Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Aplicación de una red de sensores inalámbricos distribuidos de bajo coste para el análisis de las propiedades dinámicas en 3 dimensiones de barcos en posiciones estacionarias

Grado en Ingeniería Mecánica

ALUMNO: Rafael Montero de Espinosa Alés

DIRECTORES: José Antonio González Prieto

Gerardo González-Cela Echevarría

CURSO ACADÉMICO: 2020-2021

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Aplicación de una red de sensores inalámbricos distribuidos de bajo coste para el análisis de las propiedades dinámicas en 3 dimensiones de barcos en posiciones estacionarias

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General / Infantería de Marina

Universidade Vigo

RESUMEN

La navegación es uno de los medios de transporte de mercancías y personas más importantes y variados. Dedicado principalmente al almacenaje y a la seguridad del entorno marítimo alrededor del mundo, hoy en día uno de los problemas más comunes para la ciencia es la respuesta a la pregunta de cómo mejorar las herramientas y sistemas de navegación, tanto en lo relativo a temas como la optimización de métodos de propulsión, los sistemas de navegación autónoma o los sistemas asociados a la seguridad del tráfico marítimo .

Recientemente, la investigación en el área de la navegación sin emplear sensores no inerciales (como el GPS) ha cobrado mayor importancia, sobre todo como un mecanismo estratégico que permita navegar con seguridad en situaciones en las que la señal GPS no esté disponible. Para ello se analizan los métodos de fusión de diferentes sensores como cámaras (navegación mediante odometría visual), magnetómetros (orientación del objeto respecto a los ejes Norte/Sur/Este/Oeste) y las unidades inerciales. Las unidades de medida inercial (IMU) son dispositivos electrónicos, normalmente de bajo coste, que se utilizan para obtener la información de las aceleraciones lineales y las velocidades angulares del objeto en el que se encuentran instaladas, de forma que mediante técnicas de fusión y filtrado permiten estimar la orientación (ángulos) y desplazamiento relativo (posiciones x,y,z) del objeto de estudio.

En este trabajo nos centramos en la aplicación de este tipo de sensores inerciales de forma que se procede a elaborar un prototipo en el que, mediante estos sensores, estudiaremos el comportamiento de dos objetos flotantes (con diferentes propiedades) dentro de una cubeta de agua por medio de la realización de una serie de tests que provoquen el movimiento de la estructura. En cada uno de los objetos se instalan una serie de sensores inerciales cuya información se envía por medio de una red wifi (protocolo MQTT) a un servidor que los recoge y almacena para su posterior procesado. Se aplicarán, además, métodos para el filtrado del ruido de las señales obtenidas y corrección del desvío provocado por los offsets en la señal recibida en reposo. Por último, se diseñarán algoritmos para la estimación de la orientación angular del objeto (*roll y pitch*) así como una introducción a un posible sistema de detección y localización de impactos del objeto flotante.

PALABRAS CLAVE

Dinámicas de cuerpos flotantes, Sensores inerciales, Filtrado, Estimación, Detección de choques

AGRADECIMIENTOS

En primer lugar, quería agradecer todo el apoyo que he tenido por parte de mi familia no solo a lo largo de este proyecto sino durante mi estancia en la Escuela Naval y en cualquier otra situación dentro de mi carrera militar, animándome a seguir día a día y sobre todo intentando sacarme una sonrisa en cada momento. Destacando sobre todo a mi madre, por su entrega diaria, recordándome siempre aquellos valores que hacen de mí una mejor persona y consiguiendo que nunca decaiga. Sin vuestro apoyo nada hubiese sido posible.

Agradecerle también a mi novia Natalia por haberme acompañado siempre en esta etapa final de mi carrera, por su paciencia infinita, por compartir conmigo los buenos y malos momentos y por conseguir que todo sea fácil.

Agradecer enormemente al profesor Don José Antonio González Prieto por ayudarme en todo momento con cualquier duda que pudiese surgirme, abrirme el camino hacia la programación, una parte de la ingeniería que hasta ahora no había descubierto y por haber estado en todo momento cuando le requería para solucionar cualquier problema. Sobre todo, agradecerle haberme hecho sentir un compañero más, por haberme hecho partícipe de esta gran idea que hemos llevado a cabo en el TFG y sobre todo por haber conseguido que unos meses de trabajo intenso se convirtiesen en mi afán por seguir mejorando día a día. Así mismo, agradecerle también al Capitán de Fragata Don Gerardo González-Cela Echevarría por su aportación magnífica del barco a escala que hemos utilizado para los experimentos.

A todos mis compañeros, por continuar muchos más años unidos y por empezar esta nueva etapa en nuestra carrera militar de la mejor manera posible.

CONTENIDO

Contenido 1

Índice de Figuras 3

Índice de Tablas 6

1 Introducción y objetivos 7

1.1 Preámbulo 7

1.2 Motivación 8

1.3 Objetivos 8

1.4 Estructura de la memoria 9

2 Estado del arte 10

2.1 Hidrodinámica de embarcaciones marinas 10

2.1.1 Concepto de embarcación 10

2.1.2 Grados de libertad y movimiento de una embarcación marina 11

2.1.3 Tipos de Modelos según su propósito 13

2.1.4 Teorías clásicas en el ámbito naval 14

2.2 Cinemática 18

2.2.1 Sistemas de coordenadas 18

2.2.2 Transformaciones entre BODY y NED 22

2.2.3 Transformaciones entre ECEF y NED 27

2.3 Filtrado y estimación 29

2.3.1 Filtros G-H 29

2.3.2 Filtro de Kalman 30

2.3.3 Filtro complementario 32

3 Desarrollo del TFG 34

3.1 Introducción 34

3.2 Instrumentación y medios materiales 34

3.2.1 Hardware 34

3.2.2 Software 36

3.3 Evolución del prototipo 42

3.3.1 Primer prototipo 42

3.3.2 Prototipo final: “Bloque” 44

3.4 Barco 45

3.5 Establecimiento de la conexión 46

4 Resultados / Validación / Prueba....49
4.1 Introducción....49
4.2 Calibraciones....50
4.2.1 Bloque....51
4.2.2 Barco....56
4.3 Estimación de posición....58
4.3.1 Bloque....60
4.3.2 Barco:....63
4.4 Identificación de choques....65
4.4.1 Bloque....65
4.4.2 Barco....66
Conclusiones y líneas futuras....68
Conclusiones y propuestas de mejora....68
Líneas futuras....68
5 Bibliografía....70
Anexo I: Código c (arduino)....73
Sensor inercial en módulo ESP8266....73
Anexo II: Código python....77
Script de toma de datos de calibración y tests dinámicos....77
Script de toma de análisis, filtrado y generación de gráficas de resultados....78
Funciones auxiliares empleadas por el script de análisis de resultados....80

ÍNDICE DE FIGURAS

Figura 2-1: Embarcaciones operando. [4] 11
Figura 2-2: Grados de libertad. [5] 12
Figura 2-3: Modelos utilizados en navegación, guiado y control [1] 13
Figura 2-4: Ecuaciones derivadas del modelo lineal. 15
Figura 2-5: Sistema de ejes coordenados, "*seakeeping theory".* [1] 17
Figura 2-6: Unified theory [7] 17
Figura 2-7: Rotación de la Tierra respecto al sistema ECI fijo [8] 18
Figura 2-8: Sistema de coordenadas ECEF (*e*) que rota respecto al ECI fijo (*i*) [1] 19
Figura 2-9: Sistema de coordenada NED [9] 20
Figura 2-10: Sistema de coordenadas BODY [1] 20
Figura 2-11: Área de flotación de una embarcación 21
Figura 2-12: Efectos de la flotabilidad [10] 21
Figura 2-13: Notación transformación de matrices [1] 23
Figura 2-14: Rotación en torno a λ. 24
Figura 2-15: Matrices de rotación [1] 24
Figura 2-16: Secuencia de rotación (zyx). El submarino se rota de $\{n\}$ a $\{b\}$. [1] 25
Figura 2-17: Velocidad lineal BODY [1] 25
Figura 2-18: Ecuaciones de la cinemática [4] 26
Figura 2-19: Sistemas NED y ECEF [4] 27
Figura 2-20: Rotaciones principales [12] 27
Figura 2-21: Transformación ECEF-NED [4] 28
Figura 2-22: Representación terrestre [4] 28
Figura 2-23: Ejemplo de estimación con un filtro G-H [13] 30
Figura 2-24: Principio de funcionamiento algoritmo Kalman [14] 31
Figura 2-25: Ecuaciones Newton 3D [15] 32
Figura 2-26: Ejemplo de aplicación filtro de Kalman 32
Figura 2-27: $\boldsymbol{\alpha}$=0.4 vs $\boldsymbol{\alpha}$=0.1 [16] 33
Figura 3-1: IMU MPU6050 [17] 35
Figura 3-2: Esquema interno IMU [20] 35
Figura 3-3: NodeMCU [21] 36
Figura 3-4: Logo software Arduino [22] 37
Figura 3-5: Entorno Anaconda 38

Figura 3-6: Legibilidad Python [25] 39
Figura 3-7: The Visualization Toolkit [26] 39
Figura 3-8: Tipos de gráficos [27] 40
Figura 3-9: Conexión MQTT. [28] 41
Figura 3-10: Subscribirse al servidor. [28] 41
Figura 3-11: Publicar. [28] 41
Figura 3-12: MQTT TFG [Fuente Propia] 42
Figura 3-13: Montaje en Protoboard [Fuente Propia] 43
Figura 3-14: Primer prototipo estructura acuática [Fuente Propia] 44
Figura 3-15: Centro de la estructura, punto de estimaciones [Fuente Propia] 44
Figura 3-16: Prototipo final [Fuente Propia] 45
Figura 3-17: Barco [Fuente Propia] 46
Figura 3-18: Comando Mosquitto [Fuente Propia] 46
Figura 3-19: Suscripción mosquitto [Fuente Propia] 47
Figura 3-20: Publicar en tópico [Fuente Propia] 47
Figura 3-21: Abrir framework [Fuente Propia] 47
Figura 3-22: Framework. Gráficas PyQt y representación 3D VTK. [Fuente Propia] 47
Figura 3-23: Establecimiento de Conexión broker. [Fuente Propia] 48
Figura 4-1: Calibración bloque. Aceleraciones sin filtrar. S2. [Fuente Propia] 51
Figura 4-2: Calibración bloque. Velocidades angulares del *roll, pitch y yaw. S2.* [Fuente Propia] 52
Figura 4-3: Calibración bloque. Histogramas de aceleraciones y velocidades angulares. S2. [Fuente Propia] 52
Figura 4-4: Calibración bloque. Histogramas de aceleraciones y velocidades angulares. S1, S2, S3 y S4. [Fuente Propia] 53
Figura 4-5: Calibración bloque. Aplicación filtro GH y offset para aceleraciones. S3. [Fuente Propia] 54
Figura 4-6: Calibración bloque. Aplicación filtro GH y offset para velocidades angulares. S3. [Fuente Propia] 54
Figura 4-7: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S3. [Fuente Propia] 55
Figura 4-8: Calibración bloque. Aplicación filtro GH y offset para aceleraciones. S1. [Fuente Propia] 56
Figura 4-9: Calibración bloque. Aplicación filtro GH y offset para velocidades angulares. S1. [Fuente Propia] 56
Figura 4-10: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S1. [Fuente Propia] 57
Figura 4-11: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S2. [Fuente Propia] 57
Figura 4-12: Kalman Filter [30] 58

Figura 4-13: Pitch y roll estimados mediante el filtro de Kalman-Test Lateral 1. [Fuente Propia].60

Figura 4-14: Aceleraciones Sensor 1. [Fuente Propia].....61

Figura 4-15: Pitch y roll estimados mediante el filtro de Kalman-Test Lateral 2. [Fuente Propia].61

Figura 4-16: Pitch y roll estimados mediante el filtro de Kalman-Test Longitudinal 1. [Fuente Propia].....62

Figura 4-17: Pitch y roll estimados mediante el filtro de Kalman-Test Longitudinal 2. [Fuente Propia].....63

Figura 4-18: Barco. Test lateral 2. Estimación del pitch y el roll. [Fuente Propia] 63

Figura 4-19: Barco. Test lateral 1. Estimación del pitch y el roll. [Fuente Propia] 64

Figura 4-20: Barco. Test Lateral 1. Aceleraciones. [Fuente Propia].....64

Figura 4-21: Bloque. Detección de colisiones en todos los sensores. [Fuente Propia] 66

Figura 4-22: Bloque. Probabilidad de colisión obtenida en cada sensor. [Fuente Propia].....66

Figura 4-23: Barco. Detección de colisiones en todos los sensores. [Fuente Propia] 66

Figura 4-24: Barco. Probabilidad de colisión obtenida en cada sensor. [Fuente Propia].....67

ÍNDICE DE TABLAS

Tabla 1: Notación SNAME 19
Tabla 2: Calibración bloque. Parámetros de offset obtenidos. 55
Tabla 3: Calibración bloque. Parámetros de dispersión del ruido. 55

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Preámbulo

La navegación marítima es un arte que ha estado presente durante más de 10000 años. Desde las primeras civilizaciones que utilizaban troncos tallados como medio de transporte a través del agua, pasando por los primeros barcos egipcios o fenicios fabricados muchos de ellos con papiros, hasta la construcción hoy en día de embarcaciones autónomas, mercantes con velas rígidas autodesplegables o barcos con alimentación fotovoltaica, entre otros. Las estrellas, el sol, la luna, el viento, las corrientes e incluso cuervos que avisaban ante avistamientos de tierra, era la forma de abrirse paso en los mares. Día a día, se han ido desarrollando nuevos sistemas de navegación y de posicionamiento que eran inimaginables hace unas décadas. El rápido avance de la tecnología hace que desde hace más de 70 años se integren nuevos tipos de procesadores que, por su abaratamiento, ocupan puestos importantes dentro de los numerosos equipos que componen los puentes de gobierno de los barcos, facilitando así el trabajo y la vida de las dotaciones.

La dependencia de los sistemas de control de estabilidad y navegación de un barco con respecto al análisis de los datos recibidos puede llevar a una situación en la que el fallo de uno de los sistemas de control pueda tener consecuencias irreversibles. Por tanto, la necesidad de contar con sistemas que, mediante técnicas de fusión de datos nos permitan estimar las propiedades dinámicas para diseñar sistemas que permitan navegar de **manera segura (y posiblemente autónoma en un futuro próximo)**, es un problema cuya solución se encuentra en desarrollo. La estimación correcta de las velocidades, la posición y orientación de la embarcación constituyen una herramienta que proporciona flexibidad y adaptabilidad a los algoritmos de control que permiten mejorar sus prestaciones. Debemos tener en cuenta que, en la actualidad, probablemente la mayor parte de los equipos que encontramos en los buques disponen de sistemas de posicionamiento no inercial (sistemas GPS) que les permiten conocer la información relativa a las variables de navegación. Si embargo, la fusión de esta información de GPS con datos procedentes de sensores inerciales (IMU) integrados en el barco, permitiría mejorar estas estimaciones aplicar su conocimiento al diseño de mejoras en lo que respecta a la estabilidad del buque y su capacidad de maniobra

Por otra parte, en sistemas en donde esta información inercial se obtiene de una única fuente, padecen de problemas de tolerancia a fallos, ya que, en caso de fallo del sensor, el sistema de control podría dejar de funcionar correctamente. Se necesita, por tanto, crear los denominados **sistemas tolerantes a fallos** en donde se emplea un conjunto de sensores que permite seguir operando en caso de fallo de un sensor, y además es posible detectar los fallos mediante métodos de análisis específicos.

En consecuencia, disponer de más de una fuente independiente que nos proporcione datos para poder estudiar el comportamiento de la embarcación navegando hace que la investigación sobre las combinaciones de distintos sistemas sea, en la actualidad, objeto de investigación.

1.2 Motivación

A nivel profesional y personal, como Alférez de Fragata alumno de la Escuela Naval Militar y estudiante de Ingeniería Mecánica, la idea de afrontar un trabajo que involucrase tanto el ámbito naval, a la hora de estudiar nuevos sistemas de navegación, como el ámbito ingenieril, que involucra la manera de realizar este proyecto, permite profundizar en todas aquellas necesidades técnicas y poder llegar a una serie de conclusiones. Es por ello, que este trabajo es un buen ejemplo de ello, no solo por el enfoque didáctico que contiene sino también por el práctico.

La realización de este TFG me hace sentir partícipe de lo que puede ser un punto de inflexión dentro de los futuros sistemas de navegación que pudieran ser de interés no solo en la náutica sino en el ámbito militar. La posibilidad de desarrollar un proyecto de este tipo en el que la utilidad tenga aplicación directa hace que despierte en mí las ganas de seguir trabajando posteriormente en la mejora de estos sistemas y de poder implementarlos en diferentes prototipos a escala para poder, en un futuro, utilizarlo en nuestros barcos.

Así mismo, aprender a extraer diferentes conclusiones y aplicar la lógica a través de diferentes programas informáticos, realizar un estudio sobre todas aquellas áreas que abarca este proyecto, así como el constante perfeccionamiento de aquellos problemas que nunca supusieron un obstáculo, sino algo nuevo que aprender, fueron mis mayores estímulos durante la realización de este trabajo. Todo ello ha mejorado mi capacidad de adaptación a nuevas situaciones conforme íbamos avanzando en el proyecto pues cada día integrábamos una nueva mejora o una manera más sencilla de conseguir nuestros objetivos.

1.3 Objetivos

Son muchos los objetivos que se pueden abarcar en este TFG, sin embargo, el objetivo principal consiste en la aplicación de una red de sensores inalámbricos de bajo coste, para ser instalados en un prototipo elaborado a lo largo de este proyecto, que va a ser empleado para realizar el análisis de las propiedades dinámicas en 3 dimensiones de objetos flotantes en posiciones estacionarias. En concordancia con ello, se pueden definir una serie de objetivos secundarios dependientes del objetivo principal.

- Diseño de la carcasa en donde se integren 4 sensores inerciales, un módulo encargado de la conexión a la red wifi y la batería de alimentación.
- Instalación de los sensores en el prototipo.
- Instalación de los sensores en maqueta de barco a escala.
- Programación de los sensores para el envío de mensajes al servidor MQTT (“Message Queing Telemetry Transport”).
- Programación en Python+QT de la aplicación que recibe los mensajes, los almacena correctamente y genera gráficas donde visualizar los datos de los sensores, así como una representación 3D.
- Detección de choque a través de los datos recibidos.
- Diseño de algoritmos para la estimación del estado actual y futuro de nuestro modelo.
- Detección de choque a través de los datos recibidos.

- Integración con los resultados del TFG-11 para poder utilizar este sistema como generador de posición verdadera (ground truth) que permita evaluar numéricamente el resultado obtenido en la estimación de los ángulos de *roll y pitch*.

1.4 Estructura de la memoria

Para concluir, tras la contextualización del presente TFG y definir los objetivos del mismo, a continuación, se describe la organización de la memoria con una visión global de cada uno de los apartados.

En primer lugar, el apartado 0 se divide en dos partes fundamentales. En la primera de ellas, se describe el estado del arte de la navegación. Para ello, se presenta inicialmente una definición de embarcación en sí y de su comportamiento hidrodinámico, para a continuación pasar a definir aquellos modelos que se utilizan para situar y describir el movimiento de un barco navegando. Con el fin de representar cómo será el comportamiento de nuestro prototipo durante distintos experimentos que se llevarán a cabo durante el proyecto, se describe en el apartado 2.2 todos aquellos sistemas de referencia que pueden adoptarse, así como la transformación de coordenadas entre diferentes sistemas de referencia. Para concluir, pasamos a describir el método utilizado para filtrar el ruido procedente de las señales obtenidas de las unidades inerciales, así como la base teórica de los algoritmos que nos permitirán estimar la orientación angular del objeto.

Al comienzo del apartado 3 se incluye la instrumentación y los medios utilizados para la elaboración de este proyecto. Tras esto, se muestra la evolución del prototipo inicial con las mejoras que se han ido implementando hasta llegar al bloque y el barco. En ambos objetos flotantes se instalarán los respectivos sensores y se realizarán las diferentes pruebas. Además, se realiza un desglose del modo de proceder para el funcionamiento del sistema desde el inicio en reposo hasta el envío de datos por pantalla.

