



UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
AERONÁUTICA Y DEL ESPACIO
GRADO EN INGENIERÍA AEROESPACIAL

TRABAJO FIN DE GRADO
Plataforma Giroscópica de Pruebas
para Sistemas de Orientación Inercial

AUTOR: Pablo RAMÍREZ ESCUDERO

ESPECIALIDAD: Navegación y Sistemas Aeroespaciales

TUTOR DEL TRABAJO: José Félix ALONSO ALARCÓN

Julio de 2021

Índice general

Resumen	v
Abstract	v
Lista de figuras	IX
Lista de tablas	XI
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	4
2. Tratamiento Matemático	5
2.1. Ejes de referencia	5
2.1.1. Ejes inerciales	5
2.1.2. Ejes del vehículo	6
2.1.3. Ejes cuerpo	6
2.2. Representación de la orientación	6
2.2.1. Ángulos de Euler	6
2.2.2. Cuaterniones	7
3. Unidades Inerciales	9
3.1. Acelerómetros	9
3.2. Giróscopos	11
3.3. Magnetómetros	13
4. Determinación de la Orientación	17
4.1. Detección de la gravedad	17

4.1.1. Filtro Paso Bajo	18
4.2. Medición del campo magnético	18
4.2.1. Calibración del magnetómetro	19
4.3. Integración del giróscopo	22
4.4. Fusión de sensores	23
4.4.1. Filtro Complementario	23
4.4.2. Filtro de Kalman	24
4.4.3. Filtro de Kalman Extendido	25
5. Montaje Experimental	29
5.1. Plataforma giroscópica	29
5.2. Componentes electrónicos	30
5.2.1. Arduino UNO	31
5.2.2. Servomotores	33
5.2.3. Sensores de ángulo AS5600	35
5.2.4. Multiplexor I2C TCA9548A	36
5.2.5. Arduino Nano BLE33 Sense	37
5.3. Diseño y fabricación	38
5.3.1. Diseño 3D	38
5.3.2. Impresión 3D	41
5.4. Construcción de la plataforma	43
5.4.1. Montaje de los componentes	43
5.4.2. Sistema eléctrico	47
5.4.3. Resultado final	48
5.5. Programación de la plataforma	50
5.5.1. Lectura de la unidad de medición inercial	50
5.5.2. Control de la plataforma	50
5.5.3. Adquisición de datos	52
5.6. Consideraciones	53
5.6.1. Error magnético	53
5.6.2. Vibraciones mecánicas	53
6. Análisis de los datos y resultados	55
6.1. Detección de la gravedad y el campo magnético	57

6.2. Integración del giróscopo	58
6.3. Filtro complementario	59
6.4. Filtro de Kalman extendido	60
6.5. Comparativa	61
7. Conclusiones y Trabajo Futuro	65
7.1. Conclusiones	65
7.2. Trabajo Futuro	66
Bibliografía	68
A. Planos de la plataforma	71
B. Diagrama eléctrico de la plataforma	77
C. Códigos Arduino	79
C.1. Código de control de la unidad inercial	79
C.2. Código de control de la plataforma	81
D. Códigos Matlab	85
D.1. Función del estimador de acelerómetro y magnetómetro	85
D.2. Función del estimador por integración del giróscopo	86
D.3. Función del filtro complementario	87
D.4. Función del filtro de Kalman extendido	89
D.5. Script de comparación de algoritmos	91
D.6. Función de conversión de ángulos de euler a cuaterniones	95
D.7. Función de conversión de cuaterniones a ángulos de euler	96
D.8. Función de calibración del magnetómetro	97
D.9. Script de adquisición de datos de la plataforma	98



UNIVERSIDAD
POLITÉCNICA
DE MADRID

ÍNDICE GENERAL



Resumen

Este trabajo se basa en la construcción de una plataforma para la evaluación de sistemas de orientación inercial. Las unidades de medida inercial (IMU) proporcionan datos sobre la aceleración lineal y velocidad angular, pero no pueden medir directamente la orientación. Para conocer la orientación del sistema se emplean algoritmos de procesamiento de datos. Estos algoritmos emplean las medidas de los sensores de la unidad inercial para obtener una estimación de la orientación del sistema. Dependiendo de los datos que combine un algoritmo y su capacidad para lidiar con el ruido se obtendrán mejores o peores estimaciones. Es por ello interesante disponer de algún método que permita comparar la eficacia de diferentes algoritmos. La plataforma de pruebas construida permite rotar la unidad inercial de forma controlada y generar un set de datos con el que poder comparar distintos algoritmos para la estimación de la orientación de forma cuantitativa. Se han implementado cuatro algoritmos para la comparativa, siendo el más relevante de ellos el filtro de Kalman extendido (EKF).

Abstract

This work is based on the construction of a platform for the evaluation of inertial orientation systems. Inertial measurement units (IMU) provide data of linear acceleration and angular velocity, but cannot directly measure the orientation. Data processing algorithms are used to determine the orientation of the system. These algorithms use the measurements from the sensors of the inertial unit to obtain an estimation for the orientation of the system. Depending on the data that an algorithm combines and its ability to deal with noise better or worse estimations will be obtained. It is therefore interesting to have a method to compare the performance of different algorithms. The test platform built allows to rotate the inertial unit in a controlled way and to generate a dataset with which to compare different algorithms for orientation estimation in a quantitative way. Four algorithms have been implemented for the comparison, the most relevant of them is the extended Kalman filter (EKF).



UNIVERSIDAD
POLITÉCNICA
DE MADRID

ÍNDICE GENERAL



Lista de figuras

1.1. Cohete V2	1
1.2. Acelerómetro giroscópico de un V2	1
1.3. Unidad Inercial del Apolo	2
1.4. Giróscopo láser	3
1.5. Giróscopo de fibra óptica	3
1.6. Unidad inercial MEMS	3
1.7. Prototipo de giróscopo cuántico	3
2.1. NED frame	5
2.2. Ejes cuerpo y ángulos de euler	6
3.1. Diagrama de un acelerómetro	10
3.2. Acelerómetro MEMS	11
3.3. Funcionamiento de un giróscopo CVG	11
3.4. Micrografía de un giróscopo MEMS	12
3.5. Efecto Hall	13
3.6. Mapa de declinación magnética de 2015	14
3.7. Mapa de inclinación magnética de 2015	15
4.1. Lecturas del acelerómetro con y sin filtrar	18
4.2. Errores soft-iron y hard-iron	20
4.3. Calibración del magnetómetro en 3D	21
4.4. Calibración del magnetómetro en el plano XY	22
5.1. Gimbal usado para estabilizar una cámara o smartphone	29
5.2. Plataforma para la calibración de unidades inerciales	30
5.3. Componentes de un servomotor	31

5.4. Arduino UNO	32
5.5. Arduino IDE	32
5.6. Diagrama de pines del Arduino UNO	33
5.7. Componentes de un servomotor	34
5.8. Señal PWM vs posición angular	34
5.9. Servomotores MG995 y TR-1160A	35
5.10. Sensores de ángulo AS5600	36
5.11. Separación entre el sensor AS5600 y el imán	36
5.12. Multiplexor I2C TCA9548A	37
5.13. Arduino Nano BLE 33 Sense	37
5.14. Captura de Fusion 360 con los componentes electrónicos	39
5.15. Captura de Fusion 360 con la plataforma diseñada	39
5.16. Sección XY de la plataforma giroscópica	40
5.17. Sección XZ de la plataforma giroscópica	40
5.18. Vista 3D de la plataforma giroscópica con las piezas rotadas	41
5.19. Ender 3 de Creality	42
5.20. Bobinas de filamento PLA utilizadas en impresión 3D	42
5.21. Preparación de la pieza base en Ultimaker Cura	43
5.22. Revisión del G-CODE generador en Ultimaker Cura	43
5.23. Paso 1 del montaje	44
5.24. Paso 2 del montaje	44
5.25. Paso 3 del montaje	45
5.26. Paso 4 del montaje	45
5.27. Paso 5 del montaje	46
5.28. Paso 6 del montaje	46
5.29. Fotografía del módulo de control	47
5.30. Diagrama eléctrico del circuito de alimentación	47
5.31. Fotografía de la plataforma	48
5.32. Ejes de la plataforma	49
5.33. Ejes de los sensores del Arduino Nano BLE 33 Sense	49
5.34. Captura del puerto serie de Arduino	52
5.35. Plataforma giroscópica orientada al norte	53

6.1. Datos adquiridos durante el ensayo lento	55
6.2. Datos adquiridos durante el ensayo rápido	56
6.3. Estimaciones del acelerómetro y el magnetómetro para el ensayo lento	57
6.4. Estimaciones del acelerómetro y el magnetómetro para el ensayo rápido	57
6.5. Estimaciones del giróscopo para el ensayo lento	58
6.6. Estimaciones del giróscopo para el ensayo rápido	58
6.7. Estimaciones del filtro complementario para el ensayo lento	59
6.8. Estimaciones del filtro complementario para el ensayo rápido	59
6.9. Estimaciones del filtro Kalman extendido para el ensayo lento	60
6.10. Estimaciones del filtro de Kalman extendido para el ensayo rápido	61
6.11. Errores en las estimaciones para el ensayo lento	61
6.12. Errores en las estimaciones para el ensayo rápido	62

LISTA DE FIGURAS

Lista de tablas

5.1. Características de los servomotores utilizados	35
5.2. Características del módulo inercial LSM9DS1	38
5.3. Estructura del paquete de datos enviado por la plataforma	51
6.1. Valores de RMSE para el ensayo lento	63
6.2. Valores de RMSE para el ensayo rápido	63

LISTA DE TABLAS

1. Introducción

1.1. Motivación

Los sistemas de orientación inercial están presentes en muchos aspectos del mundo actual. Se pueden encontrar en satélites, aviones, drones, teléfonos móviles, brazos robóticos e incluso en instrumental médico. Los sistemas de navegación inercial permiten conocer la posición, orientación y velocidad de un vehículo proporcionándoles, únicamente, la posición inicial. Estos sistemas miden el movimiento sin hacer uso de información externa, como señales de satélites GPS o referencias visuales del terreno. Esto los hace inmunes a las interferencias o a la pérdida de las comunicaciones, siendo de vital importancia para la industria aeronáutica y espacial.

Las primeras unidades inerciales datan de los años 40. Aparecieron durante la Segunda Guerra Mundial, para proporcionar el guiado a los primeros misiles balísticos, los V2 alemanes [1]. Estas unidades eran pesadas, costosas y de gran tamaño. Estaban formadas por componentes mecánicos como giróscopos que aprovechaban la conservación del momento angular y acelerómetros formados por una masa de prueba suspendida por muelles.



Figura 1.1: Cohete V2

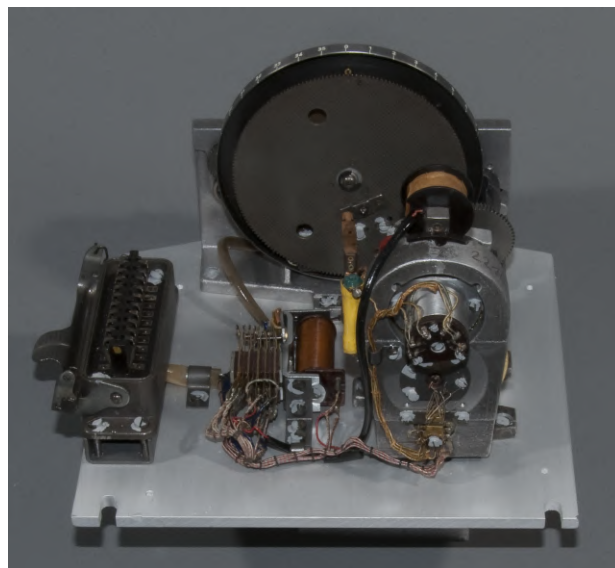


Figura 1.2: Acelerómetro giroscópico de un V2

En los años 60, la carrera espacial impulsó el desarrollo de los sistemas de navegación inercial. Los ordenadores aún no estaban suficientemente desarrollados como para poder integrar estos sistemas en aviones comerciales. Sin embargo, sí fueron integradas en las naves espaciales del

programa Apolo de la NASA, que llevaban una unidad inercial conectada a uno de los primeros computadores de vuelo de la historia.

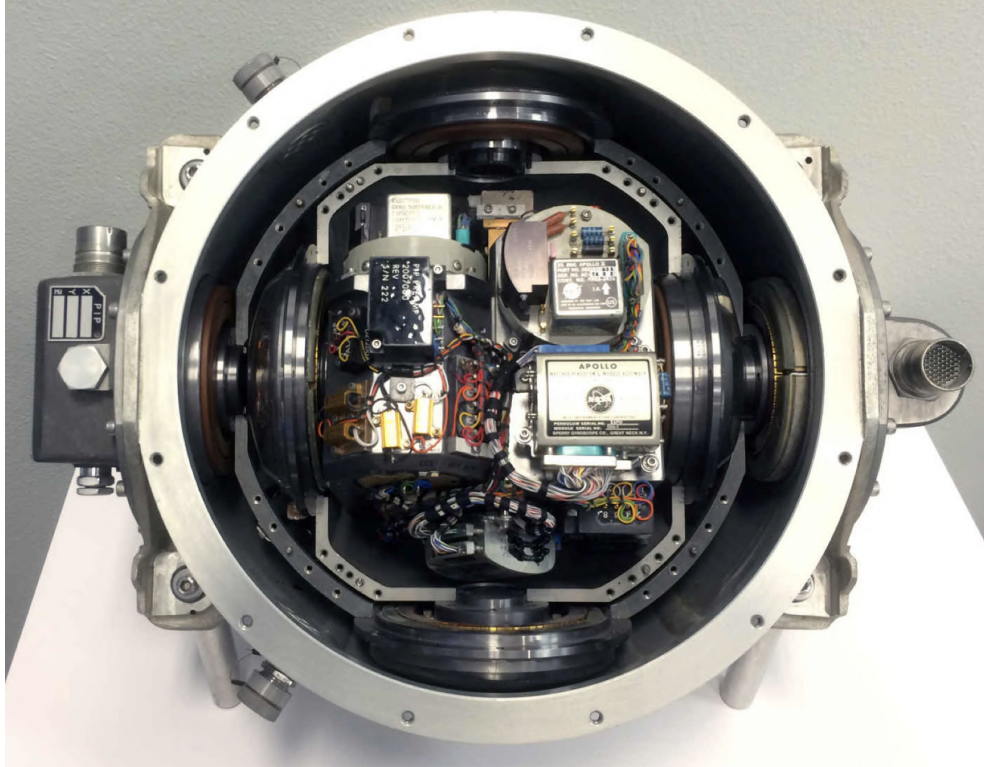


Figura 1.3: Unidad Inercial del Apolo

El uso comercial de las plataformas inerciales se popularizó en los años 70 [1]. Las anteriores unidades estaban formadas por una plataforma giro-estabilizada sobre la que se disponía un grupo de acelerómetros. En esta época, el desarrollo de las computadoras y el aumento en la precisión de estos sensores permitió el desarrollo de la plataforma strapdown. Una plataforma strapdown consiste en un grupo de acelerómetros y giróscopos fijados al cuerpo de un vehículo. Las lecturas de estos sensores son combinadas por una computadora de vuelo que, mediante la ejecución de un algoritmo, permite determinar el estado dinámico del vehículo.

En los años 80, aparecieron los giróscopos ópticos, aumentando la precisión y reduciendo el peso de estos sensores. Estos giroscopios se basan en el efecto Sagnac, interferencia generada por dos haces de luz que recorren distancias distintas por un camino circular, cuando este rota. Los primeros en aparecer fueron los giróscopos de anillo láser o RLG (del inglés, Ring Laser Gyroscope). La segunda generación de giróscopos ópticos vino de la mano de la fibra óptica con los giróscopos FOG (del inglés, Fiber Optic Gyroscope) y permitió reducir aún más su tamaño y precio.



Figura 1.4: Giróscopo láser



Figura 1.5: Giróscopo de fibra óptica

Los avances en sistemas MEMS (del inglés, Micro Electro-Mechanic Systems), junto al desarrollo de giróscopos vibratorios de efecto Coriolis y acelerómetros piezoeléctricos y capacitivos, ha permitido integrar las unidades inerciales dentro de un chip de silicio. Esto ha posibilitado su uso en dispositivos comerciales de pequeño tamaño y ha facilitado la llegada de los drones de uso recreativo, que comenzaron aparecer en 2010 y no han parado de desarrollarse desde entonces.

Actualmente, los últimos avances se están produciendo en el campo de la física cuántica. Destacan los giróscopos de resonancia nuclear magnética, basados en la medida del spin de los núcleos atómicos; y los giróscopos y acelerómetros cuánticos de átomo frío, que son potentes candidatos para la fabricación de sensores inerciales extremadamente precisos [1].

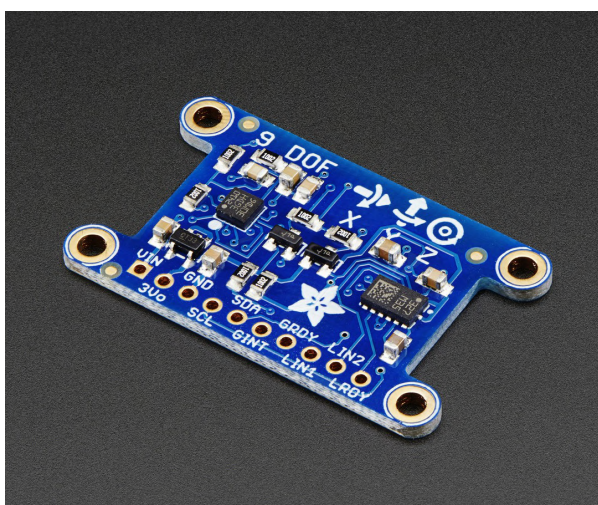


Figura 1.6: Unidad inercial MEMS

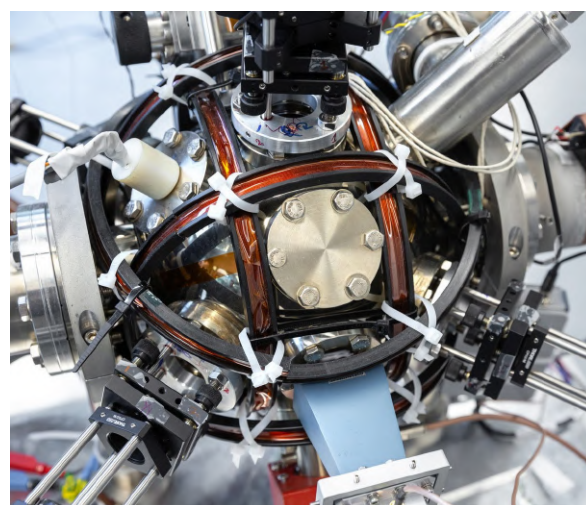


Figura 1.7: Prototipo de giróscopo cuántico

La aparición de las unidades inerciales MEMS permitió su uso en dispositivos de pequeño tamaño como drones, smartphones y estabilizadores para cámaras fotográficas. Sin embargo, estas unidades presentan un ruido elevado y tienen poca precisión, además de ser muy afectadas por la temperatura y las fuerzas exteriores.

Todo esto hace que las unidades inerciales MEMS necesiten mejores algoritmos estimadores que otros tipos de IMUs. Algoritmos que deben fusionar las medidas de varios sensores para compensar los diferentes errores a los que están sujetos. El desarrollo y mejora de los sistemas de orientación inercial es de gran interés para la industria aeronáutica, especialmente en el campo de los vehículos no tripulados o RPAS. Un sector de la industria en auge y con un gran crecimiento económico.

1.2. Objetivos

Este trabajo contiene dos objetivos principales: el diseño y construcción de una plataforma giroscópica para el estudio de sistemas de orientación inercial y la implementación y comparación de cuatro algoritmos estimadores de la orientación.

Para poder comparar cuantitativamente el desempeño de estos algoritmos, es necesario un valor de referencia de la orientación del sistema. Para ello se ha diseñado y construido una plataforma giroscópica que permite rotar la unidad inercial en sus tres grados de libertad.

La plataforma posee tres encoders magnéticos de posición angular que proporcionan la orientación real del sistema. De este modo, se puede generar un set de datos con las medidas de los sensores de la IMU y los valores reales de los ángulos de la plataforma. Las medidas tomadas de la IMU serán procesadas por los algoritmos implementados y se realizará una comparativa de las estimaciones obtenidas.

Los cuatro algoritmos de estimación estudiados son los siguientes:

- **Detección de la gravedad y del campo magnético:** emplea un acelerómetro y magnetómetro tri-axial para estimar la orientación. El acelerómetro determina la inclinación de la plataforma y el magnetómetro, compensado en inclinación, determina el rumbo magnético.
- **Integración del giróscopo:** emplea un giróscopo tri-axial para obtener la orientación por integración.
- **Filtro Complementario:** combina la estimación por detección de la gravedad y del campo magnético con la de la integración del giróscopo, realizando una media ponderada. Los pesos aplicados en esta ponderación tienen un valor constante.
- **Filtro Kalman:** actúa como un filtro complementario, pero computando una ganancia dinámica que depende del error instantáneo de cada sensor. Esto permite al filtro dar más peso al sensor que mejor se esté comportando en cada momento.

2. Tratamiento Matemático

Para el estudio de los sistemas inerciales es importante definir cómo se representa la orientación de un cuerpo y respecto a qué sistema de referencia está definida esta orientación. Este capítulo describe los sistemas de referencia, los métodos para representar la orientación y las transformaciones que se realizarán entre estos.

2.1. Ejes de referencia

En aeronáutica es común el uso de tres sistemas de referencia de coordenadas cartesianas para representar la orientación y posición de un cuerpo en sus seis grados de libertad. El primer sistema es denominado inercial S^i y se encuentra anclado al suelo. El segundo sistema, denominado ejes del vehículo S^v , permite definir la posición, pues es paralelo al inercial pero se encuentra desplazado respecto a este. El tercer sistema, llamado ejes cuerpo S^b , permite definir la orientación. Este sistema está rotado respecto a los ejes del vehículo, pero coincide en origen.

2.1.1. Ejes inerciales

El sistema de coordenadas inerciales S^i es un sistema coordenado fijado a la tierra cuyo origen suele ser el punto de inicio de misión del vehículo. El primer vector unitario i^i apunta al norte, el vector j^i apunta al este y el vector k^i apunta al centro de la tierra, hacia abajo.

Este sistema suele ser llamado NED, por sus siglas en inglés: North-East-Down [2].

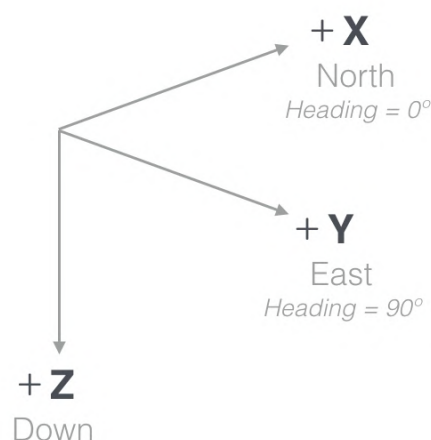


Figura 2.1: NED frame

2.1.2. Ejes del vehículo

Este sistema S^v posee su origen en el centro de masas del vehículo. Todos los ejes de este sistema son paralelos a los del inercial y también se denominan ejes NED. Se usa para representar la posición del vehículo respecto al sistema inercial S^i .

2.1.3. Ejes cuerpo

Este sistema S^b se encuentra anclado al vehículo, siendo su origen el centro de masas, al igual que S^v . El vector unitario i^b apunta hacia el morro de la aeronave, el vector j^b apunta hacia el extremo del ala derecha y el vector k^b apunta hacia la barriga del vehículo. Los ejes cuerpo se usan para definir la orientación del sistema respecto a los ejes del vehículo.

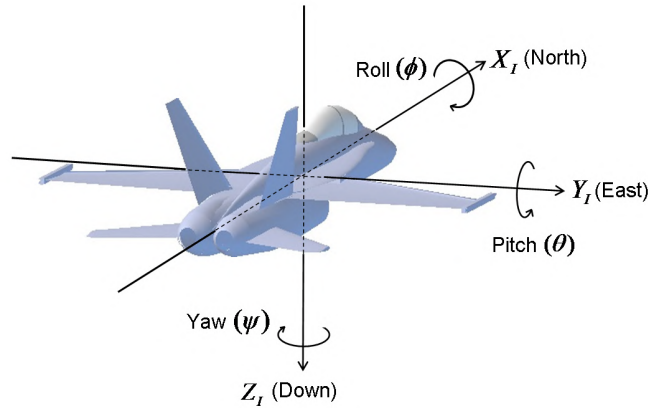


Figura 2.2: Ejes cuerpo y ángulos de euler

Es necesario definir otros dos sistemas, entre los ejes cuerpo S^b y los ejes S^v , para poder caracterizar la orientación. Estos ejes se denominan S^{v1} y S^{v2} . Su función es definir los pasos intermedios de rotación para cada ángulo de euler.

El sistema S^{v1} rota respecto al S^v , entorno al eje k^v , según el ángulo de guiñada ψ (o yaw, en inglés). A continuación, el sistema S^{v2} rota respecto al S^{v1} , entorno al eje k^{v1} , según el ángulo de cabeceo θ (o pitch, en inglés). Finalmente, el sistema S^b rota respecto al S^{v2} , entorno al eje i^{v2} , según el ángulo de balance o alabeo ϕ (o roll, en inglés) [2].

2.2. Representación de la orientación

Es importante definir el sistema de orientación a usar. En general, se suelen usar los tres ángulos de Euler o de Tait-Bryan, debido a su simplicidad y a que son intuitivos. Sin embargo, sufren de varias limitaciones como el problema de bloqueo del gimbal o que dependiendo de la secuencia de rotación, se obtienen orientaciones distintas.

2.2.1. Ángulos de Euler

Los ángulos clásicos de Euler son la guiñada (yaw) ψ , el cabeceo (pitch) θ y el alabeo (roll) ϕ . En aeronáutica, se emplea la secuencia yaw-pitch-roll para representar la orientación del vehículo.

Además, se establece el valor nulo de los ángulos de Euler para un vehículo en vuelo horizontal orientado al norte y unos rangos de valor de $\pm 90^\circ$ para el pitch θ , de $\pm 180^\circ$ para el roll ϕ y de $0^\circ - 360^\circ$ para el yaw ψ .

La matriz de rotación de los ejes S^v a los ejes S^{v1} y que define el ángulo de guiñada ψ es

$$R_v^{v1}(\psi) = \begin{pmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.1)$$

Para el ángulo de cabeceo θ , la matriz de transformación de S^{v1} a S^{v2} se expresa como

$$R_{v1}^{v2}(\theta) = \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix} \quad (2.2)$$

En el caso del ángulo de alabeo ϕ , la matriz de rotación entre el sistema S^{v2} y los ejes cuerpo S^b es

$$R_{v2}^b(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \quad (2.3)$$

Teniendo en cuenta la secuencia de rotación yaw-pitch-roll, la matriz de transformación del sistema S^v a ejes cuerpo S^b queda como

$$R_v^b(\phi, \theta, \psi) = R_{v2}^b(\phi) R_{v1}^{v2}(\theta) R_v^{v1}(\psi) = \begin{pmatrix} C\theta C\psi & C\theta S\psi & -S\theta \\ S\phi S\theta C\psi - C\phi S\psi & S\phi S\theta S\psi + C\phi C\psi & S\phi C\theta \\ C\phi S\theta C\psi + S\phi S\psi & C\phi S\theta S\psi - S\phi C\psi & C\phi C\theta \end{pmatrix} \quad (2.4)$$

donde $C\theta$ corresponde al $\cos \theta$ y $S\theta$ corresponde al $\sin \theta$.

Para realizar la transformación inversa, de ejes S^b a ejes S^v , bastaría con realizar su traspuesta $R_b^v = (R_v^b)^T$, debido a que las matrices de rotación son matrices ortogonales.

2.2.2. Cuaterniones

Un problema que presentan los ángulos de euler es el denominado bloqueo del gimbal. Cuando el ángulo de cabeceo (pitch) presenta un valor de $\pm 90^\circ$ el ángulo de guiñada (yaw) deja de estar definido. Para solucionar esto se emplean los cuaterniones (quaternions, en inglés).

Los cuaterniones añaden un cuarto parámetro para definir la orientación. Las ventajas que presentan son que no sufren de bloqueo del gimbal y que son computacionalmente más eficientes. Sin embargo, son más difíciles de visualizar y requieren de un tratamiento matemático aparte.

$$\mathbf{q} = (q_\omega, q_x, q_y, q_z) \quad (2.5)$$

Un cuaternión define un vector unitario $\mathbf{v} = (q_x, q_y, q_z)$ y una rotación sobre este q_ω . Sea Θ el giro definido sobre el vector $\mathbf{v}$, las componentes del cuaternión quedarían como

$$q_\omega = \cos \frac{\Theta}{2} \quad (2.6)$$

$$\mathbf{v} \sin \frac{\Theta}{2} = \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} \quad (2.7)$$

Se trabajará tanto con ángulos de euler como con cuaterniones. Es por ello necesario definir las transformaciones entre estas dos notaciones de la orientación.

La transformación de cuaterniones a ángulos de euler, utilizada en [2], es la siguiente:

$$\psi = \text{atan2}(2(q_\omega q_x + q_y q_z), (q_\omega^2 + q_z^2 - q_x^2 - q_y^2)) \quad (2.8)$$

$$\theta = \text{asin}(2(q_\omega q_y - q_x q_z)) \quad (2.9)$$

$$\phi = \text{atan2}(2(q_\omega q_z + q_x q_y), (q_\omega^2 + q_z^2 - q_x^2 - q_y^2)) \quad (2.10)$$

Donde $\text{atan2}(y, x)$ es la fórmula del arcotangente que evalúa el cuadrante y devuelve un resultado entre $-\pi$ y π .

La transformación de ángulos de euler a cuaterniones, según [2], es

$$q_\omega = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \cos \frac{\phi}{2} + \sin \frac{\psi}{2} \sin \frac{\theta}{2} \sin \frac{\phi}{2} \quad (2.11)$$

$$q_x = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2} - \sin \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2} \quad (2.12)$$

$$q_y = \cos \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2} + \sin \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2} \quad (2.13)$$

$$q_z = \sin \frac{\psi}{2} \cos \frac{\theta}{2} \cos \frac{\phi}{2} - \cos \frac{\psi}{2} \sin \frac{\theta}{2} \sin \frac{\phi}{2} \quad (2.14)$$

La matriz de rotación 2.4 se puede expresar en cuaterniones 2.15, utilizando la expresión de [3].

$$R_v^b(\mathbf{q}) = \begin{pmatrix} q_\omega^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y + q_z q_\omega) & 2(q_x q_z - q_y q_\omega) \\ 2(q_x q_y - q_z q_\omega) & q_\omega^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z + q_x q_\omega) \\ 2(q_x q_z + q_y q_\omega) & 2(q_y q_z - q_x q_\omega) & q_\omega^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix} \quad (2.15)$$

3. Unidades Inerciales

Las unidades de medida inercial o IMU (del inglés, Inertial Measurement Unit) son dispositivos electrónicos que miden la velocidad angular y las aceleraciones lineales usando una combinación de acelerómetros y giróscopos. La IMU es el principal componente de los sistemas de navegación inercial usados en aviones, satélites, misiles y barcos, entre otros.

Las IMUs son usadas para determinar la orientación y posición mediante el uso de algoritmos. Estos algoritmos emplean modelos físicos para realizar un ajuste de las medidas de los sensores y poder realizar una estimación de la orientación.

Las IMUs tratadas en este trabajo se denominan de 9 grados de libertad o 9DOF (del inglés, 9 Degrees Of Freedom). Las IMUs 9DOF se componen de 3 acelerómetros, 3 giróscopos y 3 magnetómetros, todos ellos tri-axiales. Estos sensores están dispuestos ortogonalmente, de modo que pueden medir cada magnitud en las tres dimensiones del espacio.

