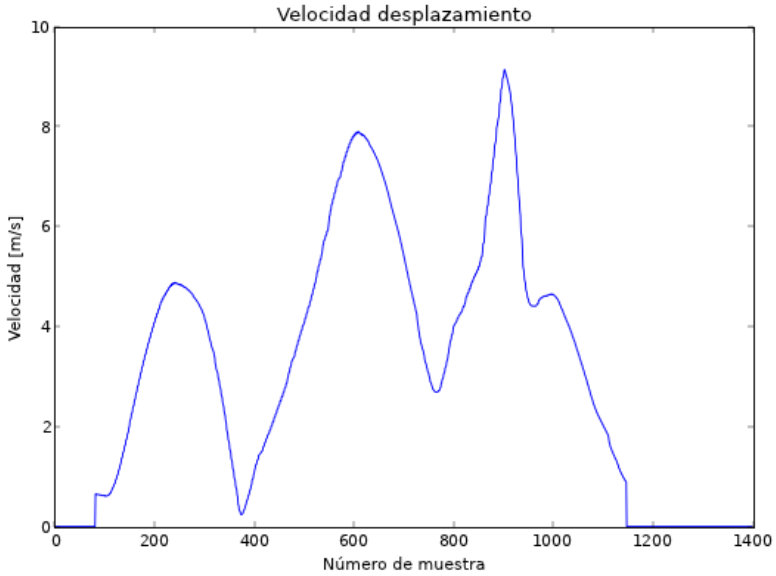


Figura 92. Velocidad del lanzamiento

Cuando ha terminado el proceso, en el dispositivo se muestra el mensaje *"Transmisión finalizada"* (Figura 93).

Figura 93. Transmisión finalizada

Llegado a este punto es posible volver a realizar otra captura pulsando el botón derecho del dispositivo. Aparecerá de nuevo una cuenta atrás para que el lanzador se vaya preparando.

El ciclo vuelve a comenzar.

5. VALIDACIÓN DEL SISTEMA

Ante la imposibilidad de probar este dispositivo en una situación real de lanzamiento de disco, se ha simulado un espacio del tamaño similar al de la zona de lanzamiento y se ha realizado un movimiento similar (en la medida de lo posible) al que realizaría un lanzador profesional. Un ejemplo se muestra en la Figura 94.

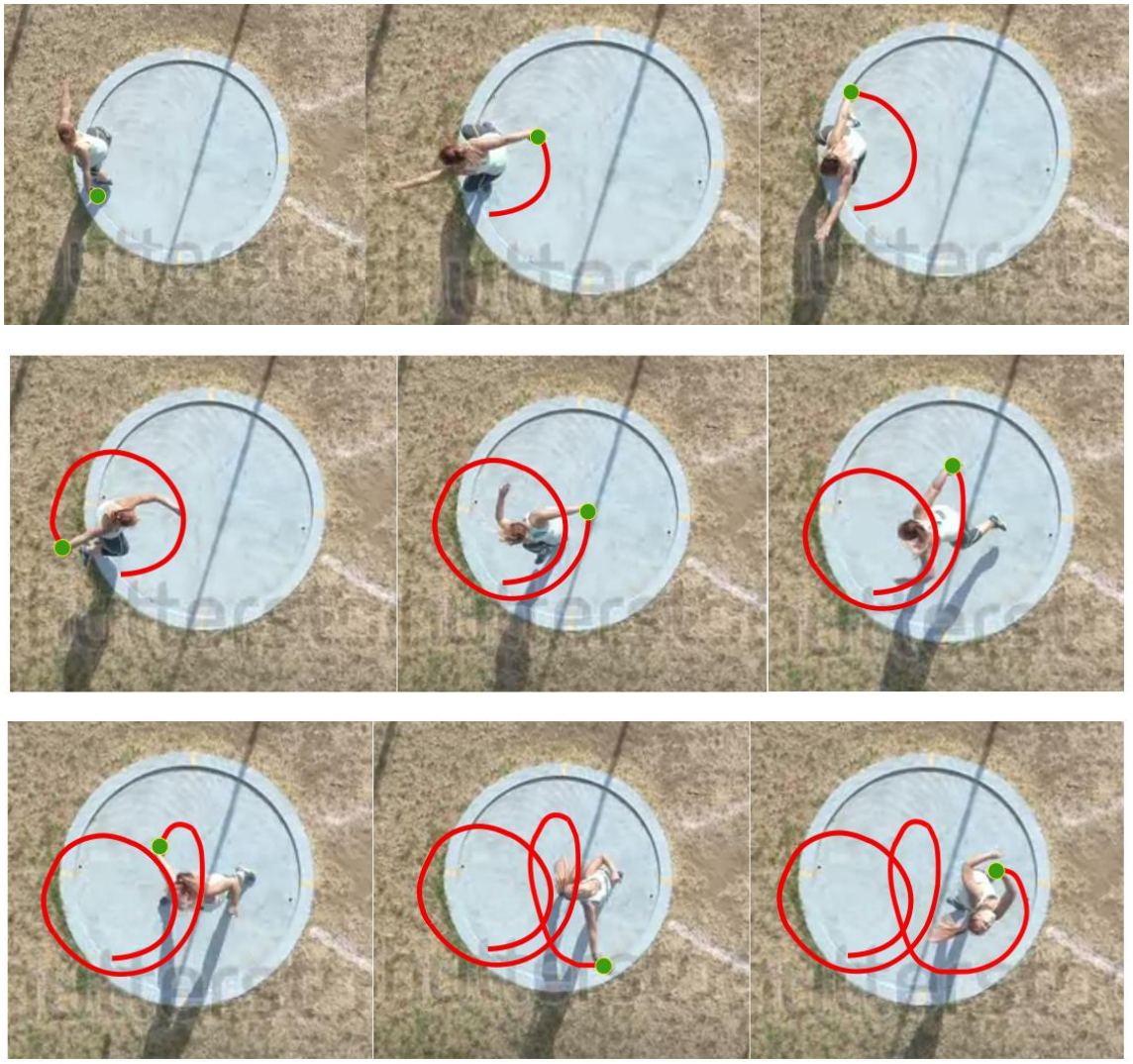


Figura 94. Proceso del lanzamiento realizado por lanzadora profesional. Fuente: shutterstock.com

Fueron realizados varias simulaciones del movimiento del lanzador de disco, y 5 de éstas se muestran a continuación. Dado que el programa ofrece la representación de la trayectoria en 3D desde varios ángulos, solo se mostrarán los que sean significativos para su seguimiento.

- **Simulación nº 1.**

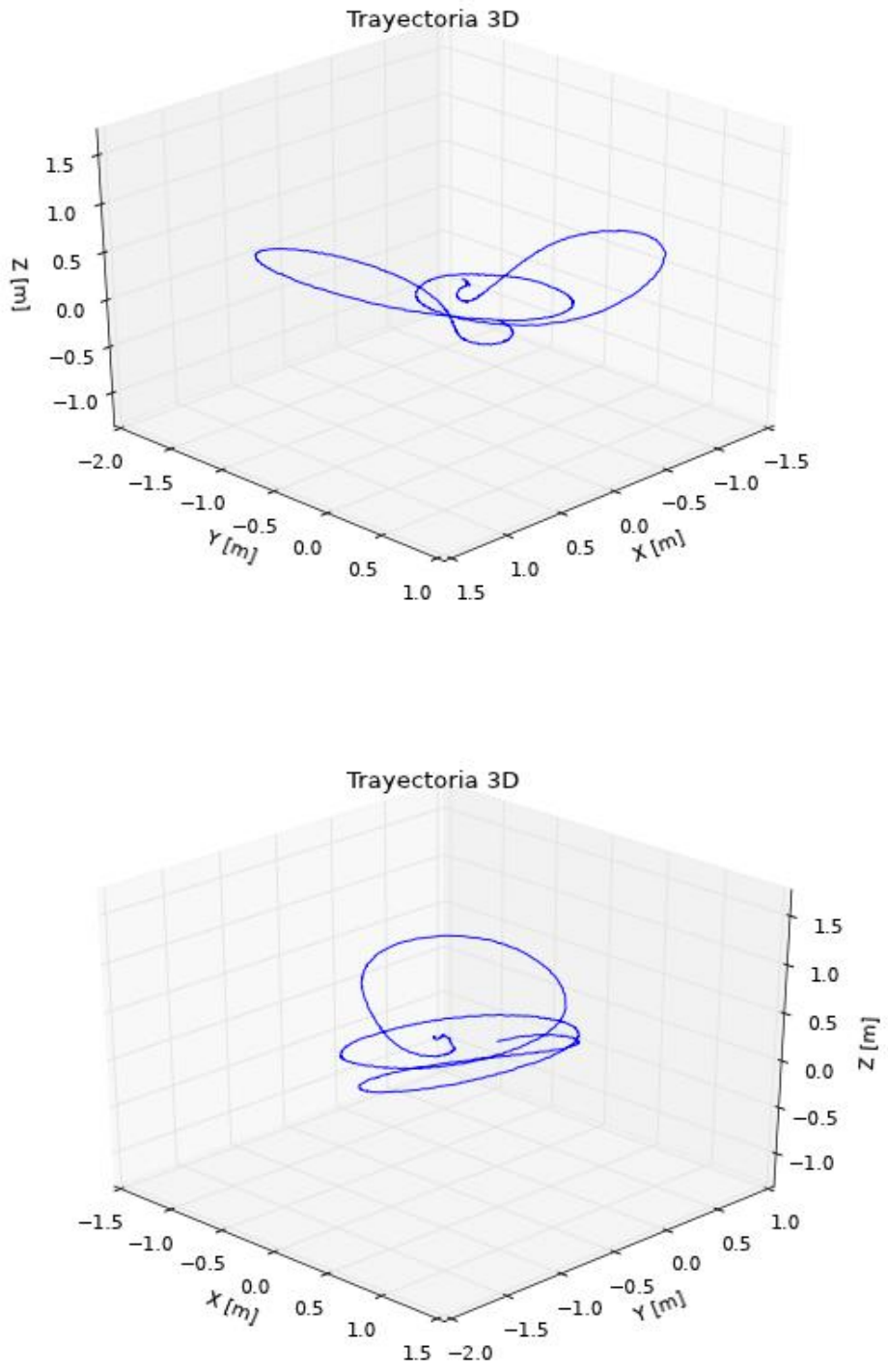


Figura 95. Trayectorias 3D. Simulación nº 1

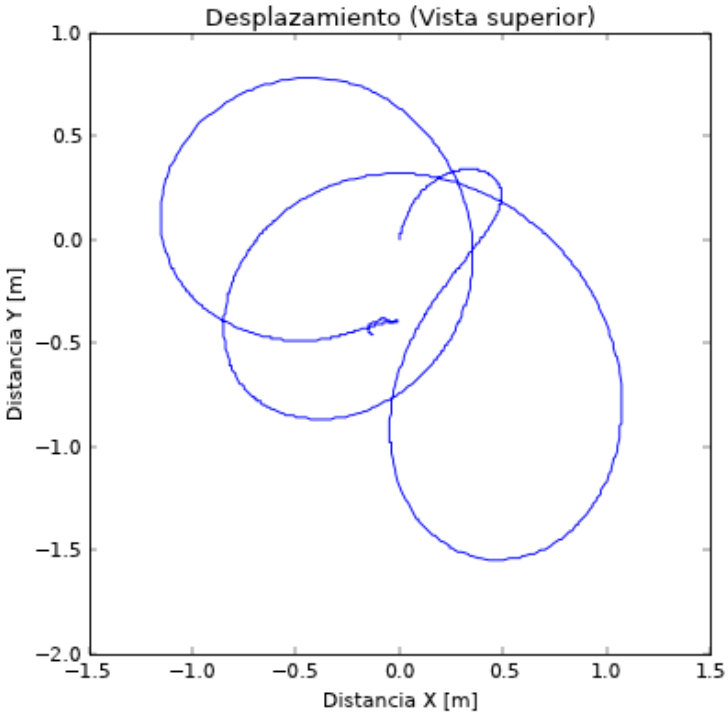


Figura 96. Vista superior. Simulación nº 1

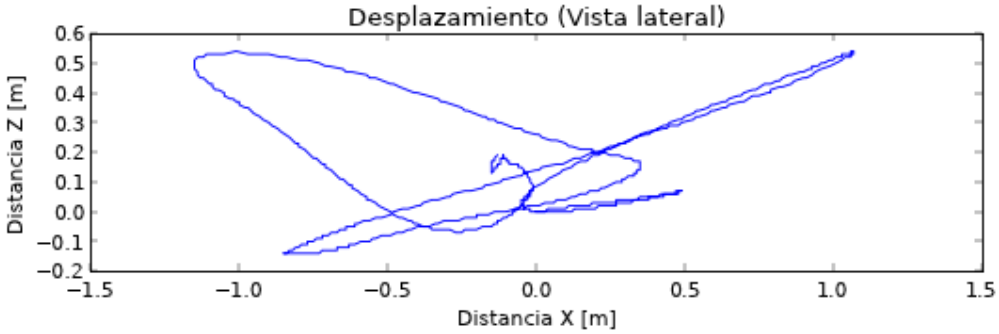


Figura 97. Vista lateral. Simulación nº 1

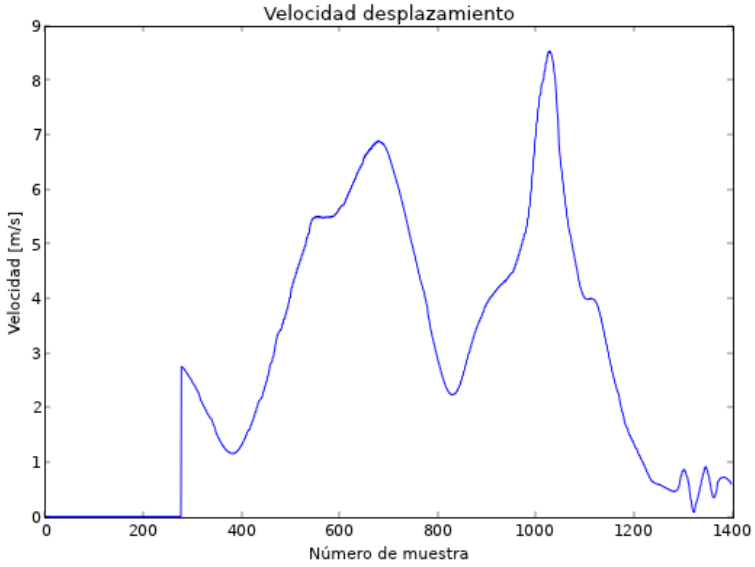


Figura 98. Velocidad. Simulación nº 1

Observaciones de la simulación 1.