En el último apartado se plasman todos aquellos resultados obtenidos durante el desarrollo del trabajo, puntualizando las pruebas que han sido realizadas: calibraciones y movimientos en la cubeta con agua. Se expone el plan de pruebas con los estudios que se han llevado a cabo para el posterior análisis de los resultados.

En el apartado de Conclusiones y líneas futuras se exponen todas aquellas conclusiones a las que se han llegado tras la realización de este TFG junto con una serie de propuestas de mejora que dan lugar a las líneas de trabajo futuras.

En el apartado 5 se cierra el proyecto presentando la bibliografía de la que nos hemos apoyado para la realización tanto del estudio como del desarrollo del trabajo.

Para finalizar se pueden consultar los anexos del proyecto, correspondiente al código Arduino que compilaremos en los sensores, así como las diferentes funciones que diseñaremos en Python para la realización de las diferentes calibraciones, tests, presentación de histogramas, filtrado del ruido, estimaciones y detección de colisiones.

2 ESTADO DEL ARTE

2.1 Hidrodinámica de embarcaciones marinas

2.1.1 Concepto de embarcación

Tal y como se ha comentado en los objetivos del TFG, el estudio de las dinámicas asociadas al movimiento de embarcaciones en diferentes situaciones de navegabilidad tiene múltiples aplicaciones relacionadas con la creación de sistemas de monitorización para el desarrollo de maniobras seguras o para ser empleados en sistemas robóticos móviles que se encuentran a bordo de la embarcación (robots de carga, plataformas de estabilización, sistemas de control de disparo, etc.). Para ello es necesario realizar una serie de definiciones que establezcan el marco de trabajo asociado a los modelos matemáticos que serán empleados para el tratamiento del análisis propuesto.

La definición de [1] define una embarcación del siguiente modo:

"estructura hueca hecha para flotar sobre el agua con fines de transporte y navegación; especialmente, uno que sea más grande que un bote de remos".

De esta forma cuando nos referimos a una embarcación podemos considerar que es posible incluir bajo esta definición barcos, embarcaciones de alta velocidad, semi-sumergibles, lanchas, botes, submarinos, veleros, piraguas, vehículos submarinos, torpedos, campos aéreos flotantes, plataformas de carga flotante o incluso plataformas de generación de energía por el movimiento de las olas Aunque habitualmente suele emplearse el término barco para referirse de forma genérica a las embarcaciones de grandes dimensiones, y bote a las pequeñas, es conveniente definirlos con cierta precisión.

De esta forma, es habitual realizar la distinción entre un barco y un bote por su tamaño mediante la siguiente proposición [2]:

"Barco: cualquier embarcación flotante capaz de navegar en aguas abiertas. Hoy en día, está referido a aquellas con mas de 500 toneladas de desplazamiento. Los barcos sumergibles generalmente se denominan barcos independientemente de su tamaño".

Para realizar el estudio de las dinámicas de las embarcaciones, es necesario introducir los conceptos físicos asociados al movimiento de las embarcaciones, así como los sistemas de referencia empleados para la creación de los modelos dinámicos.

Desde un punto de vista hidrodinámico una embarcación marina puede clasificarse según su velocidad máxima operativa. Para ello, suele usarse el número de Froude:

$$Fn = \frac{U}{\sqrt{gL}}$$

Donde U es velocidad de la embarcación, L es la longitud total sumergida y g es la aceleración de la gravedad. La presión que soporta la embarcación puede dividirse en presión hidrostática e hidrodinámica.

Es habitual que, empleando el número de Froude, se emplee la siguiente clasificación de los barcos que navegan a una velocidad constante U [3]:

***"Buques de desplazamiento** (Fn <0,4): la fuerza de flotabilidad domina en relación con las fuerzas hidrodinámicas.*

***Buque de semidesplazamiento** (0.4−0.5 <Fn <1.0−1.2): La fuerza de flotabilidad no es dominante a la velocidad máxima de operación para un tipo de embarcación con casco sumergido de alta velocidad.*

***Vaso de planeo** (Fn> 1.0−1.2): La fuerza hidrodinámica soporta principalmente el peso. Habrá una fuerte separación del flujo y las fuerzas aerodinámicas de elevación y arrastre comenzarán a desempeñar un papel."*

Figura 2-1: Embarcaciones operando. [4]

2.1.2 Grados de libertad y movimiento de una embarcación marina

Para entender correctamente los movimientos y oscilaciones de las embarcaciones en la mar como resultado de todas las fuerzas actuantes sobre ellas, es necesario definir los grados de libertad que surgen de los mismos.

Una embarcación experimenta 6 grados de libertad (DOFs: "*Degrees of Freedom"*) que son los desplazamientos y rotaciones que especifican la posición y orientación de la embarcación.

Los tres tipos de movimientos de traslación son: vertical (ascenso y descenso o arfada), lateral a una banda u otra (abatimiento) y longitudinal (avance o retroceso).

La embarcación, además, puede girar alrededor de su eje vertical, transversal y longitudinal. El giro sobre el eje 'Z' describe el rumbo del barco y se denomina guiñada o virada. Respecto al eje 'Y' recibe el nombre de cabeceo y en torno al 'X' escora; aunque si es oscilatorio puede llamarse balance.

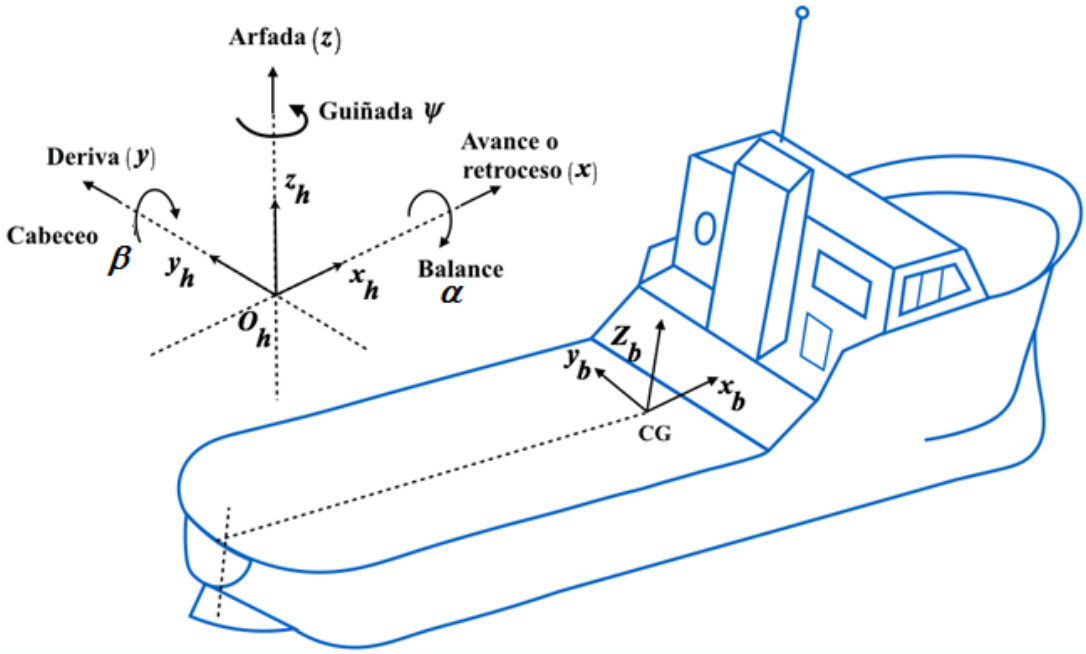

Figura 2-2: Grados de libertad. [5]

Sin embargo, a la hora de diseñar los sistemas de control de una embarcación, se utilizan a menudo modelos de orden reducido ya que muchas embarcaciones no tienen efectos en todos los grados de libertad en las situaciones que son objeto de estudio. Estos modelos pueden ser:

- **1 Grado de Libertad**: diseño de controladores de velocidad de avance (avance o retroceso), pilotos automáticos (virada) o sistemas de amortiguación de balanceo (escora).

- **3 Grados de Libertad**: Avance/retroceso, virada y abatimiento. Barcos, semi-sumergibles o submarinos con sistemas de posicionamiento dinámico, seguimiento de trayectorias.

- **4 Grados de Libertad**: Se incluye al modelo de plano horizontal de 3 Grados de Libertad la **escora**. Estos modelos se utilizan en situaciones de maniobra donde es importante incorporar la escora para así tratar de reducirla mediante control activo de aletas, timones o tanques de líquido estabilizador.

- **6 Grados de Libertad**: Arfada, abatimiento, cabeceo, virada, escora, avance/retroceso. Estos modelos contienen todas las ecuaciones de movimiento para simular y estimar el movimiento de las embarcaciones.

2.1.3 Tipos de Modelos según su propósito

Los diferentes modelos definidos anteriormente, cuya diferencia principal se encuentra en el número de grados de libertad entre cada uno de ellos, pueden ser utilizados para diferentes finalidades, entre las que se pueden destacar: estimaciones, simulaciones en tiempo real y diseño controlador-observador.

Podemos distinguir tres modelos:

- **Modelo de simulación:** Es la descripción más adecuada de un sistema, se trata de un modelo de alta fidelidad de 6 Grados de Libertad para simulación de numerosos mecanismos en el dominio del tiempo. Incluye la dinámica de la embarcación, sistema de propulsión, de medida y las fuerzas del entorno como pueden ser las olas, el viento o las corrientes oceánicas.

- **Modelo de control:** Es una versión simplificada del modelo anteriormente descrito y es utilizado para diseñar los sistemas de mecanismo de control.

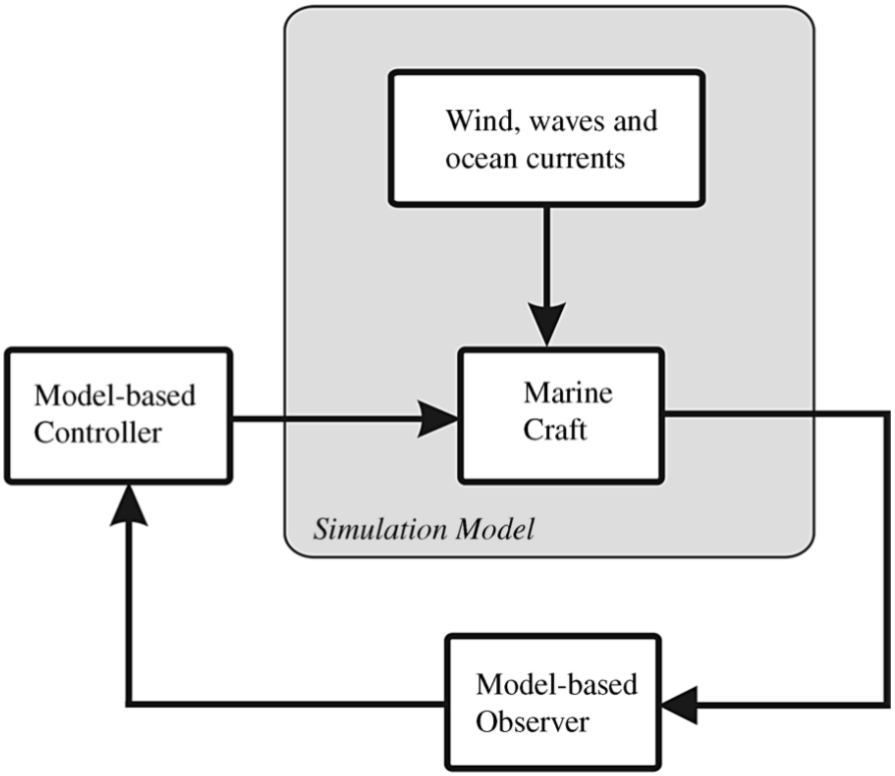

Figura 2-3: Modelos utilizados en navegación, guiado y control [1]

- **Modelo de observador:** este modelo tiene como propósito capturar las dinámicas adicionales asociadas a los sensores de navegación, así como las perturbaciones. Es una versión simplificada del modelo de simulación donde se le da más importancia a la medida del ruido, situaciones de fallo, capacidad de estima, filtrado y predicción de movimiento.

2.1.4 Teorías clásicas en el ámbito naval

2.1.4.1 "Maneuvering theory"

En esta teoría se encuadran todos aquellos barcos que navegan a **velocidad constante positiva *U* en mar en calma** donde se asume que los coeficientes hidrodinámicos son independientes de la frecuencia (sin excitación de onda). En su representación más simple, el modelo de maniobra será lineal, mientras que las representaciones no lineales se pueden derivar usando diferentes métodos para representar las fuerzas disipativas. Esto se combina con modelos que describen los coeficientes potenciales, la cinemática y la cinética del cuerpo rígido para obtener un modelo completo.

Por tanto, se asume que el barco se desplaza en aguas tranquilas restringidas, como por ejemplo en un puerto, velocidad positiva *U* y bajo un supuesto de excitación de onda de frecuencia cero, de modo que la masa y la amortiguación agregadas se pueden representar utilizando derivadas hidrodinámicas (parámetros constantes).

A velocidades altas, es decir, en una situación real donde sea necesario maniobrar, tendrán efecto tanto el fluido como fuerzas no lineales, así como la viscosidad. Esto conlleva un enfoque diferente cuando el barco se encuentra maniobrando.

Al asumir que la frecuencia es cero, se suprimen así todas aquellas matrices que dependen de la misma. Sin embargo, asumir esto solo es válido para el modelo de **3 Grados de Libertad** descrito en (2.1.2) donde los periodos varían entre 100-150 s y la frecuencia para 150s es cercana a cero:

$$\omega_n = \frac{2\pi}{T} \approx 0.04\, rad/s$$

La **cinética** de cuerpo rígido puede expresarse como:

$$M_{RB}\dot{v} + C_{RB}(v)v = \tau_{RB}$$

$$\tau_{RB} = \tau_{hyd} + \tau_{hs} + \tau_{wind} + \tau_{wave} + \tau_{control}$$

Ecuación 2-1

Donde M_{RB} se refiere a la matriz de inercia, C_{RB} a la de Coriolis y τ_{RB} al vector de fuerzas.

Así mismo, la **velocidad** generalizada de una embarcación se puede describir como:

$$v = [u, v, \omega, p, q, r]^{\intercal}$$

Las primeras tres componentes se refieren al avance/retroceso, deriva y arfada y las tres últimas a la escora, cabeceo y guiñada, como se describe en la Figura 2-2: Grados de libertad. [5]

Respecto a la **fuerza** generalizada que actúa sobre la embarcación, esta se expresa:

$$\boldsymbol{\tau_i} = [\boldsymbol{X_i} + \boldsymbol{Y_i} + \boldsymbol{Z_i} + \boldsymbol{K_i} + \boldsymbol{M_i} + \boldsymbol{N_i}]^{\intercal} \qquad \boldsymbol{i} \in \{\boldsymbol{hid}, \boldsymbol{hs}, \boldsymbol{viento}, \boldsymbol{oleaje}, \boldsymbol{control}\}$$

Donde los subíndices describen, por orden, la masa hidrodinámica añadida, amortiguación potencial debido a la radiación de las olas y la amortiguación viscosa, fuerzas hidrostáticas, viento, oleaje y fuerzas de control y propulsión.

Este modelo deriva de la segunda ley de Newton $F = ma$ donde la *F* representa la fuerza a la que esta sometido el sistema de estudio, *m* a la masa de este y *a* la aceleración que sufre. Los términos Coriolis y Centrípeto se refieren a la rotación del sistema de coordenadas fijo en el cuerpo ("*BODY*") respecto al sistema de referencia inercial ("ECI") descritos en la Figura 2-8: Sistema de coordenadas ECEF (*e*) que rota respecto al ECI fijo (*i*) [1]. Las fuerzas hidrodinámicas que se incluyen en la Ecuación 2-1 pueden describirse según teorías lineales o no lineales:

2.1.4.1.1 Modelos lineales

Para el caso de sistemas con **6 Grados de Libertad** habría en total 36 masas y 36 elementos amortiguadores proporcionales a la velocidad y aceleración. Además, habrá fuerzas de propulsión y medioambientales. De la fuerza hidrodinámica τ_{hyd} obtendríamos la masa lineal añadida y las fuerzas amortiguadoras, según la notación [6]:

$$\begin{aligned} X_1 &= X_u u + X_v v + X_w w + X_p p + X_q q + X_r r \\ &\quad + X_{\dot{u}} \dot{u} + X_{\dot{v}} \dot{v} + X_{\dot{w}} \dot{w} + X_{\dot{p}} \dot{p} + X_{\dot{q}} \dot{q} + X_{\dot{r}} \dot{r} \\ &\vdots \\ N_1 &= N_u u + N_v v + N_w w + N_p p + N_q q + N_r r \\ &\quad + N_{\dot{u}} \dot{u} + N_{\dot{v}} \dot{v} + N_{\dot{w}} \dot{w} + N_{\dot{p}} \dot{p} + N_{\dot{q}} \dot{q} + N_{\dot{r}} \dot{r} \end{aligned}$$

Figura 2-4: Ecuaciones derivadas del modelo lineal.

2.1.4.1.2 Modelos no lineales

La aplicación de teorías no lineales tiene efecto en aquellos sistemas donde es necesario incorporar efectos con características complejas (como puede ser la interacción entre variables de control, efectos de viscosidad variable, etc.) que dificultan mucho el modelo y análisis de este tipo de sistemas dinámicos. En ocasiones se trabaja empleando aproximaciones lineales de carácter local, es decir, en el punto de trabajo actual, para simplificar el análisis (se conoce como aproximación linealización de un sistema en un punto, que puede realizarse de varias formas) y en otras ocasiones se procede a trabajar directamente con modelos no lineales que requieren un análisis teórico avanzado de sus propiedades, así como un análisis numérico de las soluciones obtenidas. Algunos ejemplos de los métodos empleados son los siguientes:

- **Series de Taylor**: El teorema de Taylor y su fórmula asociada, la serie de Taylor, es de gran valor en el estudio de métodos numéricos. En esencia, la serie de Taylor proporciona un medio para predecir el valor de una función en un punto en términos del valor de la función y sus derivadas en otro punto. En particular, el teorema establece que cualquier función suave puede aproximarse como un polinomio. El teorema se define de la siguiente manera:

    - Si una función *f* y su primera derivada de orden *n +1* son continuas en un intervalo que contenga *a* y *x* el valor de la función en *x* será:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n$$

Las funciones que admiten la expansión de Taylor en un punto son denominadas funciones analíticas, y permiten asegurar que la función es diferenciable (necesario para poder crear una aproximación local) y continua en el punto. Mediante este tipo de expansión se crearon los denominados sistemas dinámicos lineales. En el caso de sistemas mecánicos, aplicando las leyes de Newton, las fuerzas y momentos aparecen en las derivadas segundas de las posiciones lineales y angulares (F=ma; M=Iα), por lo que la expansión de Taylor a emplear para reflejar la presencia de fuerzas y momentos será de tercer orden.

- **Primeros principios:** A diferencia de los modelos basados en expansiones de la serie de Taylor de tercer orden, los modelos basados en primeros principios requieren un menor número de parámetros. Además, los efectos hidrodinámicos como la sustentación y la resistencia se modelan utilizando modelos bien establecidos. Esto da como resultado modelos lagrangianos físicamente sólidos que preservan las propiedades energéticas.

2.1.4.2 "Seakeeping theory"

Tal y como se ha explicado en el apartado anterior (2.1.4.1), "*maneuvering theory*" estudia el movimiento de una embarcación navegando con la mar en calma. "*Seakeeping theory*", por otro lado, es el estudio cuando hay excitación de las olas y el barco mantiene constante su rumbo ψ y su velocidad U (que incluye el caso de velocidad cero). Esto introduce una fuerza disipativa conocida como efectos de memoria fluida. Aunque ambas áreas se ocupan de los mismos temas, estudio de movimiento, estabilidad y control, la separación entre ambos casos permite realizar diferentes suposiciones que simplifican su estudio.