Las medidas y errores asociados a cada tipo de sensor de una unidad inercial son los siguientes:

- **Acelerómetros:** miden la gravedad terrestre para detectar la inclinación del sistema con respecto al suelo, pero son sensibles a las fuerzas externas. Un acelerómetro afectado por la fuerza centrífuga de un viraje, o por una caída libre, perderá la referencia del vector gravedad y no podrá usarse para estimar la inclinación.
- **Giróscopos:** miden las velocidades angulares en cada eje, que por integración, permiten calcular los ángulos de orientación del sistema. Sin embargo, al integrar el error del sensor, los ángulos calculados van divergiendo con el tiempo respecto al valor real. A este fenómeno se le denomina deriva del giróscopo.
- **Magnetómetros:** miden el campo magnético local para detectar la dirección del norte magnético y obtener el rumbo. Son afectados por elementos que distorsionan el campo magnético en torno al sensor, causando dos errores conocidos como soft-iron y hard-iron.

Además de los errores propios del sensor, las lecturas de todos ellos se ven afectadas por ruido blanco gaussiano y errores de calibración de escala y offset.

3.1. Acelerómetros

Los acelerómetros son sensores que emplean una masa de prueba suspendida por muelles para medir la aceleración. La medida del desplazamiento de esta masa permite determinar la aceleración, al conocerse su masa y la constante elástica de los muelles.

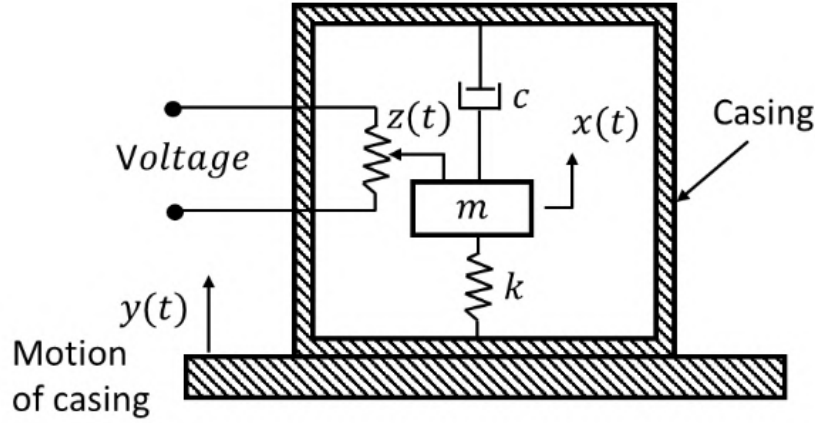


Figura 3.1: Diagrama de un acelerómetro

Los acelerómetros miden las componentes de la aceleración lineal, de coriolis y de fuerzas externas aplicadas al sensor. Sin embargo, no miden la gravedad, debido a que esta actúa sobre la masa de prueba y la caja que lo contiene del mismo modo. Es decir, la medida de un acelerómetro es la fuerza total que afecta al sistema menos la fuerza gravitacional entre la masa [2].

$$\mathbf{a}_{accel} = \frac{1}{m}(\mathbf{F}_{total} - \mathbf{F}_{grav}) \quad (3.1)$$

Cuando un acelerómetro se deja en reposo sobre una mesa horizontal, con su eje z orientado hacia arriba, dará una medida positiva de la aceleración gravitatoria en la dirección del eje z. Esto es debido a que mide la reacción normal del acelerómetro sobre la mesa. Si el acelerómetro se encontrara en caída libre, la medida de la aceleración en todos sus ejes sería nula.

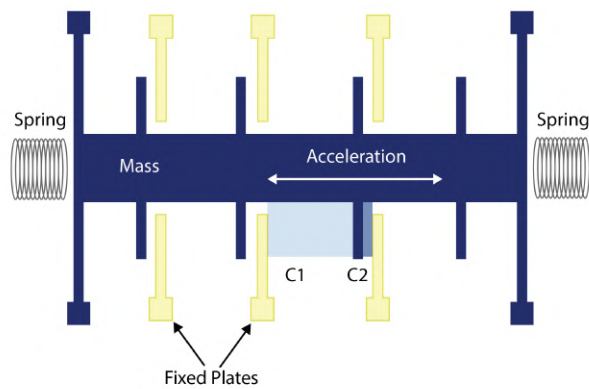
La salida idealizada de un acelerómetro según [2], orientado en ejes NED, puede modelarse de la siguiente manera:

$$a_{accel,x} = \frac{f_x}{m} + g \sin \theta \quad (3.2)$$

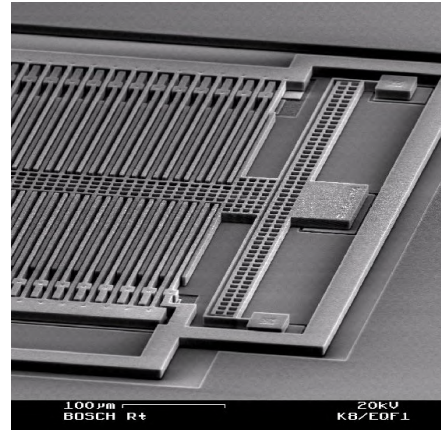
$$a_{accel,y} = \frac{f_y}{m} - g \cos \theta \sin \phi \quad (3.3)$$

$$a_{accel,z} = \frac{f_z}{m} - g \cos \theta \cos \phi \quad (3.4)$$

Los acelerómetros MEMS capacitivos se componen de una pequeña masa de prueba suspendida por resortes en sus esquinas. En los laterales de la masa se disponen una serie de condensadores de peine, que con el movimiento de la masa varían la cercanía de sus placas, y por tanto, su capacitancia. La medida de esta capacitancia permite determinar el desplazamiento de la masa de prueba, y por tanto, hallar la aceleración que afecta al sistema.



(a) Diagrama de funcionamiento



(b) Micrografía

Figura 3.2: Acelerómetro MEMS

3.2. Giróscopos

Los giróscopos son sensores que pueden medir directamente la orientación aprovechando el principio de conservación del momento angular o realizar una medida indirecta de esta determinando la velocidad angular. Los giróscopos que miden la velocidad angular pueden ser ópticos, empleando el efecto Sagnac para comparar la diferencia de distancia recorrida por dos haces de luz, o vibratorios de efecto Coriolis. Estos últimos, son los que se encuentran en una unidad inercial MEMS.

Un giróscopo vibratorio de efecto Coriolis o CVG (del inglés, Coriolis Vibratory Gyroscope) aprovecha el principio de que un objeto en vibración continúa vibrando en el mismo plano aunque su soporte rote [1]. El efecto Coriolis provoca la aparición de una fuerza sobre dicho soporte y la medición de esta fuerza permite determinar la velocidad angular.

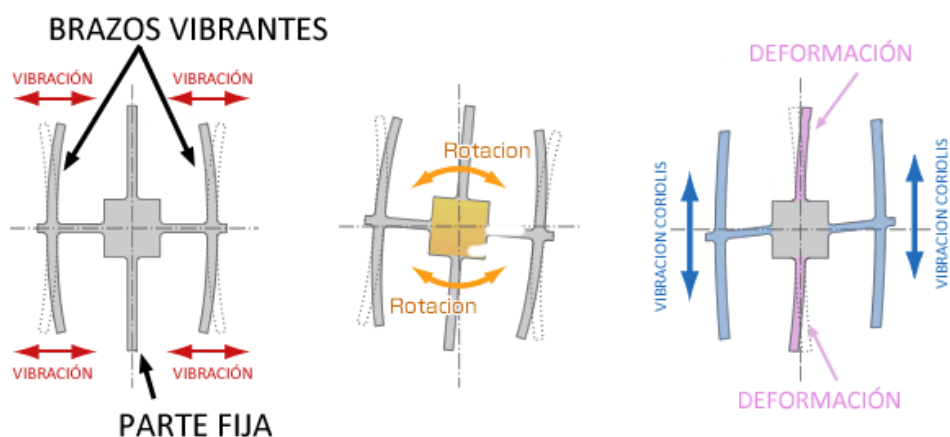


Figura 3.3: Funcionamiento de un giróscopo CVG

Un punto en traslación sobre un cuerpo en rotación experimenta una aceleración proporcional a la velocidad del punto $\mathbf{v}$ y a la velocidad angular del cuerpo Ω denominada aceleración de Coriolis [2].

$$\mathbf{a}_{\text{coriolis}} = 2\Omega \times \mathbf{v} \quad (3.5)$$

Este punto en traslación es la masa de prueba, a la cual se le provoca una vibración a su frecuencia de resonancia ω_n . La velocidad de la masa de prueba puede definirse como

$$\mathbf{v} = A\omega_n \sin \omega_n t \quad (3.6)$$

La salida del giróscopo 3.7 es la aceleración de Coriolis multiplicada por una constante de calibración k_c .

$$\Omega_{\text{gyro}} = k_c |\mathbf{a}_{\text{coriolis}}| = 2k_c |\Omega \times \mathbf{v}| = 2k_c \Omega |\mathbf{v}| = 2k_c \Omega |A\omega_n \sin \omega_n t| = 2k_c A\omega_n \Omega \quad (3.7)$$

Siendo $k_c = 1/(2k_c A\omega_n)$, la salida del giróscopo sería una medición directa de la velocidad angular. A esta salida idealizada habría que añadir un error de tipo bias y un ruido blanco gaussiano.

Un giróscopo vibratorio MEMS está compuesto por una masa de prueba accionada eléctricamente y suspendida por unos resortes sobre un marco externo [4]. Sobre esta masa, que actúa como marco interno, se sitúa un acelerómetro, como el tratado anteriormente, que permite medir la aceleración de Coriolis.

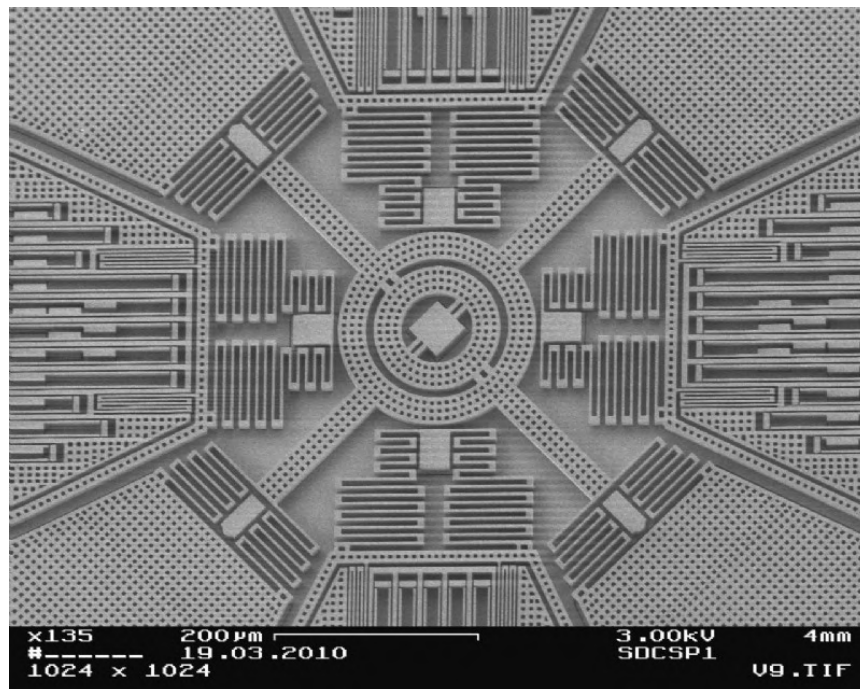


Figura 3.4: Micrografía de un giróscopo MEMS

3.3. Magnetómetros

Un magnetómetro es un sensor que mide el campo magnético local. Suele utilizarse acompañado de un acelerómetro, como brújula magnética giro-compensada. Los magnetómetros MEMS emplean el efecto Hall para detectar el campo magnético.

El efecto Hall aparece cuando se aplica un campo magnético sobre un conductor por el que circula una corriente eléctrica. El campo magnético altera el flujo de corriente, haciendo que las cargas negativas se desplacen a un lado del conductor y las positivas al lado opuesto. Esto provoca una diferencia de potencial entre los dos lados del conductor. Midiendo este voltaje puede determinarse el campo magnético aplicado.

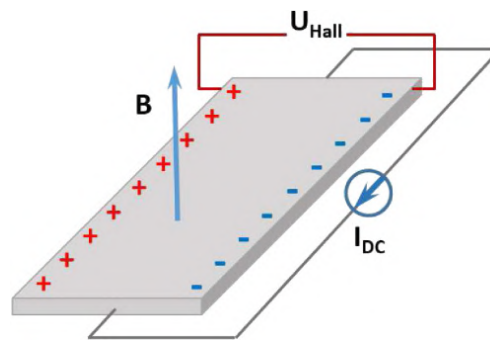


Figura 3.5: Efecto Hall

Otros sensores emplean el efecto magneto-resistivo, que aprovecha las propiedades de algunos materiales que alteran su resistencia eléctrica en presencia de campos magnéticos.

Los magnetómetros son usados para determinar la dirección del Norte Magnético. Sin embargo, el norte magnético cambia de dirección según la latitud, la longitud y la fecha.

Para obtener una referencia universal se emplea el norte geográfico. La diferencia entre la dirección del norte geográfico y el magnético se denomina declinación magnética δ . La relación entre el rumbo geográfico ψ_{geo} y el magnético ψ_{mag} es

$$\psi_{geo} = \psi_{mag} + \delta \quad (3.8)$$

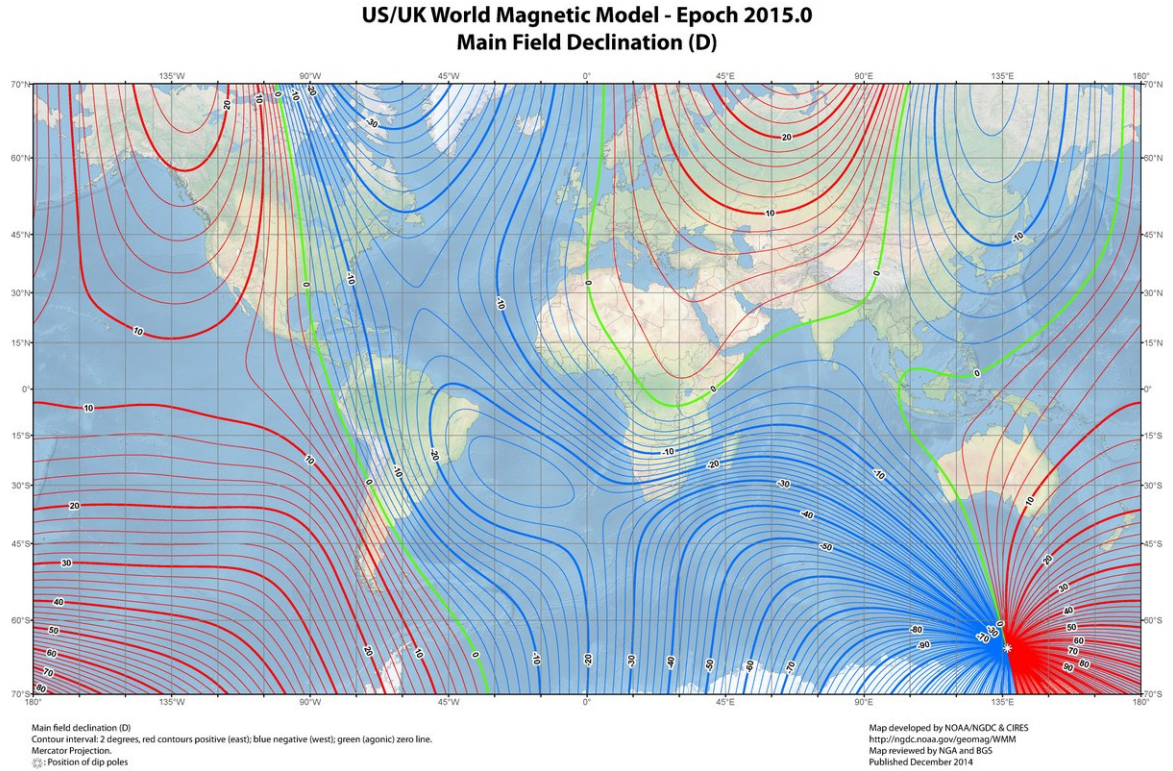


Figura 3.6: Mapa de declinación magnética de 2015

El modelo para calcular la salida de una magnetómetro tri-axial se obtiene rotando el vector campo magnético local a ejes cuerpo [2].

$$\mathbf{m}_{mag} = R_v^b(\phi, \theta, \psi) \mathbf{h} \quad (3.9)$$

El vector campo magnético terrestre local se puede modelar como

$$\mathbf{h} = \begin{pmatrix} h_N \\ h_E \\ h_D \end{pmatrix} = |H| \begin{pmatrix} \cos \gamma \\ 0 \\ \sin \gamma \end{pmatrix} \quad (3.10)$$

siendo γ la inclinación magnética y $|H|$ la magnitud del campo magnético local.

El ángulo de inclinación magnética γ es el ángulo que forma el campo magnético terrestre con el plano horizontal tangente a la superficie de la Tierra. Este modelo no tiene en cuenta la declinación magnética δ , que será compensada a posteriori, una vez determinado el rumbo magnético para obtener el rumbo geográfico.

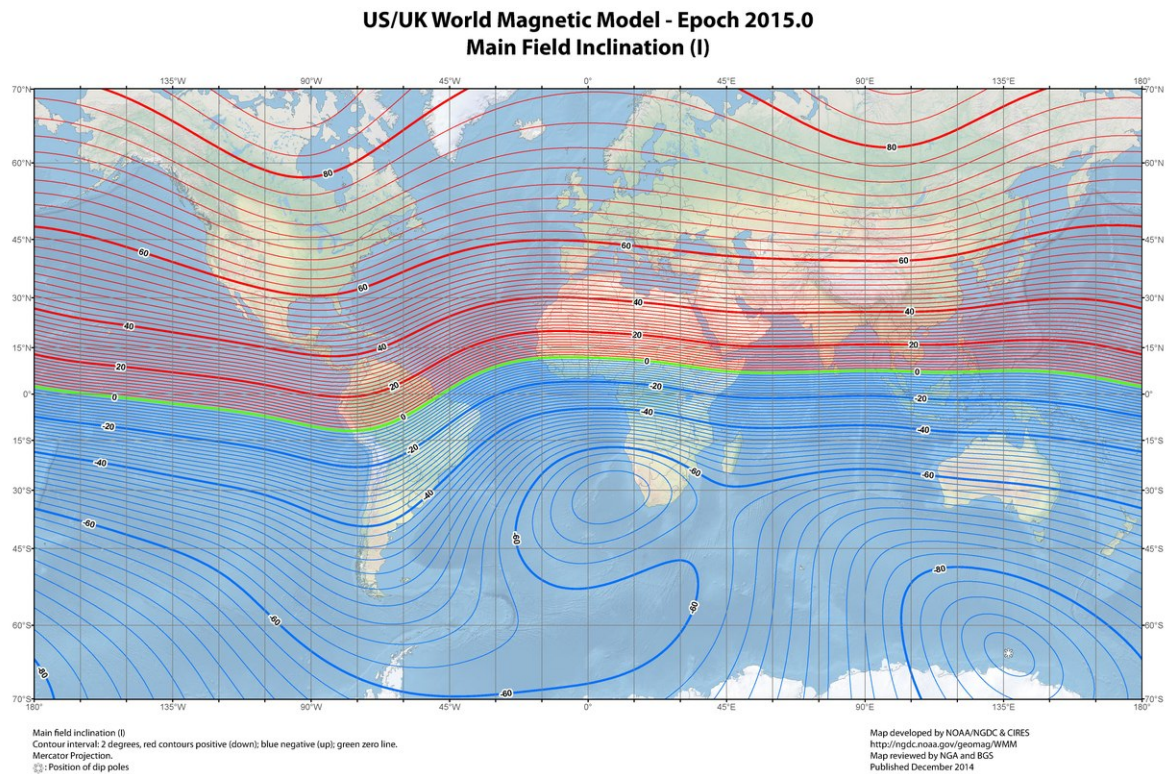


Figura 3.7: Mapa de inclinación magnética de 2015

A la salida idealizada del magnetómetro 3.10 habría que aplicar un error de offset denominado hard-iron, un error soft-iron que deforma los campos magnéticos en escala y dirección y un ruido blanco gaussiano típico de los sensores MEMS.

4. Determinación de la Orientación

Ya se han definido los modelos físicos de los sensores que integra una IMU, así como los ejes de referencia y métodos para representar la orientación. El siguiente paso es definir los algoritmos que se emplearán para estimar la orientación a partir de las lecturas tomadas de la unidad inercial.

En primer lugar, se estudiarán los métodos que permiten obtener la orientación para cada tipo de sensor de forma independiente. Esto se realiza por integración, en el caso del giróscopo; o por inversión del modelo trigonométrico, en el caso del acelerómetro y el magnetómetro.

Por último, se plantearán dos algoritmos de fusión de sensores que combinan las lecturas de diversos tipos de sensores para realizar una mejor estimación de la orientación.

4.1. Detección de la gravedad

Para determinar la orientación del vehículo utilizando el campo gravitatorio terrestre se invierte el modelo del acelerómetro expresado en las ecuaciones 3.2, 3.3 y 3.4, en ausencia de fuerzas externas.

En primer lugar, se normaliza la medida del acelerómetro $\mathbf{a}/\sqrt{a_x^2 + a_y^2 + a_z^2}$, quedando $\|\mathbf{a}\| = 1$.

A continuación, se despeja el ángulo de roll ϕ y de pitch θ obteniendo las expresiones 4.1 y 4.2.

$$\phi = \text{atan2}(-a_y, -a_x) \quad (4.1)$$

$$\theta = \text{asin}(a_x) \quad (4.2)$$

Se ha empleado el arco tangente de dos parámetros atan2 para obtener el ángulo de roll ϕ en un rango de $(-\pi, \pi]$. El ángulo de pitch θ se expresa en un rango de $[-\pi/2, \pi/2]$ y se puede determinar empleando el arco seno.

El signo de los parámetros de la ecuación 4.1 es resultado de invertir las ecuaciones 3.3 y 3.4. Si se hubiera utilizado el arco tangente de un solo parámetro no habría sido necesario evaluar el signo.

Este algoritmo solo funciona en ausencia de fuerzas externas. Durante el viraje de una aeronave, por ejemplo, aparece una aceleración centrífuga que se suma a la aceleración de la gravedad. Si el viraje fuese coordinado, el vector resultante sería perpendicular a la base del vehículo, causando que la estimación del ángulo de alabeo realizada por este método sea nula, es decir, se creería que la aeronave se encuentra horizontalmente estabilizada.

4.1.1. Filtro Paso Bajo

Para realizar una buena estimación utilizando el acelerómetro, es conveniente reducir el ruido de sus medidas. Para ello se ha empleado un filtro paso bajo.

Un filtro paso bajo es un filtro electrónico caracterizado por permitir el paso de las frecuencias más bajas y atenuar las frecuencias más altas. Un tipo de filtro paso bajo muy empleado en electrónica digital es el filtro exponencial EMA (del inglés, Exponential Moving Average).

El filtro EMA realiza una ponderación entre el valor actual y el pasado. Cuanto mayor sea el valor de la ganancia, mayor peso se dará al valor pasado y mayor filtrado será aplicado. Sin embargo, a mayor grado de filtrado mayor será el retardo de la señal.

La ecuación de un filtro EMA [5] se puede expresar como

$$y_k = (1 - \alpha)x_k + \alpha y_{k-1} \quad (4.3)$$

donde y_k es la salida del filtro, x_k es la medida actual, x_{k-1} es la salida anterior del filtro y α es la ganancia del filtro.

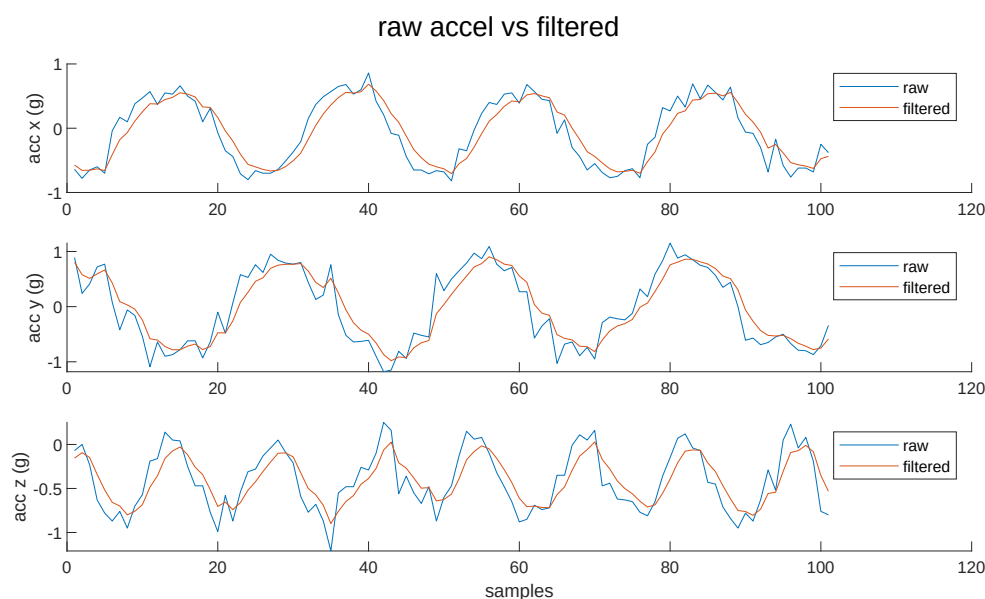


Figura 4.1: Lecturas del acelerómetro con y sin filtrar

En la figura 4.1 se puede observar una comparativa de la señal tomada directamente del acelerómetro con la señal filtrada. Se ha empleado la ganancia $\alpha = 0.60$, que ha sido determinada empíricamente, porque proporciona el mejor filtrado sin causar un retardo excesivo.

4.2. Medición del campo magnético

El acelerómetro solo permite estimar los ángulos de roll y pitch. Para poder determinar el ángulo de yaw, es necesario emplear otra referencia. El método más usado para estimar el yaw es la medición del campo magnético terrestre.

El primer paso consiste en normalizar la medida del magnetómetro $\mathbf{m}/\sqrt{m_x^2 + m_y^2 + m_z^2}$, quedando $\|\mathbf{m}\| = 1$.

A continuación, se rota la medida en ejes cuerpo S^b del magnetómetro al plano horizontal S^{v1} . Para ello, se emplea la estimación del roll y el pitch realizada con el acelerómetro.

$$\mathbf{r} = R_b^{v1}(\phi, \theta)\mathbf{m} = \begin{pmatrix} C\theta & S\phi S\theta & -C\phi S\theta \\ 0 & C\phi & -S\phi \\ -S\theta & S\phi C\theta & C\phi C\theta \end{pmatrix} \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} \quad (4.4)$$

Siendo $\mathbf{m}$ la medida del campo magnético en ejes cuerpo S^b y $\mathbf{r}$ la medida rotada al plano horizontal S^{v1} . Despejando las componentes de cada eje de la expresión 4.4 obtenemos las medidas del magnetómetro compensadas en inclinación [6].

$$r_x = m_x \cos \theta + m_y \sin \phi \sin \theta + m_z \cos \phi \sin \theta \quad (4.5)$$

$$r_y = m_y \cos \phi - m_z \sin \phi \quad (4.6)$$

$$r_z = -m_x \sin \theta + m_y \sin \phi \cos \theta + m_z \cos \phi \cos \theta \quad (4.7)$$

Despreciando la declinación magnética en la ecuación 3.10, el vector campo magnético terrestre en ejes S^v quedaría como $\mathbf{h}_v = [1 \ 0 \ 0]$. Rotando este vector a ejes S^{v1} , según el ángulo de yaw, se obtiene

$$\mathbf{r} = R_{v1}^v(\psi)\mathbf{h}_v = |H| \begin{pmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = |H| \begin{pmatrix} \cos \psi \\ -\sin \psi \\ 0 \end{pmatrix} \quad (4.8)$$

Despejando el ángulo de yaw de la expresión 4.8 y expresándolo en función de los parámetros de las ecuaciones 4.5 y 4.6 se obtiene

$$\psi = \text{atan2}(-r_y, r_x) \quad (4.9)$$

4.2.1. Calibración del magnetómetro

El campo magnético terrestre local se encuentra distorsionado por objetos que deforman el campo magnético que los atraviesa o que generan su propio campo. Para caracterizar esta distorsión se introducen dos errores denominados hard-iron y soft-iron.

El error hard-iron está causado por interferencias magnéticas externas que se suman al campo magnético terrestre local, causando un error de offset o desplazamiento del origen. El error soft-iron está causado por componentes integrados en la placa del magnetómetro que deforman el campo magnético local, causando un error de desalineación y escala en la medida.

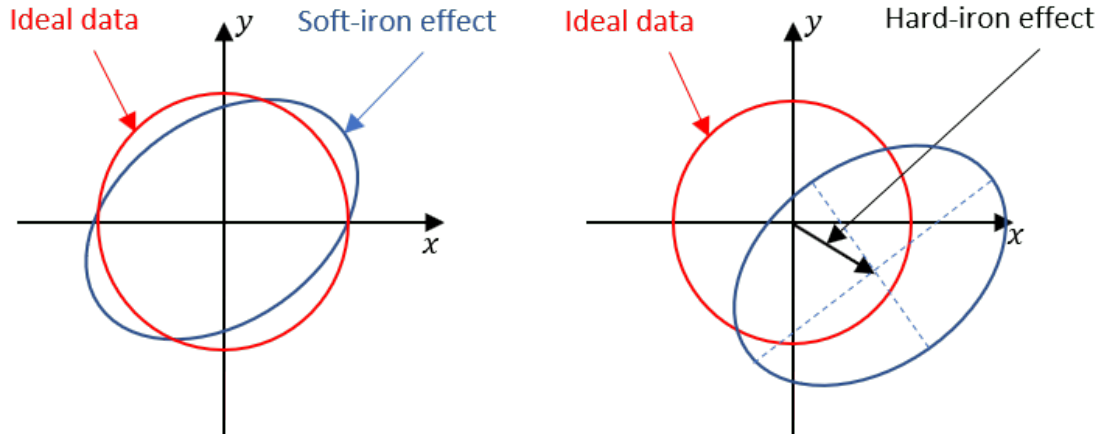


Figura 4.2: Errores soft-iron y hard-iron

Si se rota el magnetómetro en todas las direcciones posibles y se representan las medidas como puntos cuyas coordenadas corresponden a las medidas en cada eje, se obtendría una esfera de radio $|H|$ centrada en el origen de coordenadas.

El error hard-iron provoca el desplazamiento de esta esfera, mientras que el error soft-iron hace que se transforme en un elipsoide. El error soft-iron se representa con una matriz de transformación y el error hard-iron con un vector [7]. Este modelo puede expresarse como

$$\mathbf{m} = A\mathbf{h} + \mathbf{b} \quad (4.10)$$

donde A es la matriz del error soft-iron y $\mathbf{b}$ es el vector del error hard-iron.

Para calibrar las medidas del magnetómetro se realizará un ajuste esférico que permita determinar el error de offset o hard-iron. No se determinará el error soft-iron debido a que es significativamente menor al error hard-iron y a la complejidad que presenta realizar un ajuste elipsoidal.

El campo magnético local se puede expresar como $\mathbf{h} = \mathbf{m} - \mathbf{b}$.