El inicio del movimiento no fue justo cuando el dispositivo comenzó la captura. Esto provocó que el algoritmo de "Aproximación a 0 de la velocidad" detectara que el dispositivo estaba detenido durante el tiempo establecido y forzó la velocidad a 0, de ahí el cambio brusco de velocidad.

La trayectoria registrada (Figura 95), la distancia de los radios de giro (visualizado en la vista aérea, Figura 96) y la de la elevación del brazo durante el lanzamiento (vista lateral, Figura 97) se aproximan bastante a la realidad.

En la gráfica de la velocidad (Figura 98) se observa como el valor más alto coincide con el del lanzamiento del disco.

- **Simulación nº 2.**

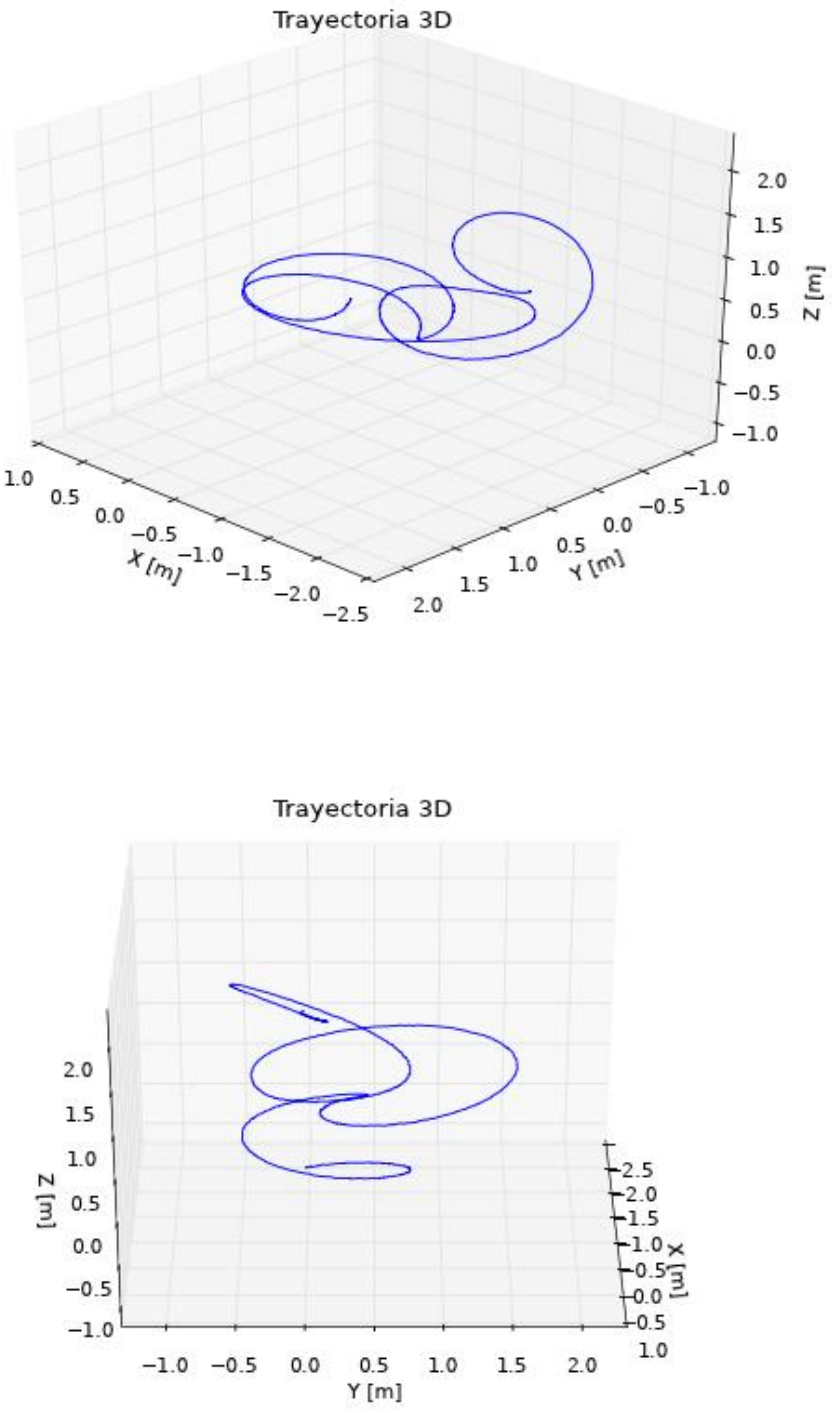


Figura 99. Trayectorias 3D. Simulación nº 2

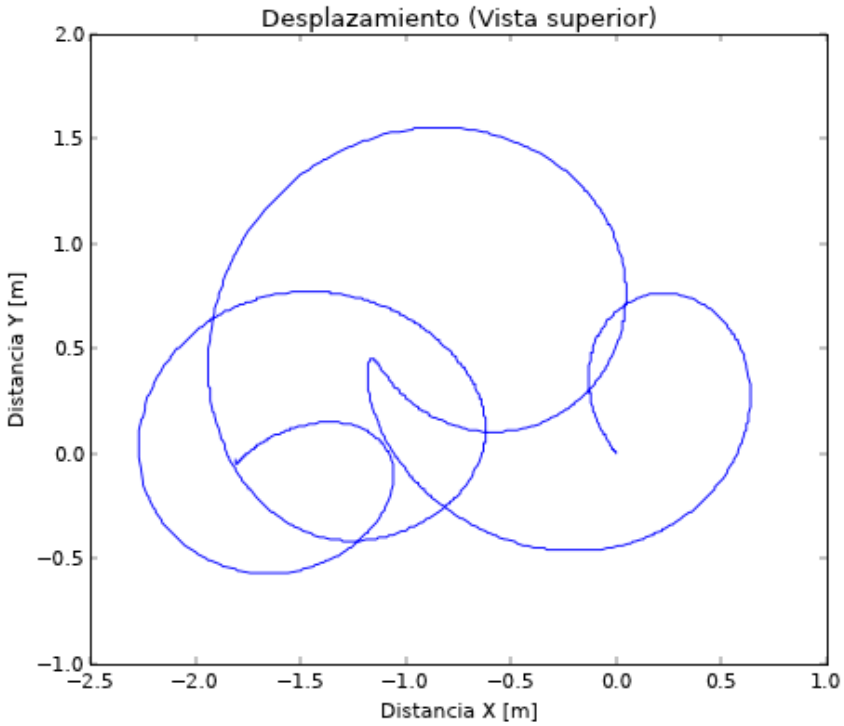


Figura 100. Vista superior. Simulación nº 2

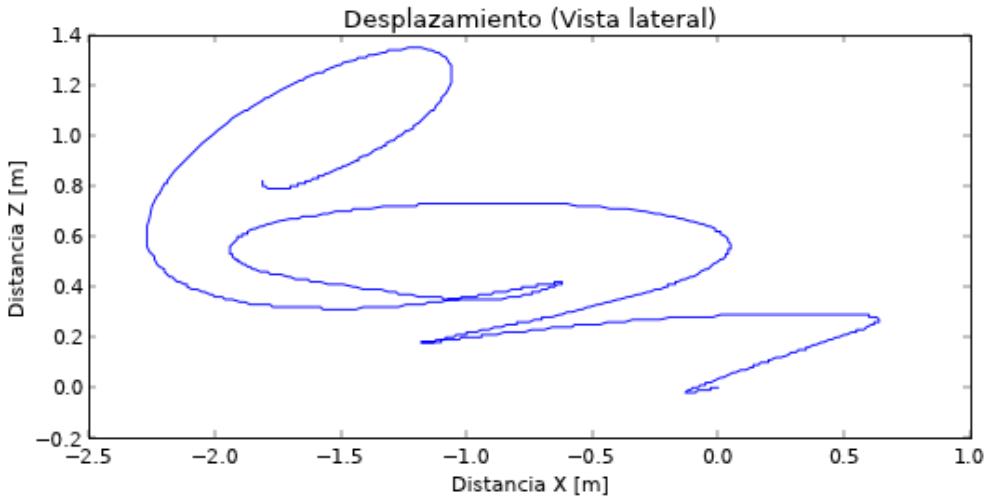


Figura 101. Vista lateral. Simulación nº 2

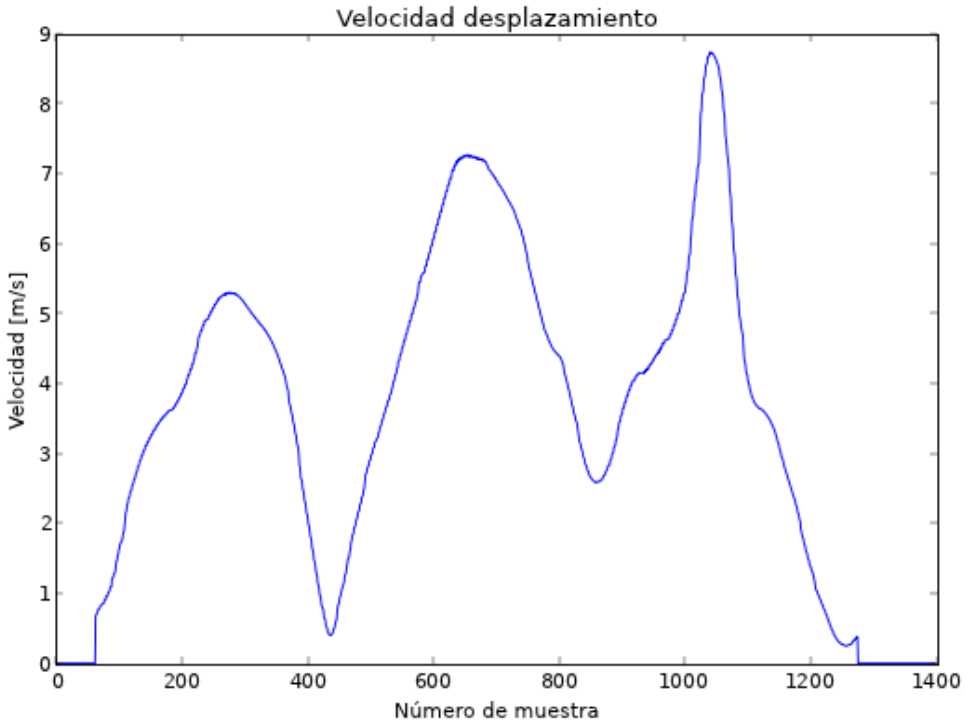


Figura 102. Velocidad. Simulación nº 2

Observaciones.

En la gráfica de la velocidad (Figura 102) se observan 3 picos de velocidad bastante definidos. El primero de ellos se corresponde con el movimiento de medio arco de circunferencia realizado en sentido opuesto. El algoritmo de "Aproximación a 0 de la velocidad" no lo detectó correctamente y no forzó a 0 la velocidad justo en el momento de cambio de sentido. Como consecuencia se observa en la vista aérea un ensanchamiento de la curva al comienzo del movimiento después del cambio de sentido.

La distancia de los radios de giro (visualizado en la vista aérea, Figura 100) se aproxima bastante a la realidad, pero el de la elevación del brazo durante el lanzamiento (vista lateral, Figura 101) es un poco mayor al real. Del mismo modo, la trayectoria reflejada (Figura 99) difiere un poco de la realidad. Este error posiblemente viene arrastrado por el fallo cometido al principio del movimiento.

En la gráfica de la velocidad se observa como el valor más alto coincide con el del lanzamiento del disco.