El uso principal de esta teoría está en conexión con el diseño de nuevos barcos. Es importante, para el diseñador de barcos, tener un buen conocimiento de las limitaciones de la embarcación aplicando las teorías y métodos creados bajo la hipótesis de estas circunstancias. Como ejemplo de ello, un problema común en el diseño de embarcaciones comerciales surge de aspectos relacionados con su comportamiento en la mar que se descubren demasiado tarde para ser corregidos por el proceso de diseño. Después de que todos los parámetros principales, que influyen en el comportamiento en la mar, se hayan fijado en función de otras consideraciones, como la resistencia al agua en calma, es posible que surjan nuevas situaciones de navegación en las que la configuración propuesta no obtenga buenos resultados. Es decir, hay que tener en cuenta qué teoría utilizar que optimice el proceso de diseño de una embarcación, así como el estudio de esta en situaciones reales, y determinar las capacidades de adaptación a situaciones inesperadas de maniobrabilidad.

"*Seakeeping theory*" se formula empleando un sistema de coordenadas $\{s\}$, donde el vector de estado $\xi = [\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6]^\top$ representa las perturbaciones respecto a un estado de equilibrio fijo. En la siguiente Figura se puede apreciar que, ante la acción del oleaje, el sistema de referencia $\{b\}$ rota respecto a $\{s\}$.

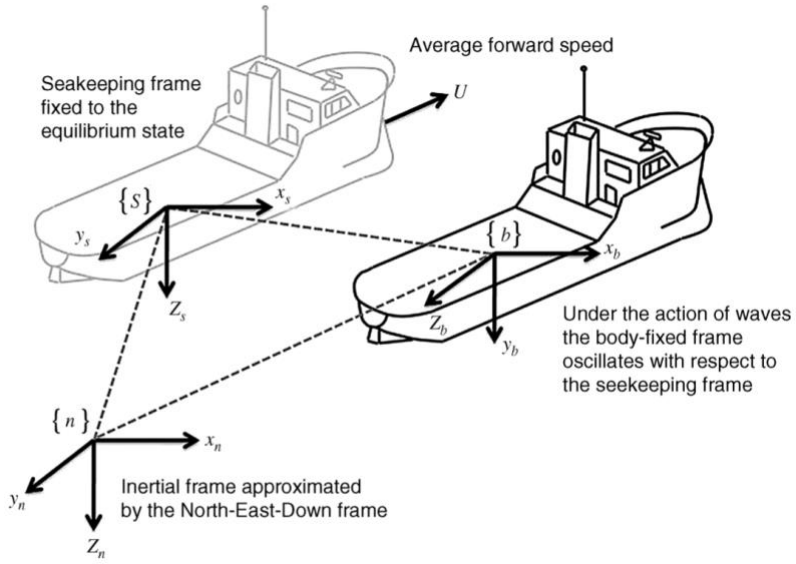

Figura 2-5: Sistema de ejes coordenados, "*seakeeping theory*". [1]

Por último, podemos expresar matemáticamente este modelo como una ecuación diferencial:

$$[\boldsymbol{M}_{RB} + \boldsymbol{A}(\omega)]\ddot{\boldsymbol{\xi}} + \boldsymbol{B}_{\text{total}}(\omega)\dot{\boldsymbol{\xi}} + \boldsymbol{C}\boldsymbol{\xi} = \boldsymbol{\tau}_{\text{wind}} + \boldsymbol{\tau}_{\text{wave}} + \delta\boldsymbol{\tau}$$

Ecuación 2-2

2.1.4.3 Teoría unificada

Tras presentar las dos posibles teorías que pueden describir una embarcación el comportamiento de una embarcación, es necesaria una teoría unificada que nos permita realizar simulaciones en el dominio del tiempo de embarcaciones en vías marítimas. Para ello, se hace uso de la Ecuación 2-2 de "*Seakeeping Theory*" y se asume una superposición lineal tal que las fuerzas inducidas por las olas se pueden sumar para diferentes velocidades *U* y estados del mar. Se utiliza una suposición similar para agregar fuerzas de amortiguación y restauración no lineales de modo que el modelo resultante sea un modelo no lineal unificado que combina los términos más importantes tanto de "*Maneuvering Theory*" como de "*Seakeeping Theory*". Por último, hay que tener cuidado al aplicar esta teoría unificada debido el problema de que los efectos hidrodinámicos se pueden modelar dos veces cuando se combinan los resultados de dos teorías.

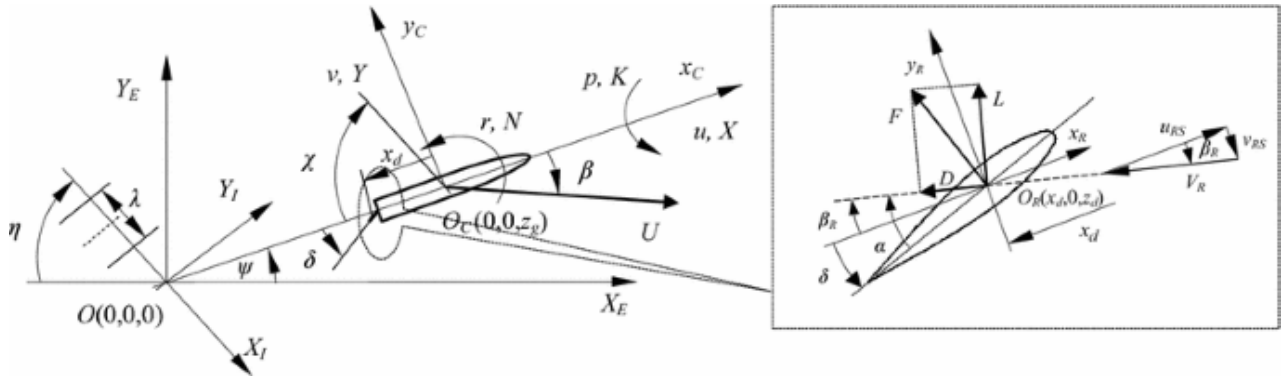

Figura 2-6: Unified theory [7]

2.2 Cinemática

2.2.1 Sistemas de coordenadas

En la navegación, guiado y el control de una embarcación, hay varios sistemas de coordenadas (o marcos) que se utilizan de forma intensiva en el diseño y el análisis. Para poder analizar el comportamiento de una embarcación con 6 grados de libertad, en este apartado se resumen aquellos sistemas de coordenadas que pueden ser adoptados y así poder entender el comportamiento de nuestra estructura acuática en los diferentes experimentos, que incluyen:

2.2.1.1 Sistemas de coordenadas centrados en la Tierra

ECI ("*The Earth-centered inertial*"): es un sistema de coordenadas inercial (es decir, "no acelerado") utilizado en la navegación terrestre y en el que se aplican las leyes de Newton y adquieren casi la totalidad de su validez, por lo que nos permite visualizar fácilmente otros marcos de referencia inerciales. Su origen se encuentra en el centro de masa de la Tierra y están **fijos** respecto a las estrellas, por lo que la rotación de la Tierra se hace sobre el eje Z (coincide con el eje polar), su eje X apunta al equinoccio Vernal y el Y coincide con el Ecuador. Debido a las fuerzas que ejercen el Sol y la Luna, el plano ecuatorial de la Tierra se mueve con respecto a la esfera celeste. La Tierra gira mientras que el sistema de coordenadas ECI no. Es utilizado en los sistemas de navegación inerciales.

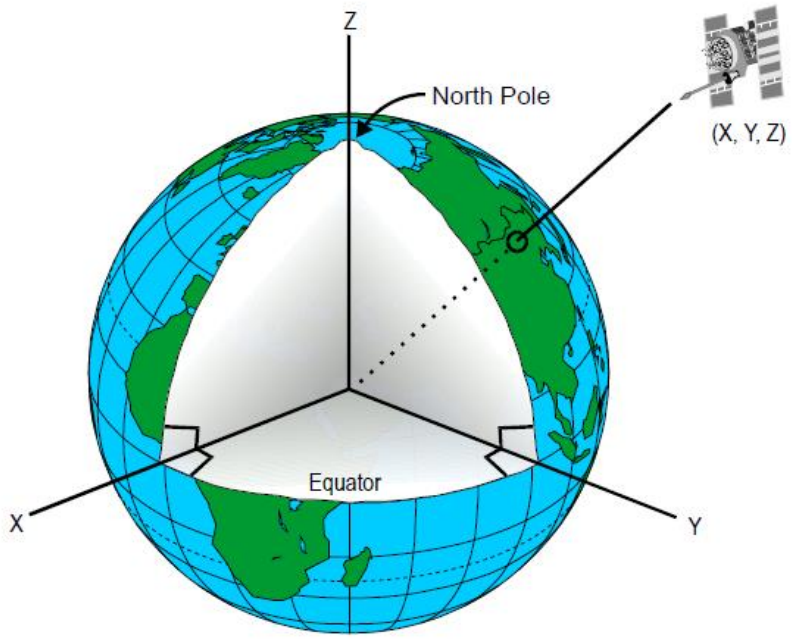

Figura 2-7: Rotación de la Tierra respecto al sistema ECI fijo [8]

ECEF {*e*} ("*Earth-centered Earth-fixed*"): en contraste con ECI, se trata de un sistema de coordenadas que gira alrededor del eje de rotación de la Tierra con respecto a las estrellas. Sus ejes están alineados con el polo de referencia internacional y el meridiano de referencia internacional que se encuentran fijos con respecto a la superficie de la Tierra. Sin embargo, en este tipo de sistemas la Tierra no gira alrededor del eje Z. El eje Z se extiende a través del norte verdadero, que no coincide con el eje de rotación instantáneo de la Tierra. Esto significa que ECEF gira con la tierra y, por lo tanto, las coordenadas de un punto fijo en la superficie de la tierra no cambian.

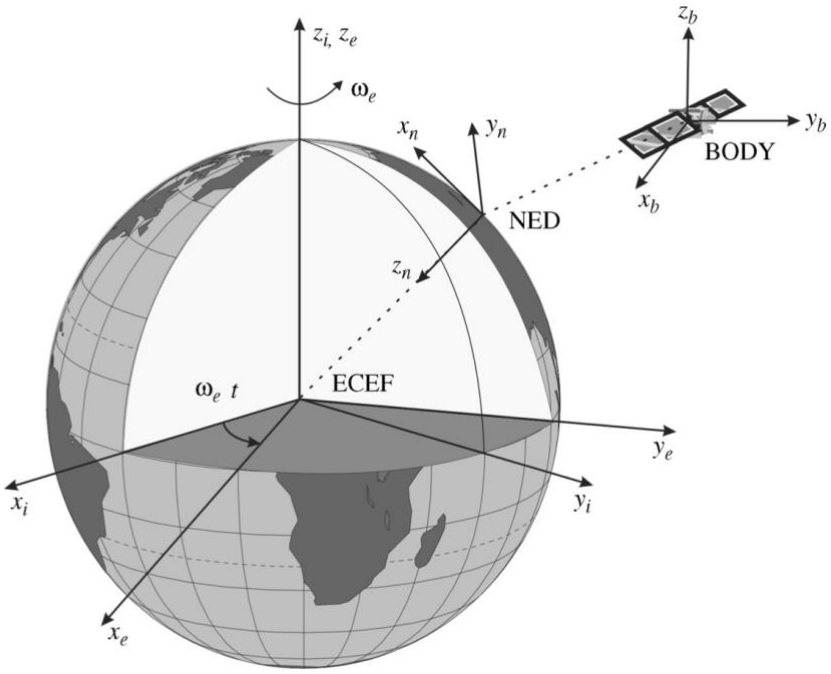

Figura 2-8: Sistema de coordenadas ECEF (*e*) que rota respecto al ECI fijo (*i*) [1]

La notación según la [6] "*Nomenclature for Treating the Motion of a Submerged Body Through a Fluid*" para buques será:

DOF		Forces and moments	Linear and angular velocities	Positions and Euler angles
1	Motions in the *x*-direction (surge)	X	u	x^n
2	Motions in the *y*-direction (sway)	Y	v	y^n
3	Motions in the *z*-direction (heave)	Z	w	z^n
4	Rotation about the *x*-axis (roll)	K	p	ϕ
5	Rotation about the *y*-axis (pitch)	M	q	θ
6	Rotation about the *z*-axis (yaw)	N	r	ψ

Tabla 1: Notación SNAME

2.2.1.2 Sistemas de coordenadas geográficos

NED $\{n\}$**:** Sistema de referencia local al que nos referimos en nuestro día a día en la mar y que se basa en la localización de los sistemas inerciales (latitud y longitud). Sus ejes X Y se encuentran en un plano **tangente** a la Tierra con origen en un punto situado en dicho plano y se mueven con respecto a la embarcación. Sin embargo, no tienen la misma orientación que los ejes fijos a la embarcación en movimiento. Normalmente, el eje X apuntará al Norte, el Y al Este y el Z hacia abajo, es por ello que se le denomina "*North East Down*" aunque deberá especificarse previamente.

Para las embarcaciones marinas que operan en un área local, de longitud y latitud aproximadamente constantes, se utiliza un plano tangente fijo a la Tierra en la superficie para la navegación. Esto generalmente se conoce como navegación por la Tierra plana y las leyes de Newton todavía siguen teniendo vigencia. Cabe destacar que este tipo de sistema de coordenadas presenta dificultades en latitudes cercanas a los polos.

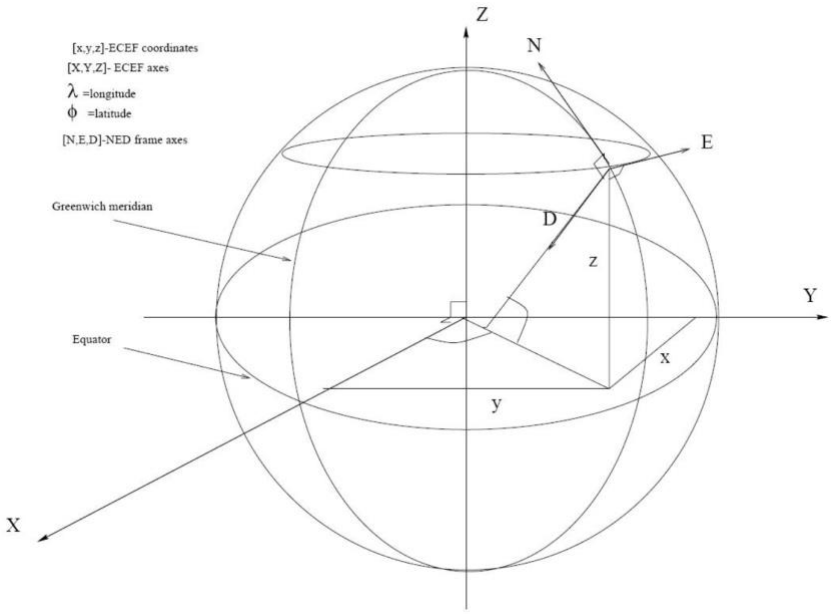

Figura 2-9: Sistema de coordenada NED [9]

BODY $\{b\}$**:** se trata de un sistema de coordenadas en movimiento **fijo a la embarcación** cuya posición y orientación se describen en relación con el sistema de referencia inercial (ECI) mientras que las velocidades lineales y angulares deben expresarse en el sistema de coordenadas fijas en el cuerpo. Normalmente, el origen se elige para que coincida con un punto en el centro del barco en la línea de flotación y los ejes coordenados se hacen coincidir con los ejes principales de inercia de la embarcación.

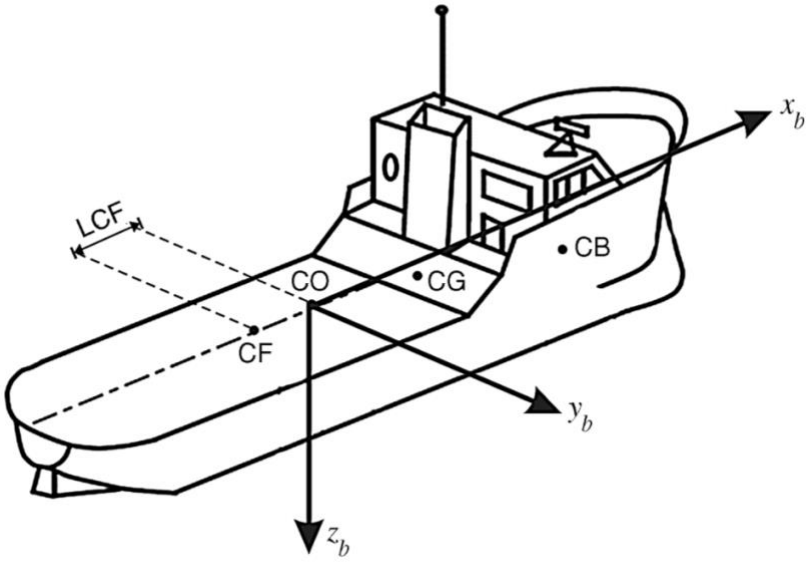

Figura 2-10: Sistema de coordenadas BODY [1]

2.2.1.3 Puntos de referencia

Los movimientos que surgen en una embarcación debido a sus 6 grados de libertad, los pesos existentes a bordo, además del propio peso del buque o el volumen sumergido provocan, a su vez, la aparición de ciertos puntos de gran importancia dentro de la misma. En primer lugar, el **centro de flotación (CF)**, que es el centroide del Área de flotación (Figura 2-11) y punto de apoyo de los movimientos de balanceo y cabezada.

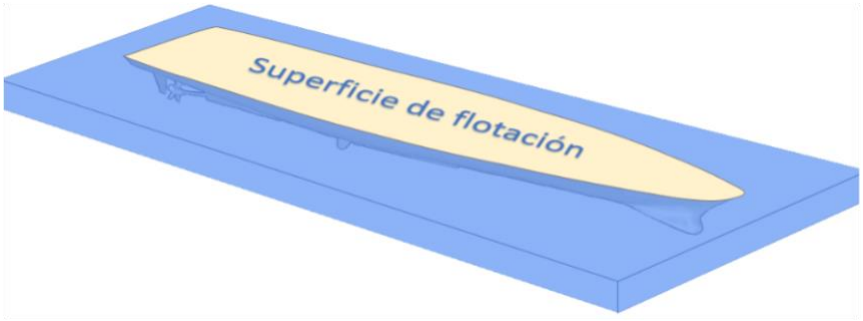

Figura 2-11: Área de flotación de una embarcación

El desplazamiento es el peso total del buque y será igual al peso del volumen del líquido desalojado por su carena. El punto de aplicación del conjunto de todos los pesos del buque, que será la suma del desplazamiento en rosca (buque vacío, tras salir del astillero) y de todos aquellos pesos añadidos posteriormente será el centro de gravedad (**G**) que provocará que el buque esté en equilibrio cuando se cumpla que el desplazamiento del mismo sea igual al empuje que sufre y se encuentren situados en el mismo plano.

La carena, o también llamada obra viva, es la parte del barco que se encuentra por debajo del Área de flotación y será diferente según la parte que se encuentre sumergida. Es decir, dependiendo de la flotación considerada, surgirá el centro de carena (**B**), que será el centro geométrico del volumen de carena para esa flotación con la que navegue el barco. Así mismo, surge el Metacentro (**M**), que será la intersección de los diferentes centros de carena en una línea que une dichos centros con el centro de gravedad cuando el buque se encuentra en equilibrio.