La ecuación paramétrica de la esfera generada durante la calibración es

$$m_x^2 + m_y^2 + m_z^2 + am_x + bm_y + cm_z + d = H^2 \quad (4.11)$$

Expresando esta ecuación en función de $\mathbf{h}$ se obtiene

$$H^2 = \mathbf{h}^T \mathbf{h} = (\mathbf{m} - \mathbf{b})^T (\mathbf{m} - \mathbf{b}) = \mathbf{m}^T \mathbf{m} - 2\mathbf{b}^T \mathbf{m} + \mathbf{b}^T \mathbf{b} \quad (4.12)$$

Para realizar un ajuste al polinomio resultante de desarrollar la ecuación 4.12, se realizará una regresión lineal por el método de mínimos cuadrados [7]. Las ecuaciones para realizar el ajuste son

$$Y = \mathbf{m}^T \mathbf{m} = m_x^2 + m_y^2 + m_z^2 \quad (4.13)$$

$$X = [m_x \ m_y \ m_z \ 1] \quad (4.14)$$

$$a = [2b_x \ 2b_y \ 2b_z \ (H^2 - b_x^2 - b_y^2 - b_z^2)] \quad (4.15)$$

$$a = (X^T X)^{-1} X^T Y \quad (4.16)$$

La ecuación 4.16 proporciona la solución del problema, pues contiene las componentes del error hard-iron y el radio de la esfera H o intensidad del campo magnético local.

En la imagen 4.3 pueden observarse los datos recogidos por el magnetómetro antes y después de la calibración, representados en 3D. En la imagen 4.4 se puede ver una proyección de los mismo datos sobre el plano horizontal. Puede apreciarse como se ha corregido el error de bias, devolviendo el centro de la esfera al origen de coordenadas.

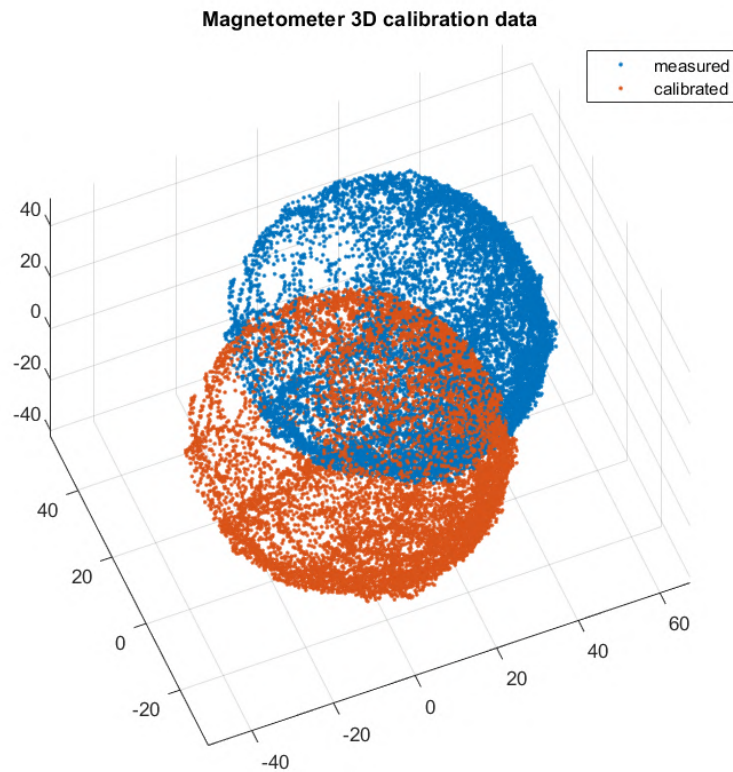


Figura 4.3: Calibración del magnetómetro en 3D

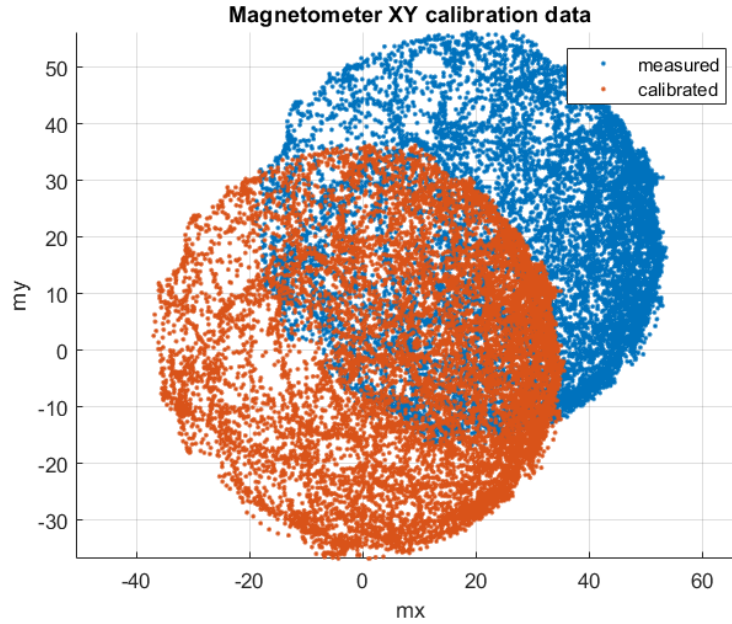


Figura 4.4: Calibración del magnetómetro en el plano XY

4.3. Integración del giróscopo

La estimación de la orientación basada en acelerómetro y magnetómetro presenta problemas si el sistema está sometido a fuerzas externas e interferencias magnéticas. Otro método muy extendido, para la estimación de la orientación, se basa en integrar las velocidades angulares proporcionadas por un giróscopo.

Para el algoritmo de integración del giróscopo se va a emplear una representación de la orientación basada en cuaterniones. Los cuaterniones no sufren de bloqueo del gimbal, problema que surgiría si empleáramos ángulos de euler para la integración y superáramos el rango de $\pm 90^\circ$ en el que está definido el ángulo de cabeceo θ .

La derivada del cuaternión [3] se expresa como

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{\Omega}(\omega) \mathbf{q} \quad (4.17)$$

siendo ω el valor de la velocidad angular en ejes cuerpo S^b y $\mathbf{\Omega}$ la matriz

$$\mathbf{\Omega}(\omega) = \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix} \quad (4.18)$$

Desarrollando la serie de Taylor para la expresión $\mathbf{q}(t + \Delta t)$ como

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \dot{\mathbf{q}}_t \Delta t + \frac{1}{2!} \ddot{\mathbf{q}}_t \Delta t^2 + \frac{1}{3!} \dddot{\mathbf{q}}_t \Delta t^3 + \dots \quad (4.19)$$

se puede aproximar la integral del cuaternión al polinomio de orden 1 de la serie [8] expresada en 4.19 como

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \dot{\mathbf{q}}_t \Delta t \quad (4.20)$$

Sustituyendo la expresión de $\dot{\mathbf{q}}$ de la ecuación 4.17, la integral del cuaternión queda como

$$\mathbf{q}_{t+1} = \left(\mathbf{I}_4 + \frac{1}{2} \boldsymbol{\Omega}(\boldsymbol{\omega}) \Delta t \right) \mathbf{q}_t \quad (4.21)$$

Un cuaternión debe tener norma unitaria para poder representar la orientación correctamente. Tras cada paso de integración es necesario normalizar el cuaternión calculado.

$$\mathbf{q}_{tn} = \frac{\mathbf{q}_t}{\|\mathbf{q}_t\|} \quad (4.22)$$

Para poder realizar la integración es necesario conocer las condiciones iniciales para $t = 0$. En la implementación realizada, este algoritmo es inicializado con el cuaternión correspondiente a los ángulos de euler de valor nulo $\phi = \theta = \psi = 0^\circ$.

Este algoritmo es recursivo y su valor depende de las condiciones iniciales y las medidas realizadas por el giróscopo. Debido a que el giróscopo presenta error en la medida y a que este error es integrado durante la ejecución del algoritmo, la orientación estimada irá divergiendo con el tiempo del valor real. Este error acumulado se denomina deriva del giróscopo.

4.4. Fusión de sensores

Los algoritmos planteados permiten estimar la orientación de un sistema tomando medidas de los diferentes sensores presentes en una unidad inercial. Sin embargo, las estimaciones proporcionadas por estos métodos presentan muchos errores.

Para solucionar este problema y obtener una mejor estimación de la orientación se realiza una combinación de estas estimaciones mediante un método denominado fusión de sensores.

Los dos métodos que se plantean en este trabajo, para combinar las lecturas de los sensores de una unidad inercial, son el Filtro Complementario y el Filtro de Kalman.

4.4.1. Filtro Complementario

El filtro complementario es una variación estática del Filtro de Kalman. Su funcionamiento es similar al de un filtro EMA estudiado en 4.1.1. El filtro complementario realiza una ponderación entre la estimación del giróscopo y la del acelerómetro y magnetómetro, a través de una ganancia α .

El filtro complementario [9] se puede expresar como

$$\mathbf{q}_t = (1 - \alpha) \mathbf{q}_{gyr} + \alpha \mathbf{q}_{acc,mag} \quad (4.23)$$

donde $\mathbf{q}_t$ es la salida del filtro en el instante t , $\mathbf{q}_{gyr}$ es la estimación realizada del giróscopo para

ese instante, $\mathbf{q}_{acc,mag}$ es la estimación del acelerómetro y el magnetómetro y α es la ganancia del filtro.

La integración del giróscopo se realiza sobre la salida anterior del filtro, como se expresa en la ecuación 4.21

$$\mathbf{q}_{gyr} = \left(\mathbf{I}_4 + \frac{\Delta t}{2} \boldsymbol{\Omega}_t \right) \mathbf{q}_{t-1} \quad (4.24)$$

El valor de α determina el peso de cada medida en la ponderación. Una ganancia baja dará mayor peso a la integración del giróscopo, mientras que una ganancia alta dará mayor peso a la estimación del acelerómetro y el magnetómetro.

4.4.2. Filtro de Kalman

El filtro de Kalman es uno de los algoritmos más usados para la estimación del estado dinámico de un sistema. Es ampliamente usado en la industria aeronáutica y espacial para la navegación y guiado de vehículos. Fue desarrollado por el ingeniero Rudolf Kalman en 1960. El filtro fue usado por primera vez en la NASA, como parte del sistema de guiado en las misiones Apollo.

El filtro de Kalman [10] combina una predicción basada en un modelo dinámico con una observación del estado actual para proporcionar la mejor estimación del estado del sistema. Para combinar predicción y observación, el filtro de Kalman realiza una media ponderada utilizando una ganancia K como peso, denominada ganancia de Kalman. Esta ganancia K se computa estimando el error cometido en la predicción y el error cometido en la medida.

Sea un sistema representado en el espacio de estados como

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_{t-1} \quad (4.25)$$

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t \quad (4.26)$$

siendo $\mathbf{x}_t$ el estado actual del sistema, $\mathbf{u}_{t-1}$ el vector de entrada y $\mathbf{z}_t$ una estimación u observación del estado. El sistema se modela a través de las matrices $\mathbf{F}$ y $\mathbf{B}$, siendo $\mathbf{F}$ la matriz de transición del sistema y $\mathbf{B}$ la matriz de entrada. La matriz $\mathbf{H}$ es la matriz de salida y modela la lectura de los sensores. Las variables $\mathbf{w}_{t-1}$ y $\mathbf{v}_t$ son ruidos blancos o gaussianos de media cero con varianzas $\mathbf{Q}_{t-1}$ y $\mathbf{R}_t$, respectivamente.

El algoritmo del filtro Kalman realiza la estimación del estado del sistema en 2 pasos: predicción y corrección.

1. El paso de predicción estima el estado del sistema $\hat{\mathbf{x}}_t$ y su error $\hat{\mathbf{P}}_t$, en el instante t , propagando las ecuaciones con el estado anterior $t - 1$. A esta estimación se le denomina a priori.

$$\hat{\mathbf{x}}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t \quad (4.27)$$

$$\hat{\mathbf{P}}_t = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q}_{t-1} \quad (4.28)$$

2. El paso de corrección realiza una medida del sistema para rectificar la estimación a priori. Esta observación se compara con una predicción de la medición para computar la ganancia de Kalman $\mathbf{K}_t$.

$$\mathbf{y}_t = \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t \quad (4.29)$$

$$\mathbf{K}_t = \hat{\mathbf{P}}_t \mathbf{H}^T (\mathbf{H} \hat{\mathbf{P}}_t \mathbf{H}^T + \mathbf{R})^{-1} \quad (4.30)$$

$$\mathbf{x}_t = \hat{\mathbf{x}}_t + \mathbf{K}_t \mathbf{y}_t \quad (4.31)$$

$$\mathbf{P}_t = \hat{\mathbf{P}}_t - \mathbf{K}_t \mathbf{H} \hat{\mathbf{P}}_t \quad (4.32)$$

El residuo $\mathbf{y}_t$ es la diferencia entre la observación y la predicción de la medición. La ganancia de Kalman $\mathbf{K}_t$ corrige las estimaciones a priori del estado del sistema $\hat{\mathbf{x}}_t$ y de su covarianza $\hat{\mathbf{P}}_t$.

4.4.3. Filtro de Kalman Extendido

El filtro de Kalman permite trabajar con sistemas lineales. Sin embargo, las ecuaciones de estado, que describen el modelo utilizado para determinar la orientación de un sistema, no lo son.

Para trabajar con sistemas no lineales se utiliza el filtro de Kalman extendido. Utilizando las series de Taylor se puede linealizar un sistema no lineal, en el entorno de la estimación actual.

Las ecuaciones del filtro de Kalman extendido [3], abreviado EKF (del inglés, Extended Kalman Filter), son las siguientes:

1. Predicción

$$\hat{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t) \quad (4.33)$$

$$\hat{\mathbf{P}}_t = \mathbf{F}_t \mathbf{P}_{t-1} \mathbf{F}_t^T + \mathbf{Q}_{t-1} \quad (4.34)$$

2. Corrección

$$\mathbf{y}_t = \mathbf{z}_t - \mathbf{h}(\mathbf{x}_t) \quad (4.35)$$

$$\mathbf{K}_t = \hat{\mathbf{P}}_t \mathbf{H}_t^T (\mathbf{H}_t \hat{\mathbf{P}}_t \mathbf{H}_t^T + \mathbf{R}_t)^{-1} \quad (4.36)$$

$$\mathbf{x}_t = \hat{\mathbf{x}}_t + \mathbf{K}_t \mathbf{y}_t \quad (4.37)$$

$$\mathbf{P}_t = \hat{\mathbf{P}}_t - \mathbf{K}_t \mathbf{H}_t \hat{\mathbf{P}}_t \quad (4.38)$$

Las matrices $\mathbf{F}_t$ y $\mathbf{H}_t$ son los Jacobianos en el instante t de las funciones $\mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t)$ y $\mathbf{h}(\mathbf{x}_t)$, respectivamente.

$$\mathbf{F}_t = \frac{\partial \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t)}{\partial \mathbf{x}} \quad (4.39)$$

$$\mathbf{H}_t = \frac{\partial \mathbf{h}(\mathbf{x}_t)}{\partial \mathbf{x}} \quad (4.40)$$

El filtro de Kalman propuesto está basado en el algoritmo detallado en [3], que utiliza las lecturas del acelerómetro y el giróscopo para determinar los ángulos de pitch y roll, pero no de yaw. Este algoritmo ha sido ampliado para incluir las lecturas del magnetómetro y el modelo de ruido del giróscopo de la implementación realizada en [11].

El algoritmo resultante combina las lecturas del giróscopo, del acelerómetro y del magnetómetro. La integración de la velocidad angular proporcionada por el giróscopo permite realizar una predicción del estado actual del sistema, mientras que el acelerómetro y el magnetómetros son usados para proporcionar una observación de la orientación.

Para este modelo se define el vector de estado $\mathbf{x}_t$ como el cuaternión que representa la orientación del sistema y el vector de entrada $\mathbf{u}_t$ como la velocidad angular en rad/s .

$$\mathbf{x}_t = \mathbf{q}_t = [q_w \quad q_x \quad q_y \quad q_z]^T \quad (4.41)$$

$$\mathbf{u}_t = \boldsymbol{\omega}_t = [\omega_x \quad \omega_y \quad \omega_z]^T \quad (4.42)$$

En el primer paso, se realiza una predicción del estado actual del sistema $\hat{\mathbf{x}}_t$ y de su error $\hat{\mathbf{P}}_{t-1}$ mediante la integración de la velocidad angular. La integración del cuaternión estado se realiza utilizando la expresión 4.21.

El Jacobiano de la matriz de transición [3] se puede calcular con la siguiente expresión:

$$\mathbf{F}_t = \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \mathbf{q}} = \mathbf{I}_4 + \frac{\Delta t}{2} \boldsymbol{\Omega}_t \quad (4.43)$$

Para calcular el error cometido en la predicción $\hat{\mathbf{P}}_{t-1}$ es necesario definir la matriz de covarianza del proceso $\mathbf{Q}_t$.

Considerando que el giróscopo es el mismo en sus tres ejes $\sigma_{gx} = \sigma_{gy} = \sigma_{gz} = \sigma_g$, la matriz de covarianza del ruido del giróscopo sería

$$\boldsymbol{\Sigma}_g = \begin{bmatrix} \sigma_{gx}^2 & 0 & 0 \\ 0 & \sigma_{gy}^2 & 0 \\ 0 & 0 & \sigma_{gz}^2 \end{bmatrix} = \sigma_g^2 \mathbf{I}_3 \quad (4.44)$$

Si el vector de estado estuviera expresado en ángulos de euler, podríamos expresar la matriz de covarianza del proceso como $\mathbf{Q}_t = \boldsymbol{\Sigma}_g$, es decir, sería igual al ruido del giróscopo. Para poder expresar en cuaterniones se utiliza la expresión 4.45 obtenida de [11].

$$\mathbf{Q}_t = \mathbf{W}_t \boldsymbol{\Sigma}_g \mathbf{W}_t^T = \sigma_g^2 \mathbf{W}_t \mathbf{W}_t^T \quad (4.45)$$

donde $\mathbf{W}_t$ es el Jacobiano de la función $\mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)$ respecto a la velocidad angular, de expresión

$$\mathbf{W}_t = \frac{\Delta t}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \quad (4.46)$$

En el segundo paso, se realiza la corrección de la predicción empleando las medidas del giróscopo y el magnetómetro.

El vector observación $\mathbf{z}_t$ contiene las lecturas del acelerómetro y del magnetómetro

$$\mathbf{z}_t = [a_x \ a_y \ a_z \ m_x \ m_y \ m_z]^T \quad (4.47)$$

A continuación, se realiza una estimación de la medida, utilizando el modelo del acelerómetro y magnetómetro y la predicción del estado realizada en el paso anterior.

El vector campo gravitatorio terrestre $\mathbf{g}$ y campo magnético $\mathbf{r}$ normalizados en ejes NED se definen como

$$\mathbf{g} = [0 \ 0 \ -1]^T \quad (4.48)$$

$$\mathbf{r} = [\cos \gamma \ 0 \ \sin \gamma]^T \quad (4.49)$$

para el vector campo magnético terrestre $\mathbf{r}$, se ha despreciado la inclinación magnética γ , quedando como $\mathbf{r} = [1 \ 0 \ 0]^T$. Se ha denominado $\mathbf{r}$ en lugar de $\mathbf{h}$ como en 3.10 para no crear confusión con la función $\mathbf{h}(\mathbf{x}_t)$.

Las estimación de la medida viene de rotar estos dos vectores a ejes cuerpo mediante la matriz 2.15

$$\mathbf{h}(\hat{\mathbf{q}}_t) = \begin{bmatrix} \hat{\mathbf{a}} \\ \hat{\mathbf{m}} \end{bmatrix} = \begin{bmatrix} R_v^b(\hat{\mathbf{q}})\mathbf{g} \\ R_v^b(\hat{\mathbf{q}})\mathbf{r} \end{bmatrix} = \frac{\partial \mathbf{h}(\mathbf{q}_t)}{\partial \mathbf{q}} = \begin{bmatrix} -2q_xq_z + 2q_wq_y \\ -2q_wq_x - 2q_yq_z \\ -q_w^2 + q_x^2 + q_y^2 - q_z^2 \\ q_w^2 + q_x^2 - q_y^2 - q_z^2 \\ 2q_xq_y - 2q_wq_z \\ 2q_xq_z - 2q_wq_y \end{bmatrix} \quad (4.50)$$

Esta expresión debe ser linealizada. Para ello, debe computarse el Jacobiano $\mathbf{H}$ de la expresión 4.50.

$$\mathbf{H}(\hat{\mathbf{q}}_t) = 2 \begin{bmatrix} q_y & -q_z & q_w & q_x \\ -q_x & -q_w & -q_z & -q_z \\ -q_w & q_x & q_y & -q_z \\ q_w & q_x & -q_y & -q_z \\ -q_z & q_y & q_x & -q_w \\ q_y & q_z & q_x & q_y \end{bmatrix} \quad (4.51)$$

Queda definir la matriz de covarianza de la observación $\mathbf{R}$, que se puede expresar directamente como las varianzas del giróscopo y del magnetómetro:

$$\Sigma_\omega = \begin{bmatrix} \sigma_{ax}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{ay}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{az}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{mx}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{my}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{mz}^2 \end{bmatrix} = \begin{bmatrix} \sigma_a^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_m^2 \mathbf{I}_3 \end{bmatrix} \quad (4.52)$$

Al finalizar cada iteración del algoritmo se debe normalizar el cuaternión estado, para que la representación de la orientación tenga sentido.

$$\mathbf{q}_{tn} = \frac{\mathbf{q}_t}{\|\mathbf{q}_t\|} \quad (4.53)$$

Al igual que en el algoritmo de integración del giróscopo 4.3 es necesario conocer las condiciones iniciales para $t = 0$. Utilizando el cuaternión correspondiente a los ángulos de euler nulos, se inicializa el vector de estado.

5. Montaje Experimental

Para poder evaluar la eficacia de distintos algoritmos, es necesario disponer de una referencia que permita compararlos. Además de conocer los datos de entrada y la orientación estimada por el algoritmo, se debe conocer la orientación real de la IMU para poder valorar el desempeño del sistema. Para ello se ha construido la plataforma giroscópica que permite rotar la IMU de forma controlada, proporcionando la orientación real de la plataforma en cada instante.

5.1. Plataforma giroscópica

Esta plataforma funciona como los gimbals usados en fotografía y drones que permiten la estabilización de un objeto, al contrarrestar las rotaciones externas con sus tres actuadores. En la plataforma giroscópica, en lugar de contrarrestar las rotaciones, éstas serán introducidas de forma controlada para lograr la orientación de la IMU deseada.



Figura 5.1: Gimbal usado para estabilizar una cámara o smartphone

La estructura planteada consiste en 4 piezas principales:

- **Eje X (cuna):** plataforma sobre la que se ubica la IMU y que rota en sus tres grados de libertad.
- **Eje Y (anillo):** estructura circular que soporta a la cuna (eje X). El motor del anillo rota al eje X respecto al Y según el ángulo de alabeo (roll) deseado.

- **Eje Z (soporte en U):** soporte en forma de U que sostiene al anillo (eje Y). El motor de este soporte rota al eje Y respecto al Z según el ángulo de cabeceo (pitch) aplicado.
- **Base:** soporte fijo al suelo respecto al que rota el eje Z, según el ángulo de guiñada (yaw), que sostiene la estructura completa.

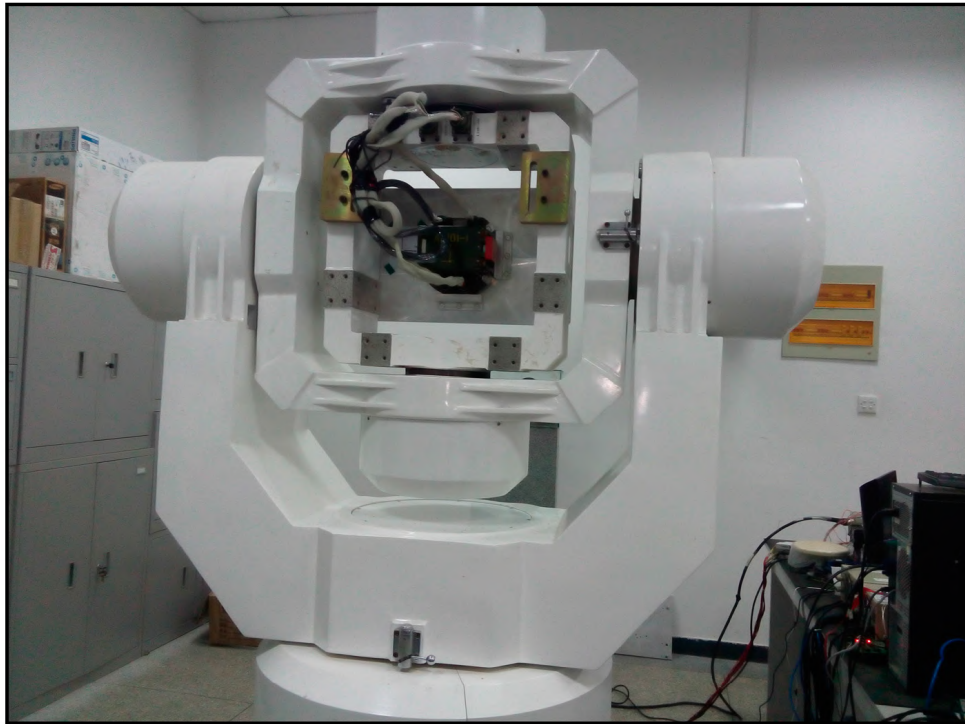


Figura 5.2: Plataforma para la calibración de unidades inerciales

La plataforma que se tratará de construir está inspirada en las plataformas giroscópicas utilizadas para calibrar y validar sensores inerciales, de uso profesional. Sin embargo, esta será una versión miniaturizada y artesanal de las mismas, con sensores de bajo coste.

Además, estas plataformas presentan un rango de rotación completo, algo que se descartó debido a su complejidad. Para hacer una plataforma que permita un rango de rotación de 360° , en cada eje, es necesario el uso de motores paso a paso y de colectores eléctricos rotativos. Esto complica el diseño, sobre todo del circuito eléctrico, además de aumentar su coste. En su lugar, se emplearán servomotores con rangos de rotación entre 90° y 180° por eje.

5.2. Componentes electrónicos

Para el funcionamiento de la plataforma se requiere de una serie de componentes electrónicos que podemos dividir en sensores, controladores y actuadores.

Para el movimiento de las piezas de la plataforma se emplearán servomotores junto a un encoder magnético de posición angular que permita conocer la rotación real de cada eje.

Para la toma de datos de la IMU se ha escogido un microcontrolador con unidad inercial integrada. Estos dispositivos permiten, además de la lectura de la IMU integrada, realizar operaciones para manipular las lecturas y controlar el bus de datos escogido para enviar la información obtenida de la IMU.

Para el control de la plataforma y comunicación con el ordenador que recopilará los datos se ha empleado un microcontrolador Arduino UNO. Las funciones de este microcontrolador son la de controlar a los servomotores, realizar la lectura de los sensores de ángulo de cada eje, recibir los medidas de la IMU y enviar los datos del ensayo al ordenador para su registro y posterior procesado.

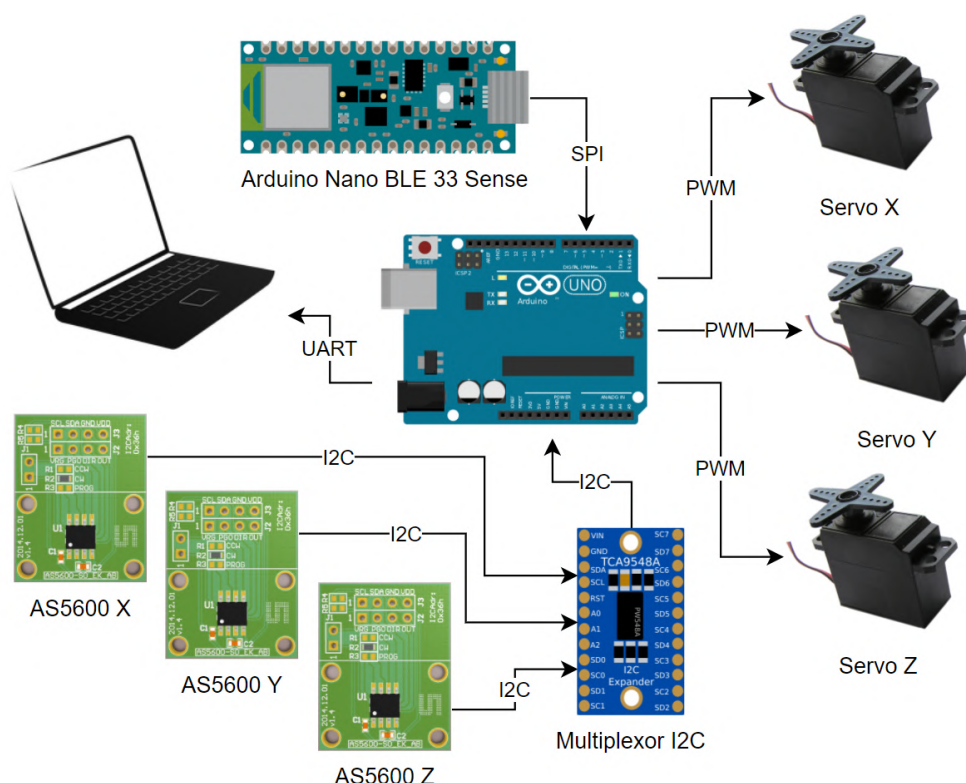


Figura 5.3: Componentes de un servomotor

La conexión entre el controlador de la plataforma y el ordenador que recopilará los datos será realizada a través del puerto de UART, mediante una conexión USB. La conexión entre los servomotores y el controlador se realizará mediante cables de señal PWM. Para la lectura de los sensores se empleará un bus I2C multiplexado. Por último, para realizar la lectura de la IMU se hará uso del bus de datos SPI.

5.2.1. Arduino UNO

El dispositivo encargado de controlar la plataforma será el microcontrolador Arduino UNO. Este microcontrolador consiste en una placa de desarrollo, basada en el microchip ATmega328P, que permite la construcción de dispositivos digitales que pueden detectar y controlar objetos del mundo real.

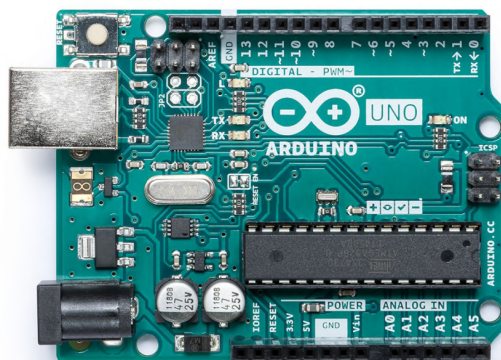


Figura 5.4: Arduino UNO

La empresa Arduino es una compañía que desarrolla Hardware y Software Libre bajo Licencia Pública General de GNU (GPL). El proyecto Arduino tiene sus orígenes en la comunidad DIY (del inglés, Do It by Yourself) como una forma de acercar a los principiantes y estudiantes el uso y programación de microcontroladores.

Arduino proporciona su propio entorno de desarrollo de software (IDE) basado en C/C++ para la programación de sus placas. La IDE de Arduino integra compiladores para diversos modelos de microcontrolador, un gestor de librerías de código y visualización del puerto serial por texto y gráficos.



Figura 5.5: Arduino IDE

La placa Arduino UNO dispone de catorce pines digitales de entrada o salida I/O (del inglés, Input/Output). Seis de ellos pueden proporcionar una señal PWM. Además, dispone de un convertidor analógico digital (ADC) multiplexado en seis canales. Para las comunicaciones integra un puerto UART, un bus I2C y un bus SPI, implementados por hardware. Su voltaje de funcionamiento es de 5V, pero también proporciona una salida de 3.3V.