- **Simulación nº 3.**

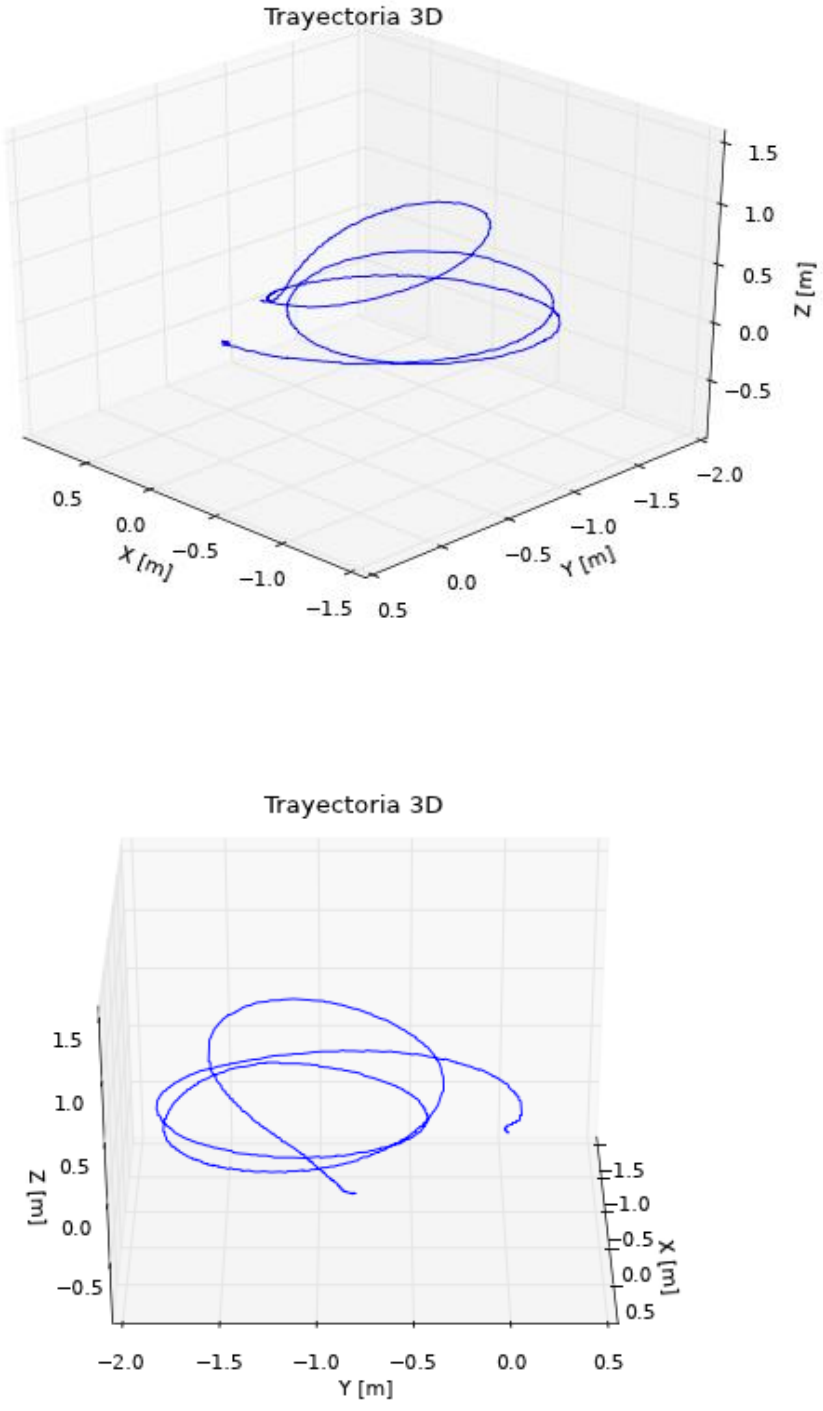


Figura 103. Trayectorias 3D. Simulación nº 3

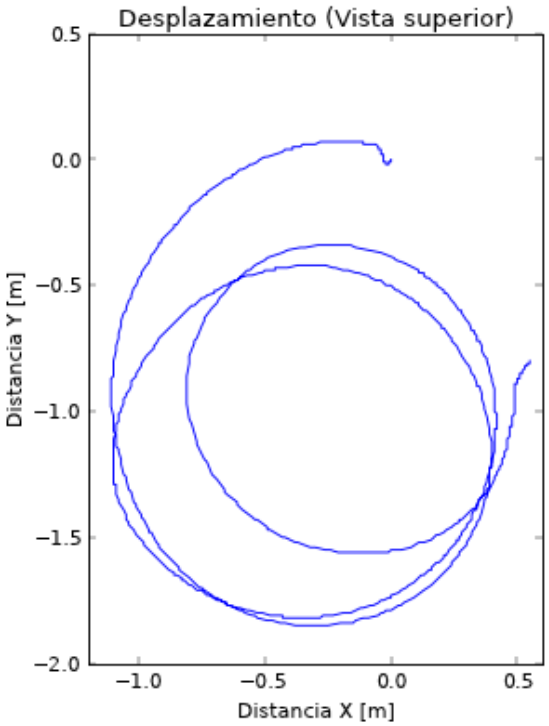


Figura 104. Vista superior y lateral. Simulación nº 3

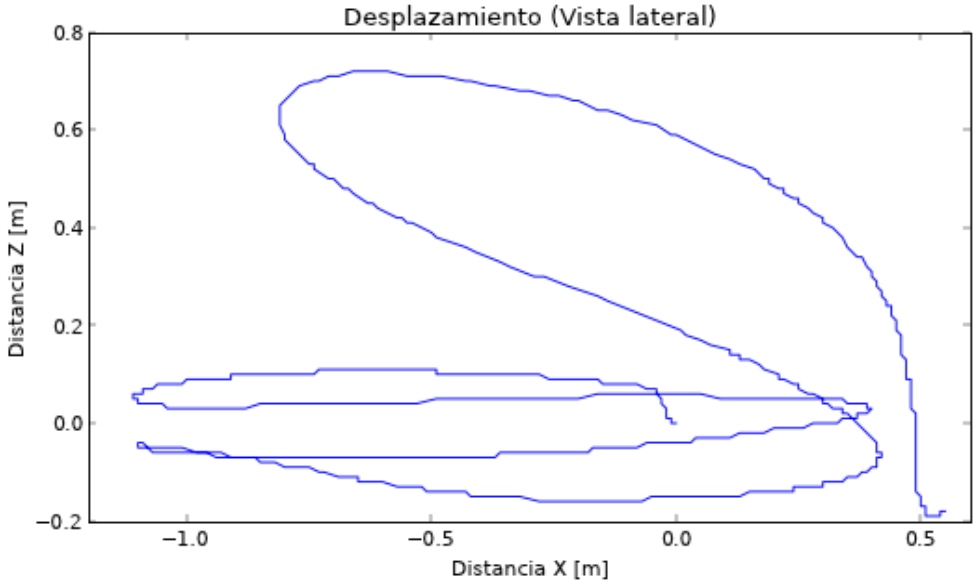


Figura 105. Vista lateral. Simulación nº 3

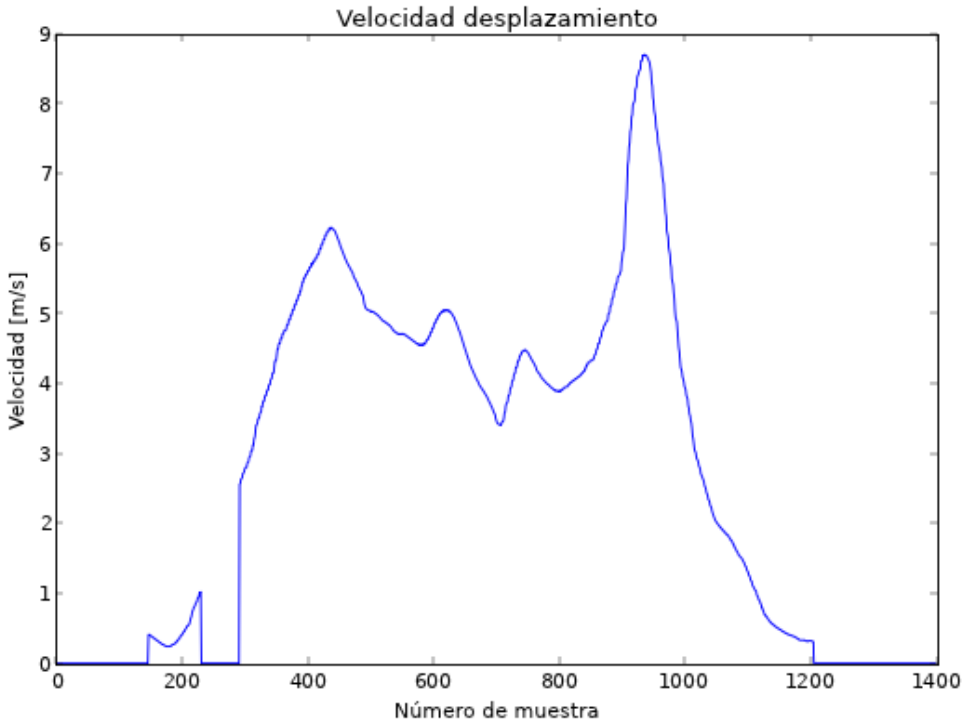


Figura 106. Velocidad. Simulación nº 3

Observaciones.

La simulación de este lanzamiento se realizó describiendo giros de menor radio y sin realizar ningún movimiento en sentido contrario al de giro.

La trayectoria registrada (Figura 103), la distancia de los radios de giro (visualizado en la vista aérea, Figura 104) y la de la elevación del brazo durante el lanzamiento (vista lateral, Figura 105) son muy precisos.

En la gráfica de la velocidad (Figura 106) se observa como el valor más alto coincide con el del lanzamiento del disco.

- **Simulación nº 4.**

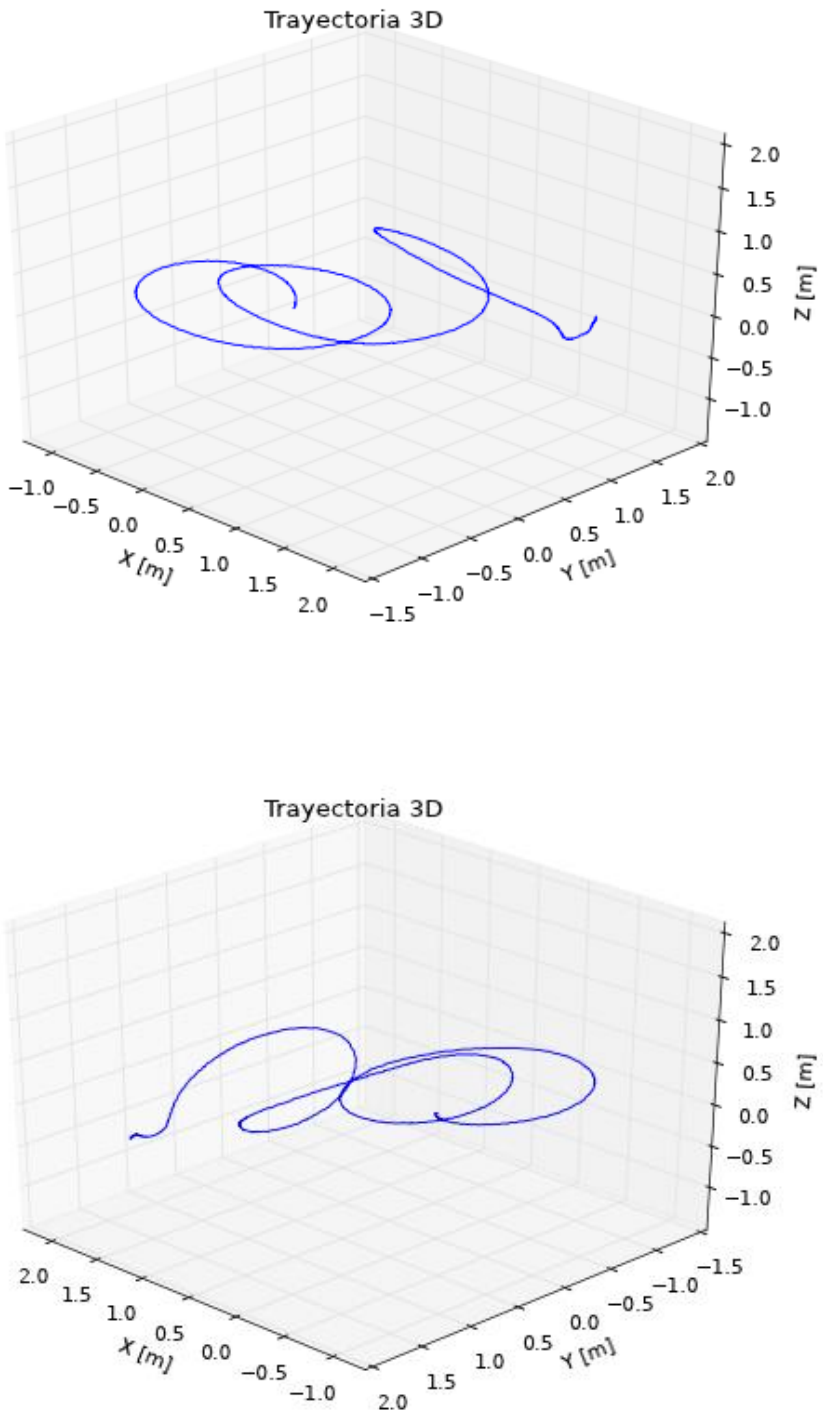


Figura 107. Trayectorias 3D. Simulación nº 4

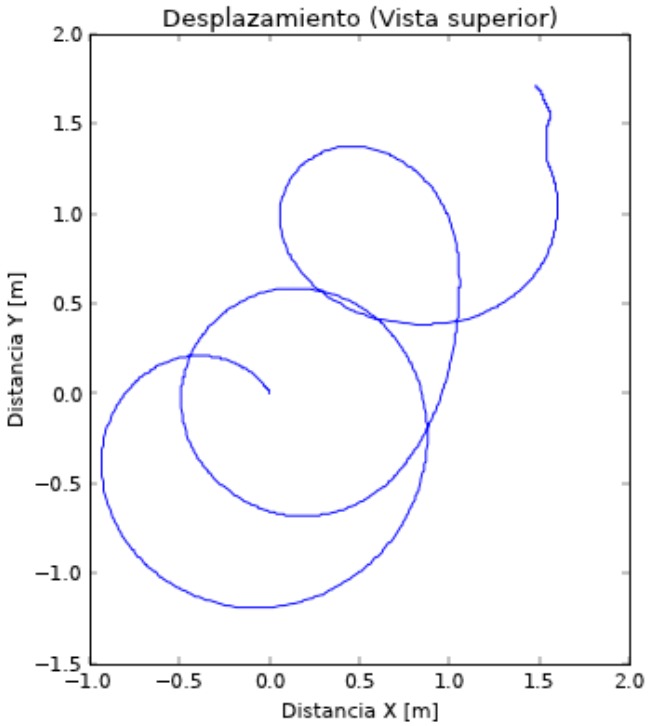


Figura 108. Vista superior. Simulación nº 4

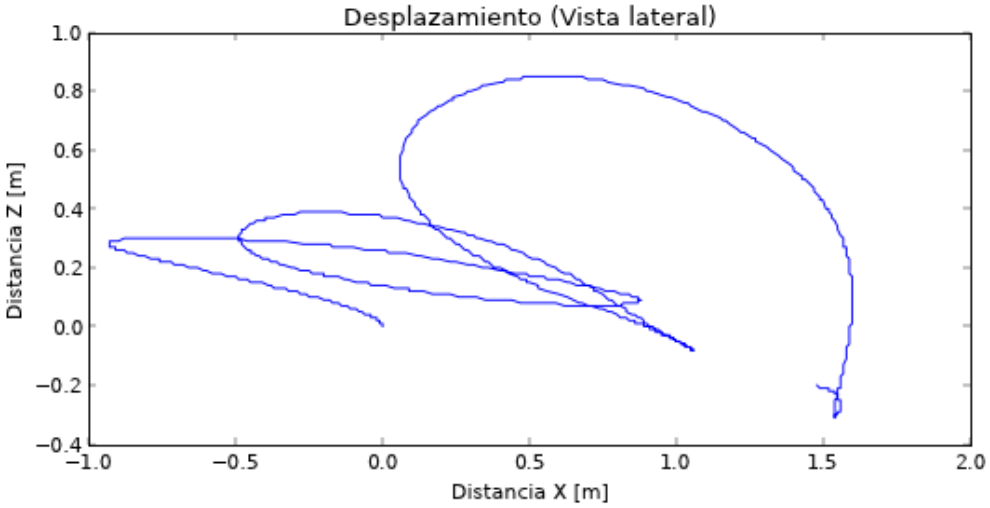


Figura 109. Vista lateral. Simulación nº 4

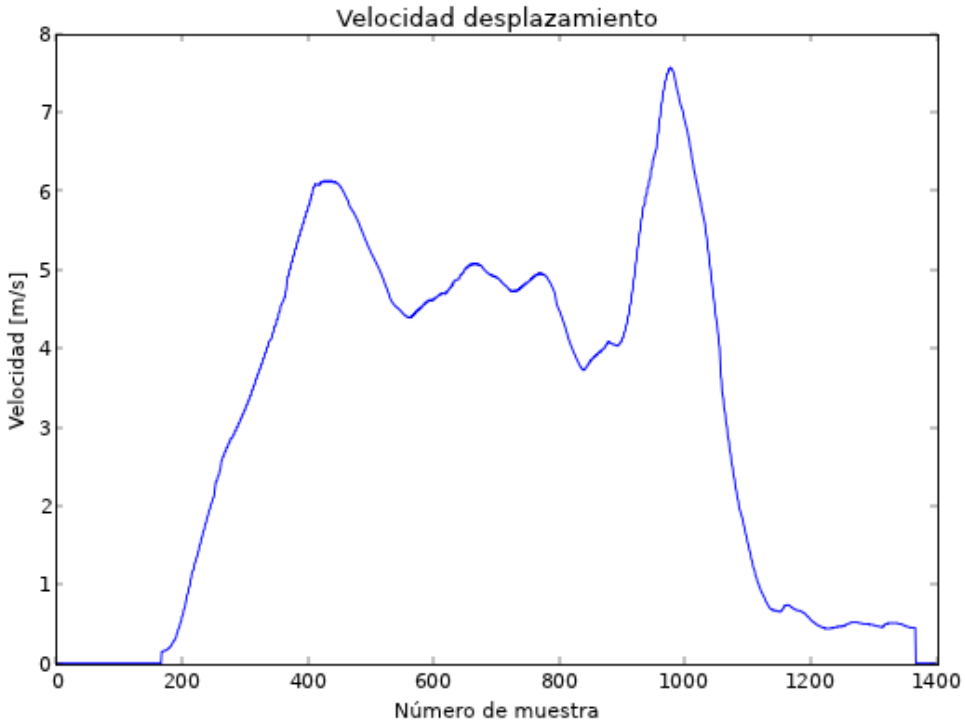


Figura 110. Velocidad. Simulación nº 4

Observaciones.