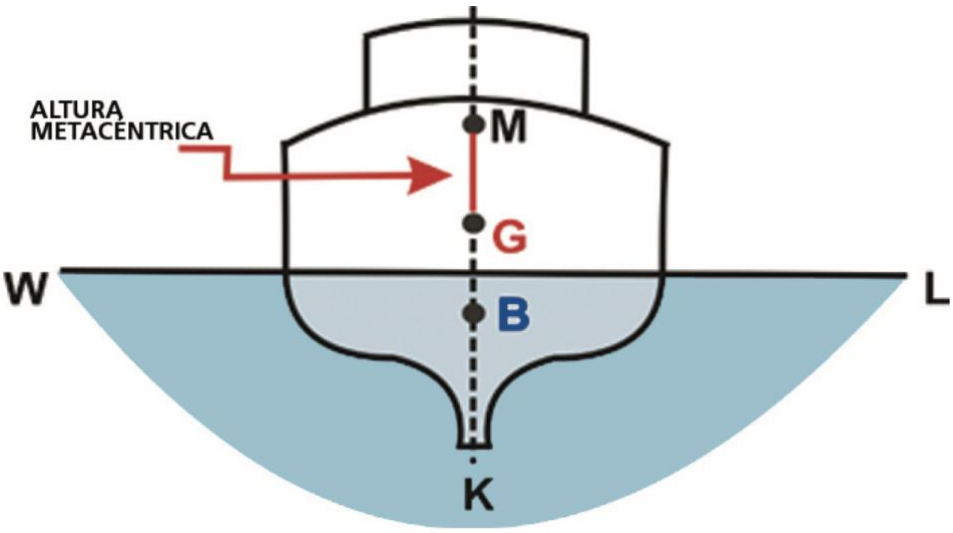

Figura 2-12: Efectos de la flotabilidad [10]

Por último, cabe destacar que los valores propios de las ecuaciones lineales de movimiento de embarcaciones con 6 Grados de Libertad son independientes del punto de referencia, pero las ecuaciones desacopladas producirán resultados incorrectos si se formulan en un punto diferente al CF.

2.2.1.4 Notación vectorial para 6 grados de libertad

Tras definir los diferentes sistemas de coordenadas que pueden adoptarse, es necesario desglosar aquellos vectores que influirán en cualquier embarcación con 6 grados de libertad. En primer lugar,

cualquier vector director a través de un segmento lineal relativo a un sistema de coordenadas vendrá expresado como:

$$\vec{u} = u_1^n \vec{n}_1 + u_2^n \vec{n}_2 + u_3^n \vec{n}_3$$

Así mismo, podemos representar este vector por su forma de coordenadas $\mathbf{u^n}$ del vector *u* en el sistema de coordenadas $\{n\}$:

$$\boldsymbol{u}^n = [u_1^n, u_2^n, u_3^n]^\top$$

Para las embarcaciones, se adoptará la siguiente notación para los diferentes vectores en los sistemas de coordenadas $\{b\}, \{n\}\ y\ \{e\}$:

$\boldsymbol{v}_{b/n}^e$ = linear velocity of the point o_b with respect to $\{n\}$ expressed in $\{e\}$

$\boldsymbol{\omega}_{n/e}^b$ = angular velocity of $\{n\}$ with respect to $\{e\}$ expressed in $\{b\}$

$\boldsymbol{f}_b^n$ = force with line of action through the point o_b expressed in $\{n\}$

$\boldsymbol{m}_b^n$ = moment about the point o_b expressed in $\{n\}$

$\boldsymbol{\Theta}_{nb}$ = Euler angles between $\{n\}$ and $\{b\}$

Por tanto, las cifras de la Tabla 1 pueden ser vectorialmente expresadas de la siguiente manera:

ECEF position	$\boldsymbol{p}_{b/e}^e = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$	Longitude and latitude	$\boldsymbol{\Theta}_{en} = \begin{bmatrix} l \\ \mu \end{bmatrix} \in \mathcal{S}^2$
NED position	$\boldsymbol{p}_{b/n}^n = \begin{bmatrix} N \\ E \\ D \end{bmatrix} \in \mathbb{R}^3$	Attitude (Euler angles)	$\boldsymbol{\Theta}_{nb} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \in \mathcal{S}^3$
Body-fixed linear velocity	$\boldsymbol{v}_{b/n}^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \in \mathbb{R}^3$	Body-fixed angular velocity	$\boldsymbol{\omega}_{b/n}^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \in \mathbb{R}^3$
Body-fixed force	$\boldsymbol{f}_b^b = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \in \mathbb{R}^3$	Body-fixed moment	$\boldsymbol{m}_b^b = \begin{bmatrix} K \\ M \\ N \end{bmatrix} \in \mathbb{R}^3$

En concordancia con lo anterior, el movimiento generalizado de una embarcación marina en 6 DOF, es decir, vectores posición, velocidad y fuerza se describen de la siguiente manera:

$$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{p}_{b/n}^n \text{ (or } \boldsymbol{p}_{b/n}^e) \\ \boldsymbol{\Theta}_{nb} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} \boldsymbol{v}_{b/n}^b \\ \boldsymbol{\omega}_{b/n}^b \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{f}_b^b \\ \boldsymbol{m}_b^b \end{bmatrix}$$

2.2.2 Transformaciones entre BODY y NED

Las matrices ortogonales de orden 3, expresadas como *O*(3), vienen definidas de la siguiente manera:

$$O(3) := \left\{ \boldsymbol{R} | \boldsymbol{R} \in \mathbb{R}^{3\times 3}, \quad \boldsymbol{R}\boldsymbol{R}^{\top} = \boldsymbol{R}^{\top}\boldsymbol{R} = \boldsymbol{I} \right\}$$

Dentro de este conjunto, se encuentra el subconjunto *SO*(3) donde se incluyen todas aquellas matrices ortogonales especiales, en concreto, se encuadran las matrices de rotación $\boldsymbol{R}$ descritas como $\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{a}}$ cuando están referidas entre dos sistemas de referencia diferentes:

$$SO(3) = \left\{ \boldsymbol{R} | \boldsymbol{R} \in \mathbb{R}^{3\times 3}, \quad \boldsymbol{R} \text{ is orthogonal and } \det \boldsymbol{R} = 1 \right\}$$

La matriz de rotación tiene las siguientes **características**:

$$\boldsymbol{R}\boldsymbol{R}^{\top} = \boldsymbol{R}^{\top}\boldsymbol{R} = \boldsymbol{I}, \qquad \det \boldsymbol{R} = 1$$

Lo que implica que se trata de una matriz **ortogonal** y, por tanto, la matriz inversa:

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^{\top}$$

La notación que se empleará para expresar el cambio de un sistema de coordenadas a otro será:

$$\boldsymbol{v}^{\text{to}} = \boldsymbol{R}^{\text{to}}_{\text{from}} \boldsymbol{v}^{\text{from}}$$

Figura 2-13: Notación transformación de matrices [1]

La matriz de rotación de uso frecuente en la guía, la navegación y el control es la matriz de rotación $\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}}$ entre los sistemas de coordenadas {n} y {b}. Posee las siguientes propiedades:

$$\boldsymbol{S} = -\boldsymbol{S}^{\top}$$

$$\boldsymbol{\lambda} \times \boldsymbol{a} := \boldsymbol{S}(\boldsymbol{\lambda})\boldsymbol{a}$$

$$\boldsymbol{S}(\boldsymbol{\lambda}) = -\boldsymbol{S}^{\top}(\boldsymbol{\lambda}) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix}$$

Según [11], el **teorema de rotación de Euler**: "*en el espacio tridimensional, cualquier desplazamiento de un cuerpo rígido tal que un punto del cuerpo rígido permanece fijo, es equivalente a una sola **rotación** alrededor de algún eje que pasa por el punto fijo*". Esta rotación viene descrita como:

$$\boldsymbol{v}_{b/n}^{n} = \boldsymbol{R}_{b}^{n}\boldsymbol{v}_{b/n}^{b}, \quad \boldsymbol{R}_{b}^{n} := \boldsymbol{R}_{\lambda,\beta}$$

En la que $R_{\lambda,\beta}$ es la matriz de rotación correspondiente a β en torno al eje λ:

$$\boldsymbol{R}_{\lambda,\beta} = \boldsymbol{I}_{3\times 3} + \sin(\beta)\boldsymbol{S}(\boldsymbol{\lambda}) + [1 - \cos(\beta)]\,\boldsymbol{S}^{2}(\boldsymbol{\lambda})$$

Ecuación 2-3

Ecuación que se expande de la siguiente manera:

$$
\begin{aligned}
R_{11} &= [1-\cos(\beta)]\,\lambda_1^2 + \cos(\beta)\\
R_{22} &= [1-\cos(\beta)]\,\lambda_2^2 + \cos(\beta)\\
R_{33} &= [1-\cos(\beta)]\,\lambda_3^2 + \cos(\beta)\\
R_{12} &= [1-\cos(\beta)]\,\lambda_1\lambda_2 - \lambda_3\sin(\beta)\\
R_{21} &= [1-\cos(\beta)]\,\lambda_2\lambda_1 + \lambda_3\sin(\beta)\\
R_{23} &= [1-\cos(\beta)]\,\lambda_2\lambda_3 - \lambda_1\sin(\beta)\\
R_{32} &= [1-\cos(\beta)]\,\lambda_3\lambda_2 + \lambda_1\sin(\beta)\\
R_{31} &= [1-\cos(\beta)]\,\lambda_3\lambda_1 - \lambda_2\sin(\beta)\\
R_{13} &= [1-\cos(\beta)]\,\lambda_1\lambda_3 + \lambda_2\sin(\beta)
\end{aligned}
$$

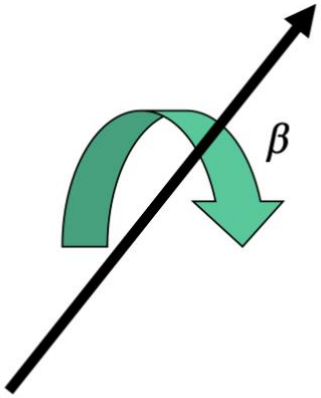

Figura 2-14: Rotación en torno a λ.

2.2.2.1 Transformación del ángulo de Euler

Para obtener las tres matrices de rotación principales en torno a los ejes x, y, z es necesario incluir lo siguiente en la Ecuación 2-3:

$$
\begin{aligned}
\boldsymbol{\lambda} &= [0,0,1]^\top \qquad \beta = \psi\\
\boldsymbol{\lambda} &= [0,1,0]^\top \qquad \beta = \theta\\
\boldsymbol{\lambda} &= [1,0,0]^\top \qquad \beta = \phi
\end{aligned}
$$

Lo que dará lugar a:

$$
\boldsymbol{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix}, \quad \boldsymbol{R}_{y,\theta} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix}, \quad \boldsymbol{R}_{z,\psi} = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

Figura 2-15: Matrices de rotación [1]

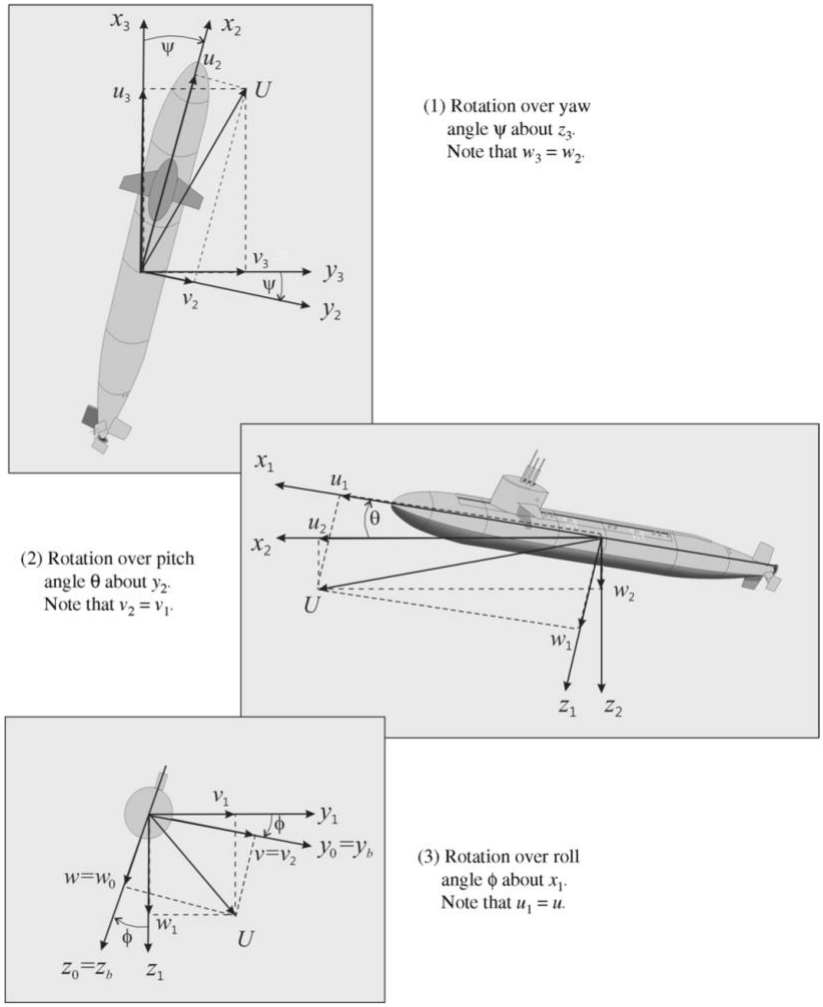

Figura 2-16: Secuencia de rotación (zyx). El submarino se rota de {*n*} a {*b*}. [1]

Transformación lineal de la velocidad:

En las aplicaciones de guiado, navegación y control de una embarcación, es común usar la convención zyx de {n} a {b} especificada en términos de los ángulos de Euler φ, θ y ψ para las rotaciones. Esta matriz se denota por $\boldsymbol{R}_n^b(\boldsymbol{\Theta}_{nb}) = \boldsymbol{R}_b^n(\boldsymbol{\Theta}_{nb})^\top$. La transposición de la matriz implica que se obtiene el mismo resultado transformando un vector de {b} a {n}, es decir, invirtiendo el orden de la transformación. Esta secuencia de rotación es equivalente a:

$$\boldsymbol{R}_b^n := \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi}$$
$$(\boldsymbol{R}_b^n)^{-1} = \boldsymbol{R}_n^b = \boldsymbol{R}_{x,\phi}^\top \boldsymbol{R}_{y,\theta}^\top \boldsymbol{R}_{z,\psi}^\top$$

Por tanto, se puede expresar el vector velocidad lineal del sistema de referencia BODY en {n} como:

$$\dot{\boldsymbol{p}}_{nb}^n = \boldsymbol{R}(\boldsymbol{\Theta}_{nb})\boldsymbol{v}_{nb}^b$$

Figura 2-17: Velocidad lineal BODY [1]

Donde, $\dot{\boldsymbol{p}}_{b/n}^n$ es el vector velocidad es el sistema de referencia NED. Utilizando el método de Euler con tiempo de muestreo *h*, se obtiene la trayectoria de la posición de la embarcación en el sistema de coordenadas NED mediante el método de integración de Euler:

$$\boldsymbol{p}_{nb}^{n}[k+1] = \boldsymbol{p}_{nb}^{n}[k] + h\boldsymbol{R}\left(\boldsymbol{\Theta}_{nb}[k]\right)\boldsymbol{v}_{nb}^{b}[k]$$

$$\begin{aligned}
\dot{x}^n &= u\cos(\psi)\cos(\theta) + v[\cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi)] \\
&\quad + w\left[\sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta)\right] \\
\dot{y}^n &= u\sin(\psi)\cos(\theta) + v[\cos(\psi)\cos(\phi) + \sin(\phi)\sin(\theta)\sin(\psi)] \\
&\quad + w[\sin(\theta)\sin(\psi)\cos(\phi) - \cos(\psi)\sin(\phi)] \\
\dot{z}^n &= -u\sin(\theta) + v\cos(\theta)\sin(\phi) + w\cos(\theta)\cos(\phi)
\end{aligned}$$

Transformación angular de la velocidad:

El vector tasa de Euler y el vector velocidad angular sujeto a un cuerpo (sistema de referencia BODY) están relacionados mediante la matriz de transformación $\boldsymbol{T}_{\Theta}(\boldsymbol{\Theta}_{nb})$ de la siguiente manera:

$$\dot{\boldsymbol{\Theta}}_{nb} = \boldsymbol{T}(\boldsymbol{\Theta}_{nb})\boldsymbol{\omega}_{nb}^{b} \qquad \boldsymbol{\omega}_{nb}^{b} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R}_{x,\phi}^{\top}\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R}_{x,\phi}^{\top}\boldsymbol{R}_{y,\theta}^{\top}\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} := \boldsymbol{T}^{-1}(\boldsymbol{\Theta}_{nb})\dot{\boldsymbol{\Theta}}_{nb}$$

Donde,

$$\boldsymbol{T}^{-1}(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} 1 & 0 & -\mathrm{s}\theta \\ 0 & \mathrm{c}\phi & \mathrm{c}\theta\mathrm{s}\phi \\ 0 & -\mathrm{s}\phi & \mathrm{c}\theta\mathrm{c}\phi \end{bmatrix} \quad \Longrightarrow \quad \boldsymbol{T}(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} 1 & \mathrm{s}\phi\mathrm{t}\theta & \mathrm{c}\phi\mathrm{t}\theta \\ 0 & \mathrm{c}\phi & -\mathrm{s}\phi \\ 0 & \mathrm{s}\phi/\mathrm{c}\theta & \mathrm{c}\phi/\mathrm{c}\theta \end{bmatrix}$$

Por último, como resumen a todo lo descrito en el punto 2.2.2, las ecuaciones cinemáticas para embarcaciones con 6 Grados de Libertad pueden expresarse vectorialmente como:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\Theta}(\boldsymbol{\eta})\boldsymbol{\nu}$$
$$\Updownarrow$$
$$\begin{bmatrix} \dot{\boldsymbol{p}}_{nb}^{n} \\ \dot{\boldsymbol{\Theta}}_{nb} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{\Theta}_{nb}) & \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{T}(\boldsymbol{\Theta}_{nb}) \end{bmatrix}\begin{bmatrix} \boldsymbol{v}_{nb}^{b} \\ \boldsymbol{\omega}_{nb}^{b} \end{bmatrix}$$

3-parameter representation

$\boldsymbol{\Theta}_{nb} = [\phi, \theta, \psi]^{\top}$

with singularity at $\theta = \pm 90^{o}$

Figura 2-18: Ecuaciones de la cinemática [4]

$$\begin{aligned}
\dot{x}^n &= u\cos(\psi)\cos(\theta) + v[\cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi)] \\
&\quad + w\left[\sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta)\right] \\
\dot{y}^n &= u\sin(\psi)\cos(\theta) + v[\cos(\psi)\cos(\phi) + \sin(\phi)\sin(\theta)\sin(\psi)] \\
&\quad + w[\sin(\theta)\sin(\psi)\cos(\phi) - \cos(\psi)\sin(\phi)] \\
\dot{z}^n &= -u\sin(\theta) + v\cos(\theta)\sin(\phi) + w\cos(\theta)\cos(\phi) \\
\dot{\phi} &= p + q\sin(\phi)\tan(\theta) + r\cos(\phi)\tan(\theta) \\
\dot{\theta} &= q\cos(\phi) - r\sin(\phi) \\
\dot{\psi} &= q\frac{\sin(\phi)}{\cos(\theta)} + r\frac{\cos(\phi)}{\cos(\theta)}, \quad \theta \neq \pm 90^{\circ}
\end{aligned}$$

2.2.3 Transformaciones entre ECEF y NED

Tal y como se definió en 2.2.1, un punto cualquiera encima o sobre la superficie terrestre está definido por **longitud** l [º], **latitud** μ [º] y **altura elipsoidal** h [m]. Así mismo, la diferencia entre el sistema de coordenadas NED y ECEF se puede apreciar visualmente en la Figura 2-19.

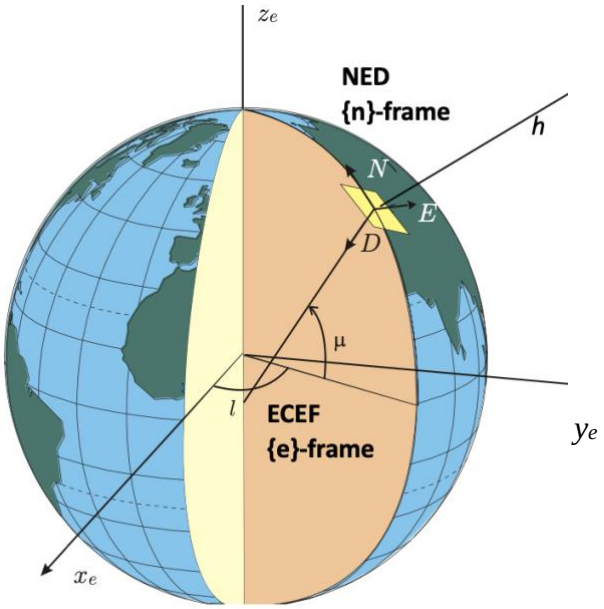

Figura 2-19: Sistemas NED y ECEF [4]

Donde, según el sistema de referencia NED:

- **N**: el eje norte apunta al norte.
- **E**: el eje este apunta al este.
- **D**: el eje hacia abajo apunta hacia abajo en la dirección normal a la superficie de la Tierra.