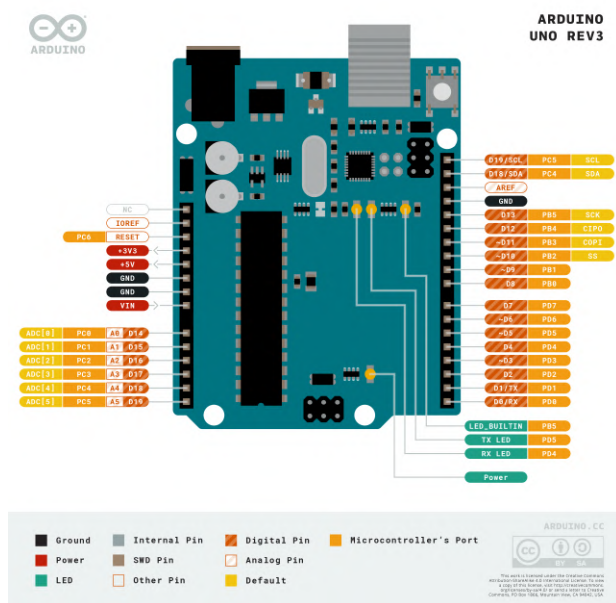


Figura 5.6: Diagrama de pines del Arduino UNO

Para el montaje propuesto se emplearán los pines 13, 12, 11 y 10 para el bus SPI; los pines 19 y 18 para el bus I2C; y los pines 6, 5 y 3 como salidas PWM. Además, se usará el puerto UART para conectar el Arduino UNO al ordenador encargado de la recogida de datos.

5.2.2. Servomotores

Un servomotor es un actuador rotativo que permite el control de la posición o de la velocidad angular. Se componen de un motor de corriente continua, un sistema de engranajes reductores, un potenciómetro que mide y codifica el ángulo del eje, una unidad de control y un cable de alimentación y control, por el que se le suministra la potencia necesaria para su funcionamiento y la señal con el ángulo deseado.

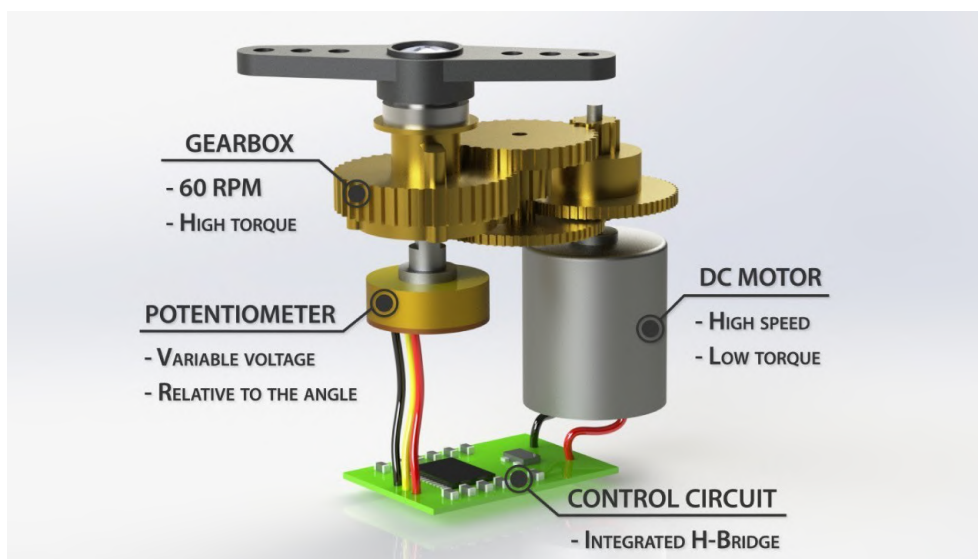


Figura 5.7: Componentes de un servomotor

La unidad de control mide el ángulo del eje, a través del potenciómetro y lo compara con el ángulo deseado. Utilizando un puente H de transistores, se controla el sentido y velocidad de giro del motor. El control del ángulo de salida suele realizarse mediante un controlador PID (siglas de Proporcional-Integrativo-Derivativo).

Para fijar la posición angular deseada se envía una señal PWM. La modulación PWM (del inglés, Pulse Width Modulation) consiste en modificar el ciclo de trabajo de una señal de onda cuadrada para transmitir información o controlar la cantidad de energía enviada. Para un servomotor, la anchura del pulso codifica la posición angular que se desea obtener.

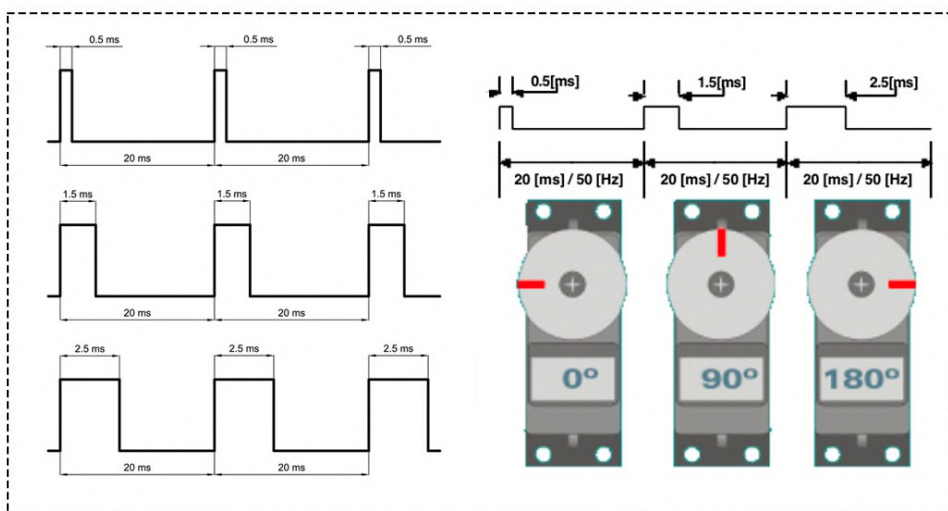


Figura 5.8: Señal PWM vs posición angular

Para el montaje de la plataforma se han empleado dos modelos de servomotores distintos. Para

mover los ejes Z e Y, se han utilizado dos unidades del modelo MG995. Para mover el eje X se ha empleado un TR-1160A, que tiene un peso menor, aunque también menos torque.

Tabla 5.1: Características de los servomotores utilizados

Modelo	Torque	Velocidad	Voltaje	Dimensiones	Peso
MG995	10 kg/cm	0.2-0.16 s	4.8-6 V	40.7 x 19.7 x 42.9 mm	55 g
TR-1160A	2.1 kg/cm	0.11 s	6 V	29.6 x 28.0 x 13.2 mm	16 g

El rango de movimiento de estos servomotores, proporcionado por el fabricante, es de 180°, 90° en cada dirección. Sin embargo, en la práctica este rango se sitúa entorno a los 120°. La velocidad indicada en la tabla 5.1 es el tiempo que el servo tarda en cubrir un giro de medio rango (60°) a máxima velocidad.

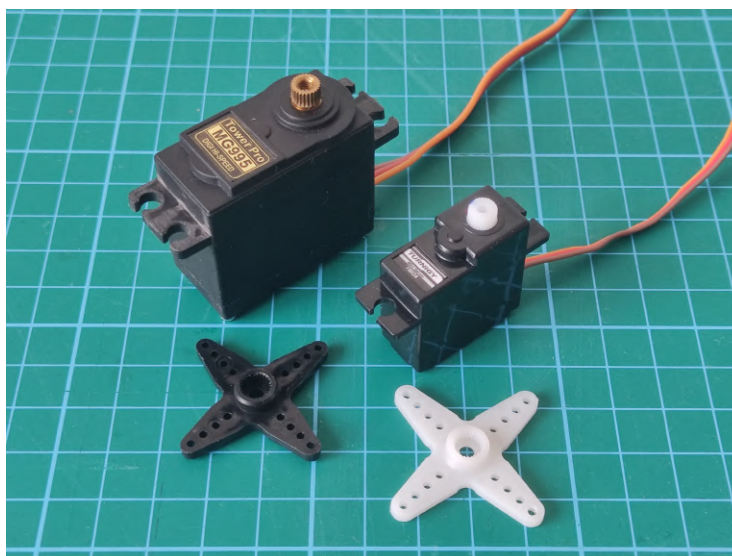


Figura 5.9: Servomotores MG995 y TR-1160A

Cabe mencionar, que el principal atractivo de los servomotores es la simplicidad de montaje y control de los mismos. Otro tipo de motor, que podría haberse usado, son los motores paso a paso. Estos motores tienen un giro suave y estable, pero requieren de un hardware más complejo para su control que el de los servomotores. Por otro lado, los servomotores presentan un giro brusco y ruidoso, y a veces se quedan oscilando en torno al ángulo demandado.

5.2.3. Sensores de ángulo AS5600

Debido a que los servomotores no mantienen perfectamente la posición angular demandada, es necesario hacer una medida de este ángulo para poder disponer del dato de orientación real. Para llevar a cabo esta tarea, se han empleado los encoders magnéticos AS5600.



Figura 5.10: Sensores de ángulo AS5600

El sensor AS5600 utiliza un sensor de efecto Hall y un imán para medir la posición angular sin contacto. El sensor presenta una resolución de 12 bits y puede proporcionar su salida de forma analógica, por señal modulada en PWM o a través del bus de datos I2C. Además, es programable y permite modificar parámetros como el punto de paso por 0° y el ángulo máximo (360° por defecto).

Se han empleado tres de estos sensores en el montaje de la plataforma, uno por cada grado de libertad. Se han calibrado los imanes de forma que el punto de paso por cero quede siempre fuera del rango de giro del eje y se ha aprovechado el imán como soporte rotativo para cada eje. Además, se ha añadido un sistema de tornillos que permita alejar o acercar el imán al sensor para poder ajustar la separación.

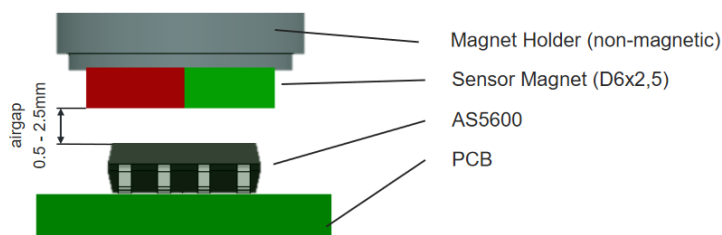


Figura 5.11: Separación entre el sensor AS5600 y el imán

5.2.4. Multiplexor I2C TCA9548A

Debido a que los sensores AS5600 poseen todos la misma dirección de memoria (0x36, en hexadecimal) y que el microcontrolador Arduino UNO solo dispone de un bus I2C, es necesario multiplexar la comunicación entre el microcontrolador y los sensores. Para esta tarea se ha escogido el multiplexor I2C TCA9548A de ocho canales.

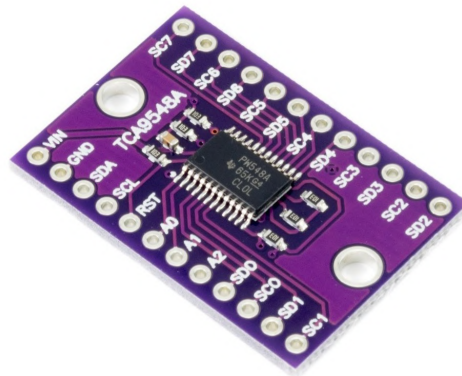


Figura 5.12: Multiplexor I2C TCA9548A

Este multiplexor tiene una dirección editable entre 0x70 y 0x77 mediante sus tres entradas A0, A1 y A2. Para seleccionar un canal, de los ocho que dispone, se envía un byte por I2C a la dirección del multiplexor. Cada posición de este byte es usada para apagar o encender los buses del multiplexor. Si enviamos el byte 0b000000, en binario, todos los buses serán apagados. Si enviamos el byte 0b00010000, se activará el bus 5.

5.2.5. Arduino Nano BLE33 Sense

La unidad inercial a ensayar está integrada en el microcontrolador Arduino Nano BLE33 Sense. El Arduino Nano BLE 33 Sense es una evolución del Arduino Nano, pero con un procesador más potente y con un grupo de sensores integrados sobre su placa.



Figura 5.13: Arduino Nano BLE 33 Sense

Los sensores que incluye son los siguientes:

- x1 IMU LSM9DS1 de 9 grados de libertad
- x1 micrófono digital omnidireccional MP34DT05

- x1 sensor digital de proximidad, luz ambiente, color y gestos APDS9960
- x1 sensor barométrico LPS22HB
- x1 sensor de temperatura y humedad HTS221

La unidad inercial MEMS LSM9DS1 de 9 grados de libertad posee tres acelerómetros, tres giróscopos y tres magnetómetros, todos ellos tri-axiales y con resolución de 16 bits. La lectura de la IMU se realiza a través del bus interno I2C, utilizando las librerías propias del sensor.

Tabla 5.2: Características del módulo inercial LSM9DS1

Sensor	Rango de medida	Sensibilidad	Offset típico
Acelerómetro	$\pm 2/\pm 4/\pm 8/\pm 16$ g	0.061/0.122/0.244/0.732 mg/LSB	± 90 mg
Giróscopo	$\pm 245/\pm 500/\pm 2000$ dps	8.75/17.50/70.00 mdps/LSB	± 30 dps
Magnetómetro	$\pm 4/\pm 8/\pm 12/\pm 16$ gauss	0.14/0.29/0.43/0.58 mgauss/LSB	± 1 gauss

Para enviar los datos de la unidad inercial al controlador de la plataforma se usará el bus SPI. El Arduino Nano BLE 33 Sense actuará de master e iniciará la comunicación con el Arduino UNO, que actuará como slave.

5.3. Diseño y fabricación

Para el diseño de la estructura de la plataforma se ha empleado el software de modelado 3D Fusion 360 de Autodesk, en su versión gratuita para estudiantes y aficionados.

Para la fabricación de la estructura diseñada, se ha empleado la impresora 3D Ender 3 de Creality, junto al software Ultimaker Cura, que permite la configuración de parámetros y generación del código para la impresora.

5.3.1. Diseño 3D

La primera fase del diseño 3D en Fusion 360 consistió en introducir los modelos 3D de los componentes electrónicos que irán montados en la plataforma.

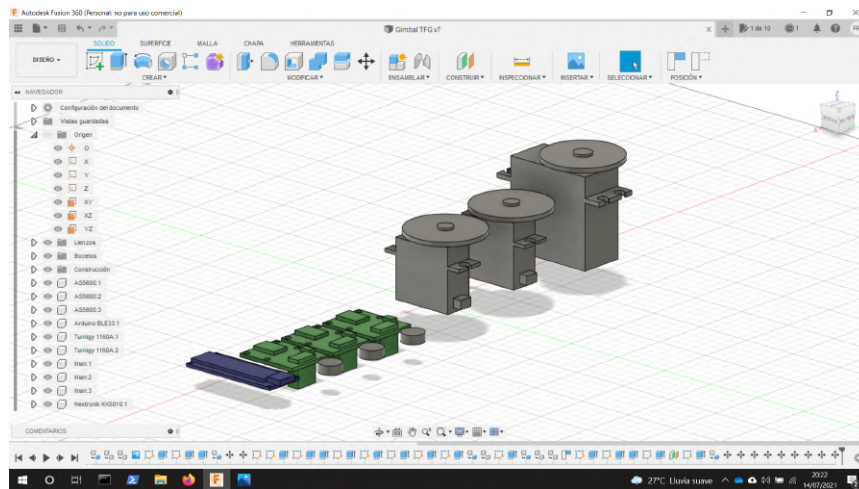


Figura 5.14: Captura de Fusion 360 con los componentes electrónicos

Una vez introducidos los modelos 3D de los componentes, estos han sido colocados en la posición espacial deseada. Las piezas de la estructura se han dibujado sobre estos componentes, generando una estructura que permita la sujeción de estos sin crear interferencias. Cabe mencionar, que la estructura se ha dimensionado sin tener en cuenta cálculos de estructuras. Sin embargo, no se han considerado necesarios debido al poco tamaño y peso de los componentes y a la elevada resistencia del material usado.

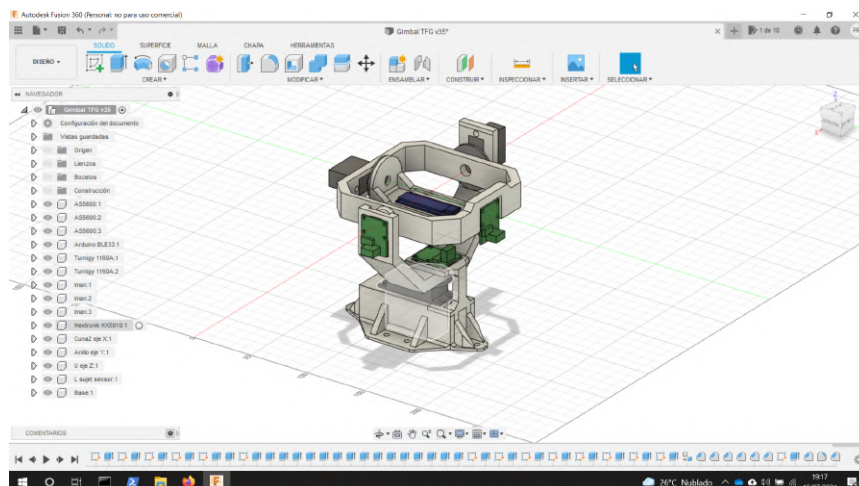


Figura 5.15: Captura de Fusion 360 con la plataforma diseñada

Cada eje está sujeto por un servomotor y por un imán del sensor AS5600, que lo sujetan al eje exterior sobre el que rota. En la figura 5.16 puede verse una sección del plano XY, donde son visibles las sujeciones a los servomotores e imanes. Los sensores AS5600 y sus imanes aparecen en rosa, mientras que los servomotores aparecen en verde.

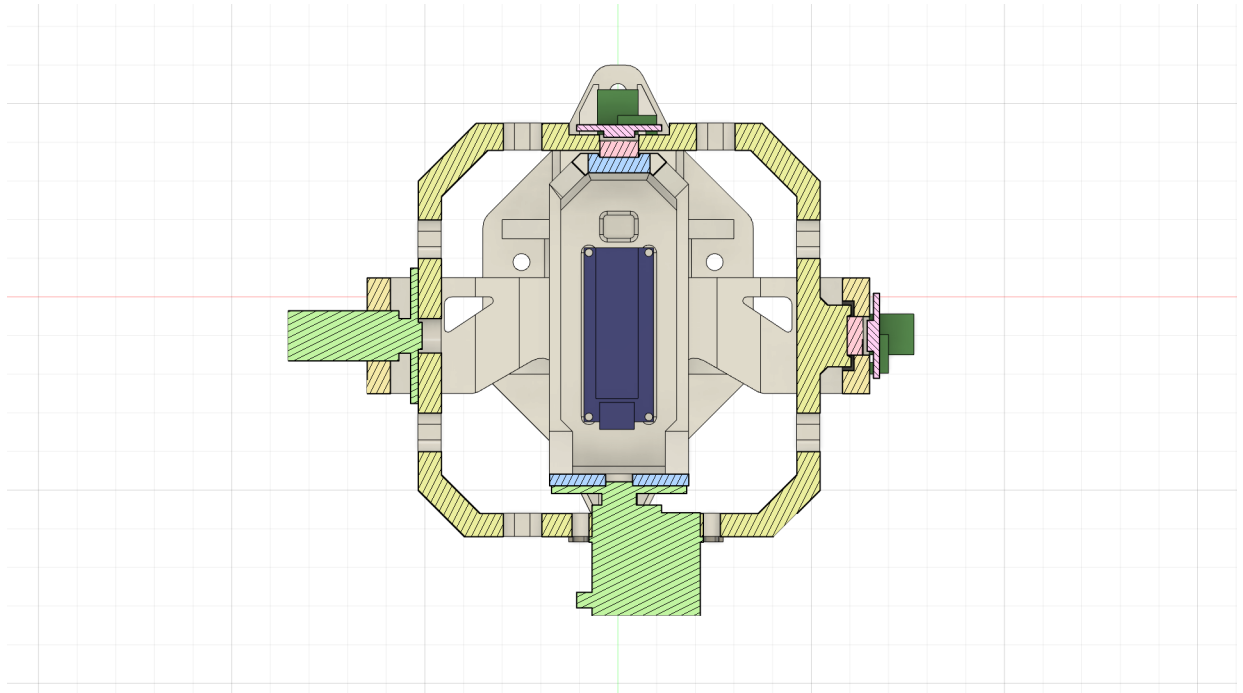


Figura 5.16: Sección XY de la plataforma giroscópica

Para la lectura del eje Z, se ha diseñado un brazo que permite colocar un sensor AS5600 sobre la base del soporte en U. En la figura 5.17 puede observarse una sección del plano XZ, donde se detalla la posición de este brazo.

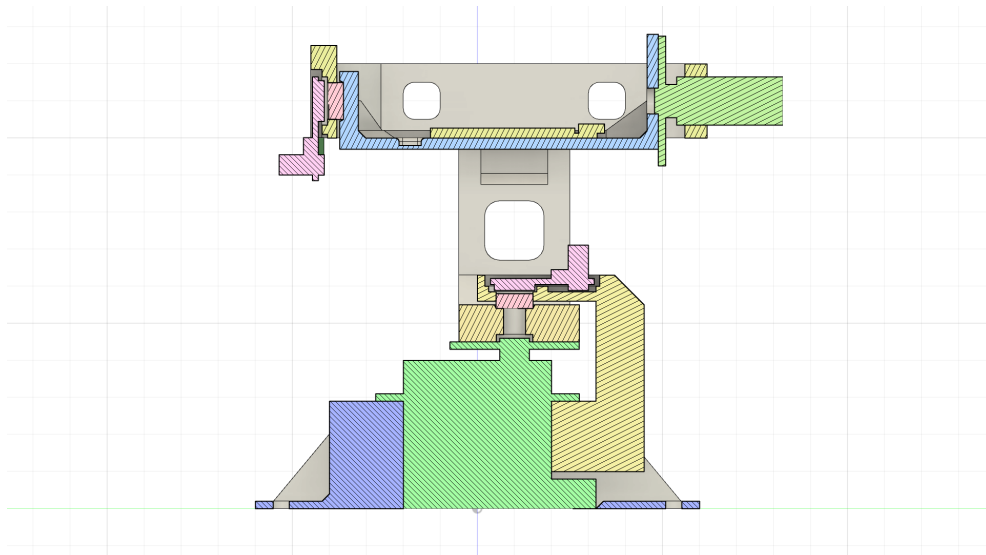


Figura 5.17: Sección XZ de la plataforma giroscópica

A continuación, se ha exportado cada una de las piezas diseñadas, para realizar un ensamble en Fusion 360. Se han vinculado con la herramienta de unión, atendiendo al tipo de restricción

que las diferentes piezas poseen entre si. Las uniones realizadas son de tipo fijo, deslizante y rotativo.

Posteriormente, se ha revisado las interferencias entre piezas, rotando los componentes y determinando los rangos máximos de movimiento. Los rangos de movimiento determinados son los siguientes:

- **Eje X (cuna):** No tiene ninguna restricción mecánica. Se ha establecido el recorrido en $\pm 90^\circ$, debido a la propia restricción del servomotor.
- **Eje Y (anillo):** Hace contacto con la parte inferior del soporte en U. Se ha establecido el recorrido en $\pm 45^\circ$ para asegurar que no entre en conflicto con el eje Y.
- **Eje Z (soporte en U):** Su giro está limitado por el soporte del sensor Z. Se ha rebajado parte de la base para permitir un mayor rango de giro. El rango establecido es de $\pm 60^\circ$.

En la imagen 5.18 se puede observar el diseño final de la plataforma con sus ejes rotados.

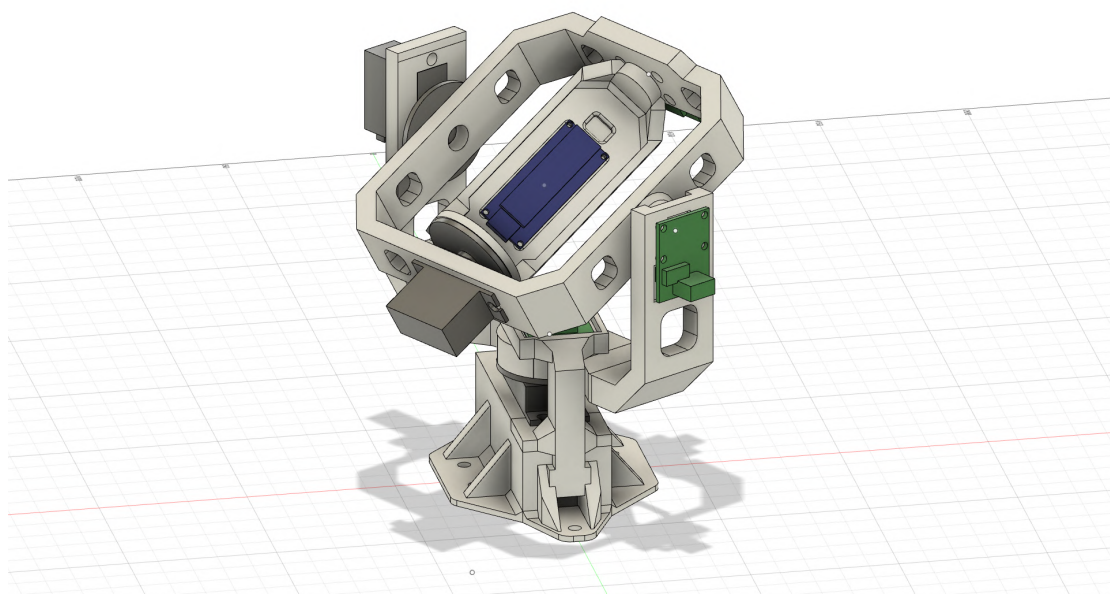


Figura 5.18: Vista 3D de la plataforma giroscópica con las piezas rotadas

En el anexo A se incluyen los planos detallados de cada pieza. Se ha añadido un procedimiento de montaje en la sección 5.4.1.

5.3.2. Impresión 3D

Para la impresión de las piezas se ha empleado una impresora 3D de deposición de plástico fundido o FDM (del inglés, Fused Deposition Modeling). La impresora utilizada es una Ender 3 de la empresa Creality.

La Ender 3 es una impresora de bajo coste fabricada con perfiles de aluminio, rodamientos de goma y correas dentadas, excepto para el eje Z que emplea un tornillo de acero. Viene en un kit

de montaje que el comprador debe ensamblar y calibrar. Dispone de un volumen de impresión de 220x220x250 mm y una precisión máxima de 0.1 mm. Soporta una temperatura máxima de 250 °C para el extrusor y de 100 °C para la cama, lo que permite imprimir una gran variedad de materiales plásticos.



Figura 5.19: Ender 3 de Creality

Para la impresión de las piezas de la plataforma se ha escogido PLA de color blanco y amarillo. El filamento PLA (ácido poliláctico) es uno de los materiales de impresión 3D más populares para impresoras FDM, debido a su facilidad de impresión, lo que lo hace ideal para principiantes. Además, el PLA es el material más sostenible y seguro de los usados en impresión 3D FDM. Se fabrica con el almidón extraído principalmente del maíz, no emite ningún tipo de residuo tóxico durante su impresión y es biodegradable. El PLA posee buenas propiedades mecánicas, por lo que es ideal para la impresión de piezas que no necesiten de características muy exigentes.



Figura 5.20: Bobinas de filamento PLA utilizadas en impresión 3D

Para la conversión del modelo 3D diseñado en Fusion 360 a código comprensible por la impre-

sora (G-CODE), se ha empleado el software Ultimaker Cura. Este software permite ajustar una serie de parámetros como la posición y orientación de la pieza sobre la plataforma, el tamaño y las dimensiones de la pieza, el grosor de las paredes, el patrón y densidad del relleno y la inclusión o no de soportes para la correcta impresión de voladizos.

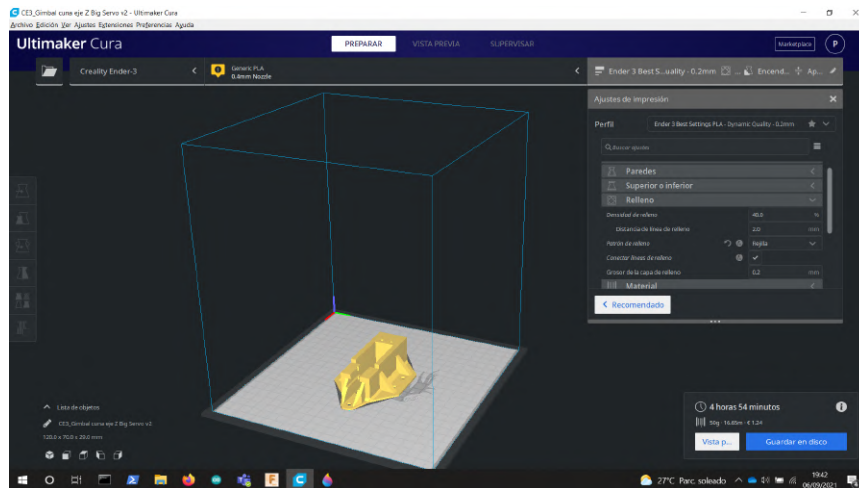


Figura 5.21: Preparación de la pieza base en Ultimaker Cura

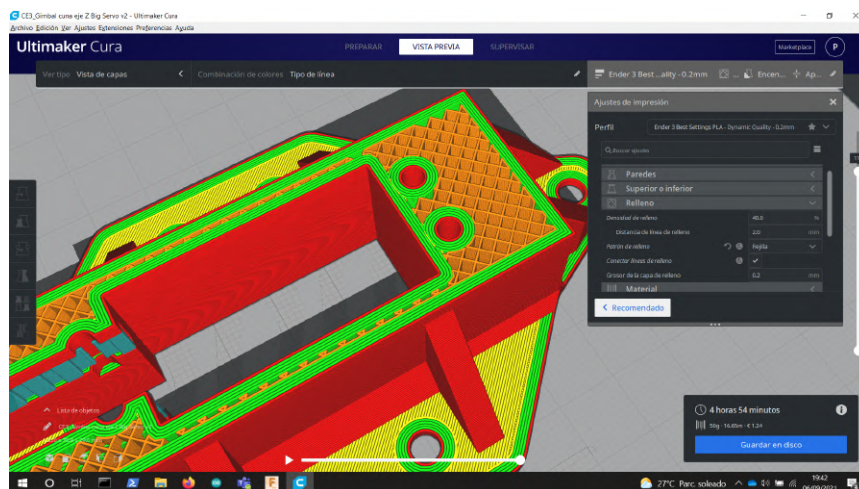


Figura 5.22: Revisión del G-CODE generador en Ultimaker Cura

5.4. Construcción de la plataforma

La construcción de la plataforma se ha dividido en dos partes: montaje de la estructura y conexionado del sistema eléctrico. El ensamblaje de los componentes se ha detallado en la sección 5.4.1. Además, se ha proporcionado un diagrama del circuito eléctrico en el anexo B.

5.4.1. Montaje de los componentes

Esta guía detalla la forma correcta de montar la plataforma en seis pasos.

- Paso 1) Se introduce el servomotor MG995 del eje Z en la base y se atornilla la base del brazo U del eje Z al eje del servomotor.

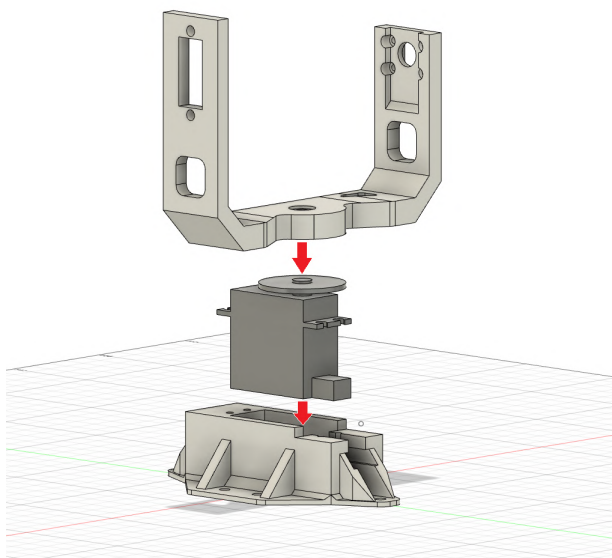


Figura 5.23: Paso 1 del montaje

- Paso 2) Primero se introduce el soporte del sensor AS5600 del eje Z en su ranura y se atornilla. A continuación, se fija el imán del sensor a su ranura con adhesivo. Por último, se atornilla el sensor al soporte y se ajusta la separación con el imán.