La simulación de este lanzamiento se ha realizado alargando el desplazamiento mientras se describían los círculos. No se ha realizado ningún movimiento en sentido contrario al de giro.

La trayectoria registrada (Figura 107), la distancia de los radios de giro (visualizado en la vista aérea, Figura 108) y la de la elevación del brazo durante el lanzamiento (vista lateral, Figura 109) son muy precisos.

En la gráfica de la velocidad (Figura 110) se observa como el valor más alto coincide con el del lanzamiento del disco.

- **Simulación nº 5.**

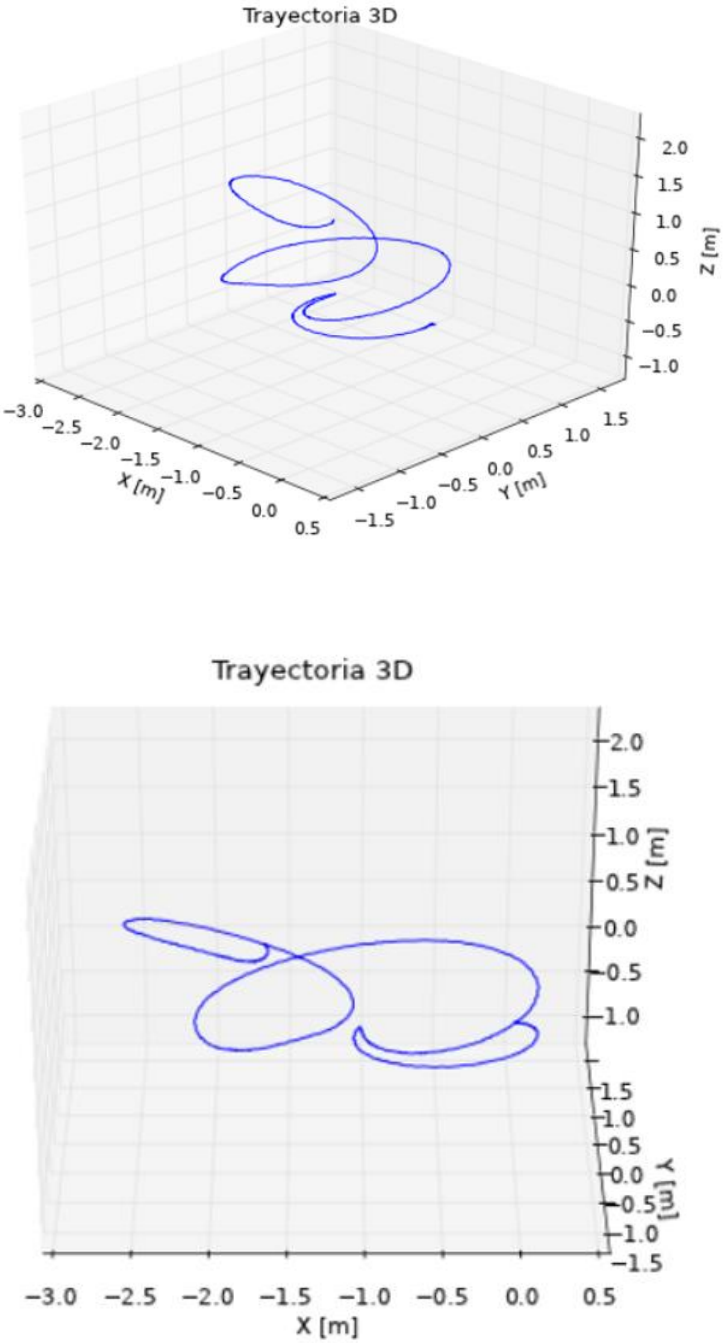


Figura 111. Trayectorias 3D. Simulación nº 5

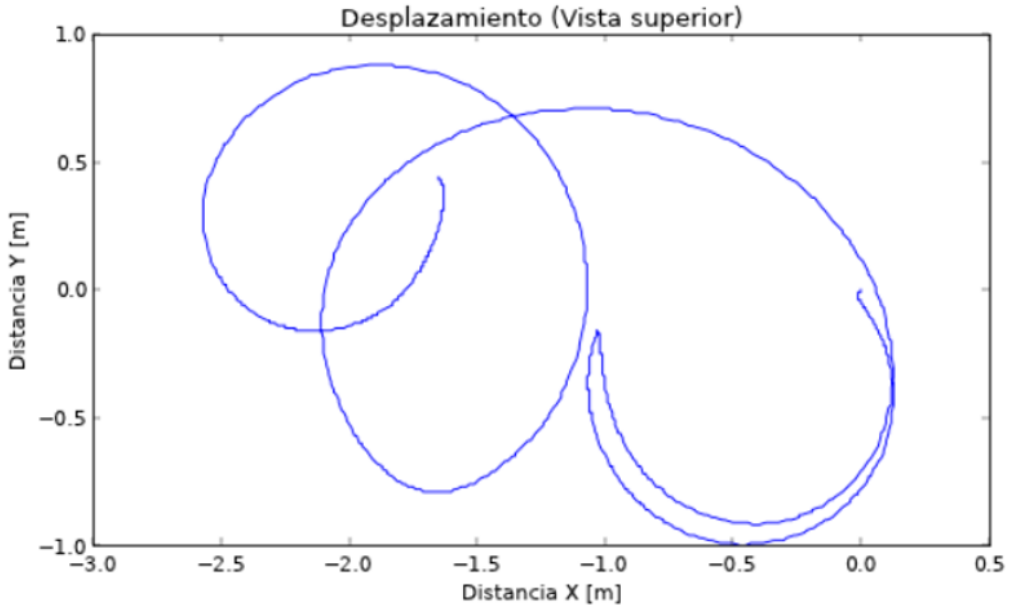


Figura 112. Vista superior. Simulación nº 5

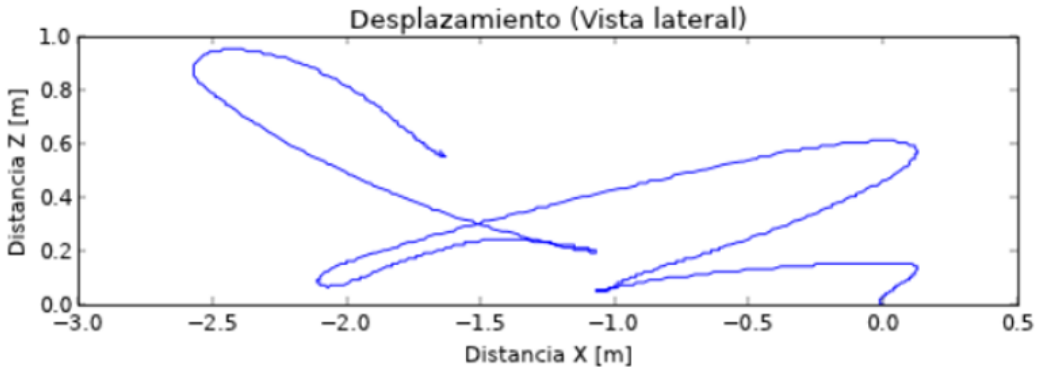


Figura 113. Vista lateral. Simulación nº 5

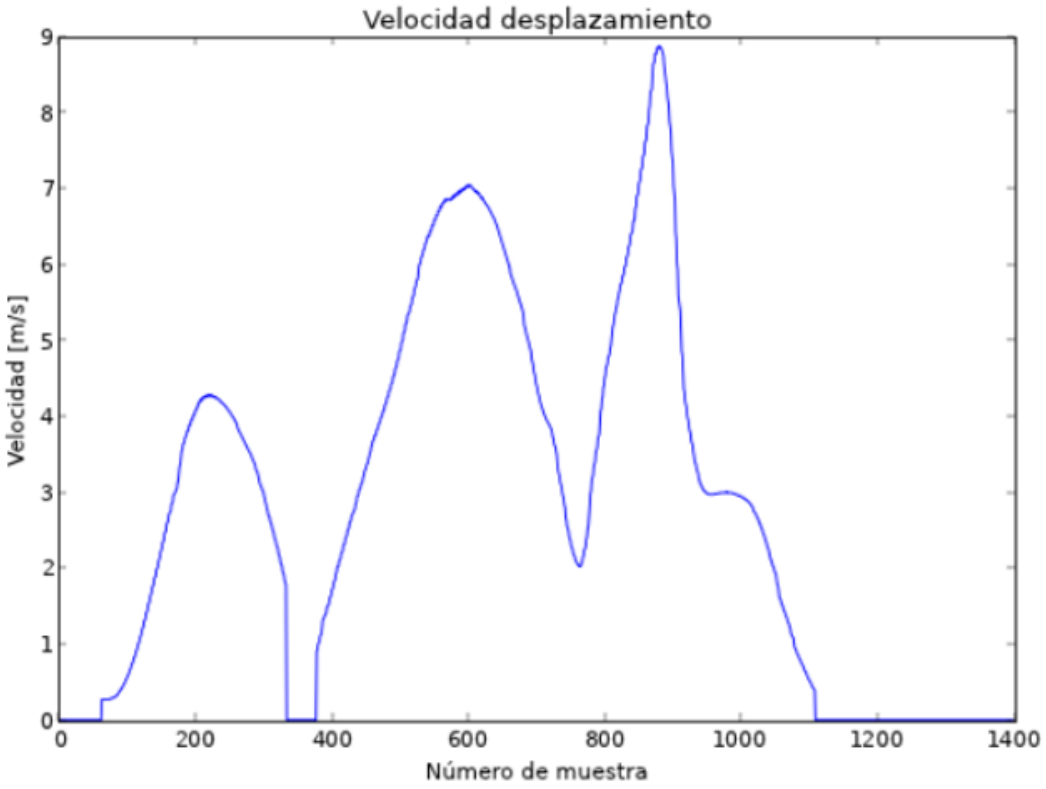


Figura 114. Velocidad. Simulación nº 5

Observaciones.

En la gráfica de la velocidad se observan 3 picos de velocidad bastante definidos. El primero de ellos se corresponde con el movimiento de medio arco de circunferencia realizado en sentido opuesto. El algoritmo de "Aproximación a 0 de la velocidad" detectó correctamente el cambio de sentido (ya que durante un instante la aceleración en este punto se aproxima a 0), forzando a que la velocidad fuese 0 en este punto. Como consecuencia de esto, el cálculo de la trayectoria es muy preciso y se genera el cambio de sentido que se observa en la vista aérea.

La trayectoria realizada (Figura 111), la distancia de los radios de giro (visualizado en la vista aérea, Figura 112) y la de la elevación del brazo durante el lanzamiento (vista lateral, Figura 113) son muy precisos.

En la gráfica de la velocidad (Figura 114) se observa como el valor más alto coincide con el del lanzamiento del disco.

6. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

En este proyecto se ha fabricado un dispositivo que es capaz de captar las variables físicas fruto del movimiento de un lanzador de disco, y posteriormente representar la trayectoria de la mano de éste en un teléfono móvil, así como su velocidad. Las pruebas realizadas para su verificación, a pesar de no poder ser realizadas por deportistas profesionales en un entorno real, han obtenido en general buenos resultados.

Todos los elementos necesarios para realizar las medidas (microcontrolador, IMU, batería, …) fueron integrados en una carcasa diseñada y fabricada para tal fin, empleando una impresora 3D. Esta carcasa, que se sujeta a la muñeca gracias a una cinta del velcro, permite que se cumpla una de las premisas de este trabajo, y es que el dispositivo sea portátil. La otra premisa, que sea de bajo coste, se cumple al emplear una placa Arduino.

El desarrollo del software del microcontrolador que gestiona la IMU ha supuesto la superación de una serie de retos que iban apareciendo conforme el proyecto iba avanzando. Entre ellos se pueden señalar:

- Calibración de los sensores y su vital importancia.
- Necesidad de un algoritmo de fusión y elección del más indicado.
- Empleo de filtros en las medidas realizadas.
- El problema de la deriva originada por el giroscopio y su corrección mediante un filtro paso alto.
- Errores en los cálculos ocasionados por la integración de la aceleración y velocidad.
- Establecimiento de la correcta orientación de los sensores según el convenio NED.
- Empleo de cuaterniones en lugar de ángulos de Euler para evitar el gimbal lock.
- Alta velocidad de ejecución de la lectura de los sensores.
- Transmisión de arrays de datos a través de BLE.

En cuanto al desarrollo de la aplicación en el teléfono móvil también surgieron una serie de dificultades, que, como en el caso anterior, también fueron superadas:

- Recepción de datos a través de BLE desde Arduino.
- Representación en 3D de la trayectoria.

Una vez superados todos estos retos, montado y programado cada una de las partes que forman el sistema, se llevaron a cabo una serie de simulaciones, que básicamente han consistido en realizar giros intentando emular a un lanzador de disco. Las representaciones de las trayectorias de estos movimientos son mostradas en forma de gráficas 2D y 3D en el teléfono móvil, teniendo la interfaz gráfica de la aplicación aún margen de mejora. De momento es algo para desarrolladores y se ejecuta en modo consola.