La transformación entre los vectores velocidad del sistema de coordenadas ECEF y NED es:

$$\dot{\boldsymbol{p}}^e_{eb} = \boldsymbol{R}^e_n \dot{\boldsymbol{p}}^n_{eb} = \boldsymbol{R}^e_n \boldsymbol{R}^n_b \boldsymbol{v}^b_{eb} \qquad\qquad \boldsymbol{\Theta}_{en} = [l, \mu]^\top \in \mathrm{S}^2$$

En la que se llevan a cabo dos rotaciones principales: rotación (l) sobre el eje ***z*** y rotación $(-\mu - \pi/2)$ sobre el eje ***y***, representadas de la siguiente manera:

$$\begin{aligned}\boldsymbol{R}^e_n &:= \boldsymbol{R}_{z,l}\boldsymbol{R}_{y,-\mu-\frac{\pi}{2}} \\ &= \begin{bmatrix} \cos(l) & -\sin(l) & 0 \\ \sin(l) & \cos(l) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\left(-\mu-\frac{\pi}{2}\right) & 0 & \sin\left(-\mu-\frac{\pi}{2}\right) \\ 0 & 1 & 0 \\ -\sin\left(-\mu-\frac{\pi}{2}\right) & 0 & \cos\left(-\mu-\frac{\pi}{2}\right) \end{bmatrix}\end{aligned}$$

$$\boldsymbol{R}^e_n = \boldsymbol{R}(\boldsymbol{\Theta}_{en}) = \begin{bmatrix} -\cos(l)\sin(\mu) & -\sin(l) & -\cos(l)\cos(\mu) \\ -\sin(l)\sin(\mu) & \cos(l) & -\sin(l)\cos(\mu) \\ \cos(\mu) & 0 & -\sin(\mu) \end{bmatrix}$$

Figura 2-20: Rotaciones principales [12]

Sin embargo, los sistemas de navegación no son útiles para el operador cuando se utiliza el sistema de coordenadas ECEF. Por tanto, la representación de la posición terrestre $p_{eb}^{e} = [x^e, y^e, z^e]^T$ deberá especificarse según los parámetros elipsoidales longitud *l*, latitud μ y altura *h*:

$$\boldsymbol{p}_{eb}^{e} = [x^e, y^e, z^e]^\top$$

Transformation

l, μ and *h*

Figura 2-21: Transformación ECEF-NED [4]

Para ello, teniendo en cuanta la disposición de los vectores y sus ángulos en la esfera terrestre (Figura 2-22).

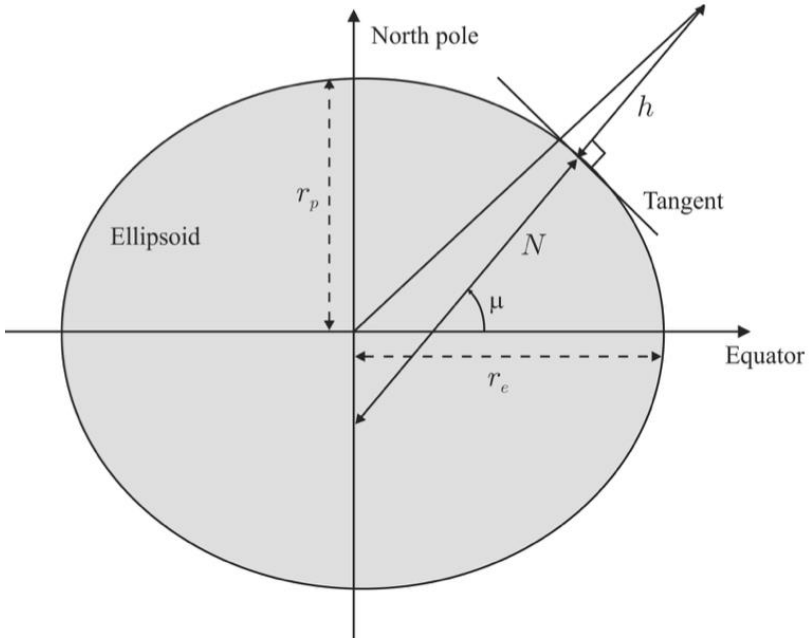

Figura 2-22: Representación terrestre [4]

Podemos obtener mediante operaciones trigonométricas las relaciones de latitud μ y altura *h* con el resto de los parámetros:

$$\tan(\mu) = \frac{z^e}{p}\left(1 - e^2\frac{N}{N+h}\right)^{-1}$$

$$h = \frac{p}{\cos(\mu)} - N$$

Por último, la transformación $\boldsymbol{\Theta}_{en} = [l, \mu]^\top$, según Figura 2-21, para alturas dadas *h* se puede expresar como:

$$\begin{bmatrix} x^e \\ y^e \\ z^e \end{bmatrix} = \begin{bmatrix} (N+h)\cos(\mu)\cos(l) \\ (N+h)\cos(\mu)\sin(l) \\ \left(\frac{r_p^2}{r_e^2}N + h\right)\sin(\mu) \end{bmatrix}$$

2.3 Filtrado y estimación

Ante la búsqueda del menor número de fallos durante la realización del proyecto, nos adentramos en ir un paso más allá y así adelantarnos a todos aquellos sistemas donde puede actuar el azar. El hecho de que un proceso sea determinista solo nos permitiría modelizarlo a partir de su comportamiento siguiendo leyes físicas conocidas u observaciones recogidas, que podremos relacionar entre sí para obtener una salida del sistema.

Todas aquellas características dominantes que recogen los modelos matemáticos, así como el ruido existente en sistemas dinámicos dejan fuera muchos efectos que no podemos modelar de manera determinista. En nuestro caso, el movimiento del agua dentro de la cubeta donde situaremos nuestra estructura de estudio, así como la temperatura, presión, viento y otros factores de gran relevancia y que somos incapaces de controlar formarían parte de la aleatoriedad que un sistema determinista no recoge. Cabe destacar, todas aquellas distorsiones que pueden contener los instrumentos de medida (tanto hardware como software) y que, en definitiva, provoca que no podamos asumir un conocimiento perfecto de todo lo necesario para describir un sistema completo.

Es por ello que nos surgen algunas preguntas: ¿Cómo podríamos modelar todas aquellas incertidumbres descritas anteriormente? ¿Podríamos elaborar un modelo óptimo a pesar de todas aquellas observaciones ruidosas o insuficientes? Para responder a todo esto, recurrimos a los **modelos estocásticos**, en concreto, técnicas de filtrado y estimaciones. A continuación, se exponen algunos de los ejemplos clásicos de este tipo de sistemas empleados en aplicaciones de filtrado y estimación.

2.3.1 Filtros G-H

El filtro G-H no es solo un filtro, es una clasificación para toda una familia de filtros que se diferencian por cómo se eligen los coeficientes G y H. En su forma mas sencilla, se seleccionan estos parámetros con valores constantes, que se obtienen a partir del análisis estadístico de las señales a filtrar. Sin embargo, existen diversas formas de seleccionar estos valores, tal y como ocurre en el caso los filtros Bayesianos, dentro de los que podemos encontrar el denominado filtro de Kalman que veremos a continuación.

En líneas generales, estos filtros trabajan de forma recursiva, de forma que, a partir de un modelo de movimiento estimado, se realiza una predicción del futuro valor a observar. Una vez que llega el nuevo valor, se evalúa la diferencia entre la predicción y lo observado (se conoce como *residual* a esta diferencia), y se utiliza este valor para actualizar la estimación del modelo de movimiento, así como de la nueva predicción a realizar. A continuación, vemos un ejemplo de código en Python para este tipo de filtro básico:

```
def filterGH(time, data, x0, dx, dt, g, h):
    x_est = x0
    results = []
    for z in data:
        # prediction step
        x_pred = x_est + (dx*dt)# update step
        residual = z - x_pred
        dx = dx + h * (residual) / dt
        x_est = x_pred + g * residual
        results.append(x_est)
    return results
```

En líneas generales, la constante G responde a como el filtro se adapta a la varianza de las medidas, es decir a su diversidad. Por ello en sistemas que presentan mucho ruido conviene usar valores bajos de G. Sin embargo, disminuir en exceso el valor de G implica que el filtro tarda más en adaptarse a los cambios en el medio, es decir, es menos sensible a las altas frecuencias de la señal de entrada.

Por otro lado, la constante H se asocia al nivel de confianza con respecto al modelo de movimiento y a las nuevas observaciones obtenidas, es decir la hipótesis acerca de cómo se producen los cambios en el sistema de estudio con el valor oculto que debemos estimar.

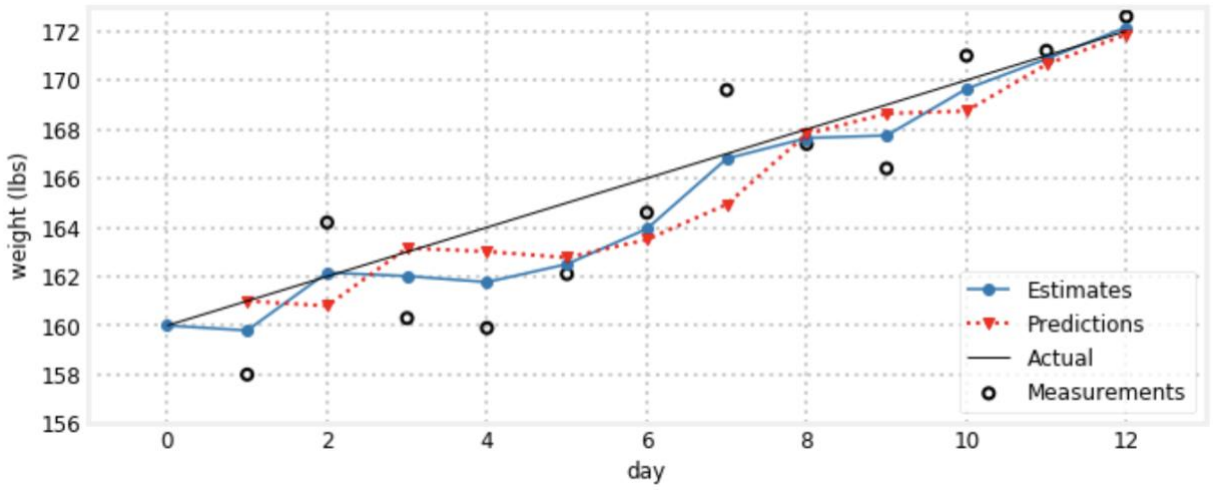

Figura 2-23: Ejemplo de estimación con un filtro G-H [13]

Una elección en particular puede funcionar bien en una situación, pero muy mal en otra. Incluso cuando se comprende el efecto de G-H, puede resultar difícil elegir los valores adecuados. De hecho, es extremadamente improbable que elija valores para G-H que sean óptimos para cualquier problema dado. Por ello, el filtro hay que diseñarlo de manera que satisfaga nuestras necesidades a la hora de realizar los experimentos expuestos en el punto 4.

2.3.2 Filtro de Kalman

El **filtro de Kalman**, que lleva el nombre de Rudolf E. Kalman (19 de mayo de 1930 – 2 de julio de 2016) es la primera solución a todas aquellas dudas que nos surgieron. Se trata de un algoritmo cuya función es estimar una variable a partir de datos medidos. Esto se consigue siguiendo dos pasos:

- Predicción del estado actual del sistema en base al estado anterior y a las ecuaciones que rigen su evolución.
- Incorporación de todas aquellas observaciones recogidas una vez corregidas.

Por tanto, el resultado de todo ello es la obtención de un estimador óptimo que sea próximo respecto al modelo sobre el que realizar de todas aquellas variables de interés basándonos en el conocimiento del sistema y de las observaciones que apreciándose han obtenido , con su correspondiente ruido y errores asociados. Al ser un **proceso recursivo**, proporciona la ventaja de que es posible incorporar nuevas observaciones sin tener que reformular el algoritmo.

Por todo ello, el filtro de Kalman es el idóneo para usar en nuestra estructura embarcada donde los sensores (3.2.1.1), al ser de bajo coste, recogen la información de las velocidades, aceleraciones y del

entorno con bastante ruido. Debemos tener en cuenta que este algoritmo requiere que las características del ruido modelado deben ser de tipo Gaussiano.

El algoritmo de Kalman queda definido por el siguiente diagrama:

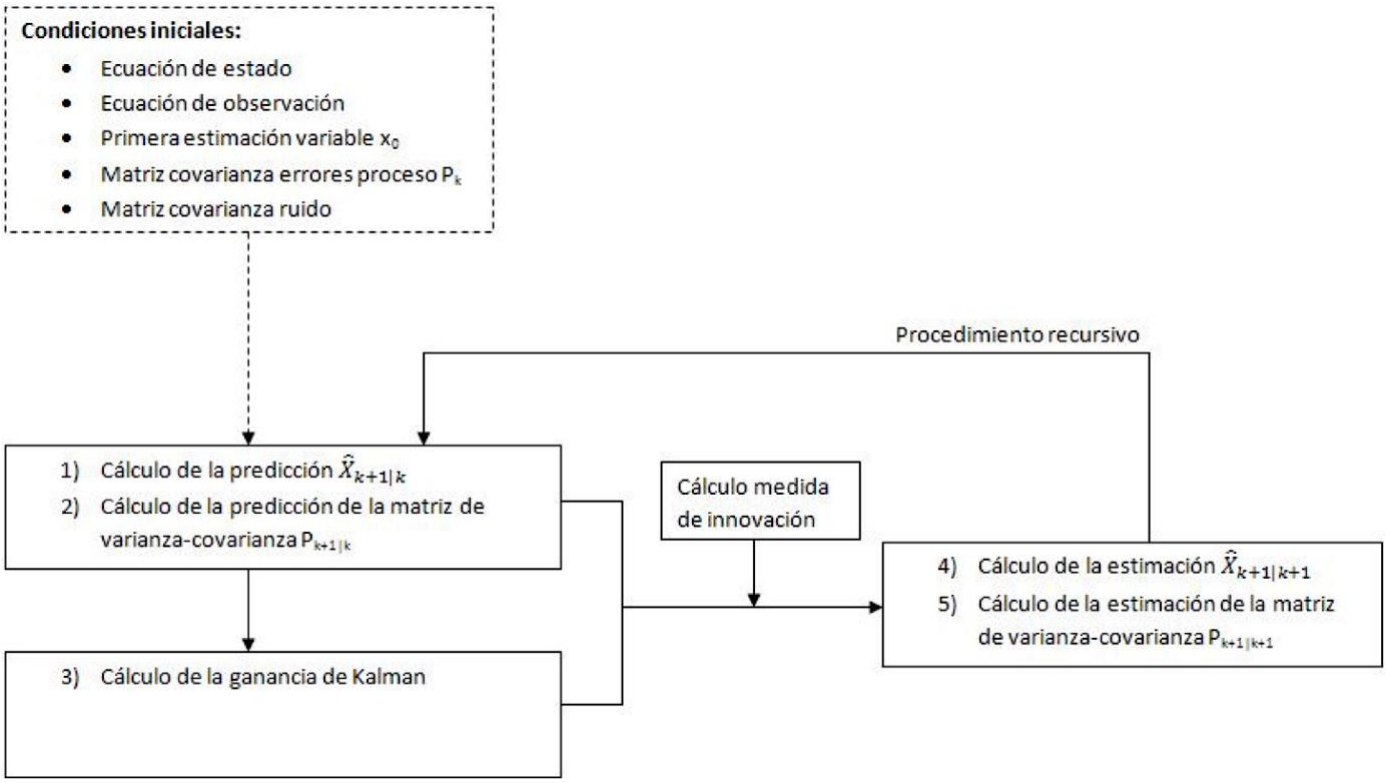

Figura 2-24: Principio de funcionamiento algoritmo Kalman [14]

Hoy en día, la mayoría de los sistemas modernos que disponemos en los barcos están equipados por sensores que nos proporcionan una estimación de variables ocultas (para nosotros son desconocidas) en base a una serie de mediciones. Estas variables pueden ser, por ejemplo, la ubicación y velocidad que nos proporciona un sistema GPS a nuestra embarcación. Por ello, la gran preocupación será proporcionar una estimación lo más exacta posible y precisa de las variables ocultas en presencia de incertidumbre.

El filtro de Kalman será el encargado de producir estimaciones de variables basadas en mediciones inexactas e inciertas y de proporcionarnos una forma de predecir el comportamiento dinámico del sistema en un futuro horizonte temporal.

Como ejemplo de uso, podríamos imaginar un radar de seguimiento que envía un haz en dirección del blanco en camino. Con esto, el radar sería capaz de estimar la posición y velocidad actual del objetivo además de la posición futura en el momento que envíe el siguiente haz.

En general, es habitual encontrar diseños del filtro de Kalman en donde las ecuaciones del movimiento, para un ejemplo en 3 dimensiones teniendo en cuenta derivadas de segundo orden, son de la forma:

$$\begin{cases} x = x_0 + v_{x0}\Delta t + \frac{1}{2}a_x\Delta t^2 \\ y = y_0 + v_{y0}\Delta t + \frac{1}{2}a_y\Delta t^2 \\ z = z_0 + v_{z0}\Delta t + \frac{1}{2}a_z\Delta t^2 \end{cases}$$

Figura 2-25: Ecuaciones Newton 3D [15]

Donde las variables de estado son las posiciones (x,y,z) así como las velocidades (v_x,v_y, v_z), es decir, las variables que definen el estado del sistema y que serán empleadas para establecer el modelo dinámico del blanco del que se está haciendo seguimiento. Sin embargo, con esto no podríamos definir el estado futuro del objetivo fácilmente, ya que el radar contiene errores de calibración, ancho de haz y numerosos parámetros que introducen ruido. Es por ello que, ante la incertidumbre del proceso y de la medición, necesitamos algún otro parámetro que la tenga en cuenta, como es el caso de los filtros de Kalman.

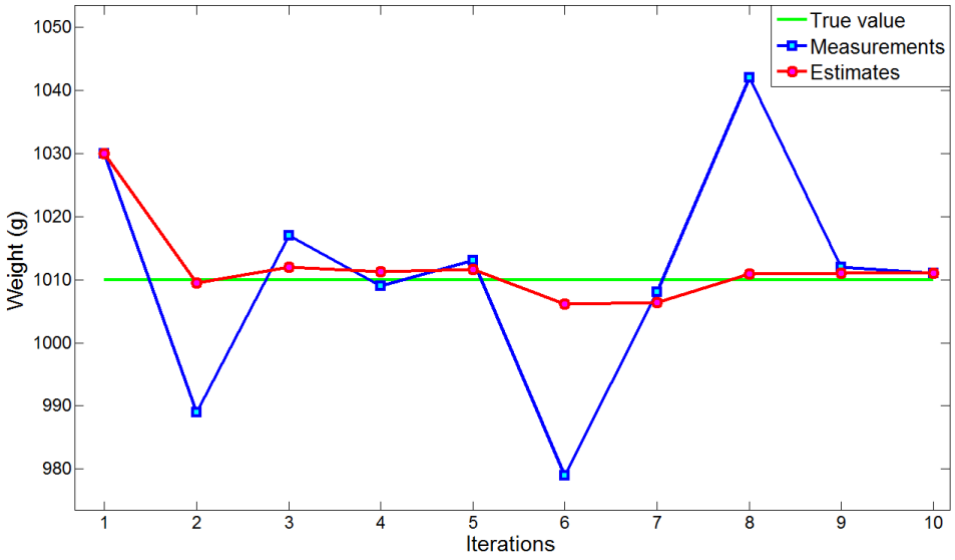

Figura 2-26: Ejemplo de aplicación filtro de Kalman

2.3.3 Filtro complementario

El filtro complementario funciona combinando las características óptimas de baja frecuencia de ciertos tipos de sensores con las de alta frecuencia de otros. Un ejemplo de esto lo podemos encontrar en las IMUs, en donde el proceso de estimación de los ángulos de roll y pitch puede aprovecharse de las características de respuesta a baja frecuencia del acelerómetro con las de alta frecuencia asociadas al giróscopo. Básicamente, con los valores obtenidos del acelerómetro con un filtro de media móvil simple e integrando el ruido de los datos del giróscopo, podemos combinar ambas estimaciones y obtener una **única estimación mejorada**. El método puede describirse en los siguientes pasos:

- En primer lugar, escoger una constante α donde $0<\alpha<1$. A mayor α, las medidas del acelerómetro son más "verídicas". Cuando α tiende a 0, se basan las estimaciones mayormente en las medidas del giróscopo.
- Se inicia un estimador de estado.
- Para cada instante de tiempo:
    - Se recuperan las lecturas del acelerómetro y del giróscopo de la IMU sin procesar.
    - Se calcula una estimación del ángulo a partir de los datos del acelerómetro.
    - Se combina esta estimación con la integral de los datos del giróscopo transformados. Lo que da lugar a la ecuación del filtrado:

$$\hat{\phi}_{t+1} = (1-\alpha)\cdot(\hat{\phi}_t + \dot{\phi}_G \cdot \Delta t) + \alpha \cdot \hat{\phi}_{Acc}$$

La diferencia principal entre las medidas es que el acelerómetro utiliza un filtro de paso bajo mientras que el giróscopo uno de paso alto. Ambas señales se combinan, dependiendo del valor de α, para dar lugar a la estimación del estado final.