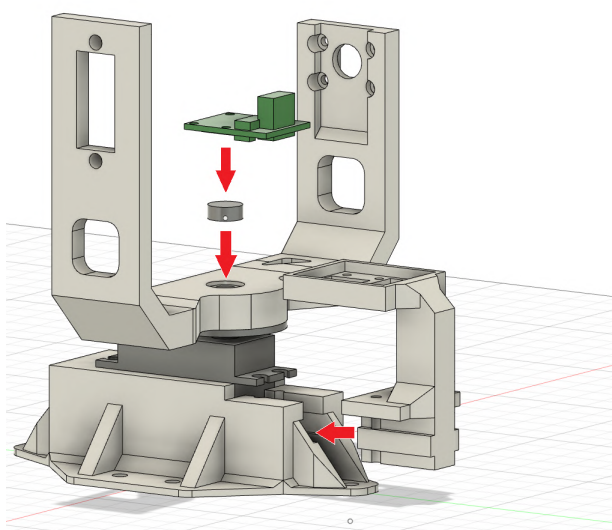


Figura 5.24: Paso 2 del montaje

- Paso 3) Primero se fija el imán del eje Y a la ranura del anillo. Después, se coloca el sensor AS5600 del eje Y en su soporte. Por último, se atornilla el soporte rotativo del motor del eje Y al anillo.

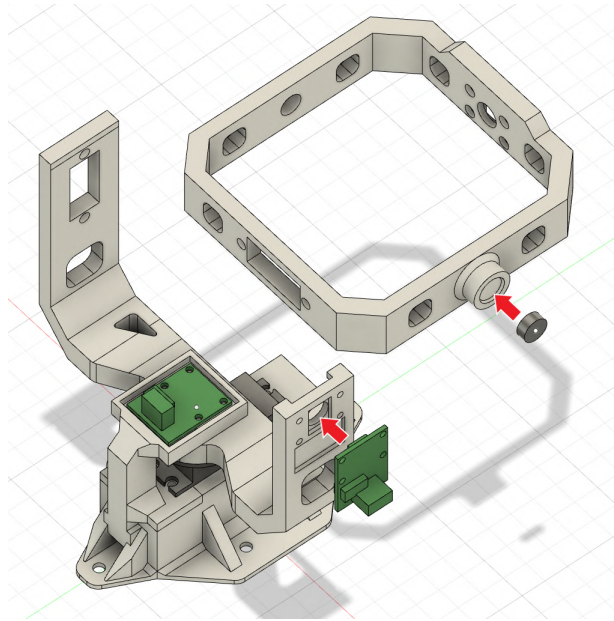


Figura 5.25: Paso 3 del montaje

- Paso 4) Se coloca el anillo del eje Y en su posición y se introduce el servomotor MG995 del eje Z, haciendo que encaje con el soporte rotativo previamente fijado al anillo.

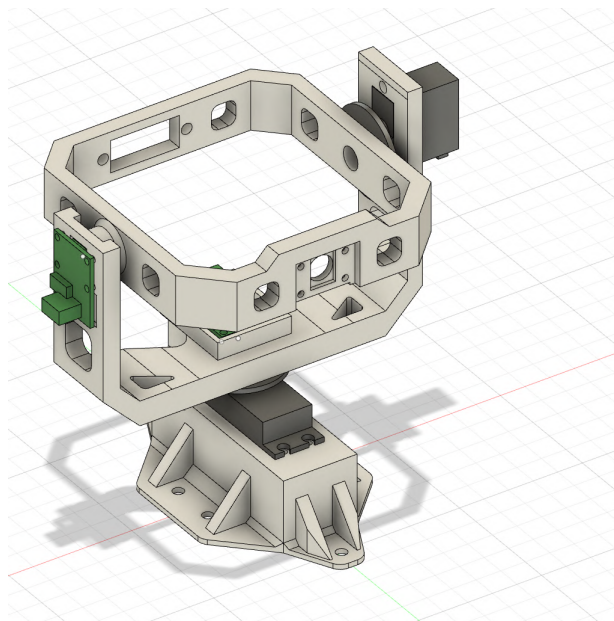


Figura 5.26: Paso 4 del montaje

- Paso 5) Se fija el imán del eje X a la ranura de la cuna. Se coloca el sensor AS5600 del eje X en su posición y se atornilla el soporte rotativo del eje X a la cuna.

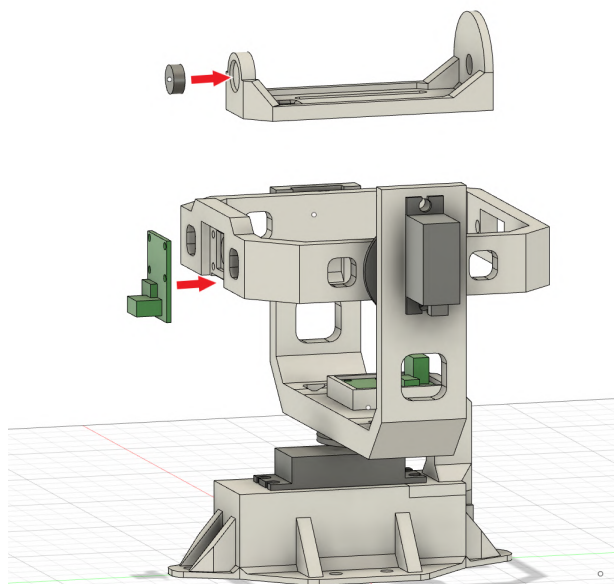


Figura 5.27: Paso 5 del montaje

- Paso 6) Se introduce el servomotor TR-1160A en su soporte, haciendo que encaje con el soporte rotativo previamente fijado. Por último, se coloca el Arduino BLE 33 en su posición sobre la cuna.

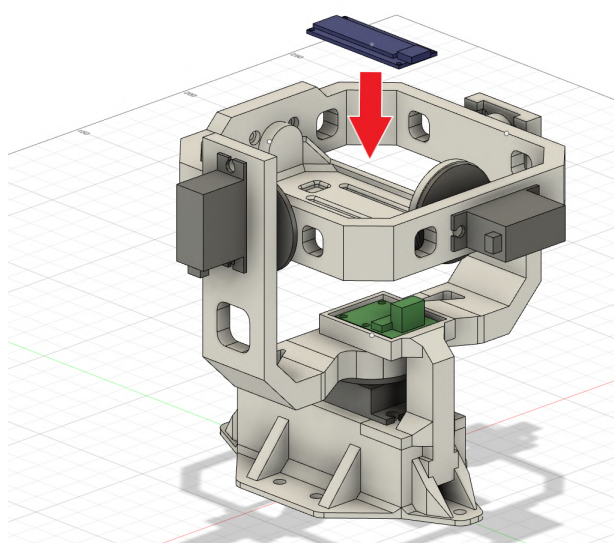


Figura 5.28: Paso 6 del montaje

5.4.2. Sistema eléctrico

El sistema eléctrico se puede dividir en dos partes: módulo de control y cableado de la plataforma. El esquema eléctrico completo está disponible en el anexo B.

El módulo de control se ha fabricado sobre una placa de pruebas o protoboard. El módulo incluye una línea de alimentación filtrada para los servos, un multiplexor I2C TCA9548A, un microcontrolador Arduino UNO y conectores para los servomotores y los sensores de posición angular AS5600. Todo el cableado se ha realizado con cables puente para prototipos o jumpers.

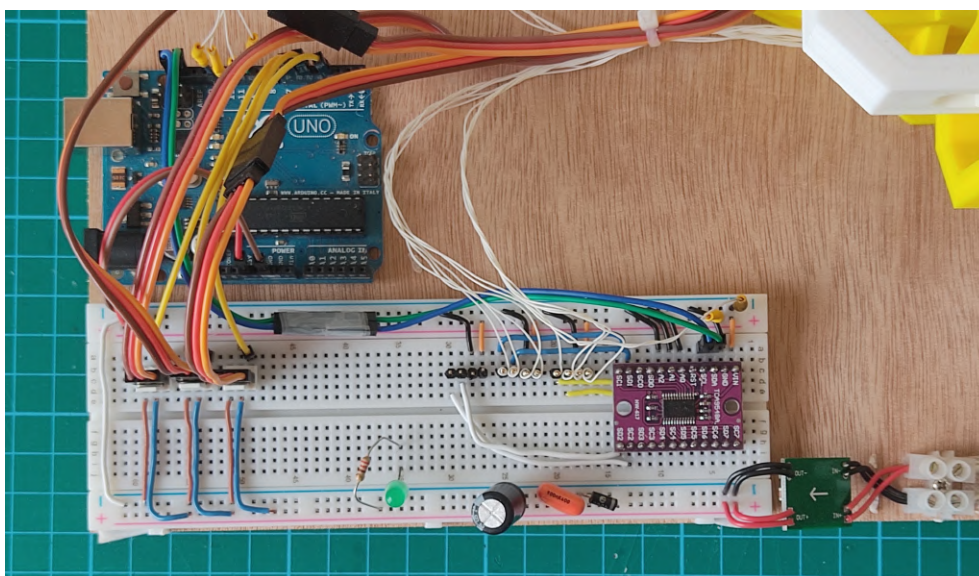


Figura 5.29: Fotografía del módulo de control

Para alimentar los servomotores, se ha utilizado una línea de tensión separada de la electrónica. Los servomotores presentan picos de consumo que pueden causar interferencias a otras cargas alimentadas a través la misma línea. Esta línea de tensión se compone de una fuente de alimentación de corriente continua de 12V, un convertor DC/DC de 12 a 5 voltios, un filtro formado por dos condensadores para absorber los picos de consumo y un diodo en configuración fly-back para proteger la línea de las corrientes de retorno provenientes del bobinado de los motores. Además, se ha añadido un interruptor para poder apagar los servomotores sin desactivar la electrónica y un LED para indicar si la línea está alimentada o no.

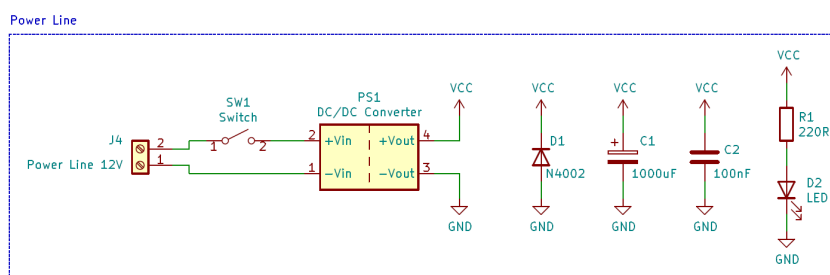


Figura 5.30: Diagrama eléctrico del circuito de alimentación

El resto del cableado de la plataforma está formado por las conexiones de los pines PWM del Arduino UNO a los cabezales de conexión de los servos y la conexión del bus I2C del Arduino al multiplexor I2C y a los cabezales de los sensores de posición angular.

Para conectar los servomotores a sus cabezales, se han utilizados cables extensores de servo-motor. Para conectar los sensores de ángulo y la unidad inercial, se ha empleado la técnica de enrollamiento o wire-wrap. Esta técnica consiste en unir dos contactos eléctricos mediante un fino cable enrollado. Para pelar, enrollar y quitar el cable se utiliza una herramienta denominada amarradora o wrapping tool.

5.4.3. Resultado final

El resultado final del montaje se refleja en la figura 5.31.

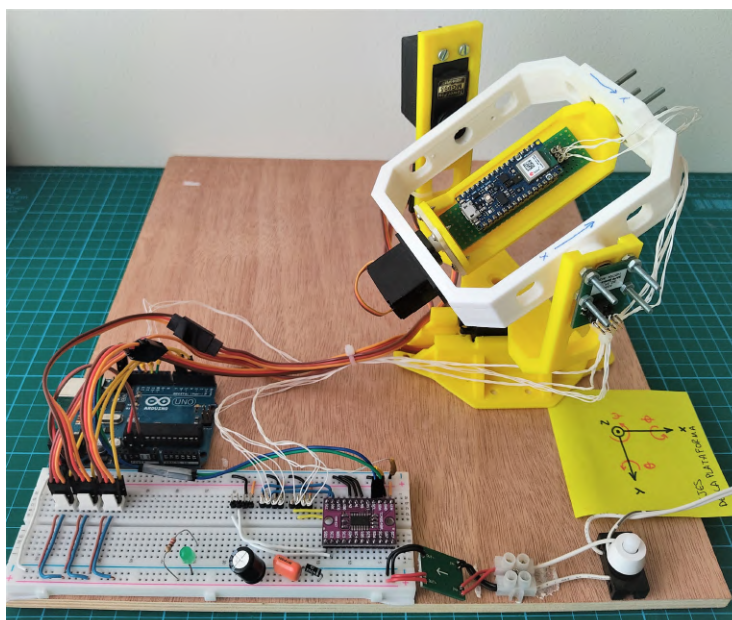


Figura 5.31: Fotografía de la plataforma

No se ha colocado el soporte del sensor de ángulo del eje Z, debido a que no se disponía de un tercer sensor a fecha del montaje. Posteriormente, se ha comprobado que no es necesaria una lectura precisa de la posición angular del eje Z, debido a las interferencias magnéticas comentadas en la sección 5.6.1.

Como se puede apreciar en la figura 5.32, el sistema de ejes coordenados utilizado para la lectura de los sensores y el control de los servomotores de la plataforma es distinto al sistema de referencia detallado en 2.1. El cambio de los ejes utilizados en la plataforma a ejes NED, será realizado como se detalla en la figura 5.33.

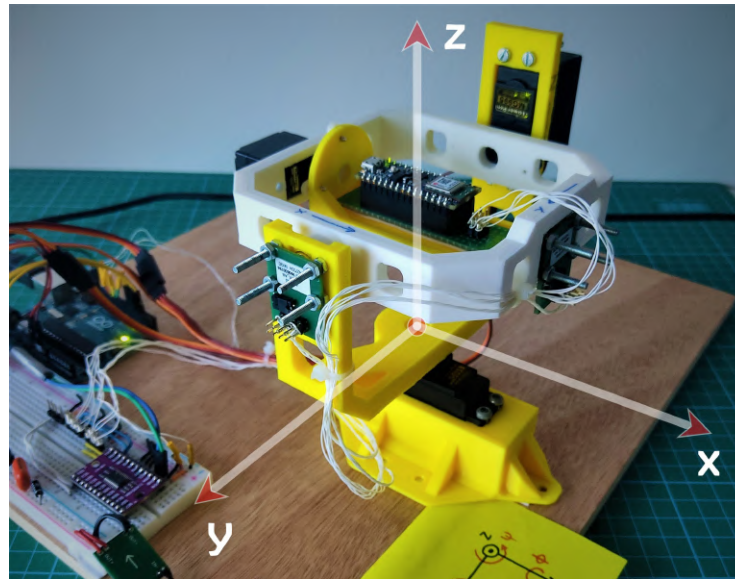


Figura 5.32: Ejes de la plataforma

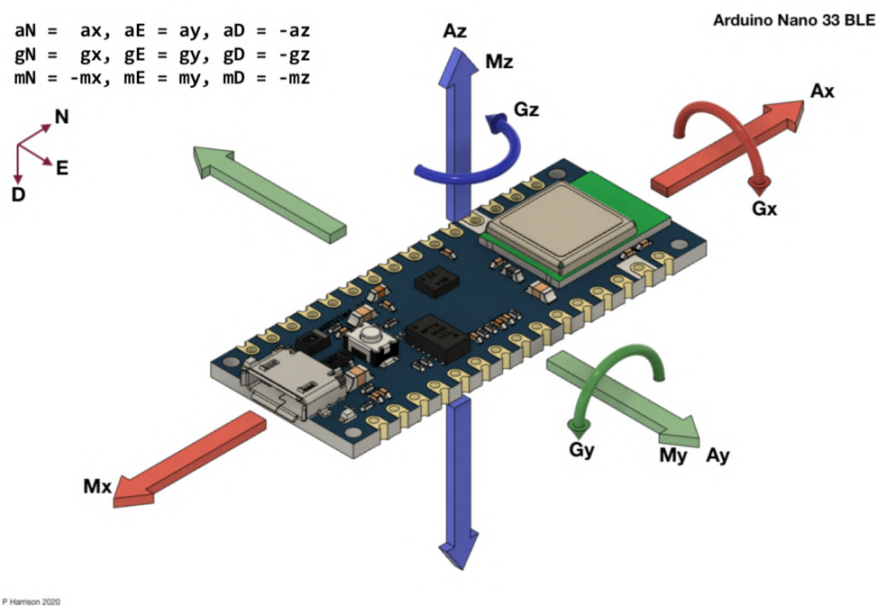


Figura 5.33: Ejes de los sensores del Arduino Nano BLE 33 Sense

Para expresar las lecturas del acelerómetro y del giróscopo en ejes NED, solamente es necesario cambiar el signo de la medida en el eje Z. Para cambiar los ejes de las lecturas del magnetómetro es necesario cambiar el signo del eje X y del eje Z, ya que los ejes del magnetómetro poseen distinta orientación al resto de sensores.

5.5. Programación de la plataforma

El funcionamiento de la plataforma requiere el uso de tres softwares: el código de lectura de la unidad inercial, el código de control de la plataforma y un programa para registrar y procesar los datos enviados por la plataforma.

5.5.1. Lectura de la unidad de medición inercial

El programa que realiza la lectura de la IMU se ejecuta sobre un Arduino Nano BLE 33 Sense. Este código realiza dos funciones: leer los datos de la IMU y enviar estos datos por SPI al controlador de la plataforma, un Arduino UNO.

La lectura de las medidas de la IMU se realiza a través de la librería propia `Arduino_LSM9DS1`. El envío de datos por bus SPI se realiza mediante la librería `SPI`. El código utilizado se detalla en el anexo C.

Para el envío de datos por SPI, se ha establecido al Arduino BLE 33 como master de la conexión. Se ha configurado la conexión para que se realice la transmisión de datos al Arduino UNO, siempre que se produzca una nueva lectura de la IMU.

5.5.2. Control de la plataforma

El programa de control de la plataforma es ejecutado sobre un Arduino UNO y realiza las siguientes funciones: lectura de los sensores de ángulo AS5600, control de la posición de los servomotores, recepción de la lecturas de la plataforma inercial (Arduino BLE 33) y envío de datos al ordenador conectado.

Para la lectura de los sensores AS5600, se ha empleado la librería `AS5600` y la librería `Wire`, que permite controlar el bus I2C de Arduino. Cabe mencionar, que la lectura del sensor del eje Z se ha sustituido por la posición angular enviada al servomotor del eje Z.

El control de los servomotores se realiza a través de la librería `Servo`. Para realizar un barrido de todas las posibles combinaciones de posiciones angulares, se han definido unas funciones sinusoidales de distintas frecuencias y amplitud. El ángulo comandado a cada servo es

$$\theta_x = A_x \sin 2\pi f_x + O_x \quad (5.1)$$

$$\theta_y = A_y \sin 2\pi f_y + O_y \quad (5.2)$$

$$\theta_z = A_z \sin 2\pi f_z + O_z \quad (5.3)$$

donde θ_x corresponde al ángulo enviado al servo del eje X, A_x a su rango de recorrido, f_x a la frecuencia de barrido deseada y O_x al ángulo central del servo.

La recepción de datos a través del SPI se ha realizado asignando una interrupción. Una interrupción es un mecanismo que permite asociar una función ISR (del inglés, Interrupt Service Routine) a un evento. Cuando este evento ocurre, el procesador sale del flujo normal del programa y ejecuta la función ISR asociada. Cuando la ejecución de la función ISR finaliza, se vuelve al flujo normal del programa. En este caso, se ha asociado la activación de la línea SS del SPI con una función que toma los datos de la IMU, los junta con los datos tomados de los sensores AS5600 en un mismo paquete de datos y los envía al ordenador conectado a través del puerto Serial (UART).

La ejecución de un programa en Arduino consta de dos fase: el setup y el loop. El setup es la parte del código que se ejecuta una sola vez, al iniciar el microcontrolador. El loop es el código ejecutado en bucle y de forma indefinida, después de finalizar el setup.

La primera parte del setup programado inicializa los buses de datos. A continuación, se colocan los servomotores en su posición central y se comprueba el estado de los sensores AS5600. Si todo es correcto, se envían 1000 muestras de datos en estático, para permitir inicializar y calibrar los algoritmos. Una vez enviadas estas muestras, se inicia el loop.

Durante la ejecución del loop, se calcula y comanda la posición angular de cada servomotor, se realiza la lectura de los sensores AS5600 y se envían los datos por UART. La recepción de datos del SPI se realiza a través de una interrupción y mantiene la lectura actualizada sin necesidad de llamar ninguna función desde el loop.

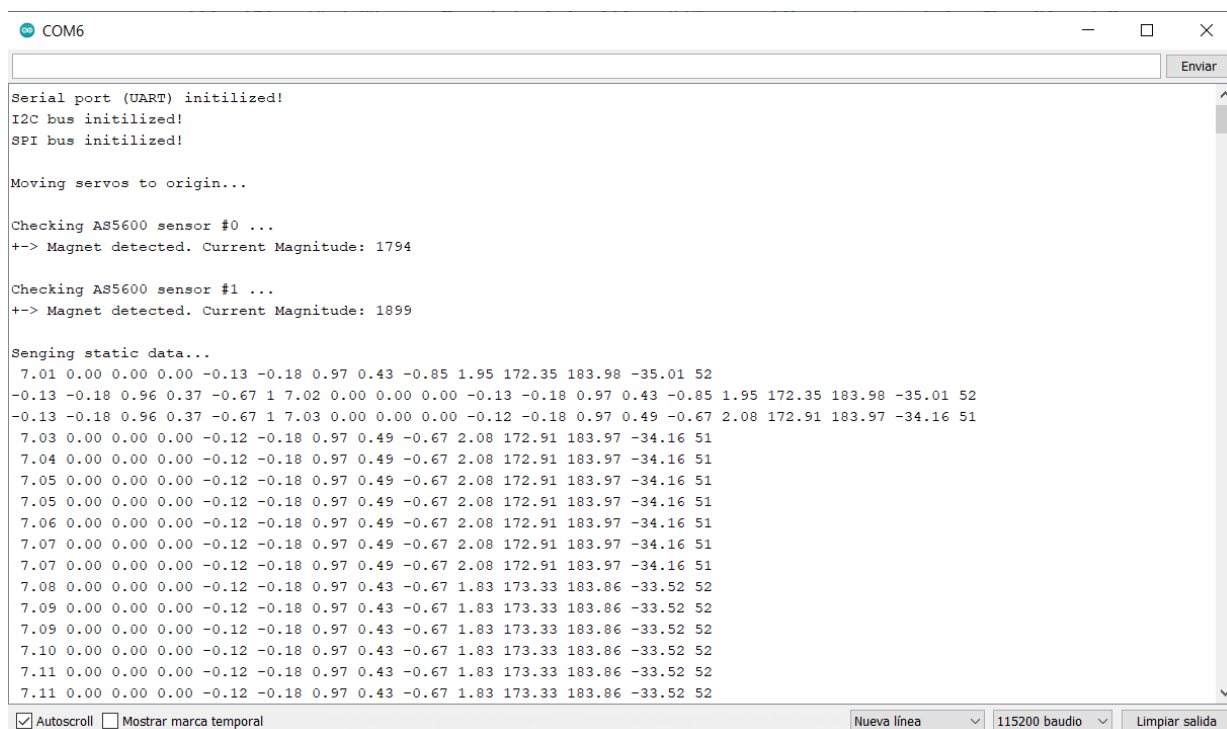
En la tabla 5.3 se detalla el formato del paquete de datos enviado por el Arduino UNO al ordenador conectado.

Tabla 5.3: Estructura del paquete de datos enviado por la plataforma

Nº	Dato	Unidades	Formato	Descripción
1	t	s (segundos)	float	Tiempo de programa del Arduino UNO
2	tx	deg (grados)	float	Medida del sensor AS5600 del eje X
3	ty	deg (grados)	float	Medida del sensor AS5600 del eje Y
4	tz	deg (grados)	float	Medida del sensor AS5600 del eje Z
5	ax	g (mili Gs)	float	Medida del acelerómetro del eje X
6	ay	g (mili Gs)	float	Medida del acelerómetro del eje Y
7	az	g (mili Gs)	float	Medida del acelerómetro del eje Z
8	gx	dps (grados/segundo)	float	Medida del giróscopo del eje X
9	gy	dps (grados/segundo)	float	Medida del giróscopo del eje Y
10	gz	dps (grados/segundo)	float	Medida del giróscopo del eje Z
11	mx	uT (micro Tesla)	float	Medida del magnetómetro del eje X
12	my	uT (micro Tesla)	float	Medida del magnetómetro del eje Y
13	mz	uT (micro Tesla)	float	Medida del magnetómetro del eje Z
14	dt	ms (mili-segundos)	integer	Periodo de muestreo de la IMU

En la imagen 5.34 se puede observar una captura del puerto serial de Arduino, al momento de iniciar la plataforma. Nótese, que los ángulos leídos de los sensores AS5600 son nulos, debido a que estas muestras corresponden a las 1000 primeras enviadas en estático para la calibración de los sensores.

El envío de datos se realiza con un baudrate de 115200 y se logra una tasa de datos media de una muestra cada 53 ms, es decir, se logran enviar aproximadamente 19 muestras por segundo. El envío se realiza como cadena de texto compuesta de caracteres ASCII.



```

Serial port (UART) initialized!
I2C bus initialized!
SPI bus initialized!

Moving servos to origin...

Checking AS5600 sensor #0 ...
+-> Magnet detected. Current Magnitude: 1794

Checking AS5600 sensor #1 ...
+-> Magnet detected. Current Magnitude: 1899

Sensing static data...
7.01 0.00 0.00 0.00 -0.13 -0.18 0.97 0.43 -0.85 1.95 172.35 183.98 -35.01 52
-0.13 -0.18 0.96 0.37 -0.67 1 7.02 0.00 0.00 0.00 -0.13 -0.18 0.97 0.43 -0.85 1.95 172.35 183.98 -35.01 52
-0.13 -0.18 0.96 0.37 -0.67 1 7.03 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.03 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.04 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.05 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.05 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.06 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.07 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.07 0.00 0.00 0.00 -0.12 -0.18 0.97 0.49 -0.67 2.08 172.91 183.97 -34.16 51
7.08 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
7.09 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
7.09 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
7.10 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
7.11 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
7.11 0.00 0.00 0.00 -0.12 -0.18 0.97 0.43 -0.67 1.83 173.33 183.86 -33.52 52
  
```

Figura 5.34: Captura del puerto serie de Arduino

Puede observarse que algunas líneas son erróneas. Sin embargo, esto no es problema, pues el programa de adquisición de datos realizado en MATLAB®, es capaz de descartar las líneas de datos erróneas.

5.5.3. Adquisición de datos

Para la adquisición de datos se ha empleado un programa en MATLAB®. Este programa utiliza la función `serialport()` para iniciar la comunicación y la función `str2num()` para convertir las cadenas de texto recibidas a números.

Las primeras líneas de datos corresponden a comprobaciones de inicialización del microcontrolador. Cada cierto tiempo se reciben líneas de datos corruptas debidas a ruidos en los buses de datos. Para solventar este problema se ha dotado al programa en MATLAB® de varios sistemas para comprobar que la línea sea correcta, como un método try-catch para excluir fallos de la función `str2num()`, así como, comprobar que las lecturas recibidas estén dentro del rango del sensor.

Al finalizar la toma de datos, se cierra el puerto serie y se exportan a un fichero para su posterior análisis.

Para realizar los ensayos, la plataforma ha sido orientada hacia el norte, de este modo el ángulo estimado por el magnetómetro coincide con el ángulo rotado por el eje Z. Esta disposición puede observarse en la figura 5.35.

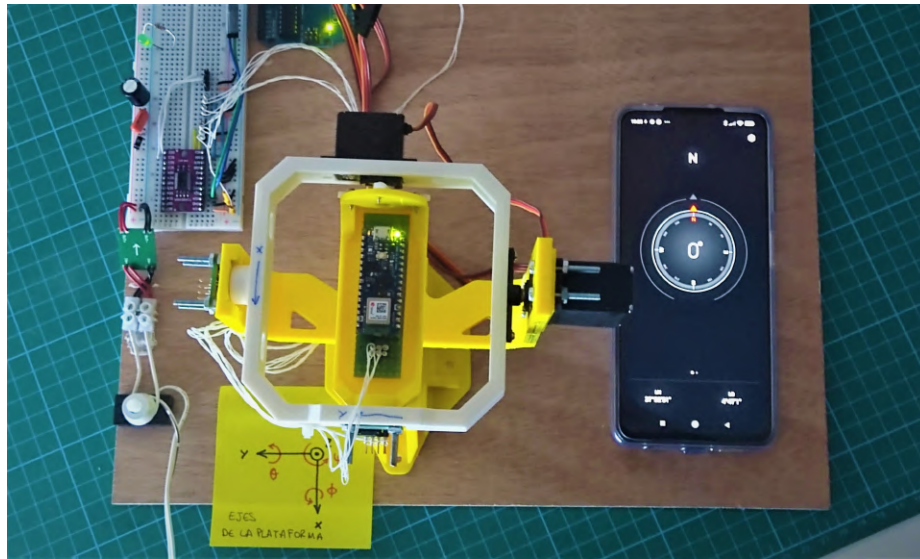


Figura 5.35: Plataforma giroscópica orientada al norte

5.6. Consideraciones

Una vez terminado el montaje y programación de la plataforma, se han realizado diversas tomas de datos, variando parámetros como el rango y velocidad de giro de los servomotores. Durante su funcionamiento, se han observado dos aspectos a tener en cuenta: error magnético y vibraciones mecánicas.

5.6.1. Error magnético

El campo magnético generado por los servomotores interfiere con el magnetómetro. Esto es debido a que, a la distancia a la que está situada la IMU de los motores, el campo magnético generado por los motores en funcionamiento es significativo en comparación con el terrestre. Esto impide al magnetómetro poder determinar la orientación respecto al norte magnético.

5.6.2. Vibraciones mecánicas

El giro proporcionado por los servomotores no es fluido. En algunos instantes, los servomotores presentan oscilaciones provocadas por el software interno de control. Además, al estar construido con un motor de corriente continua que cambia alternativamente su sentido de giro, para conservar la posición angular aparecen vibraciones que causan interferencias en la lectura de los sensores de la IMU.

6. Análisis de los datos y resultados

En este capítulo, se ha evaluado el desempeño de los cuatro algoritmos estudiados y se ha realizado una comparativa de la calidad de las estimaciones obtenidas.

Para poder evaluar la eficacia de cada algoritmo se han realizado dos ensayos de distintas características. Cada toma de datos está realizada a distintas velocidades de rotación de los ejes de la plataforma. La primera toma corresponde a oscilaciones de baja frecuencia, mientras que la segunda toma corresponde a rotaciones rápidas.

En las figuras 6.1 y 6.2 pueden observarse los datos tomados de la plataforma durante ambos ensayos.