Las pruebas de verificación realizadas al dispositivo demuestran que es capaz de captar y representar de manera aceptable el movimiento y velocidad de la mano del lanzador de disco. Se indica que los resultados son aceptables y no perfectos dado que en ciertas simulaciones el resultado difería un poco de la realidad. Esto ocurre en ocasiones cuando se realiza un movimiento de cambio de sentido de giro. A pesar de implementar un algoritmo para detectar y corregir estas circunstancias, no siempre lo consigue.

Como conclusión general se puede establecer que el empleo de este tipo de dispositivos para representar el desplazamiento de un móvil conlleva una serie de limitaciones que han de tenerse en cuenta a la hora de validar los resultados obtenidos.

La principal limitación es que no se pueden realizar registros prolongados en el tiempo sin algún mecanismo de reseteo (*dead reckoning)*, o bien, tener un elemento externo (señal GPS, visión artificial) para ir corrigiendo los errores en las medidas. Sin estos mecanismos, la deriva en la medida provocada por el giroscopio y/o el error acumulado durante la integración, primero de la aceleración y después de la velocidad, harían inservibles la adquisición de un largo trayecto. Para este trabajo, al tratarse de adquisiciones de movimientos cortos, no es necesario emplear mecanismos auxiliares.

Tras realizar diversas pruebas con el dispositivo a lo largo de este trabajo, queda de manifiesto la imperiosa necesidad de realizar una correcta calibración de los sensores antes de realizar las capturas de las medidas. El sensor que más acusa la necesidad de calibración es el magnetómetro, cuyas medidas eran drásticamente distintas antes y después de colocar la IMU en el interior de la carcasa. El acelerómetro y giroscopio también reflejaban diferencias antes y después de la calibración, pero en menor medida.

El desarrollo de este Trabajo Fin de Máster ha puesto de manifiesto la cantidad de campos de conocimiento necesarios para poder llevar a cabo un proyecto de esta índole. De manera resumida se podrían citar unos cuantos:

- Matemáticas: empleo de cuaterniones y matrices de rotación.
- Electrónica: diseño del circuito eléctrico.
- Programación: realización de un programa en C++ y otro en Python.
- Diseño industrial: diseño y fabricación de la carcasa.
- Física: conceptos de aceleración, velocidad y desplazamiento.

Todos ellos, en mayor o menor medida, son imprescindibles para la fabricación y puesta a punto de este dispositivo.

En virtud del trabajo realizado en la elaboración de este proyecto, son varias las mejoras que se pueden aplicar al mismo.

6.1. Modelo biomecánico completo.

La implantación de diversos sensores inerciales repartidos a lo largo del cuerpo del lanzador de disco permitirían representar los parámetros cinemáticos en conjunto, y si son representados interconectados al igual que los están los huesos, se podría generar una representación virtual del cuerpo del lanzador al completo, como se muestra en la Figura 115.

Figura 115. Ejemplo de modelo biomecánico. Fuente: biomech-solutions.com

Esta representación, al no limitarse solamente al movimiento de la mano, permitiría el estudio y mejora de la maniobra de lanzamiento, ya que permite el estudio del movimiento de las extremidades de manera independiente y de todo el cuerpo en conjunto.

6.2. Representación interactiva de los resultados.

La representación del movimiento del lanzador en el teléfono móvil llevada a cabo en este trabajo es estática, es decir, se representan las gráficas en 2D o 3D como una imagen plana.

Una mejora de este proyecto sería la introducción de gráficas interactivas. Mediante el empleo de este tipo de gráficas, el usuario podría rotar la trayectoria del movimiento 360º en cualquier eje, además de poder acercarse o alejarse de las mismas para poder tener una mejor perspectiva.

Para generar este tipo de gráficas en un teléfono móvil (un iPhone en este caso) se necesita realizar un programa en el entorno de programación *Xcode*, bajo el lenguaje de programación *Swift* (Figura 116).

Figura 116. Xcode y Swift logos

Un ejemplo del resultado de la gráfica en 3D resultante es el mostrado en la Figura 117:

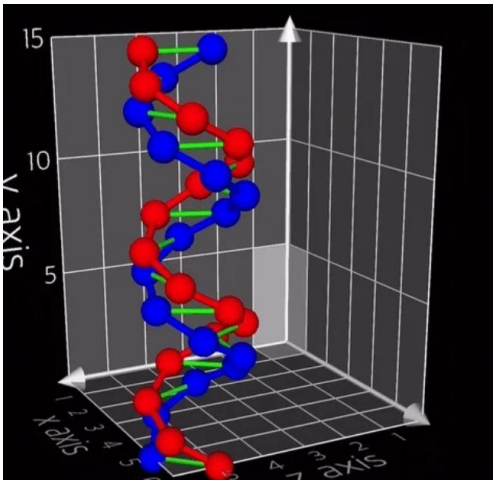


Figura 117. Ejemplo de gráfica 3D. Fuente: betterprogramming.pub

7. ANEXOS

7.1. Anexo I. Planos dispositivos

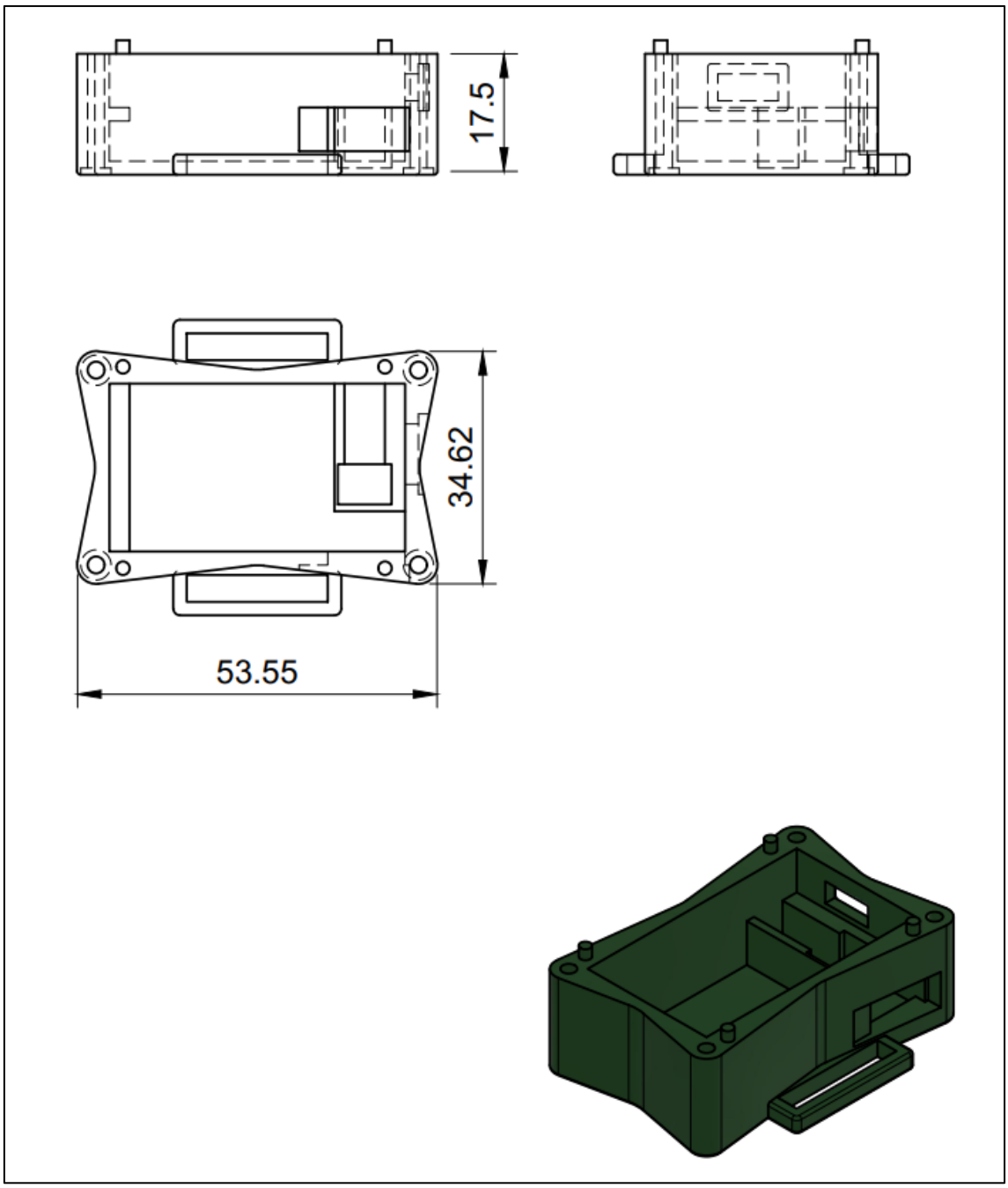


Figura 118. Planos caja

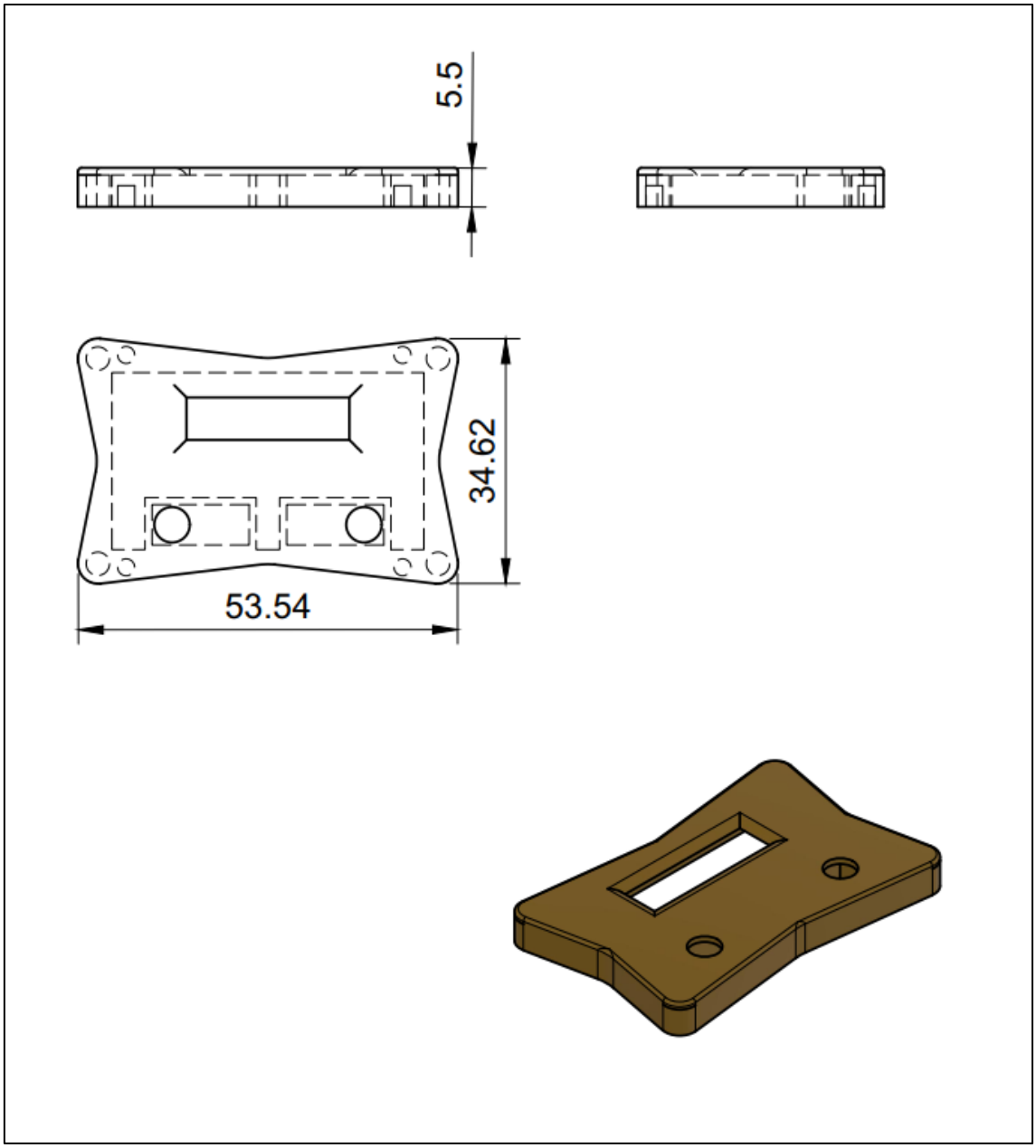


Figura 119. Planos tapa

7.2. Anexo II. Programa Arduino

Este es el programa que va cargado en la placa Arduino Nano 33 BLE y que realiza la captura y proceso de datos desde la IMU al teléfono móvil.

```
#include <Arduino.h>
#include <Arduino_LSM9DS1.h>
#include "SensorFusion.h"
#include <math.h>
#include <BasicLinearAlgebra.h>
#include <filters.h>
#include <ArduinoBLE.h>
#include <Wire.h>
#include "SSD1306Ascii.h"
#include "SSD1306AsciiWire.h"
#include <Ewma.h>

Ewma adcFilter_x(0.1);   // Filtrado de ruido para cada una de las aceleraciones (X, Y, Z)
Ewma adcFilter_y(0.1);
Ewma adcFilter_z(0.1);

// FASES
int fase = 0;
bool mensaje_mostrado = false;
bool mensaje_mostrado_2 = false;

// OLED
#define I2C_ADDRESS 0x3C
#define RST_PIN -1
SSD1306AsciiWire oled;

// BLE
BLEService posiciones("1A3AC132-31ED-758C-BC52-54A61958EF82"); //Declare service for Gyroscope
BLECharacteristic buffCharacteristic("1A3AC132-31ED-758C-BC52-54A61958EF83", BLERead | BLENotify,20,(1==1) );
char buf[20];

// Filtrado de la velocidad para reducir el efecto de la componente continua
const float cutoff_freq   = 50.0;   // Frecuencia de corte
const float sampling_time = 0.00001; // Velocidad de muestreo

Filter fhp_x(cutoff_freq, sampling_time, IIR::ORDER::OD2, IIR::TYPE::HIGHPASS);
Filter fhp_y(cutoff_freq, sampling_time, IIR::ORDER::OD2, IIR::TYPE::HIGHPASS);
Filter fhp_z(cutoff_freq, sampling_time, IIR::ORDER::OD2, IIR::TYPE::HIGHPASS);
float sign_raw, sign_filt;

// Parametros para que la velocidad sea 0 si la aceleracion esta entre una ventana
// un numero de muestras consecutivas
#define n_ceros 60        // Nº de muestras consecutivas
int limite_acel = 1.0;     // Limite inferior y superior
unsigned int n_ax = 0;
unsigned int n_ay = 0;
unsigned int n_az = 0;
unsigned int i = 0;

SF fusion;                // Creación del objeto 'fusion'
using namespace BLA;

#define periodo_muestreo 1/500

BLA::Matrix<3> accBodyVector;
BLA::Matrix <3> accNEDVector;
BLA::Matrix <3,3> rotMatrix;
BLA::Matrix <3,3> rotMatrixInv;

unsigned long millis_inicio;
unsigned long tiempo_transcurrido;
#define t_calentamiento 20000          // Tiempo de acondicionamiento (ms)
unsigned int n_buffer = 0;
#define t_buffer 3000      // Tamaño del buffer

float gx, gy, gz, ax, ay, az, mx, my, mz, temp;
float gx_buffer[t_buffer], gy_buffer[t_buffer], gz_buffer[t_buffer];
float ax_buffer[t_buffer], ay_buffer[t_buffer], az_buffer[t_buffer];
float mx_buffer[t_buffer], my_buffer[t_buffer], mz_buffer[t_buffer];
float velocidad[t_buffer];
float deltat_buffer[t_buffer];
float deltat;

float q((7];      // Cuaterniones
float g[3];

float aT_x, aT_y, aT_z;

// Función para formar un número con 'x' números enteros y 'x' números decimales
char *dtostrf (double val, signed char width, unsigned char prec, char *sout) {
  char fmt[20];

  sprintf(fmt, "%%%d.%df", width, prec);
  sprintf(sout, fmt, val);
  return sout;
}
```

```
String convertFloatToString(float temperature)
{ // begin function

   char temp((12];
   String tempAsString;

   // perform conversion
   dtostrf(temperature,5,2,temp);

   // create string object
   tempAsString = String(temp);

   return tempAsString;

} // end function

void setup() {

// OLED
  Wire.begin();
  Wire.setClock(400000L);

#if RST_PIN >= 0
  oled.begin(&Adafruit128x64, I2C_ADDRESS, RST_PIN);
#else // RST_PIN >= 0
  oled.begin(&Adafruit128x32, I2C_ADDRESS);
#endif // RST_PIN >= 0

  Serial.begin(9600);

  if ( !BLE.begin() )
  {
    while ( 1 );
  }

// Setup BLE
   BLE.setAdvertisedService( posiciones );
   posiciones.addCharacteristic( buffCharacteristic );
   BLE.addService( posiciones );
   BLE.setAdvertisingInterval(100); // 200 * 0.625 ms
   BLE.advertise();                                        //Starts advertising the peripheral device over bluetooth

  // start communication with IMU
  if (IMU.begin() < 0) {
    while (1) {}
  }

// CALIBRACION DE LOS SENSORES

      // Magnetometer code
   IMU.setMagnetFS(1);
   IMU.setMagnetODR(8);  // nueva calibracion
   IMU.setMagnetOffset(-2.397461, 2.932129, 24.388428);
   IMU.setMagnetSlope (0.966864, 0.963157, 0.971843);

      // Accelerometer code
   IMU.setAccelFS(3);
   IMU.setAccelODR(5);
   IMU.setAccelOffset(-0.024975, 0.002536, -0.043924);
   IMU.setAccelSlope (0.994798, 0.992403, 1.000054);

      // Gyroscope code
   IMU.setGyroFS(3);
   IMU.setGyroODR(5);
   IMU.setGyroOffset (0.043915, -0.345947, 0.325928);
   IMU.setGyroSlope (1.159869, 1.147911, 1.129216);

   millis_inicio = millis();
   n_buffer = 0;
}

void loop() {

BLEDevice central = BLE.central();         //Waits for BLE Central device to connect

switch (fase)
{
case 0:      // Fase 0: Fase inicial. El dispositivo se encuentra en reposo
   if (mensaje_mostrado == false)
   {
      oled.setFont(Cooper19);
      oled.clear();
      oled.println("   MOLADIS");   // Texto a mostrar en la pantalla
      mensaje_mostrado = true;
   }

   if (digitalRead(D4) == true)     // Si se pulsa el botón izuierdo pasa de fase
   {
      fase = 1;
      mensaje_mostrado = false;
      millis_inicio = millis();
   }
   break;

case 1:     // Fase 1: Estabilizacion de la lecturas de la IMU

   tiempo_transcurrido = millis() - millis_inicio;
```

```
   if (tiempo_transcurrido < t_calentamiento)    // Si transcurre el tiempo de calentamiento...
   {
      oled.setFont(Callibri15);
      oled.home();
      oled.println("      - Estabilizacion -      ");
      oled.print("Tiempo restante: ");
      oled.clearField(100,2,2);
      oled.setCursor(100,2);
      oled.println((t_calentamiento - (tiempo_transcurrido))/1000);

      IMU.readMagnet(mx, my, mz);
      IMU.readAccel(ax, ay, az);
      IMU.readGyro(gx, gy, gz);

      deltat = fusion.deltatUpdate();
      fusion.MadgwickUpdate(gx*PI/180.0f, gy*PI/180.0f, -gz*PI/180.0f, ax, ay, -az, -mx, my, -mz, deltat);//
   }
   else
   {
      fase = 2;
      digitalWrite(D2,HIGH);     // Generación de una pequeña vibración con el vibrador
      delay(500);
      digitalWrite(D2,LOW);
      delay(500);
      IMU.readMagnet(mx, my, mz);
      IMU.readAccel(ax, ay, az);
      IMU.readGyro(gx, gy, gz);
      deltat = fusion.deltatUpdate();
      fusion.MadgwickUpdate(gx*PI/180.0f, gy*PI/180.0f, -gz*PI/180.0f, ax, ay, -az, -mx, my, -mz, deltat);
   }

   break;

case 2:      // Fase 2: Captura de datos de la IMU en bruto
   if (mensaje_mostrado == false)
   {
      oled.clear();
      oled.setFont(Callibri15);
      oled.home();
      oled.println("         Capturando    ");
      oled.setCursor(1,2);
      oled.println("         movimiento    ");
      mensaje_mostrado = true;
   }

  // delay(5);

   IMU.readMagnet(mx_buffer[n_buffer], my_buffer[n_buffer], mz_buffer[n_buffer]);
   IMU.readAccel(ax_buffer[n_buffer], ay_buffer[n_buffer], az_buffer[n_buffer]);
   IMU.readGyro(gx_buffer[n_buffer], gy_buffer[n_buffer], gz_buffer[n_buffer]);

   deltat_buffer[n_buffer] = fusion.deltatUpdate();

   n_buffer++;

   if (n_buffer > t_buffer-1)
   {
      mensaje_mostrado = true;
      fase = 3;
      n_buffer = 0;
      mensaje_mostrado = false;

      digitalWrite(D2,HIGH);  // Se realiza una pequeña vibración para indicar que la
      delay(200);             // captura de datos ha finalizado
      digitalWrite(D2,LOW);
      delay(200);
      digitalWrite(D2,HIGH);
      delay(200);
      digitalWrite(D2,LOW);
   }
   break;

case 3:      // Fase 3: Calculo de la velocidad y posicion a partir de la aceleracion
   if (mensaje_mostrado == false)
   {
      oled.clear();
      oled.setFont(Callibri15);
      oled.home();
      oled.println("  Calculando velocidad  ");
      oled.setCursor(1,2);
      oled.println("        y posiciones    ");
      mensaje_mostrado = true;
   }

   for (n_buffer = 0; n_buffer < t_buffer; n_buffer++)
   {
      fusion.MadgwickUpdate(gx_buffer[n_buffer]*PI/180.0f, gy_buffer[n_buffer]*PI/180.0f, -gz_buffer[n_buffer]*PI/180.0f,
ax_buffer[n_buffer], ay_buffer[n_buffer], -az_buffer[n_buffer],-mx_buffer[n_buffer], my_buffer[n_buffer], -
mz_buffer[n_buffer],deltat_buffer[n_buffer]);

      memcpy(q,fusion.getQuat(),sizeof(float)*4); // Recuperación de los cuaterniones desde el algoritmo
                                                  // de fusión

      // Generación de la matriz de rotación
      rotMatrix(0,0) = q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3];
      rotMatrix(0,1) = 2 * (q[1] * q[2] + q[0] * q[3]);
      rotMatrix(0,2) = 2 * (q[1] * q[3] - q[0] * q[2]);
      rotMatrix(1,0) = 2 * (q[1] * q[2] - q[0] * q[3]);
      rotMatrix(1,1) = q[0] * q[0] - q[1] * q[1] + q[2] * q[2] - q[3] * q[3];
```

```
    rotMatrix(1,2) = 2 * (q[2] * q[3] + q[0] * q[1]);
    rotMatrix(2,0) = 2 * (q[1] * q[3] + q[0] * q[2]);
    rotMatrix(2,1) = 2 * (q[2] * q[3] - q[0] * q[1]);
    rotMatrix(2,2) = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3];

    rotMatrixInv = rotMatrix.Inverse();    // Inversa de la matriz de rortación

    g[0] = 2 * (q[1] * q[3] - q[0] * q[2]);   // Eliminación gravedad
    g[1] = 2 * (q[0] * q[1] + q[2] * q[3]);
    g[2] = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3];

    aT_x = 9.80665 * (ax_buffer[n_buffer]-g[0]);  // Se obtiene el valor real de la aceleración
    aT_y = 9.80665 * (ay_buffer[n_buffer]-g[1]);  // tras multiplicar las 'g' por 9.80665
    aT_z = 9.80665 * (az_buffer[n_buffer]+g[2]);

    accBodyVector(0) = aT_x;      // Obtención de la aceleración real en la IMU
    accBodyVector(1) = aT_y;
    accBodyVector(2) = -aT_z;

    accNEDVector = rotMatrixInv * accBodyVector; // Se multiplica las aceleraciones reales por la matriz
          // de rotación para obtener las aceleraciones en nuestro sistema de referencia

    ax_buffer[n_buffer] = accNEDVector(0);
    ay_buffer[n_buffer] = accNEDVector(1);
    az_buffer[n_buffer] = accNEDVector(2);

    ax_buffer[n_buffer] = adcFilter_x.filter(ax_buffer[n_buffer]);  // Aceleracion filtrada para eliminar
    ay_buffer[n_buffer] = adcFilter_y.filter(ay_buffer[n_buffer]);   // el ruido
    az_buffer[n_buffer] = adcFilter_z.filter(az_buffer[n_buffer]);

}

for (n_buffer = 0; n_buffer < t_buffer; n_buffer++)      // Reutilizacion de variables
{
    gx_buffer[n_buffer] = 0;
    gy_buffer[n_buffer] = 0;
    gz_buffer[n_buffer] = 0;
    mx_buffer[n_buffer] = 0;
    my_buffer[n_buffer] = 0;
    mz_buffer[n_buffer] = 0;
}

for (n_buffer = 1; n_buffer < t_buffer; n_buffer++)   // Calculo de las velocidades
{
    gx_buffer[n_buffer] = gx_buffer[n_buffer-1] + ax_buffer[n_buffer] * deltat_buffer[n_buffer];
    gy_buffer[n_buffer] = gy_buffer[n_buffer-1] + ay_buffer[n_buffer] * deltat_buffer[n_buffer];
    gz_buffer[n_buffer] = gz_buffer[n_buffer-1] + az_buffer[n_buffer] * deltat_buffer[n_buffer];

    mx_buffer[n_buffer] = fhp_x.filterIn(gx_buffer[n_buffer]);  // Velocidad filtrada sin DC
    delay(1);
    my_buffer[n_buffer] = fhp_y.filterIn(gy_buffer[n_buffer]);
    delay(1);
    mz_buffer[n_buffer] = fhp_z.filterIn(gz_buffer[n_buffer]);
    delay(1);

    // Cuando la aceleración durante 'n_ceros' está dentro de la ventana '-limite_acel' 'limite_acel'
    // la velocidad en ese tramo vale 0. Esto es para cada uno de los ejes

    if ((ax_buffer[n_buffer] < limite_acel) && (ax_buffer[n_buffer] > (-1.0)*limite_acel))
    {
        n_ax++;
        if ((n_ax == n_ceros) && (n_buffer-n_ceros >= 0))
        {
            for (i=n_buffer-n_ceros; i<=n_buffer; i++)
            {
                mx_buffer[i] = 0;
                n_ax = 0;
            }
        }
    }
    else
        n_ax = 0;

    if (ay_buffer[n_buffer] < limite_acel && ay_buffer[n_buffer] > (-1.0)*limite_acel)
    {
        n_ay++;
        if ((n_ay == n_ceros) && (n_buffer-n_ceros >= 0))
        {
            for (i=n_buffer-n_ceros; i<=n_buffer; i++)
            {
                my_buffer[i] = 0;
                n_ay = 0;
            }
        }
    }
    else
        n_ay = 0;

    if (az_buffer[n_buffer] < limite_acel && az_buffer[n_buffer] > (-1.0)*limite_acel)
    {
        n_az++;
        if (n_az == n_ceros && n_buffer-n_ceros >= 0)
        {
            for (i=n_buffer-n_ceros; i<=n_buffer; i++)
            {
                mz_buffer[i] = 0;
                n_az = 0;
            }
        }
```

```
      }
      else
         n_az = 0;

   }

   for (n_buffer = 0; n_buffer < t_buffer; n_buffer++)   // Cálculo de la velocidad
   {
      velocidad[n_buffer] = sqrt(pow(mx_buffer[n_buffer],2) + pow(my_buffer[n_buffer],2) + pow(mz_buffer[n_buffer],2));
      gx_buffer[n_buffer] = 0;
      gy_buffer[n_buffer] = 0;
      gz_buffer[n_buffer] = 0;
   }

   for (n_buffer = 1; n_buffer < t_buffer - n_ceros; n_buffer++)   // Calculo de las posiciones
   {
      gx_buffer[n_buffer] = gx_buffer[n_buffer-1] + mx_buffer[n_buffer] * deltat_buffer[n_buffer];
      gy_buffer[n_buffer] = gy_buffer[n_buffer-1] + my_buffer[n_buffer] * deltat_buffer[n_buffer];
      gz_buffer[n_buffer] = gz_buffer[n_buffer-1] + mz_buffer[n_buffer] * deltat_buffer[n_buffer];
   }

   fase = 4;
   mensaje_mostrado = false;
   break;

case 4:      // Fase 4: Envio de la informacion al telefono movil mediante BLE
   if (mensaje_mostrado == false)
   {
      oled.clear();
      oled.setFont(Callibri15);
      oled.home();
      oled.println("  Esperando conexion  ");
      oled.setCursor(1,2);
      oled.println("         Bluetooth    ");
      mensaje_mostrado = true;
      n_buffer = 1;
   }
            // Comienzo de transmision de datos por BLE

   if ( central )    // Si está conectado al teléfono móvil......
   {
      if (mensaje_mostrado_2 == false)
      {
         oled.clear();
         oled.setFont(Callibri15);
         oled.home();
         oled.println("    - Enviando datos -  ");
         mensaje_mostrado_2 = true;
      }

      String s;

      // Formación de la trama a enviar a través del BLE. Esta trama está formada por la aceleración
      // en los 3 ejes y la velocidad. El tamaño es de 20 bytes.

      s=convertFloatToString(gx_buffer[n_buffer]);
      s=s+convertFloatToString(gy_buffer[n_buffer]);
      s=s+convertFloatToString(gz_buffer[n_buffer]);
      s=s+convertFloatToString(velocidad[n_buffer]);

      s.toCharArray(buf,21);
      buffCharacteristic.writeValue(buf);    // Envío de la trama
      delay(10);

      n_buffer ++;
   }

           if (!central && n_buffer != 1)    // Si se desconecta el móvil o se han enviado todo los datos...
           {
              fase = 5;
              mensaje_mostrado = false;
           }
   break;

case 5:      // Fase 5: Estado final de reposo
   if (mensaje_mostrado == false)
   {
      oled.clear();
      oled.setFont(Callibri15);
      oled.home();
      oled.println(" Transmision finalizada  ");
      mensaje_mostrado = true;
   }

   if (digitalRead(D5) == true)  // Si se pulsa el boton derecho, el programa vuelve a comenzar
   {
      fase = 0;                  // Retorno a la fase inicial
      mensaje_mostrado = false;
      n_buffer = 0;
   }

   break;

default:
   break;
}
}
```

7.3. Anexo III. Programa en el teléfono móvil

Este es el programa escrito en Python que va cargado en el teléfono móvil y que recibe la información de la placa Arduino Nano 33 BLE para su posterior representación.