El filtro complementario tiene muchas ventajas. Por ejemplo, solo hay un parámetro ajustable (α), es fácil de implementar en unas pocas líneas de código, no requiere mucha potencia de procesamiento y, por último, ofrece un rendimiento satisfactorio.

Como ejemplo, comparando un α de 0.4 con uno de 0.1, se puede comprobar con claridad que a menor α, se consiga reducir en gran medida el ruido:

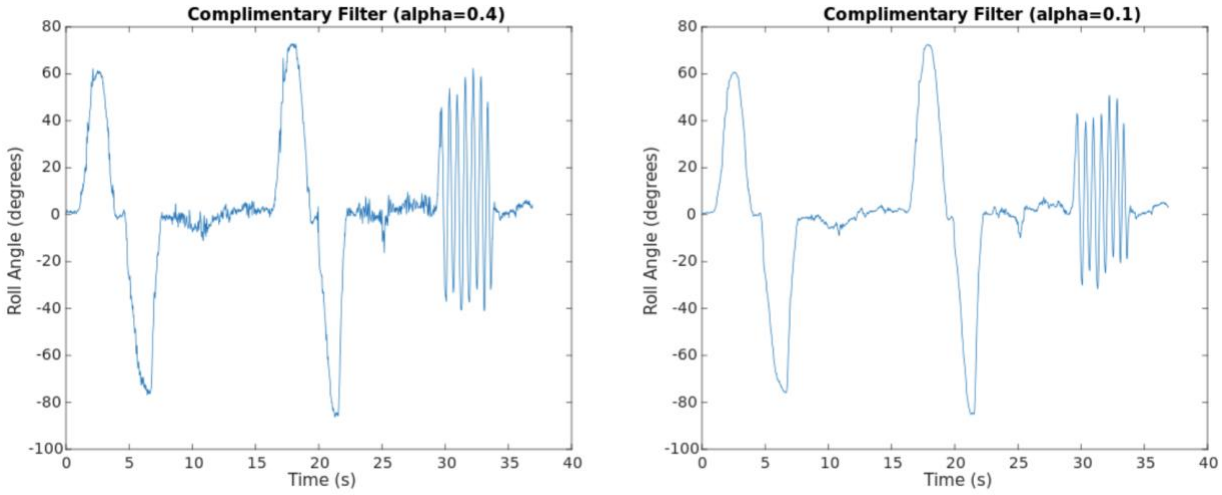

Figura 2-27: α=0.4 vs α=0.1 [16]

Las diferencias fundamentales entre los filtros G-H y los filtros de Kalman, para el ejemplo indicado de fusión de datos de aceleraciones y velocidades angulares en las IMUS, serían las siguientes:

- Los Filtros de Kalman se basan un modelo de sistemas dinámicos que no está restringido únicamente a dos estados.

- Utilizan múltiples variables observadas para corregir las estimaciones de las variables de estado, y estas no tienen que ser medidas directas de los estados individuales del sistema.

- Un filtro de Kalman emplea matrices de covarianza de ruido para estados y observaciones. Con estos, se actualiza automáticamente una estimación dependiente del tiempo de la covarianza de estado, y a partir de esto se calculan los términos de la matriz de ganancia de Kalman. Las ganancias del filtro alfa beta se seleccionan manualmente y son estáticas.

3 DESARROLLO DEL TFG

3.1 Introducción

En esta sección de la memoria, se exponen los recursos que se han utilizado para crear un conjunto hardware-software con el que poder realizar todas las calibraciones, tests, filtrados y estimaciones expuestas en el capítulo 4. Así mismo, se incluye una evolución de los prototipos que se han desarrollado para este proyecto, así como la maqueta de barco, incluyendo mejoras aplicadas con la finalidad de simular al máximo posible las dinámicas de un barco real. El objetivo práctico final es realizar un estudio de las propiedades dinámicas en 3 dimensiones del movimiento de dos objetos diferentes (un bloque y un barco de modelismo) dentro de una cubeta de agua.

3.2 Instrumentación y medios materiales

3.2.1 Hardware

En primer lugar, se expone todo el material físico adquirido para realizar los diferentes experimentos y que son la base para crear los prototipos del proyecto. Parte de ese material lo constituyen, por ejemplo, las unidades encargadas de obtener los datos de las IMUs y enviarnos al servidor central que los almacena, lo cual nos permitirán obtener una serie de conclusiones en base al análisis posterior de estos datos.

3.2.1.1 IMU

A la hora de realizar un estudio acerca del movimiento de un sistema en 3 dimensiones, del que queremos obtener mediciones para su posterior análisis, se hace uso de las *"IMUs"* (siglas en inglés: Inertial Measurement Unit) que son dispositivos electrónicos que realizan las funciones de unidades de medición inercial, es decir, de obtener las observaciones que tendría un observador situado sobre el objeto en movimiento.

Los giróscopos miden las velocidades rotacionales y los acelerómetros se encargan de medir las aceleraciones lineales (usualmente en unidades relativas a *fuerzas G*) en cada uno de los ejes del dispositivo.

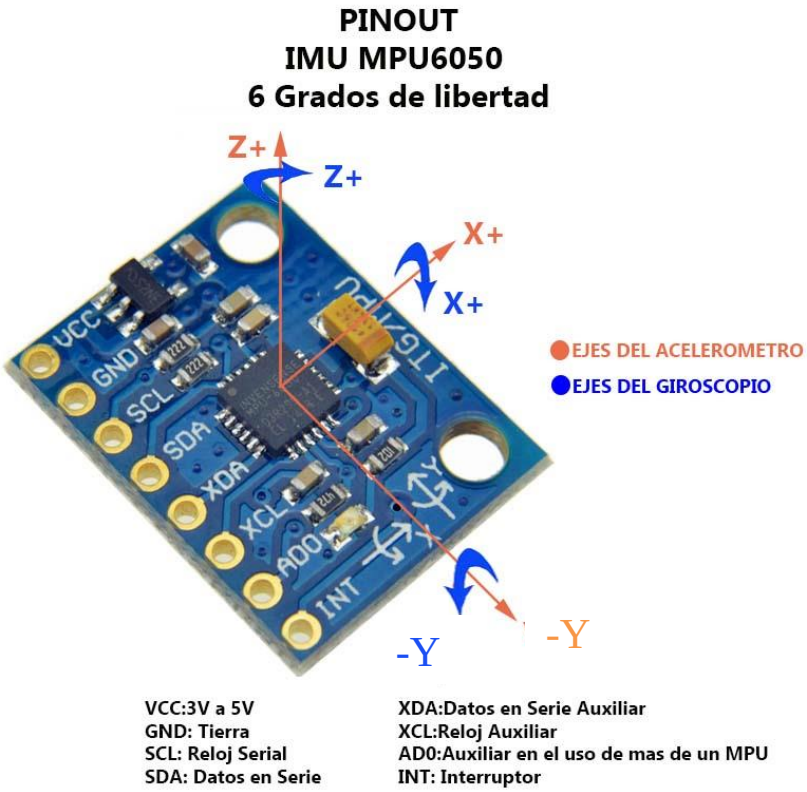

Figura 3-1: IMU MPU6050 [17]

Se trata, por tanto, del componente principal de los sistemas de navegación **inercial**, cuyo objetivo final (fuera del alcance del análisis de este trabajo) es la creación de estimaciones de las propiedades dinámicas de un objeto en movimiento que permitan, en un futuro, especular con la posibilidad de navegar únicamente empleando sensores inerciales sin depender de sistemas tipo GPS. Además, cabe destacar que *"en la actualidad los sistemas de captura de movimiento basados en IMUs han cobrado gran relevancia debido a su **bajo costo** y facilidad de uso"* [18].

El funcionamiento depende esencialmente de su DMP ("*Digital Movement Processor*") que se encarga de procesar los algoritmos de movimientos complejos de 9 ejes por lo que "*pueden incluso acceder a magnetómetros externos u otros sensores a través de un bus I2C auxiliar maestro, permitiendo reunir un conjunto completo de dispositivos sensores de datos, sin la intervención del procesador del sistema"* [19]

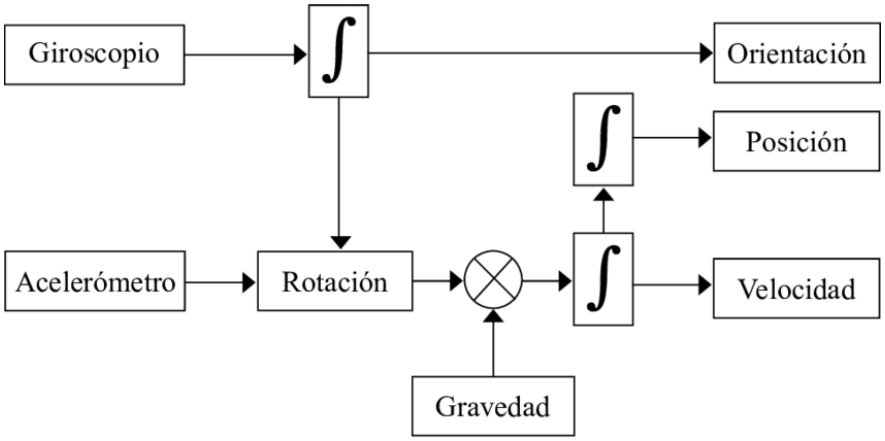

Figura 3-2: Esquema interno IMU [20]

3.2.1.2 NodeMCU

Una vez descritos los sensores que nos proporcionarán los datos necesarios para poder estimar el movimiento de nuestra estructura, es necesario un dispositivo que nos permita leer y enviar esos datos a nuestro programa para poder interpretarlos.

NodeMCU es una plataforma IoT de código abierto que incluye el firmware donde se ejecuta el módulo **ESP8266**, que es un chip de **bajo coste** que colocaremos próximo a las IMUs y que permitirá que se conecten a una **red Wi-Fi** de 2,4 GHz mediante IEEE 802.11 bgn. El módulo tiene una pila TCP / IP completa y proporciona la capacidad de procesamiento, lectura y control de datos de GPIO. Además, NodeMCU es capaz de soportar el protocolo IoT MQTT.

ESP8266 tiene muchas aplicaciones cuando se trata de IoT. Estas son solo algunas de las funciones para las que se usa el chip:

Redes: la antena Wi-Fi del módulo permite que los dispositivos integrados se conecten a enrutadores y transmitan datos.

Procesamiento de datos: incluye el procesamiento de entradas básicas de sensores analógicos y digitales para cálculos mucho más complejos con un RTOS o SDK sin sistema operativo.

Conectividad P2P: cree una comunicación directa entre los ESP y otros dispositivos utilizando la conectividad IoT P2P.

Servidor web: acceda a páginas escritas en HTML o lenguajes de desarrollo.

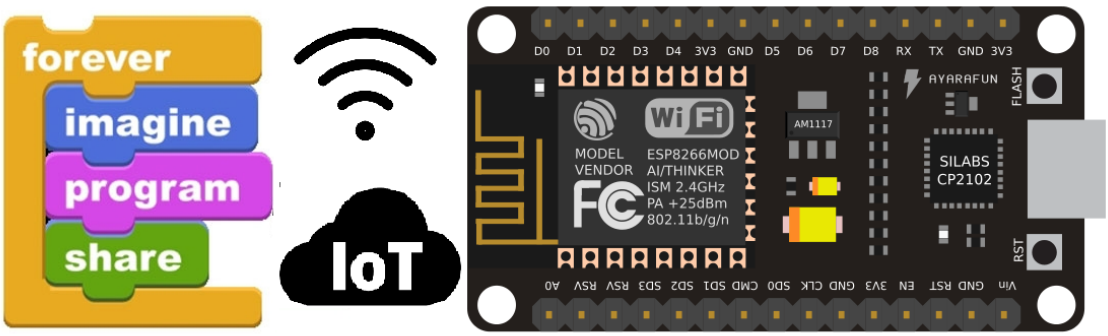

Figura 3-3: NodeMCU [21]

3.2.2 Software

A continuación, tras definir todos aquellos dispositivos que conformarán el hardware de nuestro proyecto, se describen todos aquellos programas que se encargarán de realizar las conexiones entre todos los sistemas, extraer la información que necesitamos y permitir que visualicemos lo que ocurre en el entorno.

3.2.2.1 Arduino

Para comprobar que el módulo ESP8266 nos envíe vía Wi-Fi los datos de los sensores, hicimos uso del software de Arduino donde, además, podemos visualizar si hemos establecido conexión con el servidor de mensajes MQTT y establecer protocolos para el envío de datos.

Tal y como se ha descrito anteriormente, Arduino es una plataforma electrónica de código abierto con software y hardware sencillos de utilizar. En la parte de hardware, las placas pueden leer numerosas entradas y convertirlo en salidas.

Respecto al software, que es la parte de la que hemos hecho uso, se ha utilizado el lenguaje de programación Arduino (basado en Wiring) y Software Arduino (IDE), basado en Processing.

Figura 3-4: Logo software Arduino [22]

La razón por la que escoger Arduino para realizar nuestras primeras mediciones fueron las numerosas ventajas que nos proporciona:

- **Económico**: las placas Arduino son relativamente económicas en comparación con otras plataformas de microcontroladores.

- **Multiplataforma**: el software Arduino (IDE) se ejecuta en los sistemas operativos Windows, Macintosh OSX y Linux. La mayoría de los sistemas de microcontroladores están limitados a Windows es por ello que para la realización de los experimentos se ha utilizado este sistema operativo.

- **Entorno de programación simple y claro**: el software Arduino (IDE) es fácil de usar para principiantes, pero lo suficientemente flexible para alguien que tenga más conocimientos.

- **Software de código abierto y extensible:** el software Arduino se publica como herramientas de código abierto, disponibles para su extensión por programadores experimentados.

- **Hardware extensible y de código abierto**: los planos de las placas Arduino se publican bajo una licencia Creative Commons, por lo que podemos ser capaces de crear nuestra propia versión del módulo, ampliarlo y mejorarlo.

3.2.2.2 Anaconda

La integración de todos los sensores descritos anteriormente junto con el software de Arduino queda incompleta debido a la necesidad de crear un programa donde podamos visualizar un "*framework*" donde representar todo lo que está ocurriendo en nuestro entorno, tanto el envío de mensajes de las IMUs y de datos ambientales como poder visualizarlo en 2D y 3D. El propósito de todo ello es obtener una serie de gráficas y resultados con el que podamos estimar finalmente la posición de nuestra estructura acuática.

Para ello, se hace uso de Anaconda, que integra distintos programas que utilizaremos para lo descrito anteriormente. Es una distribución libre y de código abierto, es decir, el software está licenciado de manera que todos los usuarios que hagan uso de este pueden estudiar, modificar y optimizar el diseño disponiendo del código fuente, de los lenguajes R y Python. Es principalmente utilizada en ciencia de datos y *"machine learning"* (Aprendizaje automático: permitir que las computadoras "aprendan" a través de la experiencia de uso).

Su finalidad es simplificar el despliegue y la administración de los diferentes paquetes de software instalados en nuestro equipo. Para la instalación, ejecución y actualización de los diferentes paquetes de los que vamos a hacer uso, se hace uso del **sistema de gestión de paquetes conda.**

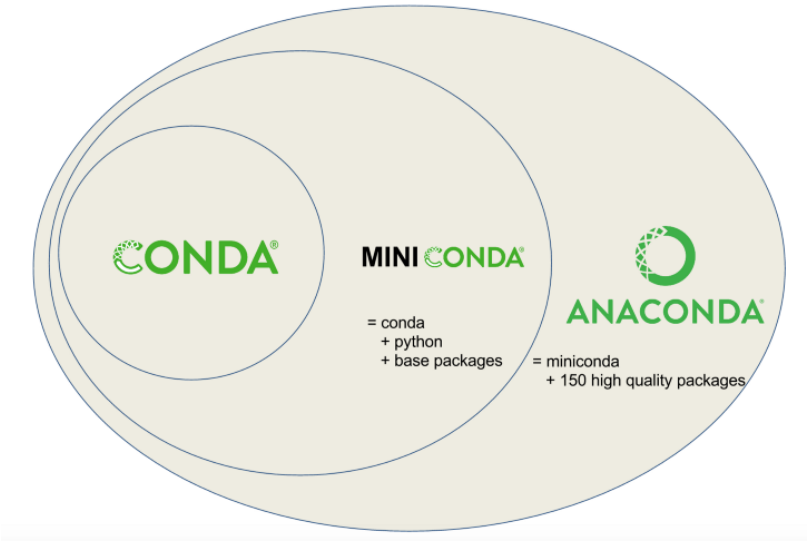

Figura 3-5: Entorno Anaconda

Como se puede comprobar en la imagen anterior, Anaconda integra tanto Conda como MiniConda. En nuestro caso, Conda será la parte encargada de instalar y ejecutar los programas necesarios para realizar nuestros experimentos. Más específicamente, según [23]:

"Conda es un sistema de gestión de paquetes de código abierto y un sistema de gestión del entorno que se ejecuta en Windows, macOS y Linux. Conda instala, ejecuta y actualiza rápidamente paquetes y sus dependencias además de crear, guardar, cargar y cambiar fácilmente entre entornos en su computadora local. Fue creado para programas Python, pero puede empaquetar y distribuir software para cualquier idioma. Conda como administrador de paquetes nos facilita encontrar e instalar paquetes.

3.2.2.3 Python

El programa que se utilizará para fusionar todos los elementos de los que disponemos para el flujo de información y datos, estudiar el entorno y poder interpretar todo en pantalla será Python, que se trata de un lenguaje de programación interpretado, cuya filosofía hace hincapié en la **legibilidad** de su código. Además, es **multiparadigma** ya que soporta programación orientada a objetos, programación imperativa y programación funcional. Permite varios estilos según el programador que este haciendo uso de este (Java, .net, entre otros) y, además, soporta diferentes paradigmas haciendo uso de extensiones.

Su éxito en el mundo de la programación se debe, principalmente, a que cuenta con una licencia de código abierto. Permite, tal y como se describe en [24] *"automatización de procesos y ejecución de tareas en tanto en entorno cliente como servidor y es ideal para trabajar con* ***grandes volúmenes de datos*** *ya que, el ser multiplataforma, favorece su extracción y procesamiento".*

Debido a su lenguaje elegante, sencillo y fácilmente legible, permite elaborar algoritmos complejos en tan solo unas pocas líneas sin necesitar altos conocimientos de programación, lo que consigue ahorrar gran cantidad de tiempo.

Python dispone también de frameworks gráficos de gran calibre, recurso que podremos explotar para obtener una representación en pantalla de todos los datos que envían los sensores y que iremos recibiendo en nuestro sistema.