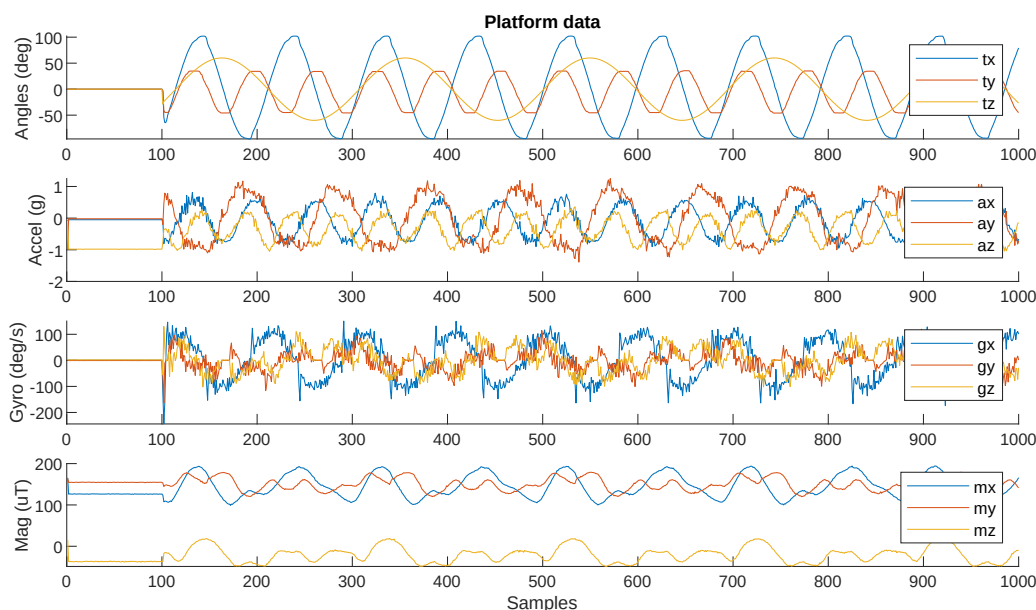


Figura 6.1: Datos adquiridos durante el ensayo lento

Las lecturas del giróscopo han aumentado de rango en el ensayo rápido respecto al ensayo lento, debido a que ha aumentado la velocidad de giro de cada eje. Las lecturas del acelerómetro y del magnetómetro no han cambiado de rango, como cabría esperar, pero se han vuelto más ruidosas.

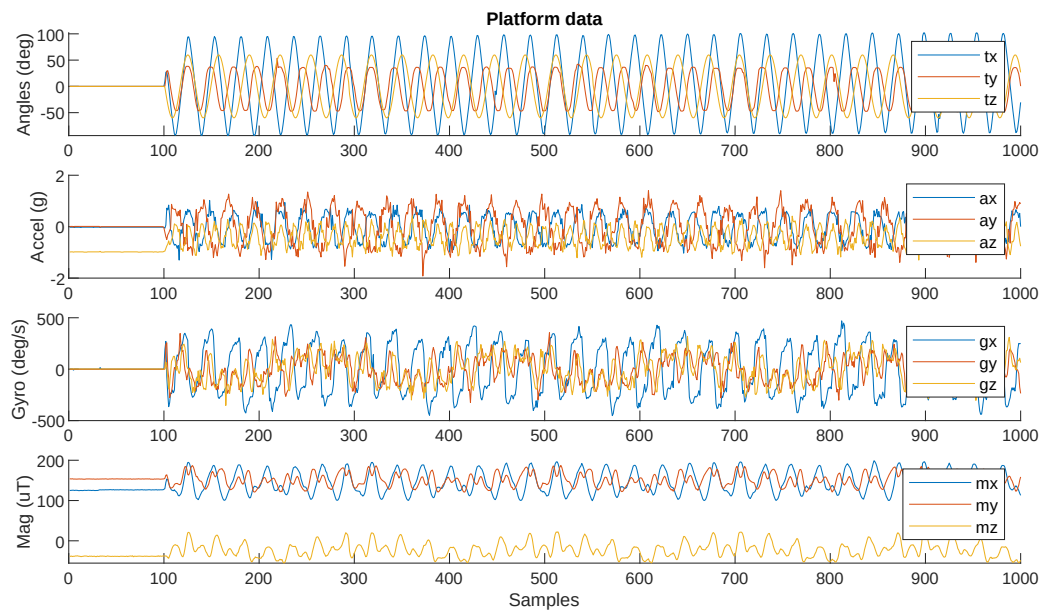


Figura 6.2: Datos adquiridos durante el ensayo rápido

Las lecturas de los ángulos de la plataforma también se han mantenido dentro del rango esperado. Para frecuencias mayores de oscilación, los servomotores no eran capaz de recorrer el rango completo, debido a que se alcanzaban sus velocidades de rotación máximas.

Cabe recordar, como se mencionó en la sección 5.4.3, que no se dispone del sensor de posición angular para el eje Z. En su lugar, se ha tomado el valor del ángulo comandado al servomotor.

6.1. Detección de la gravedad y el campo magnético

El algoritmo utilizado para estimar la orientación con las lecturas del acelerómetro y el magnetómetro produce buenos resultados en las estimaciones de pitch y roll. Sin embargo, la estimación del ángulo yaw no es satisfactoria. La divergencia entre el ángulo de yaw real y el estimado por el algoritmo se debe a las interferencias magnéticas causadas por los servomotores.

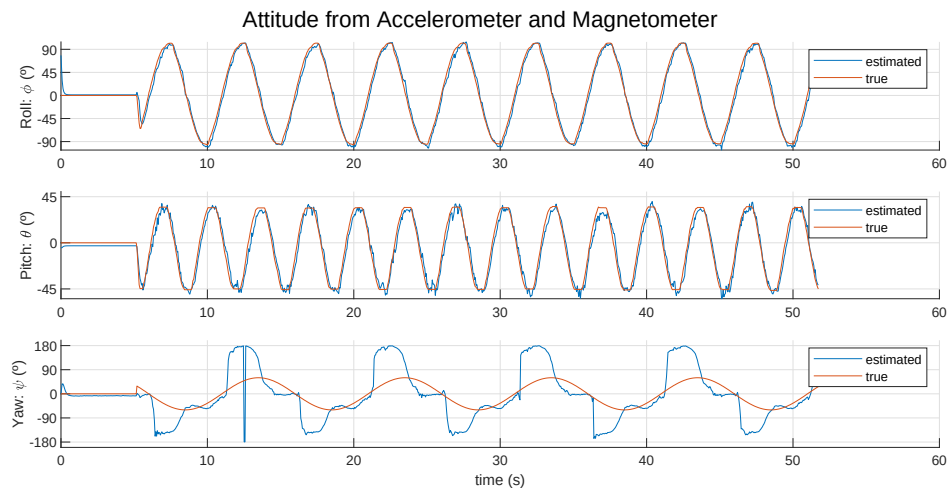


Figura 6.3: Estimaciones del acelerómetro y el magnetómetro para el ensayo lento

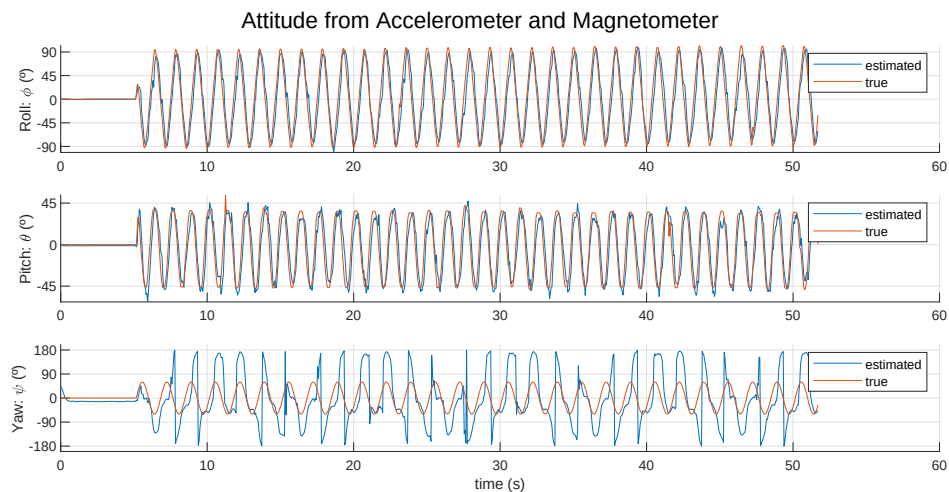


Figura 6.4: Estimaciones del acelerómetro y el magnetómetro para el ensayo rápido

6.2. Integración del giróscopo

El método de estimación por integración de la velocidad angular, presenta el problema de deriva del giróscopo. En el ensayo lento, el error se va acumulando de forma continua, mientras que en el ensayo rápido, el error hace oscilar la estimación fuera del valor real.

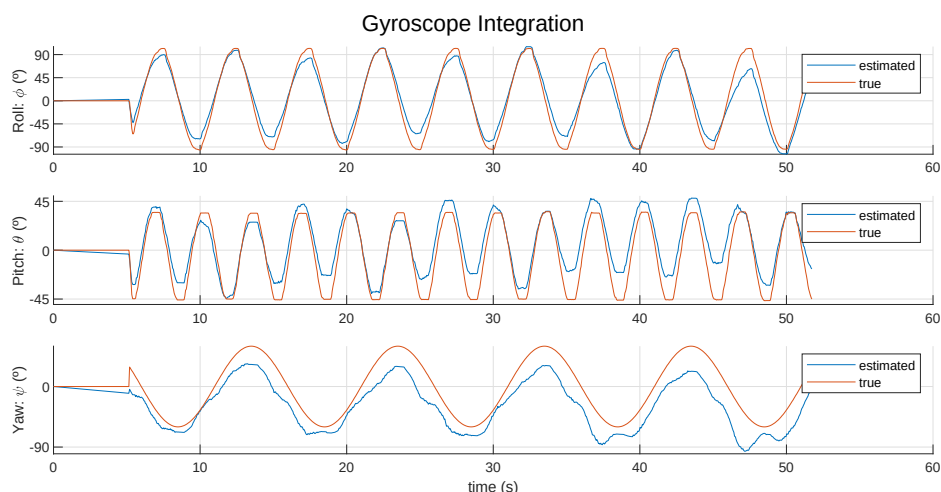


Figura 6.5: Estimaciones del giróscopo para el ensayo lento

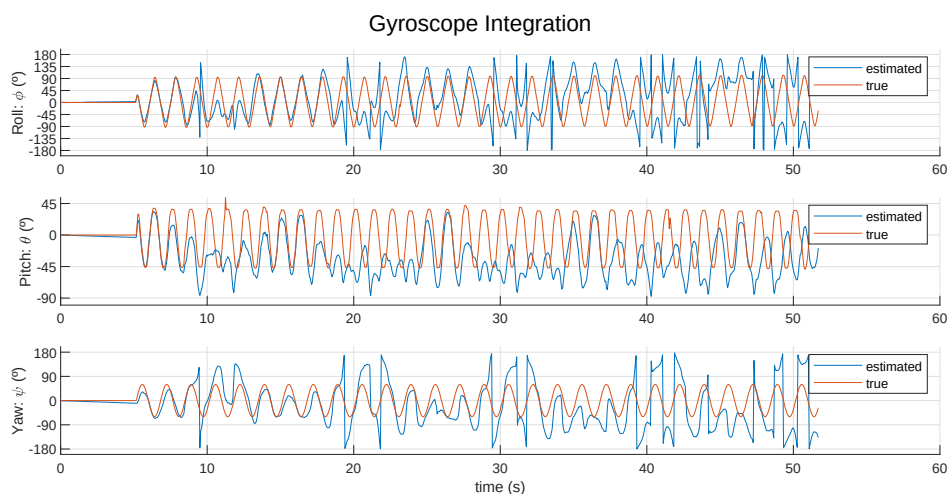


Figura 6.6: Estimaciones del giróscopo para el ensayo rápido

6.3. Filtro complementario

Las estimaciones obtenidas con el filtro complementario, son similares a las obtenidas mediante integración del giróscopo, pero ligeramente suavizadas por las lecturas del acelerómetro y el magnetómetro. Esto se debe a que la ganancia empleada en el filtro ($\alpha = 0.1$) otorga mayor peso a la predicción del giróscopo.

La ganancia de $\alpha = 0.1$ permite que el filtro actúe como un filtro paso bajo que pondera con mayor peso la estimación anterior y con menor peso la estimación actual de la orientación. La estimación anterior es corregida por la lectura del giróscopo, mientras que la estimación actual es tomada del acelerómetro y el magnetómetro.

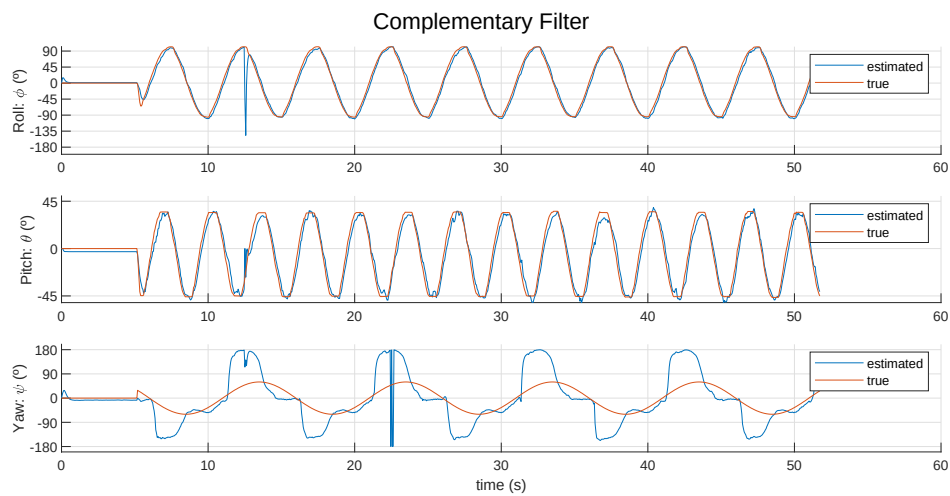


Figura 6.7: Estimaciones del filtro complementario para el ensayo lento

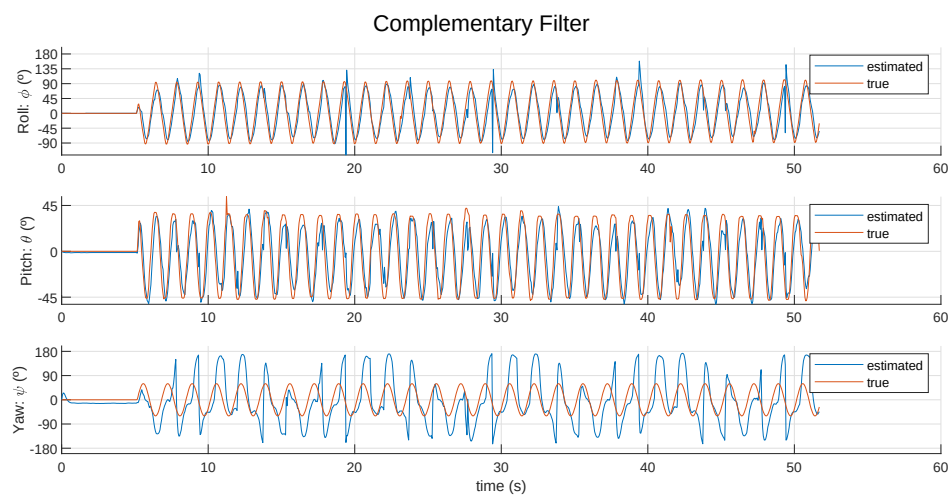


Figura 6.8: Estimaciones del filtro complementario para el ensayo rápido

6.4. Filtro de Kalman extendido

El filtro de Kalman extendido presenta la mejor estimación, aparentemente. En el ensayo lento es capaz de realizar buenas estimaciones de la orientación e incluso lidiar con las interferencias del magnetómetro, realizando una buen predicción del ángulo yaw. En el ensayo rápido, las estimaciones presentan mayor grado de error, al igual que ha ocurrido con el resto de filtros.

Para obtener un buen funcionamiento del filtro es necesario proporcionar la desviación típica del ruido de cada sensor. Esta desviación típica ha sido calculada a través del computo de la varianza de cada sensor para un ensayo estático. Los valores obtenidos han sido $\sigma_{acc} = 0.3$, $\sigma_{mag} = 0.5$ y $\sigma_{gyr} = 0.08$.

Las interferencias magnéticas que afectan al magnetómetro provocaban un error en la estimación del ángulo de yaw, pero también afectaban a la estimación de roll y pitch, debido a que las ecuaciones de observación están acopladas.

Para sintonizar correctamente el filtro se ha variado la desviación típica del magnetómetro hasta lograr la mejor estimación. También se ha modificado el valor de σ del giróscopo para mejorar el desempeño del filtro. No ha sido necesario modificar el valor del ruido del acelerómetro.

Finalmente, los valores de ruido utilizados han sido $\sigma_{acc} = 0.3$, $\sigma_{mag} = 20.0$ y $\sigma_{gyr} = 0.8$.

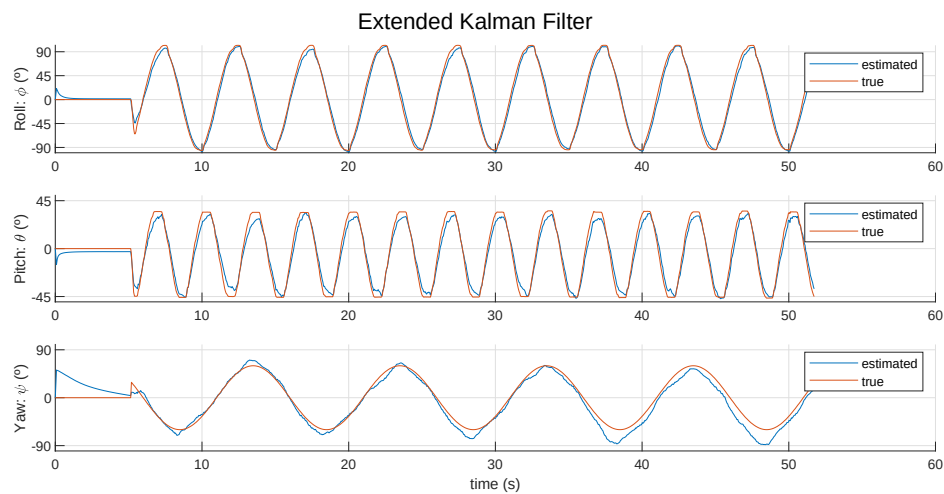


Figura 6.9: Estimaciones del filtro Kalman extendido para el ensayo lento

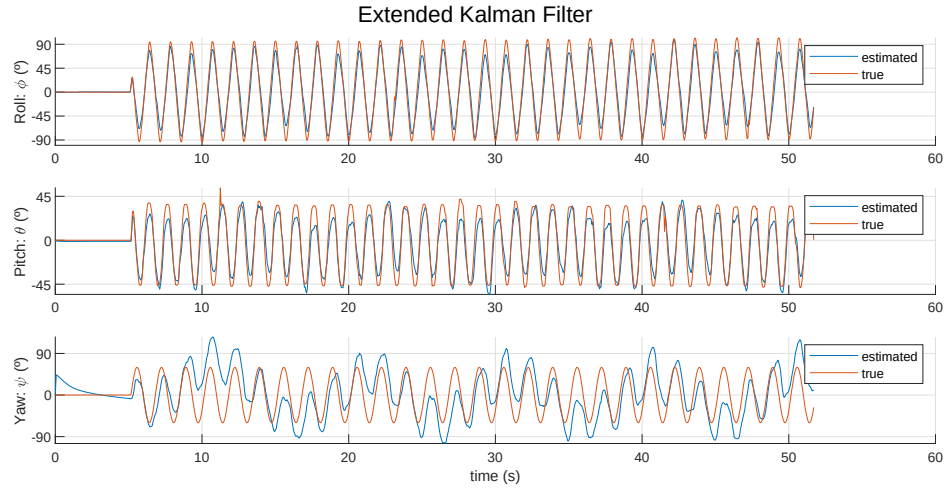


Figura 6.10: Estimaciones del filtro de Kalman extendido para el ensayo rápido

6.5. Comparativa

Para poder comparar de forma visual el desempeño de los algoritmos evaluados, se ha representado el error cometido en cada estimación. Este error $\Delta\theta$ se ha calculado como la diferencia entre el ángulo estimado $\hat{\theta}$ y el ángulo real θ .

$$\Delta\theta = \hat{\theta} - \theta \quad (6.1)$$

En las figuras 6.11 y 6.12 se ha representado el error cometido, para un fragmento del ensayo de 200 muestras.

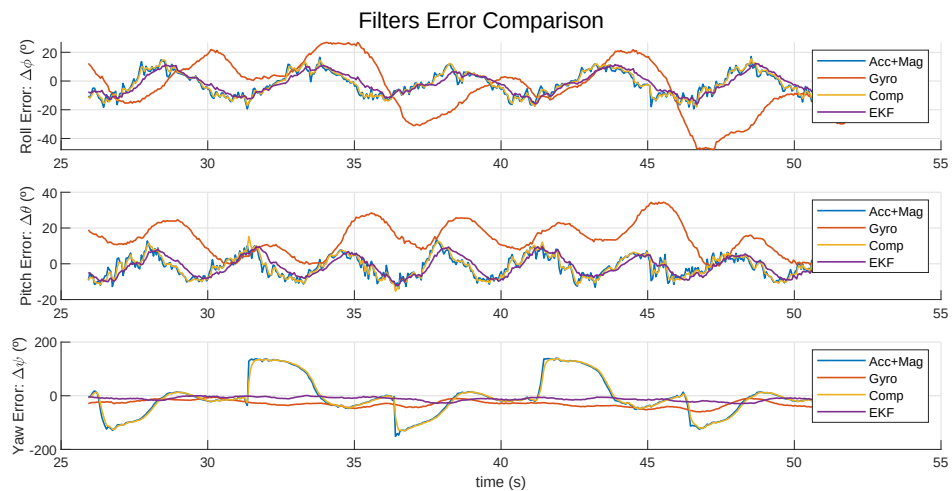


Figura 6.11: Errores en las estimaciones para el ensayo lento

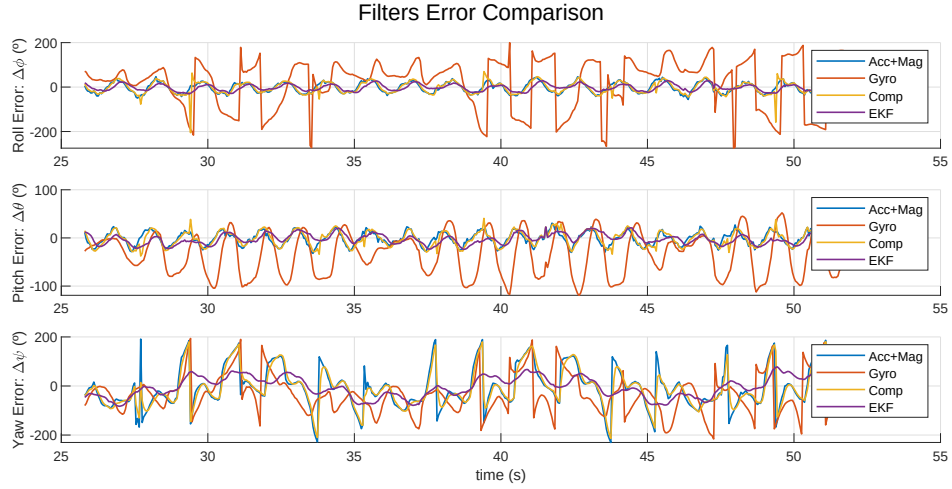


Figura 6.12: Errores en las estimaciones para el ensayo rápido

La gráfica 6.11 representa los errores cometidos durante el ensayo lento. Puede observarse que la mejor estimación es proporcionada por el filtro de Kalman (EKF), seguido del filtro Complementario (Comp) y el estimador de acelerómetro y magnetómetro (Acc+Mag). La peor estimación ha sido la proporcionada por el método de integración de la velocidad angular (Gyro), debido al gran error acumulado en la deriva. La estimación del ángulo de yaw no ha sido buena en ningún caso, debido a las fuertes interferencias magnéticas causadas por los servomotores. Sin embargo, el filtro de Kalman y el método por integración del giróscopo, han logrado acercarse al valor real de este ángulo.

En la gráfica 6.12 se representan los errores de estimación cometidos en el ensayo rápido. Los algoritmos que mejor estimación proporcionan coinciden con los comentados en el ensayo lento, siendo más significativo el error cometido por integración del giróscopo y la incapacidad de los estimadores de determinar el ángulo de yaw.

Para comparar de forma cuantitativa los cuatro algoritmos empleados se utilizará el error cuadrático medio. El error cuadrático medio o RMSE (del inglés, Root Mean Square Error) es una métrica empleada para estimar la calidad del ajuste de una serie de datos a un modelo.

La fórmula utilizada para calcular el error cuadrático medio es

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2} \quad (6.2)$$

donde $\hat{y}_i$ son los valores del ángulo estimado, y_i son los valores del ángulo medido y N el número total de muestras.

En las tablas 6.1 y 6.2 se pueden observar los valores de RMSE calculados para las distintas estimaciones de los diferentes filtros.

Tabla 6.1: Valores de RMSE para el ensayo lento

Algoritmo	RMSE roll (°)	RMSE pitch (°)	RMSE yaw (°)
Acc + Mag	8.675	5.946	68.053
Gyr Integ	16.485	13.580	27.879
Comp Filt	10.524	8.845	56.341
EKF	7.241	5.596	13.112

Tabla 6.2: Valores de RMSE para el ensayo rápido

Algoritmo	RMSE roll (°)	RMSE pitch (°)	RMSE yaw (°)
Acc + Mag	21.920	13.828	76.706
Gyr Integ	89.349	47.001	68.527
Comp Filt	23.378	18.509	52.452
EKF	13.641	10.006	41.959

Para ambos ensayos, el algoritmo que mejor estimación proporciona es el filtro Kalman Extendido (EKF), seguido del estimador de acelerómetro y magnetómetro (Acc + Mag) y del filtro Complementario (Comp Filt). La estimación proporcionada por integración del giróscopo (Gyr Integ) es la que presenta mayor error.

El filtro complementario presenta mayor error que el método de detección de gravedad y campo magnético, debido a que da mayor peso a la estimación proporcionada por el giróscopo. Se ha probado a aumentar la ganancia del filtro Complementario para que dé más peso al acelerómetro y al magnetómetro en la ponderación. Sin embargo, solo se ha logrado acercar el resultado al proporcionado por el método de detección de gravedad y campo magnético, pero no se ha logrado superar la estimación de este.

Se puede afirmar que la estimación del giróscopo no es eficaz y que empeora el comportamiento de los algoritmos que lo emplean. Sin embargo, el filtro de Kalman extendido, al usar la integración de la velocidad angular en cortos periodos de tiempo para propagar el estado del sistema, permite un correcto uso de las lecturas del giróscopo.

En conclusión, ningún estimador ha proporcionado una estimación satisfactoria del ángulo de guiñada (yaw). Los únicos sensores que pueden usarse para determinar este ángulo son el magnetómetro y el giróscopo, ambos señalados como muy ruidosos y poco fiables, si son usados de forma independiente.

7. Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este trabajo se ha realizado un estudio sobre los principales algoritmos utilizados en sistemas de orientación inercial. El estudio se ha dividido en dos objetivos: el desarrollo e implementación de cuatro algoritmos estimadores, así como el diseño y construcción de una plataforma giroscópica para el ensayo de sensores inerciales.

En primer lugar, se han estudiado dos métodos utilizados para representar la orientación: los ángulos de euler y los cuaterniones. Se ha escogido la representación en cuaterniones, debido a que no presentan bloque del gimbal y a su bajo coste computacional.

A continuación, se ha modelado el comportamiento de tres tipos de sensores inerciales MEMS: acelerómetro, giroscopo y magnetómetro. Se han estudiado sus ventajas y limitaciones, así como el ruido asociado a cada tipo de sensor

Tras el modelado de los sensores, se han detallado las ecuaciones de los cuatro algoritmos a evaluar: detección de la gravedad y el campo magnético, integración del giroscopo, filtro complementario y filtro extendido de Kalman. Estos algoritmos han sido implementados en MATLAB®.

El filtro de Kalman implementado se ha basado en un algoritmo que combina las lecturas del acelerómetro y el giroscopo para estimar la orientación. Este algoritmo ha sido ampliado para incluir las lecturas del magnetómetro y el modelo de ruido del giroscopo de otra implementación del filtro más compleja. Permitiendo así, determinar el ángulo de yaw y mejorar la estimación de la orientación proporcionada.

La inclusión de la plataforma giroscópica en el trabajo ha surgido de la necesidad de comparar los algoritmos estudiados de forma cuantitativa. La construcción de la estructura ha requerido el uso de diversas herramientas de diseño y fabricación 3D. Para el desarrollo del sistema de control de la plataforma, se ha trabajado con una amplia variedad de microcontroladores, sensores y buses de datos. Se ha detallado todo el hardware y software empleado en su construcción, así como el proceso completo de fabricación y ensamblaje.

La plataforma construida cumple con los objetivos planteados. Sin embargo, presenta una serie de inconvenientes, como las vibraciones e interferencias electromagnéticas causadas por los servomotores. Estas interferencias electromagnéticas, han impedido el correcto uso del magnetómetro.

Se han evaluado y comparado las estimaciones de los algoritmos implementados. Para la evaluación se han utilizado las lecturas de la IMU, tomadas de diversos ensayos realizados con la plataforma. Ningún algoritmo ha proporcionado estimaciones aceptables del ángulo de guiñada (yaw), por lo que se han comparado atendiendo a los ángulos de pitch y roll estimados. Como

cabía esperar, el filtro de Kalman extendido ha obtenido la mejor estimación, seguido del método por detección de la gravedad y el campo magnético.

El filtro de Kalman ha demostrado ser una poderosa herramienta matemática. Correctamente sintonizado, proporciona la mejor estimación e incluso logra hacer una buena predicción del ángulo de yaw. Ha sido capaz de compensar las lecturas con interferencias del magnetómetro, utilizando el giróscopo de forma satisfactoria.

7.2. Trabajo Futuro

La plataforma de pruebas para sistemas de orientación inercial ha demostrado ser de gran utilidad. Sin embargo, posee ciertas limitaciones que pueden ser mejoradas.

Una segunda versión de esta plataforma podría contar con las siguientes mejoras:

- Aumentar el tamaño, fuerza y velocidad de giro de la plataforma, para poder testar unidades inerciales de mayor tamaño y realizar ensayos más exigentes.
- Permitir la rotación continua de los ejes, para recorrer todos los rangos de posiciones angulares.
- Solucionar el problema de las interferencias magnéticas para poder disponer de una buena lectura del magnetómetro.
- Mejorar la experiencia de uso de la plataforma, haciendo que se puedan cambiar parámetros del ensayo sin necesidad de recompilar el código de control.
- Introducir aceleraciones externas para poder evaluar los algoritmos en entornos en los que sea necesario compensar el acelerómetro.

Para conseguir estas características, se plantea realizar las siguientes tareas:

- Reemplazar los servomotores por motores paso a paso, que permiten la rotación continua del eje y poseen mayor fuerza y velocidad.
- Utilizar motores con aislamiento magnético o añadir un encapsulado tipo jaula de Faraday, para no causar interferencias en el magnetómetro.
- Utilizar juntas eléctricas rotativas, para poder conectar los cables de un eje a otro, sin causar restricciones al movimiento.
- Diseñar una interfaz de control gráfica, que permita ajustar los parámetros del ensayo y muestre gráficas en tiempo real de las lecturas tomadas de la unidad inercial.
- Montar la plataforma sobre un vehículo como un coche o un dron, para poder someterla a fuerzas externas como la fuerza centrífuga que aparece durante un giro o un viraje. También, podría construirse un brazo centrífugo o una plataforma oscilatoria para introducir este tipo de aceleraciones.

Otra línea para mejorar los resultados de este trabajo, sería la evaluación de un mayor número de algoritmos. El filtro Kalman es el estimador más usado en sistemas autónomos. Sin embargo, existen otros algoritmos muy utilizados por su bajo coste computacional como son el filtro de Mahony y el filtro de Madgwick, entre otros.