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

# Core Bluetooth module
import cb

# iOS UI module
import ui

# Time
import time

import struct
import numpy as np

size_bufferv = 2000                     # Tamaño del buffer de recepción
pos_x = np.tile(0.0, size_bufferv)      # Array para almacenar las posiciones en X
pos_y = np.tile(0.0, size_bufferv)      # Array para almacenar las posiciones en Y
pos_z = np.tile(0.0, size_bufferv)      # Array para almacenar las posiciones en Z
velocidad = np.tile(0.0, size_bufferv)  # Array para almacenar los valores de la velocidad
index_x = 0
index_y = 0
index_z = 0
fin = False

def set_axes_equal(ax):     # Función que permite que los 3 ejes de la gráfica 3D tengan
                            # la misma escala

   x_limits = ax.get_xlim3d()
   y_limits = ax.get_ylim3d()
   z_limits = ax.get_zlim3d()

   x_range = abs(x_limits[1] - x_limits[0])
   x_middle = np.mean(x_limits)
   y_range = abs(y_limits[1] - y_limits[0])
   y_middle = np.mean(y_limits)
   z_range = abs(z_limits[1] - z_limits[0])
   z_middle = np.mean(z_limits)

   plot_radius = 0.5*max([x_range, y_range, z_range])

   ax.set_xlim3d([x_middle - plot_radius, x_middle + plot_radius])
   ax.set_ylim3d([y_middle - plot_radius, y_middle + plot_radius])
   ax.set_zlim3d([z_middle - plot_radius, z_middle + plot_radius])


# Delegate handles all BLE events
class MyCentralManagerDelegate(object):
   def __init__(self):
      self.peripheral = None
      self.characteristic = None
      self.button = None

   # Se busca el periférico llamado 'Arduino'
   def did_discover_peripheral(self, p):
      if p.name == 'Arduino' and not self.peripheral:
      print('Descubierto periférico ' + p.name)
      self.peripheral = p
      cb.connect_peripheral(self.peripheral)

   # Conectado
   def did_connect_peripheral(self, p):
      print('Conectado al periférico ' + p.name)
      p.discover_services()

   # Búsqueda de servicios
   def did_discover_services(self, p, error):
      for s in p.services:
         if s.uuid == '1A3AC132-31ED-758C-BC52-54A61958EF82':    # Se busca la UUID del servicio del Arduino
            print('Servicio encontrado')
            p.discover_characteristics(s)

   # Búsqueda de caracteristicas
   def did_discover_characteristics(self, s, error):
      for c in s.characteristics:
         if c.uuid == '1A3AC132-31ED-758C-BC52-54A61958EF83':  # Se busca la UUID de la caracteristica del Arduino
            print('Caracteristica encontrada')
            self.characteristic = c
                self.peripheral.set_notify_value(c, True)

   # Actualización de los valores que se van recibiendo a través de Bluetooth
   def did_update_value(self, c, error):
      global index_x
      global index_y
      global index_z
```

```
        global fin

    # La trama recibida de 20 bytes se divide en las variables de posición y velocidad
    pos_x[index_x] = c.value[:5]             # Parseo de la posición en X
    float(pos_x[index_x])                   # Se convierte a float la cadena recibida
    pos_y[index_x]= c.value[5:10]            # Parseo de la posición en X
    float(pos_y[index_x])                   # Se convierte a float la cadena recibida
    pos_z[index_x] = c.value[10:15]          # Parseo de la posición en X
    float(pos_z[index_x])                    # Se convierte a float la cadena recibida
    velocidad[index_x] = c.value((3]:20]     # Parseo de la posición en X
    float(velocidad[index_x])                # Se convierte a float la cadena recibida

    index_x = index_x + 1

    if index_x == size_bufferv:              # La transmisión finaliza cuando se reciben todos los datos
        cb.reset()

# Inicialiación del programa
cb.set_central_delegate(MyCentralManagerDelegate())
cb.scan_for_peripherals()

while True:
    if index_x == size_bufferv:
        pos_ini = 1
        pos_fin = size_bufferv

        plot1 = plt.figure(1)                # Creación de la primera Figura en 3D
        ax = plot1.gca(projection='3d')
        ax.plot(pos_x[pos_ini:pos_fin], pos_y[pos_ini:pos_fin], pos_z[pos_ini:pos_fin], label = 'Vista 3D')
        ax.set_title("Trayectoria 3D")
        ax.set_xlabel('X [m]')
        ax.set_ylaber('Y [m]')
        ax.set_zlabel('Z [m]')

        set_axes_equal(ax)

        for i in range(0, 360, 45):                    # Rotación y representación de la trayectoria en la reprentación 3D
            ax.view_init(None, i)        # Cada rotación es de 45º
            plt.show()

        fig, ax = plt.subplots()        # Creación de la Figura para representar la velocidad
        ax.plot(velocidad[pos_ini:pos_fin], label = 'Velocidad')
        ax.set_title("Velocidad desplazamiento")
        ax.set_xlabel('Número de muestra')
        ax.set_ylabel('Velocidad [m/s]')
        plt.show()

        fig, ax = plt.subplots()        # Creación de la Figura para representar el plano XY
        plt.gca().set_aspect('equal')
        ax.plot(pos_x,pos_y, label = 'XY')
        ax.set_title("Desplazamiento (vista superior)")
        ax.set_xlabel('Distancia X [m]')
        ax.set_ylabel('Distancia Y [m]')


        plt.show()

        fig, ax = plt.subplots()        # Creación de la Figura para representar el plano XZ
        plt.gca().set_aspect('equal')
        ax.plot(pos_x,pos_z, label = 'XZ')
        ax.set_title("Desplazamiento (vista lateral)")
        ax.set_xlabel('Distancia X [m]')
        ax.set_ylabel('Distancia Y [m]')

        plt.show()

        cb.reset()
        index_x = 0
```

7.4. Anexo IV. Proceso de calibración del magnetómetro.

En el apartado 1.2.1 se expuso la importancia de una correcta calibración de los sensores, en especial del magnetómetro.

Para llevar a cabo este proceso, donde se obtendrán los parámetros de calibración del magnetómetro, se ha empleado el programa ejemplo "DIY_Calibration_Magnetometer" [35] incluído en la librería "Arduino_LSM9DS1" [36].

Al ejecutar este programa y abrir el monitor serie, aparece una ventana con las instrucciones a seguir y con los parámetros que se pueden modificar antes de comenzar el proceso de calibración (Figura 120).

```
Calibrate Magnetometer
During measurement each of the sensor XYZ axes must be aligned in both directions with the Earth's magnetic field.
This takes about 10 seconds, if you know the local direction of the magnetic field lines. If you don't, it will take
several minutes, as you have to twist the board around, aiming every axis in every direction until the min and max
values no longer change. Info about the Earthmagnetic field https://en.wikipedia.org/wiki/Earth's_magnetic_field
E.g. in my case (Northern hemisphere)declination=0°, inclination=67°, means the aiming direction is South and a
rather steep 67° above the horizon.
The magnetic field measurement will be heavily disturbed by your set-up, so an "in-situ" calibration is advised.

 (F) Full Scale setting 0 = ±400 µT
 (R) Output Data Rate (ODR) setting 8 = 414Hz (actual value)
 (L) Local intensity of Earth magnetic field  43.54 µT  Change into your local value.
 (C) Calibrate Magnetometer, Twist board around to find min-max values or aim along earth mag field,  press enter to stop

  // Magnetometer code
  IMU.setMagnetFS(0);
  IMU.setMagnetODR(8);
  IMU.setMagnetOffset(0.000000, 0.000000, 0.000000);
  IMU.setMagnetSlope (1.000000, 1.000000, 1.000000);
```

Figura 120. Instrucciones para la calibración del magnetómetro.

El primer parámetro es el fondo de escala, que en nuestro caso es de ±800 µT (opción 1). El segundo parámetro es la frecuencia de salida del sensor. En el programa se indica que la frecuencia es de 414 Hz (opción 8). El último parámetro es la intensidad del campo magnético terrestre en la localidad donde se realizará la calibración. Para determinar este valor se ha empleado la calculadora de campo magnético disponible en la página web del "Centro Nacional para la Información Medioambiental" [37] (NOAA de sus siglas en inglés). En mi caso particular, la intensidad del campo magnético terrestre en Almería es de 43.5475 µT (Figura 121).

Magnetic Field

Model Used:	WMM-2020
Latitude:	36° 50' 30" N
Longitude:	2° 27' 49" W
Elevation:	0.0 km Mean Sea Level

Date	Declination (+ E | - W)	Inclination (+ D | - U)	Horizontal Intensity	North Comp (+ N | - S)	East Comp (+ E | - W)	Vertical Comp (+ D | - U)	Total Field
2021-08-21	0° 18' 49"	50° 34' 52"	27,652.0 nT	27,651.6 nT	151.3 nT	33,641.5 nT	43,547.5 nT
Change/year	0° 9' 3"/yr	-0° 0' 19"/yr	26.2 nT/yr	25.8 nT/yr	73.0 nT/yr	25.6 nT/yr	36.5 nT/yr
Uncertainty	0° 20'	0° 13'	128 nT	131 nT	94 nT	157 nT	145 nT

Figura 121. Campo magnético en Almería. Fuente: www.ngdc.noaa.gov

Una vez configurados los parámetros comenzamos con la calibración. El proceso consiste en girar sobre sí mismo el dispositivo hasta que los valores registrados y mostrados en pantalla no varíen (Figura 122).

```
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
Xmin = 13.70  Xmax = 14.88 Ymin = 12.91  Ymax = 14.28 Zmin = -19.72  Zmax = -16.99
```

Figura 122. Valores de ejemplo durante la calibración

Cuando pasado un tiempo no se aprecien modificaciones en las medidas, se puede detener el proceso, apareciendo a continuación los valores de la calibración (Figura 123).

```
// Magnetometer code
IMU.setMagnetFS(1);
IMU.setMagnetODR(8);
IMU.setMagnetOffset(-2.397461, 2.932129, 24.388428);
IMU.setMagnetSlope (0.966864, 0.963157, 0.971843);
```

Figura 123. Valores obtenidos para la calibración

Estos son los valores que se emplearán en el programa y que permitirán reducir los errores de medida del magnetómetro debidos al hierro blando y al hierro duro.

7.4.1. Verificación de la calibración.

Con el fin de comprobar si la calibración se ha realizado correctamente se emplea un programa que recoge las mediciones en cada eje del magnetómetro y las muestra a través del monitor. Este programa es un ejemplo de la librería "Arduino_LSM9DS1" y se llama "SimpleMagnetometer" [38]. Una vez obtenidas estas mediciones se representan gráficamente en Excel. Mientras se está ejecutando el programa es necesario rotar suavemente el dispositivo sobre sí mismo.

En primer lugar, se comprueban las mediciones sin los parámetros de calibración.

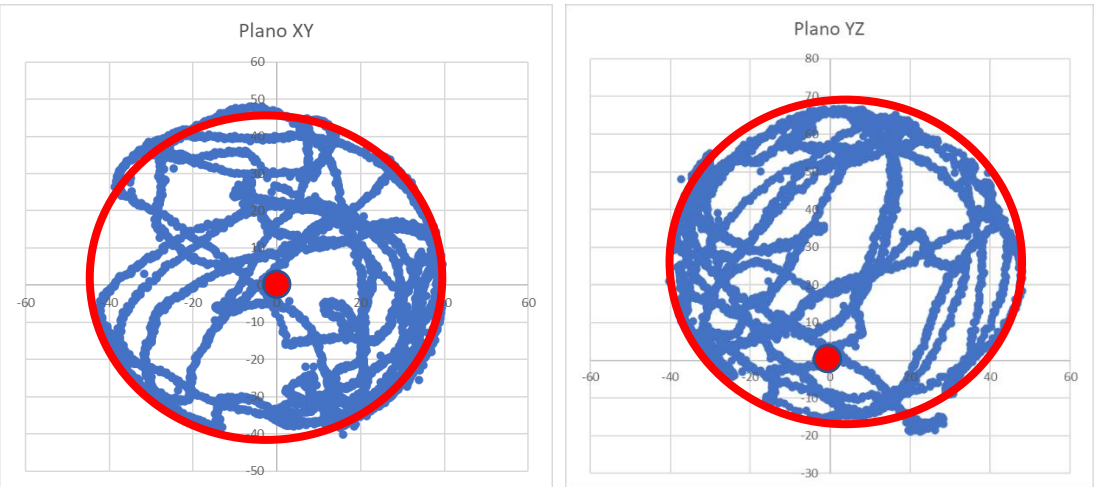


Figura 124. Mediciones sin calibración

Se puede observar en la Figura 124 cómo aparecen en las medidas del plano YZ una distorsión provocada por hierro duro, ya que la circunferencia formada por las mediciones tiene su centro desplazado del punto (0,0). Esta distorsión por hierro duro es debida a los componentes ubicados en la carcasa junta a la IMU (batería, pulsadores, controlador de carga y pantalla)

A continuación, se incorpora al programa empleado anteriormente los parámetros obtenidos en el proceso de calibración.

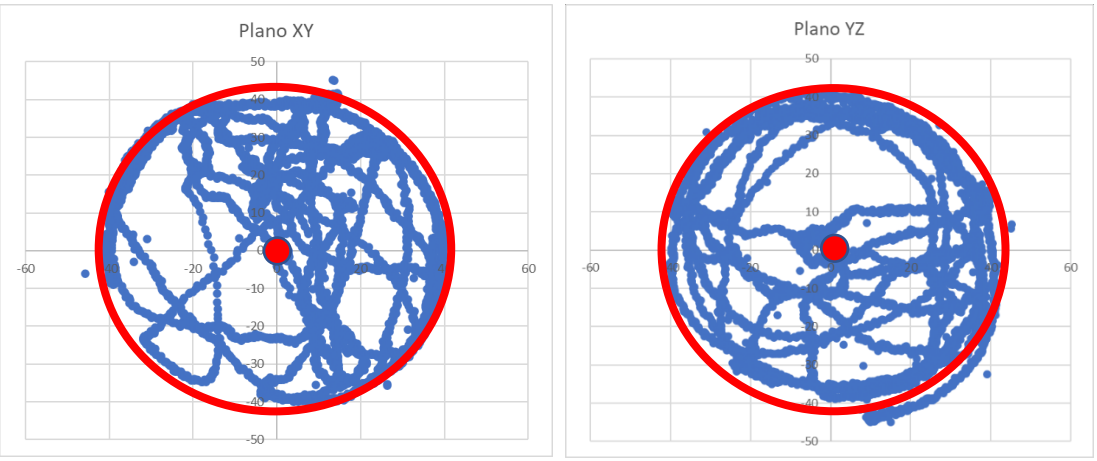


Figura 125. Mediciones con calibración

Se observa en la Figura 125 como las distorsiones que aparecían al ejecutar el programa sin ningún tipo de calibración han desaparecido, por lo que se puede determinar que la calibración del magnetómetro se ha realizado correctamente.

7.5. Anexo V. Proceso de calibración del acelerómetro.

El proceso de calibración del acelerómetro se realiza empleando el programa de ejemplo "DIY_Calibration_Accelerometer" [39] incluido en la librería "Arduino_LSM9DS1".

Al ejecutar este programa y abrir el monitor serie, aparece una ventana con las instrucciones a seguir y con los parámetros que se pueden modificar antes de comenzar el proceso de calibración (Figura 126).