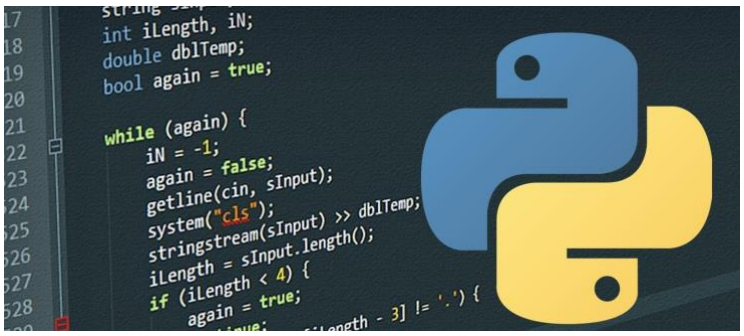

Figura 3-6: Legibilidad Python [25]

3.2.2.3.1 VTK

Ante la llegada de datos y la visualización de estos en el "*framework*", apareció la necesidad de poder representar en **3D** un prototipo de nuestros sensores ubicándolo en el centro de unos ejes coordenados fijos y así poder diferenciar la orientación de los mismos, que irán en concordancia con el "*yaw, pitch* y *roll"*. Para ello se hace uso de VTK (Visualization Toolkit) librería integrada en Python y que, según lo define su página web, [26]:

"Es un sistema de software de ***código abierto*** *para gráficos por computadora en 3D, modelado, procesamiento de imágenes, reproducción de volumen, visualización científica y visualización de información. VTK también incluye soporte auxiliar para widgets de interacción 3D, anotaciones bidimensionales y tridimensionales y computación paralela.*

Figura 3-7: The Visualization Toolkit [26]

VTK emplea el proceso de software de calidad de Kitware (CMake, Ctest, Cdash y Cpack) para construir, probar y empaquetar el sistema, lo que convierte a VTK en una aplicación ***multiplataforma*** *que depende del desarrollo impulsado por pruebas y la programación extrema, y permite que la aplicación produzca códigos robustos y de calidad. VTK se utiliza en todo el mundo en aplicaciones comerciales, así como en investigación y desarrollo*".

3.2.2.3.2 PyQtGraph

Tras la obtención de resultados en nuestro experimento y poder representar el movimiento de la estructura y así estimar posiciones futuras, se hace uso de la librería PyQtGraph, construida en NumPy, que está diseñada para aplicaciones matemáticas, científicas o de ingeniería. la biblioteca es muy rápida debido a la gran capacidad de NumPy para procesar números y al marco GraphicsView de Qt para una visualización rápida. PyQtGraph se distribuye bajo la licencia de código abierto del MIT.

La finalidad de esta librería será la de graficar todos aquellos datos que han sido recopilados en las bases de datos tras realizar los experimentos y poder llegar a una serie de conclusiones de gran utilidad en el futuro.

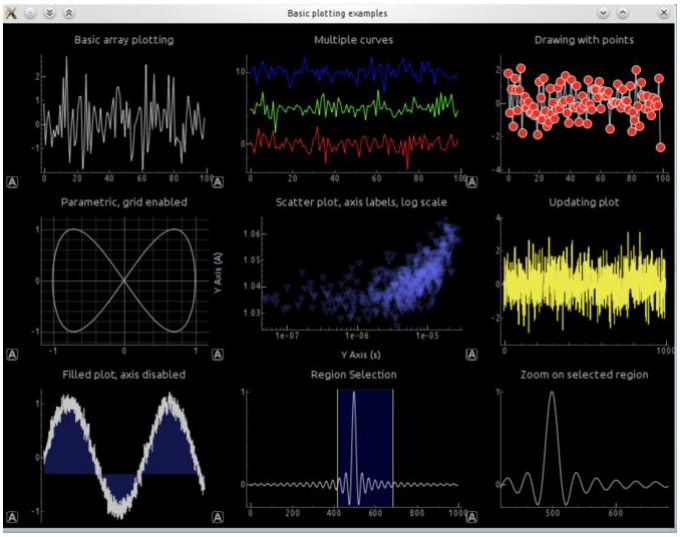

Figura 3-8: Tipos de gráficos PyQt [27]

3.2.2.4 Mosquitto

A la hora de recibir los datos por parte de los sensores, necesitamos que un intermediario nos "traduzca" esos mensajes y, así mismo, indique a los sensores cuando empezar o terminar de enviarlos o que función hacer. Para ello hacemos uso de un servidor de mensajes que se define en [28] como:

"MQTT son las siglas MQ Telemetry Transport. Es un ***protocolo de comunicación M2M (machine-to-machine) de tipo message queue****. Está* ***basado en la pila TCP/IP*** *como base para la comunicación. En el caso de MQTT cada conexión se mantiene abierta y se "reutiliza" en cada comunicación."*

El servidor o *broker*, recopila los datos que los *publishers*, los clientes o nodos publican en los tópicos (*"topics").* Es decir, es el responsable de recibir todos los mensajes y de aceptar suscripciones de otros dispositivos (aquellos que están "pendientes" de ciertos mensajes) para mostrarles los mensajes que son de su interés.

Un mensaje que envía cierto sistema puede tener lo que es llamado QoS (Quality of Service, o calidad de servicio) en el que podemos indicarle al broker si queremos que dicho mensaje sea simplemente enviado, enviado con confirmación de entrega o garantizando su entrega al completo.

Así, por ejemplo, un sensor se suscribe al tópico "TFG" y enviará datos. Sin embargo, puede también subscribirse a "PROTOCOL" y que, al enviarles un 0, pasen a estado de espera o de parada; o envíen datos cuando reciban un 1. Debido a este tipo de estructura que separa datos del sensor de los del protocolo de control de su estado, los ESP8266 (3.2.1.2) trabajan de manera bidireccional, ya que envían mensajes de datos en nuestro caso las IMU (3.2.1.1), y se suscriben para recibir instrucciones (como el inicio de la recogida y envío de los datos de las IMUs).

Para explicar el funcionamiento de este servidor es necesario conocer el modo de trabajo. Según se explica en [28]:

"Los clientes inician una conexión TCP/IP con el broker, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto, MQTT emplea el puerto 1883 y el 8883 cuando funciona sobre TLS.

Para ello el cliente envía un mensaje CONNECT que contiene información necesaria (nombre de usuario, contraseña, client-id…). El broker responde con un mensaje CONNACK, que contiene el resultado de la conexión (aceptada, rechazada, etc).

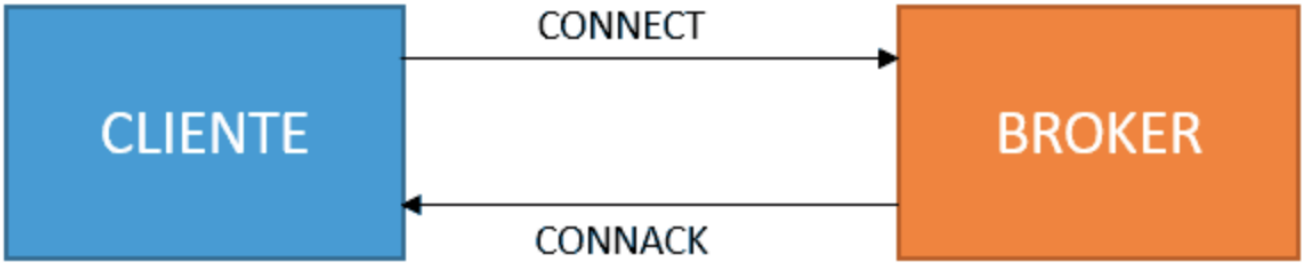

Figura 3-9: Conexión MQTT. [28]

Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el topic y el payload.

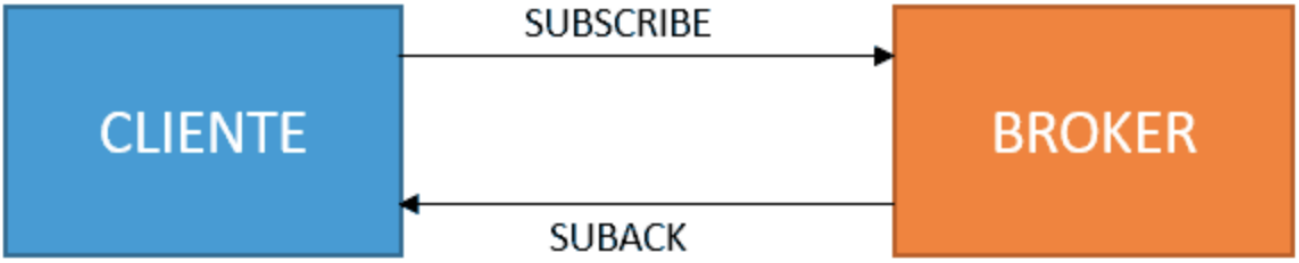

Figura 3-10: Subscribirse al servidor. [28]

Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACK y UNSUBACK.

Figura 3-11: Publicar. [28]

Por otro lado, para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT".

Para entender más fácilmente el servidor de mensajes Mosquitto integrado en este TFG, y como resumen de lo explicado anteriormente, se muestra en la Figura 3-12 la arquitectura empleada para obtener la información dinámica en tiempo real:

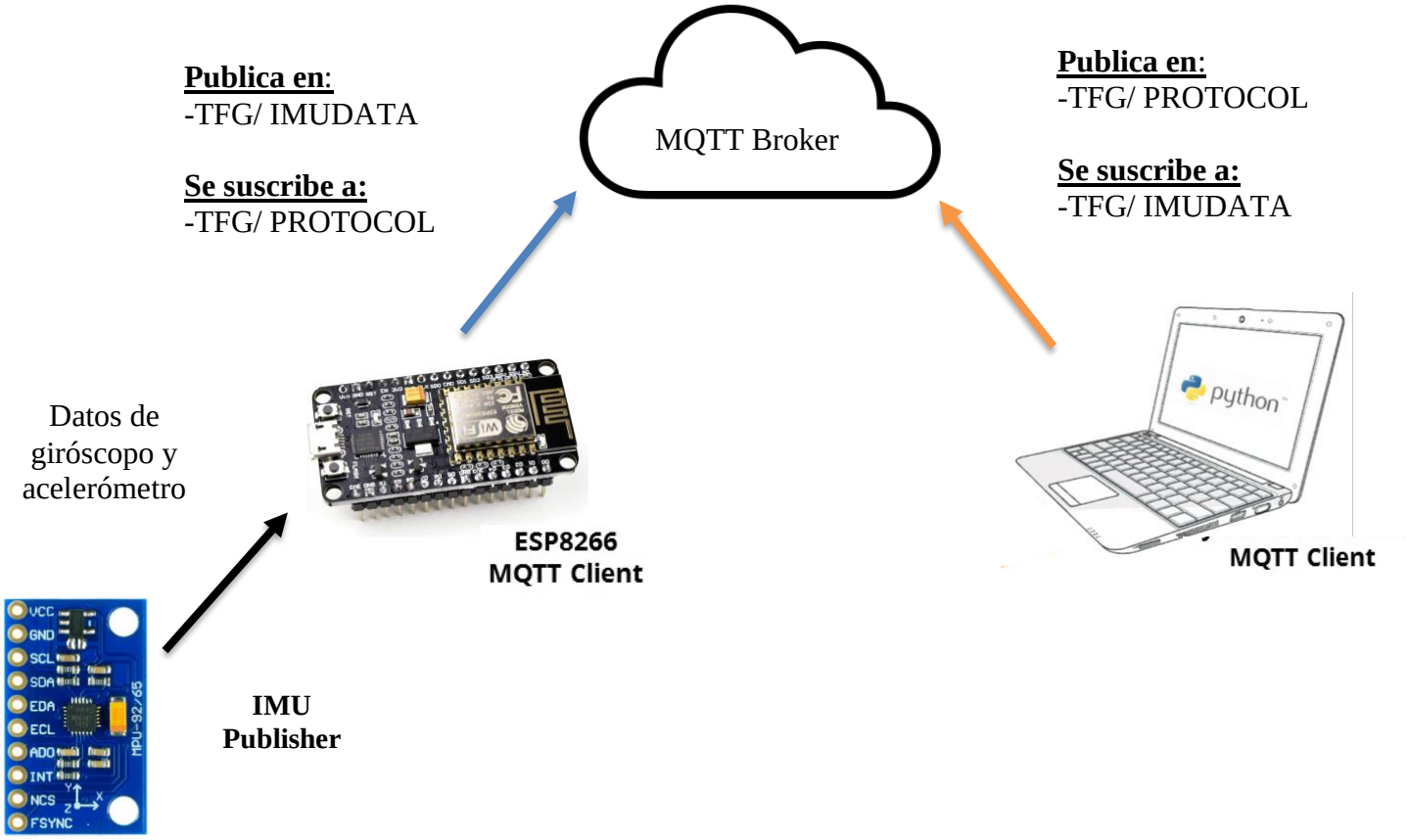

Figura 3-12: MQTT TFG [Fuente Propia]

3.3 Evolución del prototipo

3.3.1 Primer prototipo

Para poder analizar las propiedades dinámicas en 3 dimensiones, es necesario construir una estructura que contenga un número determinado de sensores, definidos dentro de este capítulo en el punto 3.2.1.1, y situarlo dentro de una cubeta de agua. Para ello, la estructura tiene que ser de un material que flote y que, a su vez, nos garantice que los sensores no sufrirán daños debido a salpicaduras.

Hay que tener en cuenta en todo momento que el objetivo de utilizar mayor número de sensores de bajo coste es doble. Por una parte, tenemos la **adaptabilidad** que proporcionan, permitiendo realizar modificaciones sin que suponga un coste considerable.

Por otra parte, a mayor número de sensores que envíen datos, la capacidad operativa del sistema puede mantenerse cuando uno de ellos falle y además el fallo ser detectado. Esto aporta una robustez a la solución planteada que un único sensor no puede ofrecer (sistemas tolerantes a fallos). De esta forma, si es posible conseguir la misma precisión que ofrece un sensor de elevado coste por medio de una red de sensores de bajo coste, se obtienen beneficios adicionales que justifican el estudio de este tipo se soluciones.

Un buen ejemplo de lo anterior podemos encontrarlo en los aviones, cuyo buen funcionamiento depende de muchos factores. Entre ellos, la redundancia de sus sistemas críticos. "*El principio de redundancia se basa en garantizar que, frente a posibles desperfectos en el funcionamiento de un equipo o sistemas, exista otro que asuma sus funciones. Así, instrumentos de vuelo como los indicadores de*

velocidad y altitud, o los sistemas de comunicaciones, se encuentran incluso triplicados en la cabina de los pilotos". [29]

En primer lugar, para colocar los sensores, se hizo uso de **ProtoBoards**, que permiten crear y modificar los circuitos con fluidez y rapidez a lo largo del proyecto ofreciéndonos flexibilidad, donde se situaron las NodeMcu (3.2.1.2) que emitirán por Wi-Fi los datos de las IMUs

El esquema de montaje de los sensores en cada una de las ProtoBoards queda de la siguiente manera (Figura 3-13):

-Conexión a **3V**

-RST: Reinicio del código Arduino introducido.

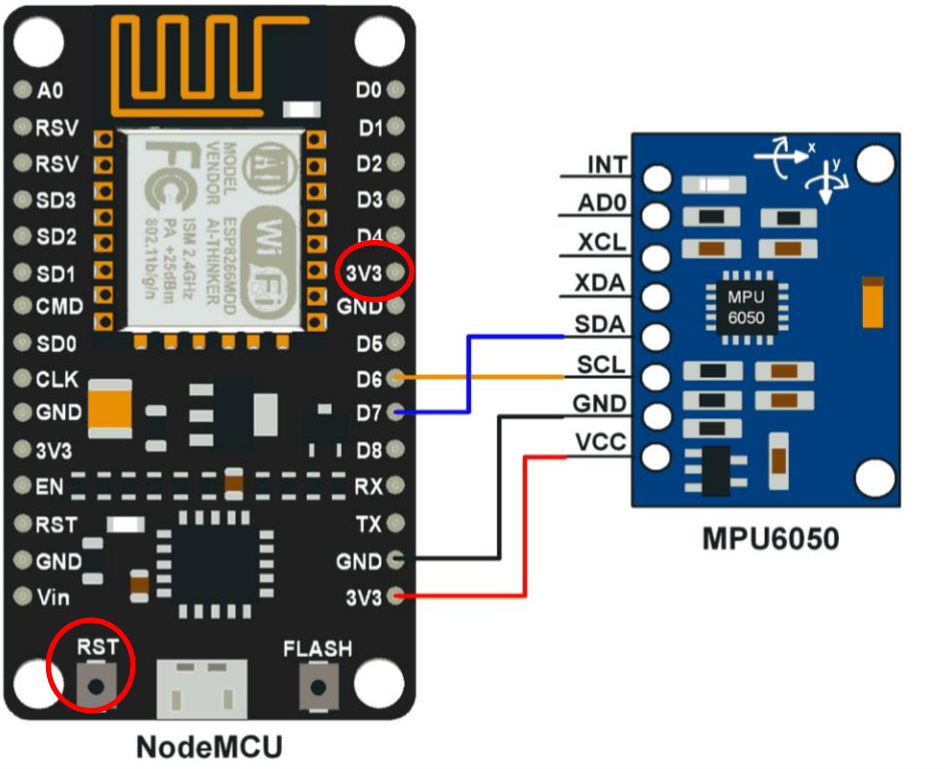

Figura 3-13: Montaje en Protoboard [Fuente Propia]

Para la construcción del prototipo de tipo bloque, se ha utilizado de base **espuma de poliuretano** (se unirán dos planchas) que, gracias a la versatilidad, estanqueidad, resistencia a la fisura, adherencia y permeabilidad del material nos permitirá construir una base estable para los sensores que no permita el paso del agua de la cubeta durante los movimientos a los que se someterá. Sin embargo, es necesario previamente definir la dirección y sentido de cada una de las IMUs y la orientación de la plataforma que simula una embarcación (Figura 3-14 y Figura 3-15).