Bibliografía

- [1] D. Tazartes, "An historical perspective on inertial navigation systems," in *2014 international symposium on inertial sensors and systems (ISISS)*, pp. 1–5, IEEE, 2014.
- [2] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012.
- [3] C. Paz, G. Infante, J. B. Carballo, F. D. Báez, and C. Cavenio, "Implementación de un filtro extendido de kalman para la estimación de la orientación de un uav utilizando el estándar cmsis," in *Proc. V Congreso de Microelectrónica Aplicada-UEA2014*, 2014.
- [4] V. Apostolyuk, "Theory and design of micromechanical vibratory gyroscopes," in *MEMS/-NEMS*, pp. 173–195, Springer, 2006.
- [5] L. Llamas, "Filtro paso bajo y paso alto exponencial (ema) en arduino." <https://www.luisllamas.es/arduino-paso-bajo-exponencial/>, 2017.
- [6] T. Ozyagcilar, "Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors," *Freescale semiconductor, AN*, vol. 4248, 2012.
- [7] T. Ozyagcilar, "Calibrating an ecompass in the presence of hard and soft-iron interference," *Freescale Semiconductor Ltd*, pp. 1–17, 2012.
- [8] M. García, "Attitude from angular rate." <https://ahrs.readthedocs.io/en/latest/filters/angular.html>.
- [9] M. García, "Complementary filter." <https://ahrs.readthedocs.io/en/latest/filters/complementary.html>.
- [10] B. Esme, "Kalman filter for dummies." <http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies>, 2009.
- [11] M. García, "Extended kalman filter." <https://ahrs.readthedocs.io/en/latest/filters/ekf.html>.

BIBLIOGRAFÍA

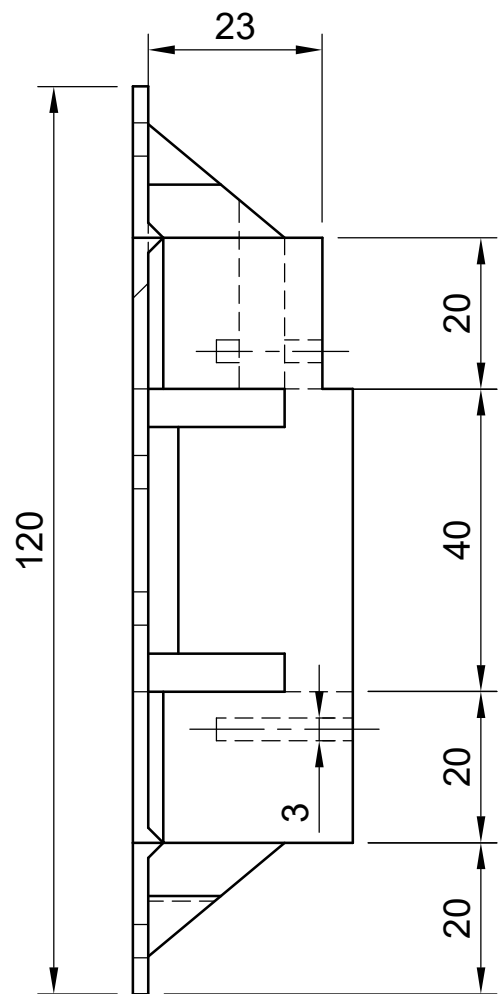
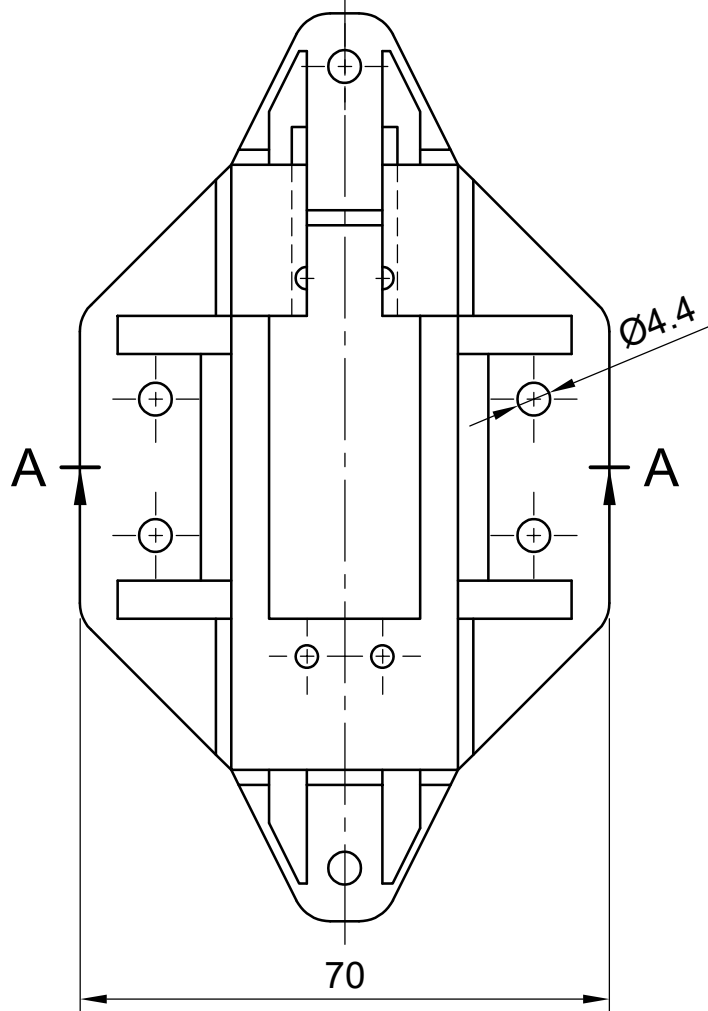
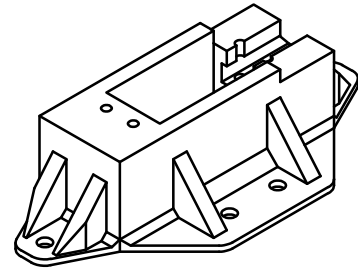
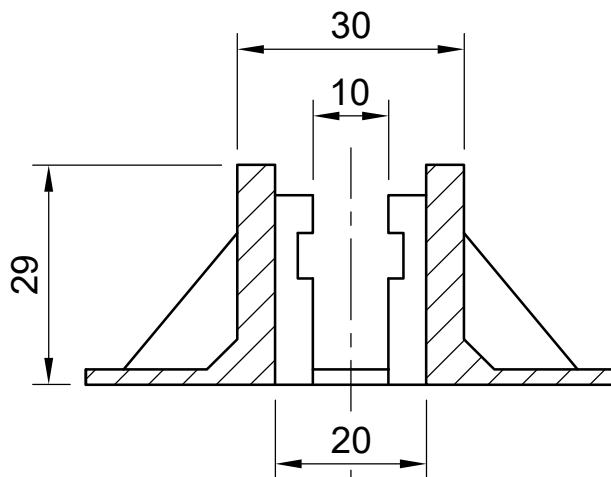
A. Planos de la plataforma

En este anexo se adjuntan los planos de las piezas utilizadas en la construcción de la plataforma. Los planos son orientativos y solo poseen las cotas de las dimensiones principales de la pieza.

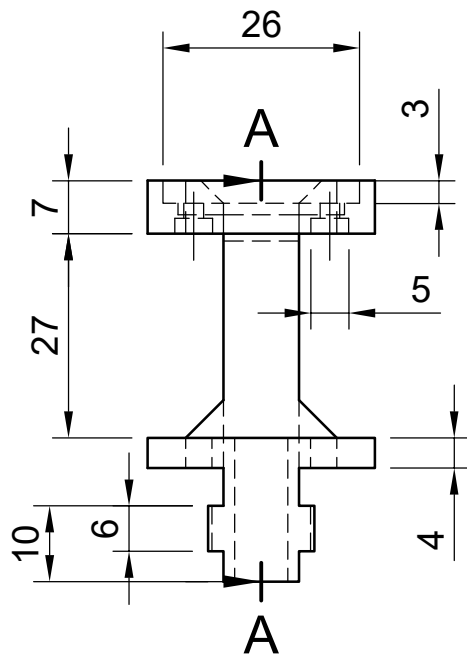
Los modelos 3D detallados están disponibles en formato STL en un repositorio de Thingiverse bajo licencia GNU - GPL.

La url de dicho repositorio es <https://www.thingiverse.com/thing:4964323>.

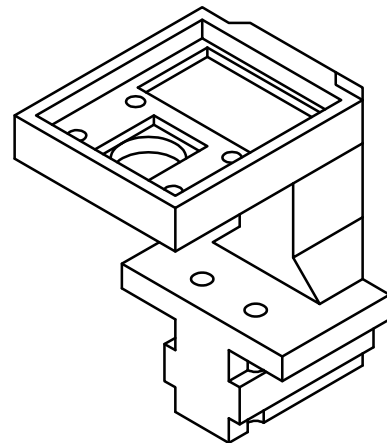
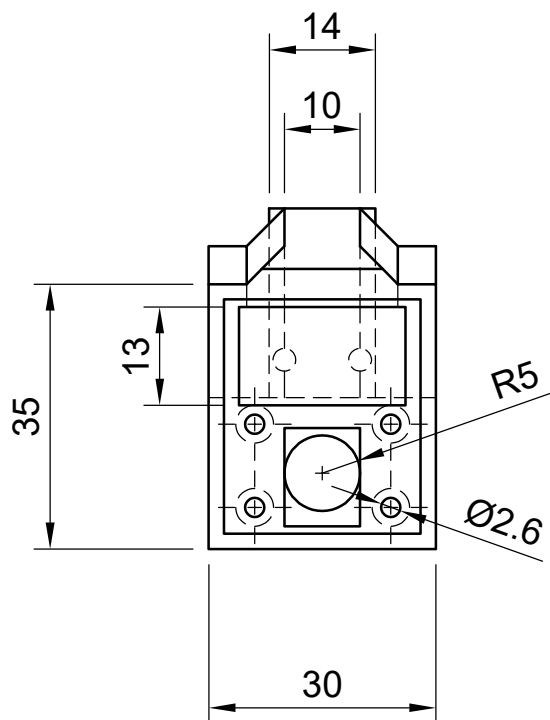
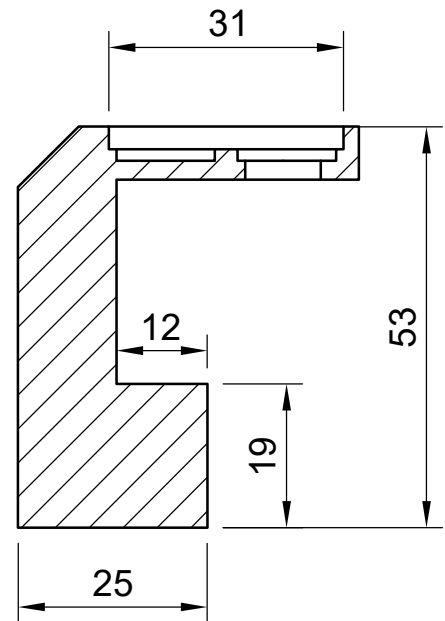
A-A (1:1)



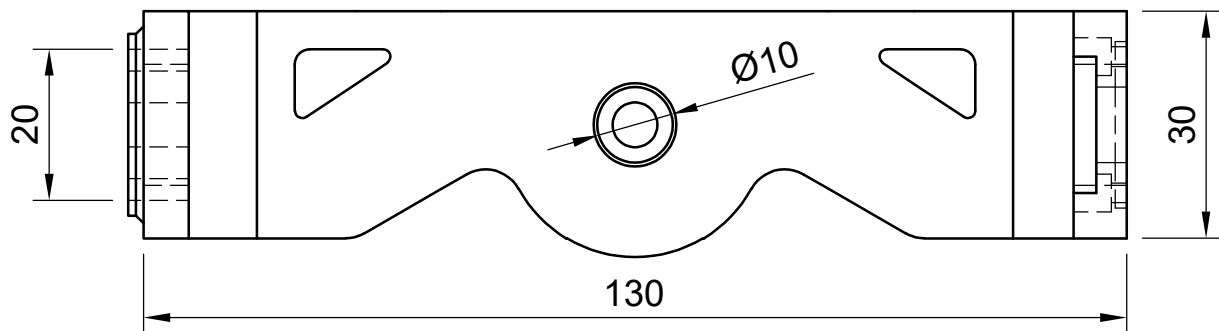
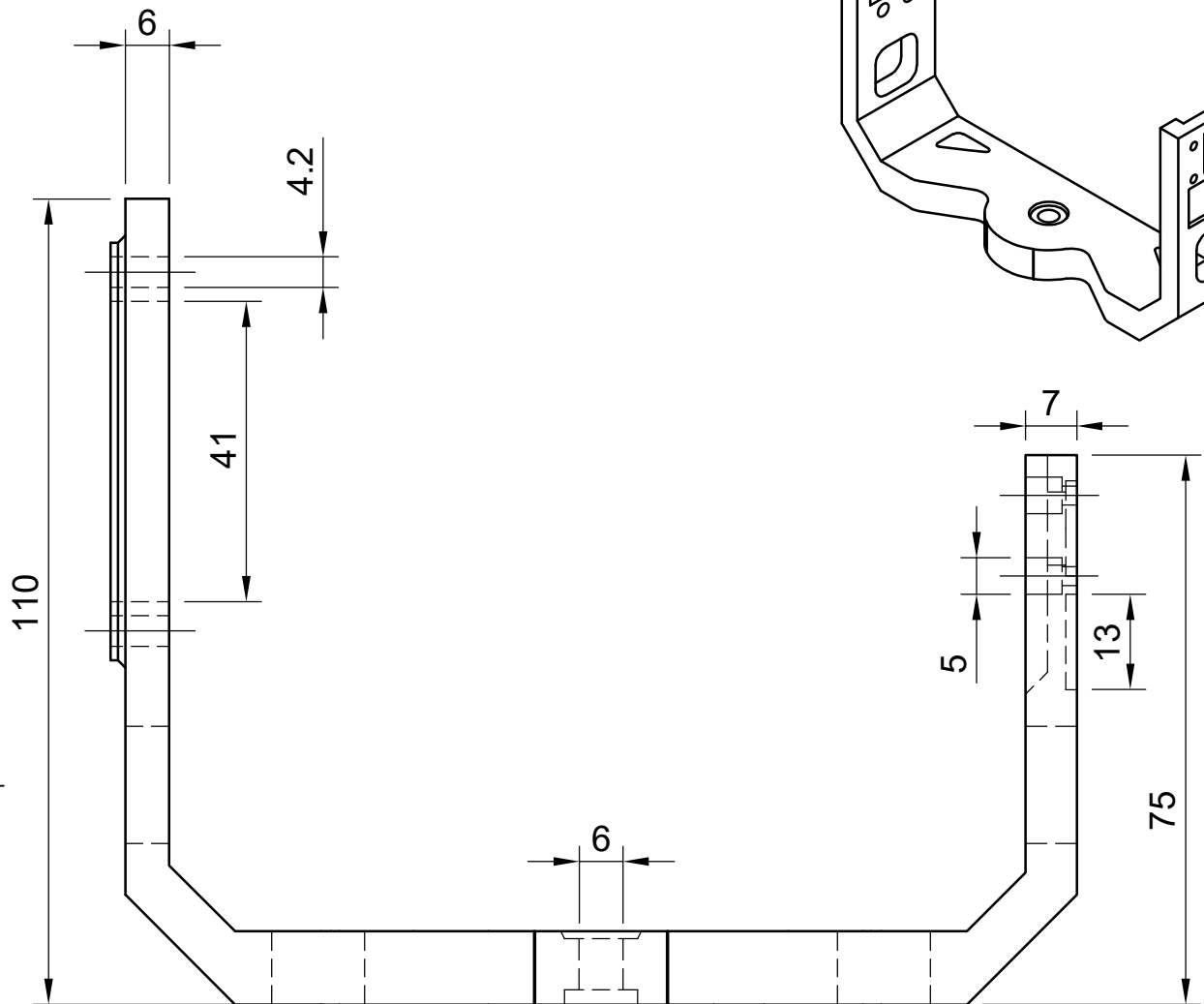
Dept.	Technical reference	Created by Pablo Ramirez 09/09/2021	Approved by		
		Document type	Document status		
		Title Soporte base Plataforma Giroscópica	DWG No. #1		
			Rev.	Date of issue	Sheet 1/1



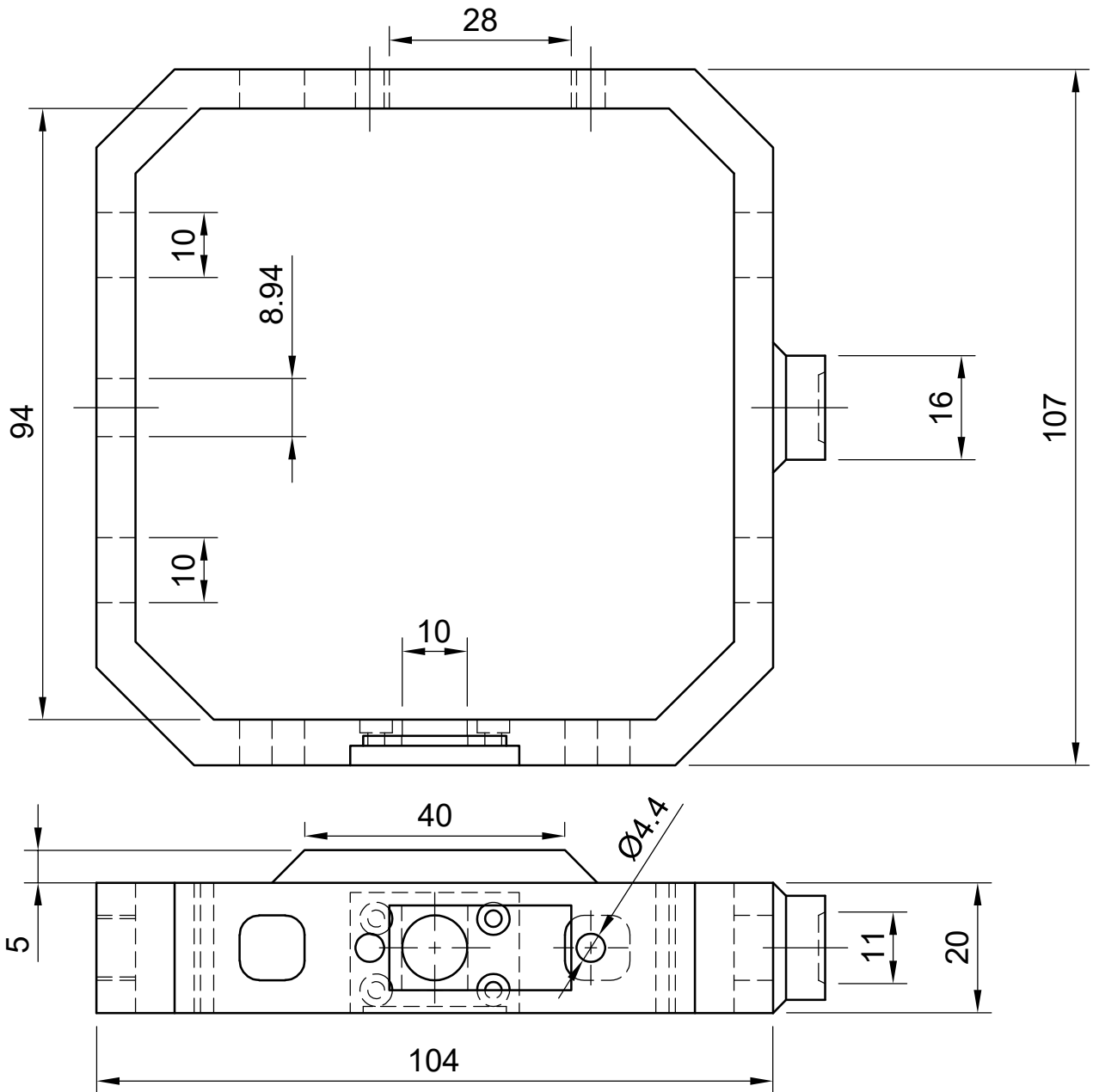
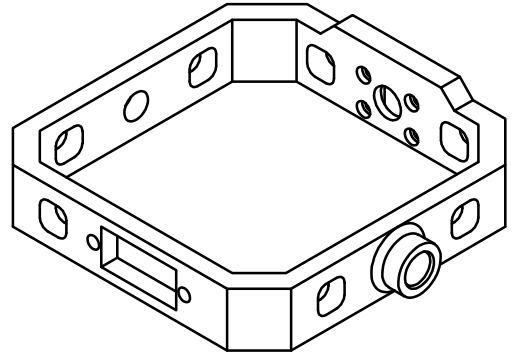
A-A (1:1)



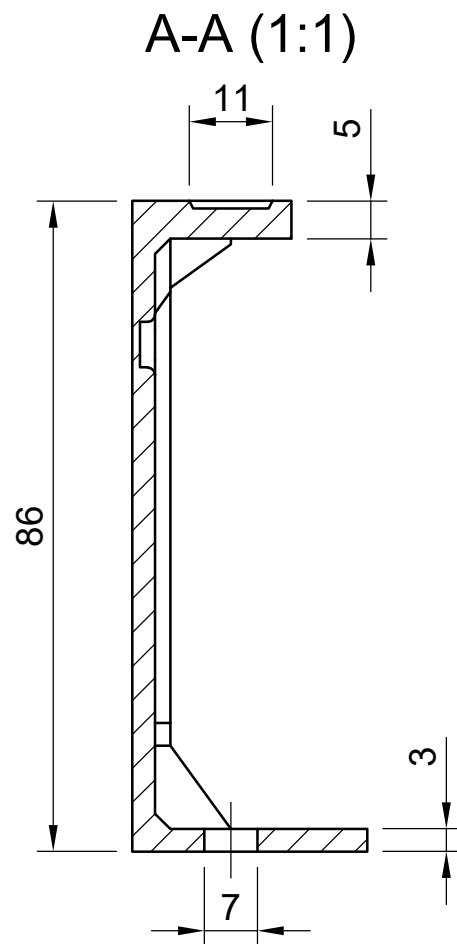
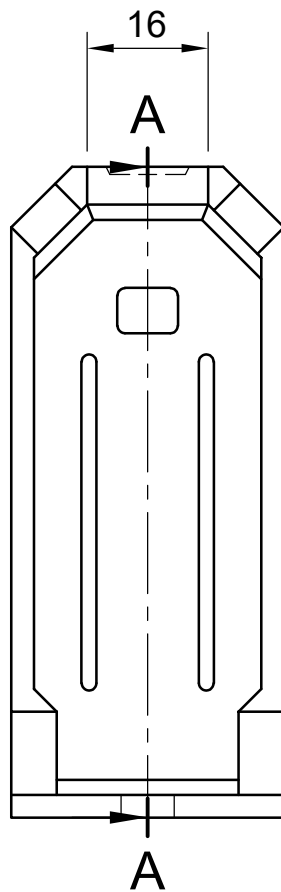
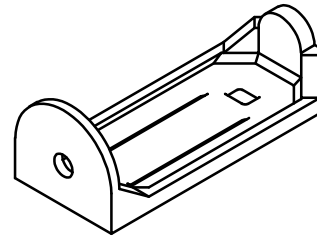
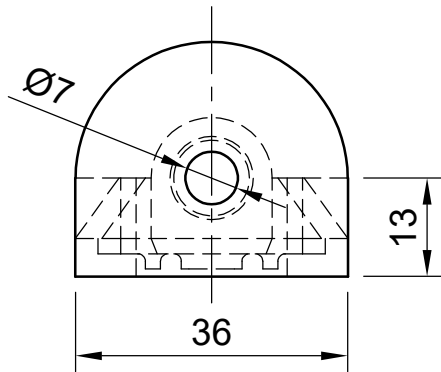
Dept.	Technical reference	Created by Pablo Ramirez 09/09/2021	Approved by		
		Document type	Document status		
		Title Brazo sensor base Plataforma Giroscópica	DWG No. #2		
			Rev.	Date of issue	Sheet 1/1



Dept.	Technical reference	Created by Pablo Ramirez 09/09/2021	Approved by		
		Document type	Document status		
		Title Soporte eje Z Plataforma Giroscópica	DWG No. #3		
			Rev.	Date of issue	Sheet 1/1



Dept.	Technical reference	Created by Pablo Ramirez 09/09/2021	Approved by		
		Document type	Document status		
		Title Anillo eje Y Plataforma Giroscópica	DWG No. #4		
			Rev.	Date of issue	Sheet 1/1

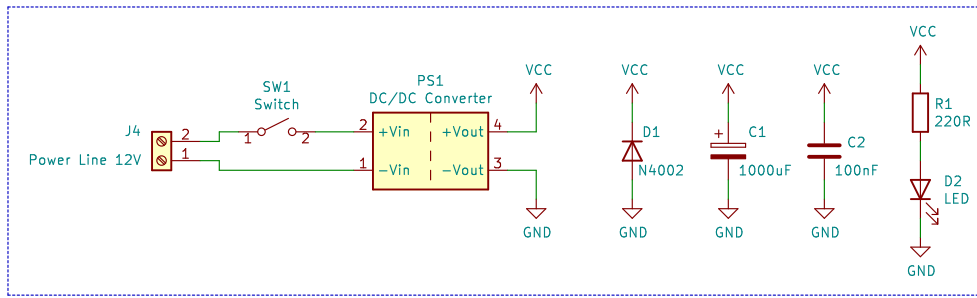


Dept.	Technical reference	Created by Pablo Ramirez 09/09/2021	Approved by		
		Document type	Document status		
		Title Cuna eje X Plataforma Giroscópica	DWG No. #5		
			Rev.	Date of issue	Sheet 1/1

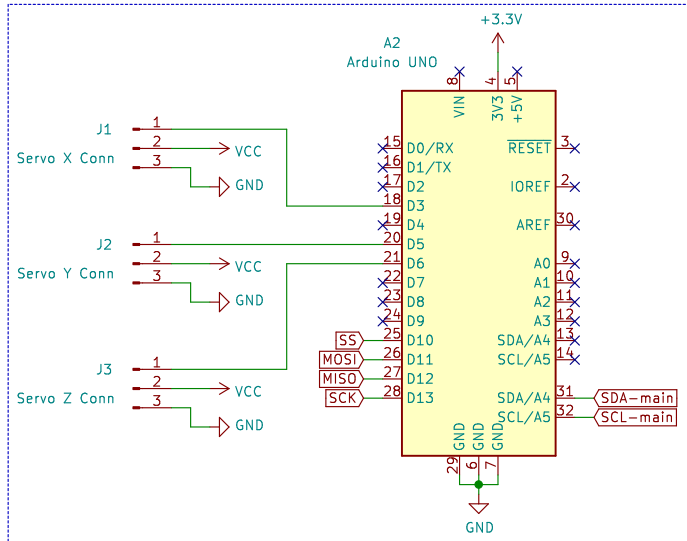
B. Diagrama eléctrico de la plataforma

El diagrama eléctrico ha sido generado con el software de diseño electrónico KiCAD EDA.

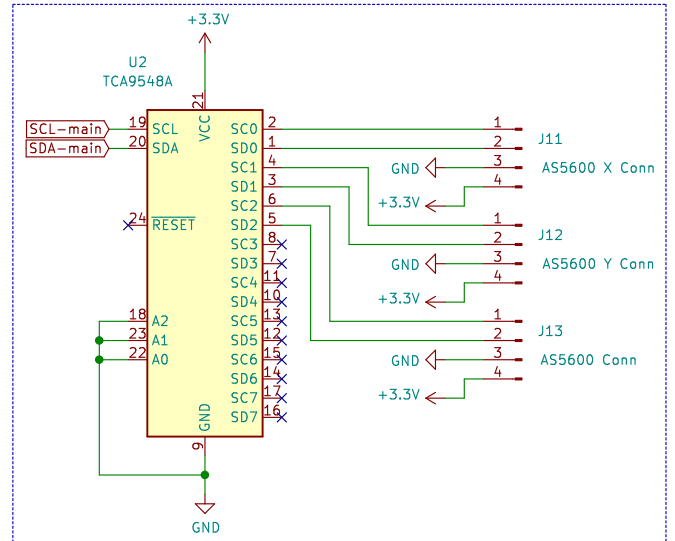
Power Line



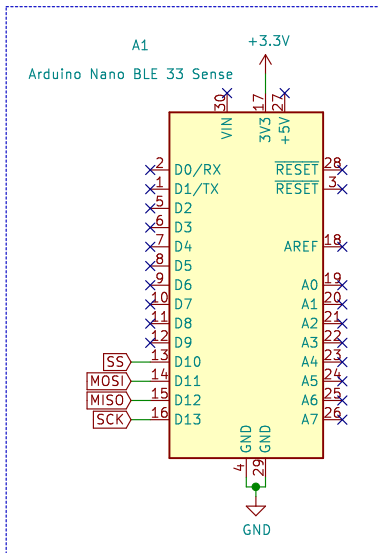
Platform Controller



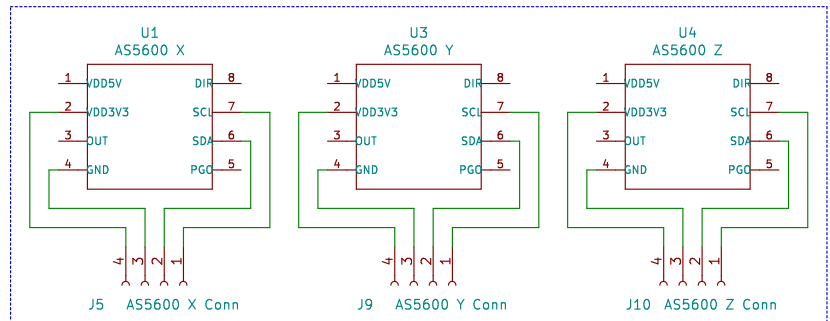
I2C Multiplexer



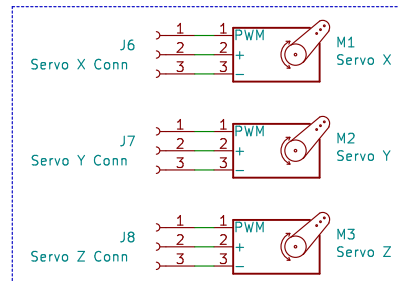
Inertial Measurement Unit



Angular Position Encoders



Servo Motors



Sheet: /

File: palataforma _gir6scopica.sch

Title: Esquema El6ctrico Plataforma Girosc6pica

Size: A4

Date: 2021-09-07

KiCad E.D.A. kicad (5.1.10)-1

Rev: 01

Id: 1/1

C. Códigos Arduino

Los códigos utilizados para la programación de los dos microcontroladores están disponibles en un repositorio en GitHub bajo licencia GNU - GPL.

La url de dicho repositorio es <https://github.com/BigKhalPablo/GimbalPlatform>.