```
Calibrate Accelerometer Offset and Slope
Before calibrating choose the Full Scale (FS) setting and Output Data Rate (ODR). The accelerometer and the
gyroscope share their ODR, so the setting here must be the same as in the DIY_Calibration_Gyroscope sketch.
Place the board on a horizontal surface with one of its axes vertical and hit enter to start a calibration
measurement. Each of the axes must be measured pointing up and pointing down, so a total of 6 measurements.
The program recognises which axis is vertical and shows which were measured successfully. If the angle is to
far oblique the measurement is not valid.

(enter)  Start a calibration measurement.
  (N)    Number of calibration samples 1000
  (F)    Full Scale setting 3 = 8g
  (R)    Output Data Rate (ODR) setting 5 = 464Hz (actual value)
Measured status of axis
X+ = not done Y+ = not done Z+ = not done
X- = not done Y- = not done Z- = not done

  // Accelerometer code
  IMU.setAccelFS(3);
  IMU.setAccelODR(5);
  IMU.setAccelOffset(0.000000, 0.000000, 0.000000);
  IMU.setAccelSlope (1.000000, 1.000000, 1.000000);
```

Figura 126. Procedimiento para la calibración del magnetómetro.

Las instrucciones a seguir para completar la calibración son:

- Colocar un lado del dispositivo sobre una superficie plana.
- Presionar 'Enter' y esperar que termine la calibración de ese lado. Si la calibración es correcta, se mostrará el mensaje "(-OK-)" en el eje del acelerómetro correspondiente a esa cara.
- Repetir el proceso hasta que se haya realizado la calibración de los seis lados y aparezcan todos los ejes 'OK' (Figura 127).

```
X+ = ( -OK- ) Y+ = ( -OK- ) Z+ = ( -OK- )
X- = ( -OK- ) Y- = ( -OK- ) Z- = ( -OK- )
```

Figura 127. Todos los ejes calibrados.

Una vez calibrado los seis lados se muestran los valores de *offset* y *slope* que se incluirán en el programa (Figura 128).

```
// Accelerometer code
IMU.setAccelFS(3);
IMU.setAccelODR(5);
IMU.setAccelOffset(-0.024975, 0.002536, -0.043924);
IMU.setAccelSlope (0.994798, 0.992403, 1.000054);
```

Figura 128. Resultado de la calibración.

7.6. Anexo VI. Proceso de calibración del giroscopio.

El proceso de calibración del giroscopio se realiza empleando el programa de ejemplo "DIY_Calibration_Gyroscope" [40] incluido en la librería "Arduino_LSM9DS1".

Al ejecutar este programa y abrir el monitor serie, aparece una ventana con las instrucciones a seguir y con los parámetros que se pueden modificar antes de comenzar el proceso de calibración.

La calibración del giroscopio se realiza en dos partes:

1. **Obtención del *offset*** (Figura 129)**.** Hay que dejar el dispositivo inmóvil sobre una superficie plana y pulsar la letra 'O'. El programa mostrará el valor del *offset.*

```
Step 1       CALIBRATE GYROSCOPE OFFSET
First choose the sample frequency (ODR) and the Full Scale value. The accelerometer and the gyroscope
share their ODR, so the setting here must be the same as in the DIY_Calibration_Gyroscope sketch.
This is far more important for the Gyroscope than for the accelerometer.
Next enter "O" to start the gyroscope offset measurement.
During this offset measurement the sensor must be kept still.

 (F) Full Scale setting 2 = 1000°/s
 (R) Output Data Rate (ODR) setting 5 = 456Hz (actual value)
 (N) Number of calibration samples 1000
 (O) Calibrate Offset (keep board still during measurement)
```

Figura 129. Obtención del offset

2. **Obtención del *slope*** (Figura 130)**.** Para la obtención de este parámetro hay que apoyar un lado del dispositivo sobre una superficie plana, pulsar la letra 'C', rotar el dispositivo 180º y pulsar "Enter". Si la calibración se ha realizado correctamente se mostrará el valor del *slope* para ese eje. De lo contrario, un mensaje indicará que es necesario repetir el proceso.

```
Step 2       CALIBRATE GYROSCOPE SLOPE
During a slope calibration the sensor must be rotated calmly about one axis over a known angle.
Change the angle to your convenience. A larger angle is more accurate, but more difficult to do.
The rotation must be pure, without much rotation about the other two axes. It can be done by hand.
Keeping the board on a flat surface with the rotation axis vertical while turning is good enough.
For an accurate result you can start and end with its side pushed against a non moving object.
When you're done turning the sensor, press (Enter) to stop measuring. Each of the axes X,Y and Z
must be measured. The program automatically detects which.

 (A) Change the measuring angle to turn the board
 (C) Calibrate Slope ==>> Turn the board over 180° and press enter when finished
 (F) Full Scale setting 2 = 1000°/s
 (R) Output Data Rate (ODR) setting 5 = 456Hz (actual value)
 (N) Number of calibration samples 1000
 (O) Calibrate Offset (keep board still during measurement)
```

Figura 130. Obtención del slope

Cuando se han completado ambos pasos tenemos los parámetros del *offset* y *slope* para ser utilizados en el programa (Figura 131).

```
IMU.setGyroFS(2);
IMU.setGyroODR(5);
IMU.setGyroOffset (0.734406, -0.679108, 0.275238);
IMU.setGyroSlope (1.000000, 1.000000, 1.000000);
```

Figura 131. Parámetros de calibración del giroscopio.

8. BIBLIOGRAFÍA

[1] Bermejo Frutos, J., Manuel Palao, J., López Elveria, J.L. (2012). Fundamentos biomecánicos del lanzamiento de disco. Parte I: técnica de lanzamiento. EFDeportes.com, Revista Digital. Buenos Aires, Año 15, Nº 166.

[2] de Hegedüs, J. (2012). Historia de los records mundiales de lanzamiento del disco, varones. EFDeportes.com, Revista Digital. Buenos Aires - Año 17 - Nº 170.

[3] Bartlett, R. (1991). The biomechanics of the discus throw: A review.

[4] Leibson, S. IMU para obtener una ubicación precisa: Parte 2: cómo utilizar el software de IMU para obtener una mayor precisión. Recuperado de: El software de IMU interpreta los datos de ubicación del sensor | DigiKey

[5] Beavers, I. The Case of the Misguided Gyro. Recuperado de: The Case of the Misguided Gyro | Analog Devices

[6] Hrisko, J. (2021). Gyroscope and Accelerometer Calibration with Raspberry Pi. Recuperado de: Gyroscope and Accelerometer Calibration with Raspberry Pi — Maker Portal (makersportal.com)

[7] Hard & Soft Iron Correction for Magnetometer Measurements. Recuperado de: FAQ: Hard & Soft Iron Correction for Magnetometer Measurements - Documents - MEMS Inertial Sensors - EngineerZone (analog.com)

[8] Hrisko, J. (2021). Calibration of a Magnetometer with Raspberry Pi. Recuperado de: Calibration of a Magnetometer with Raspberry Pi — Maker Portal (makersportal.com)

[9] Chérigo, C., Rodríguez, H. (2017). Evaluación de algoritmos de fusión de datos para estimación de la orientación de vehículos aéreos no tripulados.

[10] Quaternions – Solving Gimbal Lock. Recuperado de: Quaternions - Solving Gimbal Lock - Zaragoza MakerSpace

[11] Barrientos, A., Peñín, L.F., Balaguer, C., Aracil, R. (2007). Fundamentos de robótica. Segunda Edición. Mc Graw Hill.

[12] Kong, X. (2002). INS algorithm using quaternion model for low cost IMU. University of Technology, Sydeny, Australia. Elsevier.

[13] Olivares Garcés, D. (2018). Estudio e implementación de algoritmos para la estimación de la posición mediante sistemas inerciales con arduino. Universidad Politécnica de Valencia.

[14] Tutorial mpu6050, acelerómetro y giroscopio. Recuperado de: Tutorial MPU6050, Acelerómetro y Giroscopio (naylampmechatronics.com)

[15] H. Yin, H. Guo, X. Deng. 2020. Pedestrian Dead Reckoning Indoor Positioning with Step detection Based on foot-mounted IMU Nanchang University, China

[16] High pass filtering accelerometer data on Arduino. Recuperado de: c++ - High pass filtering accelerometer data on Arduino - Arduino Stack Exchange

[17] Introduction to Bluetooth Low Energy. Recuperado de: Introduction | Introduction to Bluetooth Low Energy | Adafruit Learning System

[18] Maximizing BLE Throughput Part 2: Use Larger ATT MTU. Recuperado de: Maximizing BLE Throughput Part 2: Use Larger ATT MTU | Punch Through

[19] Sánchez Llorach, D. (2020). Wearable de medida de rendimiento en running. Universidad Politécnica de Valencia.

[20] STEVAL-WESU1. Recuperado de: STEVAL-WESU1 - Wearable sensor unit reference design for fast time to market - STMicroelectronics

[21] Arándiga Martínez, A. L. Prototipo de dispositivo de medida de rendimiento en deportes de contacto basado en un acelerómetro triaxial y comunicación a dispositivo móvil. Universidad Politécnica de Valencia.

[22] CC2541 SensorTag Development Kit. Recuperado de: CC2541DK-SENSOR Development kit | TI.com

[23] Alexander, H., Kristof V.L. (2018). Using Wrist-Worn Activity Recognition for Basketball Game Analysis. University of Siegen.

[24] LSM9DS1 Datasheet. iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer (st.com)

[25] Getting started with the Arduino Nano 33 BLE Sense. Recuperado de: Getting started with the Arduino Nano 33 BLE Sense | Arduino

[26] BAT48 Datasheet. Small signal Schottky diode (st.com)

[27] BC847 Datasheet. 45 V, 100 mA NPN general-purpose transistors (mouser.es)

[28] Fabio. (2011). Simple gravity compensation for 9 DOM IMUs. Simple gravity compensation for 9 DOM IMUs | Varesano.net

[29] Verrastro, C., Gómez, J.C., Folino, P., Alberino, S. (2009). Control PID con Filtro Dinámico de Promedios Móviles Ponderados Exponencialmente (dEWMA-PID). XIII Reunión de Trabajo en Procesamiento de la Información y Control.

[30] Librería *EWMA* Arduino. Recuperado de: GitHub - jonnieZG/EWMA: Exponentially Weighted Moving Average Filter

[31] Librería *Filters* Arduino. Recuperado de: GitHub - JonHub/Filters: A realtime digital signal processing (DSP) library for Arduino

[32] Comprensión del UUID de Bluetooth de Android. Recuperado de: Comprensión del UUID de Bluetooth de Android - programador clic (programmerclick.com)

[33] Connecting to Bluetooth LE Peripherals. Recuperado de: cb — Connecting to Bluetooth LE Peripherals — Python 3.6.1 documentation (omz-software.com)

[34] Chapter 4. Visualization with Matplotlib. Recuperado de: 4. Visualization with Matplotlib - Python Data Science Handbook [Book] (oreilly.com)

[35] Programa calibración magnetómetro: "DIY_Calibration_Magnetometer". Recuperado de: Arduino_LSM9DS1/DIY_Calibration_Magnetometer.ino at master · FemmeVerbeek/Arduino_LSM9DS1 · GitHub

[36] Librería *Arduino_LSM9DS1*. Recuperado de: GitHub - FemmeVerbeek/Arduino_LSM9DS1: LSM9DS1 Library for Arduino

[37] Magnetic Field Estimated Values. Recuperado de: NCEI Geomagnetic Calculators (noaa.gov)

[38] Programa chequeo magnetómetro: "SimpleMagnetometer". Recuperado de: Arduino_LSM9DS1/SimpleMagnetometer.ino at master · arduino-libraries/Arduino_LSM9DS1 · GitHub

[39] Programa calibración acelerómetro: "DIY_Calibration_Accelerometer". Recuperado de: Arduino_LSM9DS1/DIY_Calibration_Accelerometer.ino at master · FemmeVerbeek/Arduino_LSM9DS1 · GitHub

[40] Programa calibración giroscopio: "DIY_Calibration_Gyroscope". Recuperado de: Arduino_LSM9DS1/DIY_Calibration_Gyroscope.ino at master · FemmeVerbeek/Arduino_LSM9DS1 · GitHub