C.1. Código de control de la unidad inercial

Este código está programado en el microcontrolador Arduino Nano BLE 33 Sense y es responsable de la lectura de la IMU y el envío de datos a la unidad de control de la plataforma (Arduino UNO).

```
1 #include <Arduino_LSM9DS1.h>
2 #include <SPI.h>
3
4 #define SPI_CLK 13
5 #define SPI_MISO 12
6 #define SPI_MOSI 11
7 #define SPI_SS 10
8
9 float ax, ay, az;
10 float gx, gy, gz;
11 float mx, my, mz;
12 int t0, t1, dt;
13 int n_readings;
14
15 // SPI data sen function
16 void send_sample(float ax, float ay, float az,
17                 float gx, float gy, float gz,
18                 float mx, float my, float mz,
19                 int dt, String separator)
20 {
21     char msg_byte;
22     String msg_string = "";
23
24     msg_string.concat(ax); msg_string.concat(separator);
25     msg_string.concat(ay); msg_string.concat(separator);
26     msg_string.concat(az); msg_string.concat(separator);
27     msg_string.concat(gx); msg_string.concat(separator);
28     msg_string.concat(gy); msg_string.concat(separator);
29     msg_string.concat(gz); msg_string.concat(separator);
30     msg_string.concat(mx); msg_string.concat(separator);
```

```

31 msg_string.concat(my); msg_string.concat(separator);
32 msg_string.concat(mz); msg_string.concat(separator);
33 msg_string.concat(dt); msg_string.concat('\n');
34
35 // activate slave device
36 digitalWrite(SS, LOW);
37
38 // sen all msg bytes
39 for (int k = 0; k < msg_string.length(); k++){
40     msg_byte = msg_string.charAt(k);
41     SPI.transfer(msg_byte);
42 }
43
44 // deactivate slave device
45 digitalWrite(SS, HIGH);
46 }
47
48
49 void setup() {
50
51     // initialize IMU object
52     if (!IMU.begin()) {
53         while (1);
54     }
55
56     digitalWrite(SS, HIGH); // initialize SPI bus with slave deactivated
57     SPI.begin(); // initialize SPI bus as master
58
59     t0 = millis(); // initialize time counter
60 }
61
62
63 void loop() {
64
65     n_readings = 0;
66
67     if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable() && IMU.
        magneticFieldAvailable()) {
68         // read IMU sensors
69         n_readings += IMU.readAcceleration(ax, ay, az);
70         n_readings += IMU.readGyroscope(gx, gy, gz);
71         n_readings += IMU.readMagneticField(mx, my, mz);
72         // check for fails in sensor reading
73         if (n_readings == 3) {
74             // compute measurement period dt
75             t1 = millis();
76             dt = t1 - t0;
77             t0 = t1;
78             // send all data throught SPI
79             send_sample(ax, ay, az, gx, gy, gz, mx, my, mz, dt, ", ");
80         }
81     }
82 }

```

C.2. Código de control de la plataforma

Este código está programado en el microcontrolador Arduino UNO y es responsable de la lectura de la IMU, la lectura de los sensores de posición angular, el control de los servomotores y la comunicación con el ordenador encargado de la recogida de datos.

```
1 #include "Wire.h" // Arduino I2C library
2 #include "SPI.h" // Arduino SPI library
3 #include <AS5600.h> // angular position sensor AS5600 library
4 #include <Servo.h> // servo motors library
5
6 // build the servo objects
7 Servo servoX;
8 Servo servoY;
9 Servo servoZ;
10
11 // define each servo PWM pin
12 #define SERVOX_PIN 3
13 #define SERVOY_PIN 5
14 #define SERVOZ_PIN 6
15
16 // sensor AS5600 working voltage (3.3 o 5)
17 #define SYS_VOL 3.3
18
19 // build the AS5600 sensor object
20 AMS_5600 ams5600;
21
22 // TCA9548A multiplexer I2C address
23 #define I2C_MUX_ADDR 0x70
24
25 // TCA9548A multiplexer buses to connect each AS5600 sensor
26 #define AS5600X_BUS 0
27 #define AS5600Y_BUS 1
28 #define AS5600Z_BUS 2
29
30 // parameters
31 const float servox_freq = 1.5, servoy_freq = 2.0, servoz_freq = 1.0; //
    servos osillating frecuency
32 const float servox_orig = 90, servoy_orig = 90, servoz_orig = 90; // servos
    central angle
33 const float servox_amp = 90, servoy_amp = 45, servoz_amp = 60; // servos
    angle max range
34 const float angx_off = 187.0, angy_off = 169.0, angz_off = 0.0; // AS5600
    sensors offset
35
36 // variables
37 int nsamples = 0; // samples counter
38 float t; // time input/output
39 float angx, angy, angz; // AS5600 angles input
40 float servox_ang, servoy_ang, servoz_ang; // servos angle output
41 float ax, ay, az, gx, gy, gz, mx, my, mz; // imu values input
42 float bx, by, bz; // gyro bias
43
```

```

44 // TCA9548A multiplexer bus selection function
45 void i2c_mux_select (uint8_t bus) {
46   Wire.beginTransmission(I2C_MUX_ADDR);
47   Wire.write(1 << bus); // send a byte to select the bus
48   Wire.endTransmission();
49 }
50
51 // AS5600 sensor checking function
52 void as5600_check (uint8_t bus) {
53   Serial.print("Checking AS5600 sensor #"); Serial.print(bus); Serial.println
54   (" ...");
55   i2c_mux_select(bus); // select bus to use
56   while(true){
57     if(ams5600.detectMagnet() == 1 ){
58       Serial.print("--> Magnet detected. Current Magnitude: ");
59       Serial.println(ams5600.getMagnitude());
60       break;
61     }else{
62       Serial.println("--> Can not detect magnet! Retrying...");
63     }
64     delay(2500);
65   }
66   Serial.println();
67 }
68
69 // AS5600 sensor reading function
70 float as5600_read(uint8_t bus) {
71   i2c_mux_select(bus); // sselect bus to use
72   word rawAngle = ams5600.getRawAngle(); // read the AS5600 sensor
73   /* Raw data reports 0 - 4095 segments, which is 0.087 of a degree */
74   float degAngle = rawAngle * 0.087; // convert raw data to degrees
75   return degAngle;
76 }
77
78 // read AS5600 sensors
79 void read_angle_sensors() {
80   angx = as5600_read(AS5600X_BUS) - angx_off;
81   angy = as5600_read(AS5600Y_BUS) - angy_off;
82   angz = servoz_ang - 90.0;
83 }
84
85 // compute angle and move servos
86 void move_servos() {
87   // compute servos angle
88   servox_ang = servox_amp*sin(2*PI*servox_freq*t) + servox_orig;
89   servoy_ang = servoy_amp*sin(2*PI*servoy_freq*t) + servoy_orig;
90   servoz_ang = servoz_amp*sin(2*PI*servoz_freq*t) + servoz_orig;
91   // write servos angle
92   servoX.write(servox_ang);
93   servoY.write(servoy_ang);
94   servoZ.write(servoz_ang);
95 }

```

```

96 // SPI bus reading main variables
97 char SPI_dataReceived [100];
98 volatile byte SPI_index;
99 volatile bool SPI_receptionEnd;
100 String SPI_dataString;
101
102 // SPI bus interrupt routine
103 ISR (SPI_STC_vect) {
104     // read received data from the SPDR register of the SPI bus
105     byte SPI_byteReceived = SPDR;
106     // insert received data to buffer
107     if (SPI_index < sizeof SPI_dataReceived) {
108         SPI_dataReceived[SPI_index++] = SPI_byteReceived;
109         // if end line received, proces the buffer
110         if (SPI_byteReceived == '\n') {
111             SPI_receptionEnd = true;
112         }
113     }
114 }
115
116 // send all data throught serial and reset SPI bus
117 void send_data_serial(String separator) {
118     if (SPI_receptionEnd) {
119         // store SPI data
120         SPI_dataReceived[SPI_index] = 0;
121         SPI_dataString = String(SPI_dataReceived); // convert to string
122         // reset SPI bus variables
123         SPI_index = 0;
124         SPI_receptionEnd = false;
125         // send data throught Serial
126         Serial.print(separator);
127         Serial.print(t); Serial.print(separator);
128         Serial.print(angx); Serial.print(separator);
129         Serial.print(angy); Serial.print(separator);
130         Serial.print(angz); Serial.print(separator);
131         SPI_dataString.replace(",", " ", separator); // replace Arduino BLE 33
            separator
132         Serial.print(SPI_dataString);
133         nsamples ++; // increase samples counter
134     }
135 }
136
137 void setup() {
138     // initialize the serial port
139     Serial.begin(115200);
140     delay(1000);
141     Serial.println("Serial port (UART) initilized!");
142
143     // initialize the I2C bys
144     Wire.begin();
145     Serial.println("I2C bus initilized!");
146
147     // activate SPI bus in slave mode

```

```

148 SPCR |= bit (SPE);
149 // configure the MISO pin as output (slave out)
150 pinMode (MISO, OUTPUT);
151 // initialize the SPI bus buffer
152 SPI_index = 0;
153 // initialize the reception bit
154 SPI_receptionEnd = false;
155 // activate the SPI bus interruption
156 SPI.attachInterrupt();
157 Serial.println("SPI bus initilized!");
158
159 Serial.println();
160
161 // attach each servo to pins
162 servoX.attach(SERVOX_PIN);
163 servoY.attach(SERVOY_PIN);
164 servoZ.attach(SERVOZ_PIN);
165 // initilize servos in the center position
166 Serial.println("Moving servos to origin...");
167 servoX.write(servox_orig);
168 servoY.write(servoy_orig);
169 servoZ.write(servoz_orig);
170 // wait to start
171 delay(2000);
172
173 Serial.println();
174
175 // check AS5600 sensors
176 as5600_check(AS5600X_BUS);
177 as5600_check(AS5600Y_BUS);
178 //as5600_check(AS5600Z_BUS);
179
180 Serial.println("Senging static data...");
181 delay(1000);
182 // send 100 samples static
183 while (nsamples <= 100) {
184     t = millis()/1000.0;
185     angx = 0.0; angy = 0.0; angz = 0.0;
186     send_data_serial(" ");
187 }
188 }
189
190 void loop() {
191     t = millis()/1000.0;
192     move_servos();
193     read_angle_sensors();
194     send_data_serial(" ");
195 }

```

D. Códigos Matlab

D.1. Función del estimador de acelerómetro y magnetómetro

```

1 function angles = accel_magnet_attitude(acc, mag)
2     % Estimate attitude using accelerometer and magnetometer data.
3     % With accelerometer tilt is calculated (roll and pitch), then
4     % calibrated magnetometer data is rotated and yaw is computed.
5     %
6     % Inputs: acc, mag, mag_bias
7     %     acc: Nx3 tri-axial accelerometer readings (ax, ay, az) in m/s^2
8     %     mag: Nx3 tri-axial magnetometer calibrated readings (mx, my, mz) in
9         % uT
10    %
11    % Outputs: angles
12    %     angles: Nx3 euler angles array (roll, pitch, yaw) in rad
13
14    a = acc./vecnorm(acc,2,2); % normalize accel readings
15    ax = a(:,1); ay = a(:,2); az = a(:,3);
16
17    % compute roll and pitch from gravity
18    roll = atan2(-ay, -az);
19    pitch = atan2(ax, sqrt(ay.^2 + az.^2));
20
21    m = mag./vecnorm(mag,2,2); % normalize magnet readings
22    mx = m(:,1); my = m(:,2); mz = m(:,3);
23
24    % compute sin and cos of roll and pitch for later use
25    sr = sin(roll); cr = cos(roll);
26    sp = sin(pitch); cp = cos(pitch);
27
28    % rotate magnetometer data to the horizontal plane
29    rx = mx.*cp + my.*sr.*sp + mz.*cr.*sp;
30    ry = my.*cr - mz.*sr;
31
32    % compute yaw as the tilt-compensated heading angle to Magnetic North
33    yaw = atan2(-ry, rx);
34
35    angles = [roll pitch yaw];
36 end

```

D.2. Función del estimador por integración del giróscopo

```

1 function q = gyro_integration(gyr, q0, Dt)
2     % Estimate attitude integrating the angular rates measured by the
3     % gyroscope. To avoid locking gymbal problem the integration will be
4     % performed with quaternions.
5     %
6     % Inputs: gyr, q0, Dt
7     %     gyr: Nx3 tri-axial gyroscope readings (wx, wy, wz) in rad/s
8     %     q0: 1x4 initial attitude as quaternion
9     %     Dt: N sampling periods array in seconds
10    %
11    % Outputs: q
12    %     q: Nx4 normalized quaternions array with the attitude
13
14    N = size(gyr, 1); % get the number of samples
15    q = zeros(N, 4);
16    q(1, :) = q0;
17    for k = 2:N
18        dt = Dt(k); % current sampling period
19        wx = gyr(k, 1); wy = gyr(k, 2); wz = gyr(k, 3);
20        % compute Omega matrix
21        Omega = [ 0.0  -wx  -wy  -wz
22                  +wx   0.0  +wz  -wy
23                  +wy  -wz   0.0  +wx
24                  +wz  +wy  -wx   0.0 ];
25        qt_prev = q(k-1, :)'; % get las quaternion as column (4x1)
26        qt = (0.5*Omega*dt + eye(4))*qt_prev;
27        qt = qt/norm(qt); % normalize the quaternion
28        q(k, :) = qt.'; % store the computed quaternion as row (1x4)
29    end
30 end

```

D.3. Función del filtro complementario

```

1 function q = complementary_filter(acc, mag, gyr, q0, Dt, gain)
2     % Estimate attitude by fusing accelerometer and gyroscope estimations
3     % with a complementary filter (linear interpolator).
4     %
5     % Inputs: acc, gyr, q0, Dt, gain
6     %     acc: Nx3 tri-axial accelerometer readings (ax, ay, az) in m/s^2
7     %     mag: Nx3 tri-axial magnetometer readings (mx, my, mz) in uT
8     %     gyr: Nx3 tri-axial gyroscope readings (wx, wy, wz) in rad/s
9     %     q0: 1x4 initial attitude as quaternion
10    %     Dt: N sampling periods array in seconds
11    %     gain: gain of filter as float in [0.0, 1.0]. If gain = 1.0 the
12    %     filter will only use the accelerometer and magnetometer data.
13    %     If gain = 0.0 the filter will only use gyroscope data.
14    %
15    % Outputs: q
16    %     q: Nx4 normalized quaternions array with the attitude
17
18    N = size(gyr, 1); % get the number of samples
19    a = acc./vecnorm(acc,2,2); % normalize accel readings
20    m = mag./vecnorm(mag,2,2); % normalize magnet readings
21
22    q = zeros(N,4);
23    q(1,:) = q0;
24    for k = 2:N
25        % Step 1) Attitude propagation integrating gyro
26        dt = Dt(k); % current sampling period
27        wx = gyr(k,1); wy = gyr(k,2); wz = gyr(k,3);
28        % compute Omega matrix
29        Omega = [ 0.0  -wx  -wy  -wz
30                  +wx   0.0  +wz  -wy
31                  +wy  -wz   0.0  +wx
32                  +wz  +wy  -wx   0.0 ];
33        q_prev = q(k-1,:)' ; % get las quaternion as column (4x1)
34        A = eye(4) + 0.5*Omega*dt;
35        q_gyr = A*q_prev;
36        q_gyr = q_gyr/norm(q_gyr); % normalize the quaternion
37        q_gyr = q_gyr'; % quaternion as row (1x4)
38
39        % Step 2) Compute attitude with accel and magnet data
40        % compute roll and pitch from gravity
41        ax = a(k,1); ay = a(k,2); az = a(k,3);
42        roll = atan2(-ay, -az);
43        pitch = asin(ax);
44        % rotate magnetometer data and compute yaw
45        mx = m(k,1); my = m(k,2); mz = m(k,3);
46        sr = sin(roll); cr = cos(roll);
47        sp = sin(pitch); cp = cos(pitch);
48        rx = mx*cp + my*sr*sp + mz*cr*sp;
49        ry = my*cr - mz*sr;
50        yaw = atan2(-ry, rx);
51        q_acc = euler2quat([roll pitch yaw]);

```

```
52  
53     % Step 3) Combine each estimation with the complementary filter  
54     q(k,:) = (1-gain)*q_gyr + gain*q_acc; % store the computed quaternion  
55     end  
56  
57 end
```

D.4. Función del filtro de Kalman extendido

```

1 function q = extended_kalman_filter(acc, mag, gyr, q0, Dt, noises)
2     % Estimate attitude by fusing accelerometer and gyroscope estimations
3     % with a complementary filter (linear interpolator).
4     %
5     % Inputs: acc, gyr, q0, Dt, gain
6     %     acc: Nx3 tri-axial accelerometer readings (ax, ay, az) in m/s^2
7     %     mag: Nx3 tri-axial magnetometer calibrated readings (mx, my, mz) in
8     %           uT
9     %     gyr: Nx3 tri-axial gyroscope readings (wx, wy, wz) in rad/s
10    %     q0: 1x4 initial attitude as quaternion
11    %     Dt: N sampling periods array in seconds
12    %     gain: gain of filter as float in [0.0, 1.0]. If gain = 1.0 the
13    %           filter only use the accelerometer data. If gain = 0.0 the filter
14    %           only use gyroscope data.
15    %
16    % Outputs: q
17    %     q: Nx4 normalized quaternions array with the attitude
18
19    % Prediction
20    % 1) x_est = F * x_prev
21    % 2) P_est = F * P_prev * F' + Q
22
23    % Correction
24    % 3) y_est = h(x_est)
25    % 4) K = P_est * H' * inv(H * P_est * H' + R)
26    % 5) x = x_est + K * (y_meas - y_est)
27    % 6) P = P_est - K * H * P_est
28
29    % get sensors noises
30    sig_a = noises(1);
31    sig_m = noises(2);
32    sig_g = noises(3);
33
34    N = size(gyr, 1); % get the number of samples
35    a = acc./vecnorm(acc,2,2); % normalize accel readings
36    m = mag./vecnorm(mag,2,2); % normalize magnet readings
37
38    % build measurement error covariance matrix
39    R = diag([sig_a^2 sig_a^2 sig_a^2 sig_m^2 sig_m^2 sig_m^2]);
40
41    q = zeros(N,4);
42    q(1,:) = q0;
43    x_prev = q0';
44    P_prev = eye(4);
45    for k = 2:N
46
47        dt = Dt(k); % current sampling period
48        ax = a(k,1); ay = a(k,2); az = a(k,3);
49        mx = m(k,1); my = m(k,2); mz = m(k,3);
50        wx = gyr(k,1); wy = gyr(k,2); wz = gyr(k,3);
51        qw = x_prev(1); qx = x_prev(2); qy = x_prev(3); qz = x_prev(4);

```



```
51
52 % Step 1) compute estate prediction
53 % compute F Jacobian matrix
54 Omega = [ 0.0 -wx -wy -wz
55           +wx 0.0 +wz -wy
56           +wy -wz 0.0 +wx
57           +wz +wy -wx 0.0 ];
58 F = 0.5*Omega*dt + eye(4);
59 x_est = F*x_prev; % prediction
60 x_est = x_est/norm(x_est); % normalize the quaternion
61
62 % Step 2) compute prediction covariance
63 % compute the process covariance matrix
64 W = 0.5*dt * [-qx -qy -qz
65               +qw -qz +qy
66               +qz +qw -qx
67               -qy +qx +qw];
68 Q = sig_g^2 * W * W';
69 P_est = F * P_prev * F' + Q;
70
71 % Step 3) estimate measurement
72 y_est = [-2*qx*qz + 2*qy*qw
73          -2*qy*qz - 2*qx*qw
74          -qw^2 + qx^2 + qy^2 - qz^2
75          qw^2 + qx^2 - qy^2 - qz^2
76          2*qx*qy - 2*qz*qw
77          2*qx*qz + 2*qy*qw];
78
79 % Step 4) compute Kalman gain
80 % build measurement model Jacobian matrix
81 H = [+2*qy -2*qz +2*qw -2*qx
82      -2*qx -2*qw -2*qz -2*qy
83      -2*qw +2*qx +2*qy -2*qz
84      +2*qw +2*qx -2*qy -2*qz
85      -2*qz +2*qy +2*qx -2*qw
86      +2*qy +2*qz +2*qw +2*qx];
87 K = P_est * H' * inv(H * P_est * H' + R);
88
89
90 % Step 5) correct predicted state and its covariance
91 y_meas = [ax; ay; az; mx; my; mz];
92 x = x_est + K * (y_meas - y_est);
93 x = x/norm(x); % normalize the quaternion
94 P = P_est - K * H * P_est;
95
96 % Store data
97 P_prev = P;
98 x_prev = x;
99 q(k,:) = x'; % quaternion as row (1x4)
100
101 end
102
103 end
```

D.5. Script de comparación de algoritmos

```

1 clear; clc; close all;
2
3 load('ensayos_banco/giroXYZ_slow.mat');
4
5 t = data(:,1); % time elapsed (seconds)
6 tx = data(:,2); % AS5600 measured angles (degrees)
7 ty = data(:,3);
8 tz = data(:,4);
9 ax = data(:,5); % BLE33 accelerometer readings (g)
10 ay = data(:,6);
11 az = data(:,7);
12 gx = data(:,8); % BLE33 gyroscope readings (deg/s)
13 gy = data(:,9);
14 gz = data(:,10);
15 mx = data(:,11); % BLE33 magnetometer readings (uT)
16 my = data(:,12);
17 mz = data(:,13);
18
19 % change data axis to NED frame
20 ang = [tx ty -tz];
21 acc = [ax ay -az]*(9.81); % g to m/s^2
22 gyr = [gx gy -gz]*(pi/180.0); % deg/S to rad/s
23 mag = [-mx my -mz]*(100); % uT to gauss
24
25 acc = low_pass_filter(acc, 0.6); % filter accel data
26 mag = low_pass_filter(mag, 0.6); % filter mag data
27
28 t = t - t(1); % time vector starting in 0s
29 dt = gradient(t); % compute dt from t vector
30
31 % calibrate magnetometer
32 mag_bias = mag_cal(mag);
33 mag = mag - mag_bias; % remove magnetometer offset error
34
35 % get initial quaternion to start integration
36 ang0 = [0 0 0];
37 q0 = euler2quat(ang0);
38
39
40 %% ATTITUDE FROM ACCEL AND MAGNET
41
42 est = accel_magnet_attitude(acc, mag);
43 est = rad2deg(est);
44 estims(1, :, :) = est; %store estimation
45
46 txt = "Attitude from Accelerometer and Magnetometer";
47 plotter(t, ang, est, txt);
48
49 rms_r = RMS(t, ang(:,1), est(:,1));
50 rms_p = RMS(t, ang(:,2), est(:,2));
51 rms_y = RMS(t, ang(:,3), est(:,3));

```

```

52
53 fprintf("%s \n", txt);
54 fprintf("RMS_roll = %.4f , RMS_pitch = %.4f , RMS_yaw = %.4f \n\n", rms_r,
    rms_p, rms_y)
55
56 %% GYRO INTEGRATION
57
58 quat = gyro_integration(gyr, q0, dt);
59 est = quat2euler(quat);
60 est = rad2deg(est);
61 estims(2,,:) = est; %store estimation
62
63 txt = "Gyroscope Integration";
64 plotter(t, ang, est, txt);
65
66 rms_r = RMS(t, ang(:,1), est(:,1));
67 rms_p = RMS(t, ang(:,2), est(:,2));
68 rms_y = RMS(t, ang(:,3), est(:,3));
69
70 fprintf("%s \n", txt);
71 fprintf("RMS_roll = %.4f , RMS_pitch = %.4f , RMS_yaw = %.4f \n\n", rms_r,
    rms_p, rms_y)
72
73 %% COMPLEMENTARY FILTER
74
75 gain = 0.5;
76 quat = complementary_filter(acc, mag, gyr, q0, dt, gain);
77 est = quat2euler(quat);
78 est = rad2deg(est);
79 estims(3,,:) = est; %store estimation
80
81 txt = "Complementary Filter";
82 plotter(t, ang, est, txt);
83
84 rms_r = RMS(t, ang(:,1), est(:,1));
85 rms_p = RMS(t, ang(:,2), est(:,2));
86 rms_y = RMS(t, ang(:,3), est(:,3));
87
88 fprintf("%s \n", txt);
89 fprintf("RMS_roll = %.4f , RMS_pitch = %.4f , RMS_yaw = %.4f \n\n", rms_r,
    rms_p, rms_y)
90
91 %% Extended Kalman Filter
92
93 noises = [0.3 20 0.8]; % standar deviation
94 quat = extended_kalman_filter(acc, mag, gyr, q0, dt, noises);
95 est = quat2euler(quat);
96 est = rad2deg(est);
97 estims(4,,:) = est; %store estimation
98
99 txt = "Extended Kalman Filter";
100 plotter(t, ang, est, txt);
101

```

```

102 rms_r = RMS(t, ang(:,1), est(:,1));
103 rms_p = RMS(t, ang(:,2), est(:,2));
104 rms_y = RMS(t, ang(:,3), est(:,3));
105
106 fprintf("%s \n", txt);
107 fprintf("RMS_roll = %.4f , RMS_pitch = %.4f , RMS_yaw = %.4f \n\n", rms_r,
    rms_p, rms_y)
108
109 %% Compare Estimators
110 k1 = 500;
111 k2 = 1000;
112 comparer(t(k1:k2), ang(k1:k2,:), estims(:,k1:k2,:), 'Filters Error Comparison
    ', {'Acc+Mag', 'Gyro', 'Comp', 'EKF'})
113
114 %% Auxiliary Functions
115 function plotter(t, true, estim, title)
116     figure()
117     sgtitle(sprintf(title))
118     subplot(3,1,1)
119     hold on
120     plot(t, estim(:,1))
121     plot(t, true(:,1))
122     hold off
123     legend('estimated','true')
124     grid on; set(gca,'ytick',[-180:45:180])
125     ylabel('Roll: \phi (deg)')
126     subplot(3,1,2)
127     hold on
128     plot(t, estim(:,2))
129     plot(t, true(:,2))
130     hold off
131     legend('estimated','true')
132     grid on; set(gca,'ytick',[-90:45:90])
133     ylabel('Pitch: \theta (deg)')
134     subplot(3,1,3)
135     hold on
136     plot(t, estim(:,3))
137     plot(t, true(:,3))
138     hold off
139     legend('estimated','true')
140     grid on; set(gca,'ytick',[-360:90:360])
141     ylabel('Yaw: \psi (deg)')
142     xlabel('time (s)')
143 end
144
145 function comparer(t, true, estims, title, legendtxt)
146     N = size(estims, 1);
147     figure()
148     sgtitle(title)
149     subplot(3,1,1)
150     ylabel('Roll Error: \Delta\phi (deg)')
151     hold on; grid on
152     subplot(3,1,2)

```

```

153 ylabel('Pitch Error: \Delta\theta (deg)')
154 hold on; grid on
155 subplot(3,1,3)
156 ylabel('Yaw Error: \Delta\psi (deg)')
157 hold on; grid on
158 for k = 1:N
159     estim = squeeze(estims(k,:,:));
160     subplot(3,1,1)
161     plot(t, estim(:,1) - true(:,1), 'LineWidth', 1)
162     subplot(3,1,2)
163     plot(t, estim(:,2) - true(:,2), 'LineWidth', 1)
164     subplot(3,1,3)
165     plot(t, estim(:,3) - true(:,3), 'LineWidth', 1)
166 end
167 subplot(3,1,1)
168 legend(legendtxt)
169 subplot(3,1,2)
170 legend(legendtxt)
171 subplot(3,1,3)
172 legend(legendtxt)
173 xlabel('time (s)')
174 end
175
176 function rms = RMS(t, true, estim)
177     N = size(t, 1);
178     err = 0;
179     for k = 1:N
180         err = err + (estim(k) - true(k))^2;
181     end
182     rms = sqrt(err/N);
183 end

```

D.6. Función de conversión de ángulos de euler a cuaterniones

```
1 function q = euler2quat(angles)
2     % Compute quaternions from euler angles
3     % Input: angles
4     %   Nx3 euler angles array (roll, pitch, yaw) in rad
5     % Output: q
6     %   Nx4 normalized quaternions array
7
8     % get the number of columns
9     N = size(angles, 1);
10
11     % get Nx1 euler angles arrays from input array
12     roll = angles(:,1);
13     pitch = angles(:,2);
14     yaw = angles(:,3);
15
16     % compute sin and cos of each angle for later use
17     sr = sin(0.5*roll);
18     cr = cos(0.5*roll);
19     sp = sin(0.5*pitch);
20     cp = cos(0.5*pitch);
21     sy = sin(0.5*yaw);
22     cy = cos(0.5*yaw);
23
24     % compute the quaternions array
25     q = zeros(N, 4);
26     q(:,1) = cr.*cp.*cy + sr.*sp.*sy;
27     q(:,2) = sr.*cp.*cy - cr.*sp.*sy;
28     q(:,3) = cr.*sp.*cy + sr.*cp.*sy;
29     q(:,4) = cr.*cp.*sy - sr.*sp.*cy;
30
31 end
```

D.7. Función de conversión de cuaterniones a ángulos de euler

```
1 function angles = quat2euler(q)
2     % Compute euler angles from quaternions
3     % Input: q
4     %   Nx4 normalized quaternions array
5     % Output: angles
6     %   Nx3 euler angles array (roll, pitch, yaw) in rad
7
8     % get the number of columns
9     N = size(q, 1);
10
11     % get Nx1 quaternions arrays from input array
12     q0 = q(:,1);
13     q1 = q(:,2);
14     q2 = q(:,3);
15     q3 = q(:,4);
16
17     % compute euler angles array
18     angles = zeros(N, 3);
19     angles(:,1) = atan2(2.0*(q0.*q1 + q2.*q3), (q0.*q0 - q1.*q1 - q2.*q2 + q3
20         .*q3));
21     angles(:,2) = asin(2.0*(q0.*q2 - q1.*q3));
22     angles(:,3) = atan2(2.0*(q0.*q3 + q1.*q2), (q0.*q0 + q1.*q1 - q2.*q2 - q3
23         .*q3));
24 end
```

D.8. Función de calibración del magnetómetro

```

1 function mag_bias = mag_cal(mag)
2     % Calibrate magnetometer by fitting its data to a sphere. The
3     % coordinates of the center of the sphere are the magnetometer bias
4     % (or hard iron error).
5     %
6     % Inputs: mag
7     %     mag: Nx3 tri-axial magnetometer readings (mx, my, mz) in gauss
8     %
9     % Outputs: mag_bias
10    %     mag_bias: 1x3 tri-axial magnetometer bias (bx, by, bz) in gauss
11
12    % measured magnetic field
13    mx = mag(:,1);
14    my = mag(:,2);
15    mz = mag(:,3);
16
17    % number of samples
18    N = size(mx, 1);
19
20    % Spherical fitting error model calibration --> h = m - b
21    % h : true local magnetic field
22    % m : measured magnetic field
23    % A : soft iron error matrix (transformation)
24    % b : hard iron error vector (bias)
25
26    % Spherical parametric equation:
27    % x^2 + y^2 + z^2 + a*x + b*y + c*z + d = H^2
28
29    % Least Squares regresion to spherical parametric equation:
30    % H^2 = h'*h = (m-b)'+*(m-b) = m'*m - 2*b'*m + b'*b
31    % Y = m'*m = mx^2 + my^2 + mz^2 ; X = [mx my mz 1]
32    % a = [2*bx 2*by 2*bz H^2-bx^2-by^2-bz^2]
33    % a = inv(X'*X)*X'*Y
34
35    Y = mx.^2 + my.^2 + mz.^2;
36    X = zeros(N,4);
37    X(:,1) = mx;
38    X(:,2) = my;
39    X(:,3) = mz;
40    X(:,4) = 1;
41    a = inv(X'*X)*X'*Y;
42
43    bx = 0.5*a(1);
44    by = 0.5*a(2);
45    bz = 0.5*a(3);
46    mag_bias = [bx by bz];
47
48 end

```

D.9. Script de adquisición de datos de la plataforma

```

1 clear
2
3 % parameters
4 PORT = 'COM6';
5 BAUD = 115200;
6 FILE = 'ensayos_banco\giroXYZ_legend.mat';
7 n_samples = 1000;
8 data = zeros(n_samples,14);
9
10 fprintf('Connecting to serial port %s at baudrate %d ...\n\n', PORT, BAUD);
11 com = serialport(PORT, BAUD);
12 disp(com)
13
14 k = 1;
15 while (k <= n_samples)
16     try
17         line = readline(com);
18         data_line = str2num(line);
19         % check if accel, gyro and mag data is inside range
20         ax = data_line(5); % BLE33 accelerometer readings (g)
21         ay = data_line(6);
22         az = data_line(7);
23         acc_check = (abs(ax) < 2) & (abs(ay) < 2) & (abs(az) < 2);
24         gx = data_line(8); % BLE33 gyroscope readings (deg/s)
25         gy = data_line(9);
26         gz = data_line(10);
27         gyr_check = (abs(gx) < 500) & (abs(gy) < 500) & (abs(gz) < 500);
28         mx = data_line(11); % BLE33 magnetometer readings (uT)
29         my = data_line(12);
30         mz = data_line(13);
31         mag_check = (abs(mx) < 400) & (abs(my) < 400) & (abs(mz) < 400);
32         dt = data_line(14); % BLE33 internal clock dt (ms)
33         dt_check = (dt >= 50 & dt < 100);
34         if acc_check && gyr_check && mag_check && dt_check
35             data(k,:) = data_line;
36             fprintf('SUCCES! %d lines stored. line: %s \n', k, line);
37             k = k + 1;
38         else
39             fprintf('ERROR! line: %s \n', line);
40         end
41     catch
42         fprintf('ERROR! line: %s \n', line);
43     end
44 end
45
46 fprintf('Closing serial port connection... \n');
47 delete(com);
48
49 fprintf('Saving collected data to %s \n', FILE);
50 save(FILE);

```