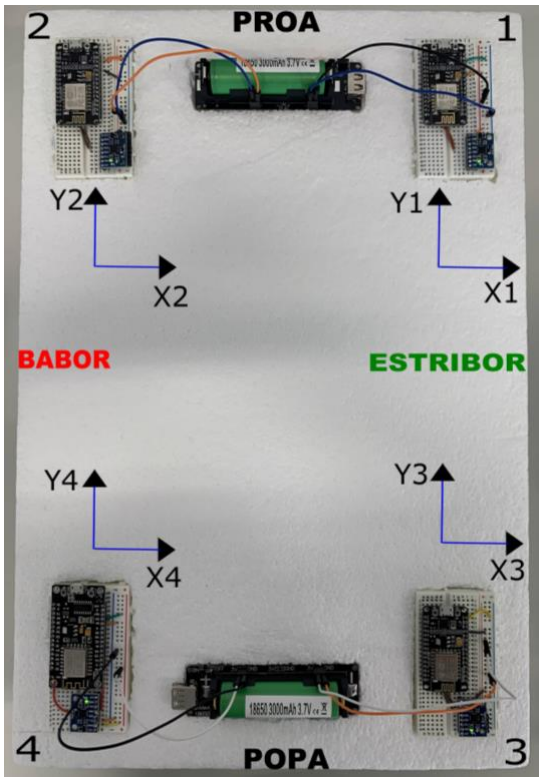

Figura 3-14: Primer prototipo estructura acuática [Fuente Propia]

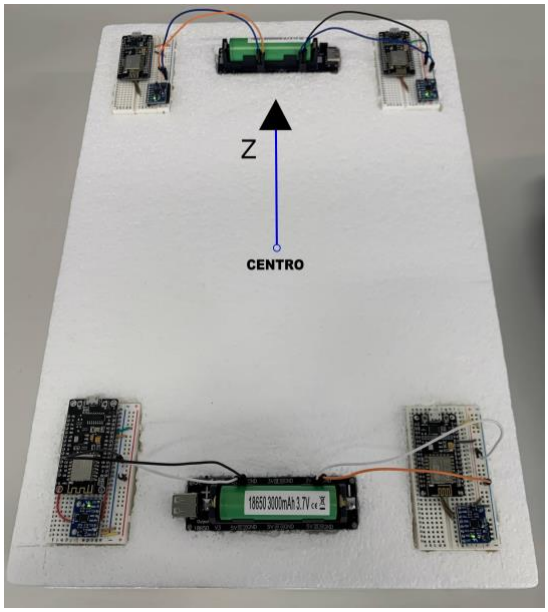

Figura 3-15: Centro de la estructura, punto de estimaciones [Fuente Propia]

3.3.2 Prototipo final: "Bloque"

Con el fin de impedir que las salpicaduras a la hora de realizar los experimentos deterioren los sensores, se diseñó un último prototipo que incluye varias capas de protección con polietileno de baja densidad y un mástil en el centro de la estructura. A este prototipo le llamaremos "**bloque**" y será sobre el que realizaremos las calibraciones y los diferentes tests. El mástil define la posición del punto de origen donde realizaremos estimaciones y analizaremos las propiedades dinámicas del conjunto utilizando los filtros expuestos en el punto 1.1 y será nuestro punto de referencia del sistema adherido al

objeto. Además, el mástil proporcionará cierta inestabilidad a la estructura con el fin de estudiar mejor su comportamiento y simular en mayor medida el movimiento de un barco en el agua. Este prototipo queda del siguiente modo:

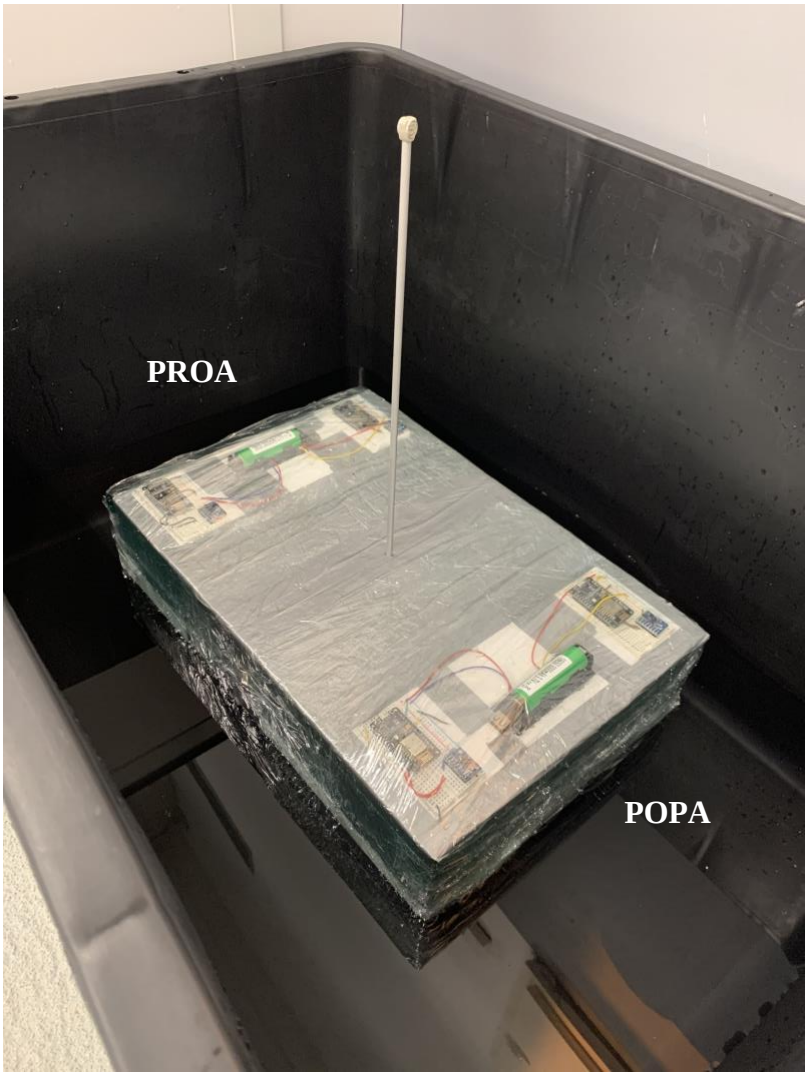

Figura 3-16: Prototipo final [Fuente Propia]

3.4 Barco

Tras finalizar todos los experimentos, instalaremos dos ProtoBoards con sus correspondientes IMUs que estudiarán el comportamiento de un velero a escala al que llamaremos "**barco**" durante las calibraciones y experimentos. Dispone además de quilla, por lo que se asemejará en mayor medida al movimiento de una embarcación navegando en el agua.

Realizaremos los mismos experimentos que con el prototipo expuesto en la Figura 3-16 y veremos cuáles son las diferencias en cuanto a la exactitud de las medidas utilizando 2 o 4 sensores.

La disposición de las ProtoBoards en el barco es la siguiente:

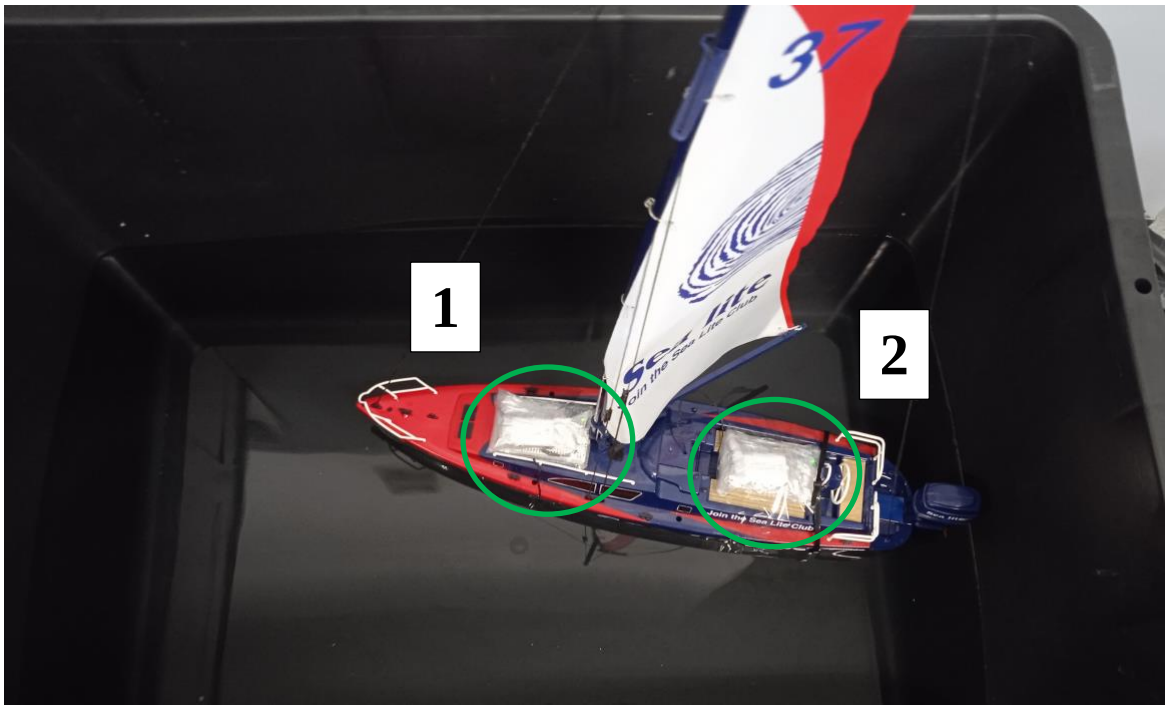

Figura 3-17: Barco [Fuente Propia]

3.5 Establecimiento de la conexión

Para que los sensores nos envíen la información y podamos verla por pantalla, es necesario seguir una serie de pasos:

- En primer lugar, hay que configurar cada uno de los NodeMCU con su correspondiente código Arduino, incluido en el **Anexo I:** , donde a cada uno se le asigne un número según la Figura 3-14. Para ello, se conecta uno a uno mediante un cable USB-micro USB al ordenador. Acto seguido, se compila el código que se le haya asignado según su número y se comprueba por monitor serie si se ha establecido la conexión vía WiFi, así como con el servidor MQTT, habiendo previamente inicializado el servidor a través del terminal Anaconda Prompt (3.2.2.2) de la siguiente manera:

```
(base) C:\Users\Rafa>cd C:\Program Files\mosquitto

(base) C:\Program Files\mosquitto>mosquitto -c mosquitto.conf -v
1614417847: mosquitto version 2.0.4 starting
1614417847: Config loaded from mosquitto.conf.
1614417847: Opening ipv4 listen socket on port 1883.
```

Figura 3-18: Comando Mosquitto [Fuente Propia]

- A continuación, a través del terminal Anaconda Prompt nuevamente, realizamos una prueba de envío de mensajes. Esto consiste en subscribir al sensor en cuestión al tópico TFG y, por otro lado, publicar un mensaje en ese tópico, que debería mostrarse en la ventana del suscriptor. En la Figura 3-19 suscribimos los sensores al tópico TFG que incluye, a su vez, tópicos como PROTOCOL y DATA.

```
(base) C:\Program Files\mosquitto>mosquitto_sub -h 192.168.1.70 -t TFG/#
```

Figura 3-19: Suscripción mosquitto [Fuente Propia]

Una vez suscritos, solo quedará publicar en cualquiera de los tópicos. Como ejemplo, se adjunta en la Figura 3-20 la publicación en PROTOCOL de un "1" o un "0" según queramos que nuestros sensores envíen o no datos:

```
(base) C:\Program Files\mosquitto>mosquitto_pub -h 192.168.1.70 -t PROTOCOL -m "1"
```

Figura 3-20: Publicar en tópico [Fuente Propia]

- Tras comprobar que hay un flujo de mensajes correcto, pasamos a abrir el framework donde visualizaremos los datos del sensor, así como una representación 3D del mismo en VTK (3.2.2.3.1) con unos ejes de coordenadas. Antes de ello, debemos indicar al terminal la dirección de la carpeta en la que se encuentra el programa y ejecutaremos el comando:

```
(base) C:\Users\Rafa>activate qt_pyside

(qt_pyside) C:\Users\Rafa>cd C:\Users\Rafa\Desktop\TFG\EXPERIMENT MANAGER

(qt_pyside) C:\Users\Rafa\Desktop\TFG\EXPERIMENT MANAGER>python main.py
```

Figura 3-21: Abrir framework [Fuente Propia]

Una vez se inicie el framework concluiremos presionando el botón (rodeado en rojo en la

- Figura 3-22) que se ha establecido para indicarle a los sensores que inicien el envío de datos de las IMUs al servidor central. Estos datos se muestran en las gráficas adecuadas y, a falta de integrarlos, se mostraría el valor de los ángulos de pitch y roll estimados:

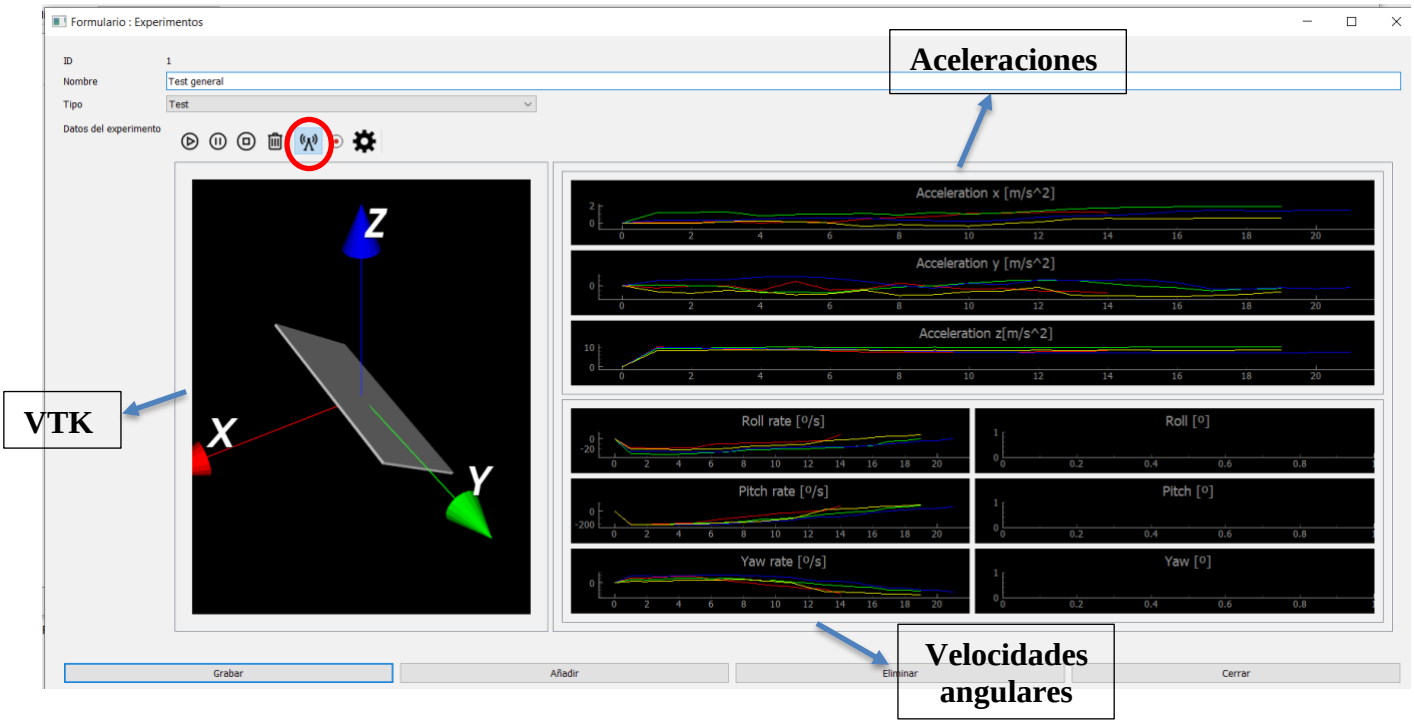

Figura 3-22: Framework. Gráficas PyQt y representación 3D VTK. [Fuente Propia]

- Tras esto, deberíamos ver tanto en Arduino como en el terminal Anaconda Prompt todos los datos llegando del sensor. Así mismo, las gráficas se irán actualizando según movamos la IMU en el eje X, Y o Z. El establecimiento de la conexión se mostraría de la siguiente manera:

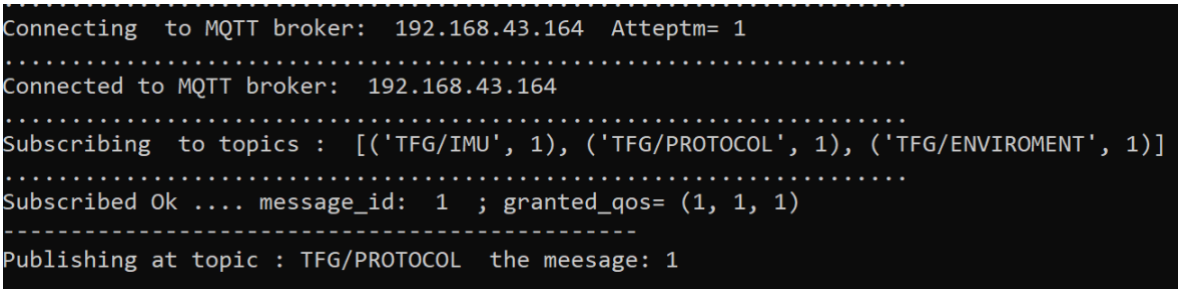

Figura 3-23: Establecimiento de Conexión broker. [Fuente Propia]

Después de repetir este proceso en cada uno de los módulos ESP8266 asociados al NodeMCU para comprobar su correcto funcionamiento, pasamos a conectarlos a las pilas, que nos darán los 3V necesarios para hacer funcionar cada sensor de forma independiente. Una vez los cuatro estén conectados por separado y se ha comprobado su correcto funcionamiento, empleamos un script de Python para realiza el proceso asociado a la toma de datos de las calibraciones y de los experimentos de movimiento con perturbaciones externas. La descripción de esta toma de datos se presentará en el siguiente capítulo.

4 RESULTADOS / VALIDACIÓN / PRUEBA

4.1 Introducción

En este capítulo se va a realizar la recogida de datos empleando los elementos hardware y el software indicado en los capítulos anteriores. A partir de estos recursos se procede a diseñar dos tipos de pruebas:

- **Calibración**: Se trata de un test donde se recogen los datos en una posición estacionaria, los cuales van a ser empleados para establecer los *offsets o bias* (existentes debido al montaje, uso de sensores de baja calidad, etc.) así como los niveles de ruido que introducen este tipo de sensores en posiciones estáticas, y cuyos valores serán empleados en la matriz de covarianza del ruido empleada en el filtro de Kalman.

- **Test dinámicos**: Se trata de la captura de datos del movimiento del bloque y del barco para realizar el análisis de sus propiedades e identificar aspectos del movimiento obtenido cuando se introducen perturbaciones en el entorno.

De esta forma los objetivos de estas pruebas son los siguientes:

- Obtener un conjunto de datos, proporcionados por las diferentes IMUs, que nos describirán el comportamiento del sistema y que podremos procesar para llegar a una serie de conclusiones tales como la obtención de parámetros de calibración o el tipo de filtrado de ruido que debe ser aplicado.

- Desarrollar un sistema que realice la estimación del *roll y el pitch* del elemento flotante, lo cual se suele denominar como *attitude*, que tiene gran importancia en el diseño de sistemas de estabilización (la posición y el *yaw* son mas indicados para sistemas de control de navegación). De esta forma en este apartado no se estimarán los estados asociados a la posición o la orientación Norte-Sur, ya que por otra parte las dimensiones de la cubeta con respecto a las embarcaciones no permitirían generar pruebas con datos que podrían considerarse de interés, ya que requieren de un sistema auxiliar de posicionamiento que verifique los resultados obtenidos.

- Desarrollar un sistema de detección de choques, que permita determinar en qué momento se ha producido un impacto con la pared de la cubeta y, si es posible, estimar en que lugar (proximidad respecto a los sensores) se ha producido dicho impacto. En el caso del barco, debido a que solo tiene dos IMUS, se podrá identificar entre choques en proa y popa, mientras que en el caso del bloque con 4 IMUS se podrá estimar además si el golpe se produce a babor o estribor.

Para lograr dichos objetivos, podemos diferenciar las siguientes etapas.

4.2 Calibraciones

Para cada uno de los sistemas, tanto bloque como barco, realizaremos una toma de datos de las IMUs en posición estacionaria. En el caso del bloque las IMUs se encuentran instaladas en su posición final, pero en el caso del barco la calibración se realiza sobre una mesa previamente a su instalación en el barco. De esta forma se pretende comprobar como el procedimiento de instalación de las IMUs puede afectar a la toma de resultados. Mediante los datos obtenidos en la calibración podremos determinar dos factores necesarios para generar las estimaciones de posición:

- Los niveles de ruido de las señales de aceleración y velocidades angulares.

- Los bias u *offsets* generados por los sensores, que tendrán un desplazamiento (sesgo) con respecto al valor nominal deseado, lo que significa que en reposo no leerán exactamente cero (o el valor de la gravedad en el caso de la aceleración del eje Z). Esto es debido a imperfecciones en el montaje interno que provoca que la toma de datos en posiciones estáticas no sea exacta y tengan desviaciones. Además, cabe destacar que al tratarse de sensores de bajo coste es necesario calibrar antes de cada prueba. El principio detrás de la fusión de sensores, como su nombre lo indica, es combinar los datos recuperados de dos o más sensores imperfectos para darnos una estimación mejorada y más confiable de un estado.

Las calibraciones se realizaron tomando 20000 registros de los 4 sensores (aproximadamente 5000 registros de media por cada sensor, aunque, debido al servidor de mensajes y al delay que tiene cada uno, no todos enviarán la misma cantidad de datos que otros). El proceso empleado para el análisis de los datos obtenidos es el siguiente:

1. Extracción del bias de cada uno de los sensores (nombrados del 1 al 4 en el bloque tal y como se muestra en la Figura 3-14 y del 1 al 2 en el barco, señalado en la Figura 3-17) en posición de reposo o “neutra”. Se calcula a partir del valor medio de cada señal y, con ello, se obtiene el offset de cada una de las señales generadas por cada sensor (3 aceleraciones y 3 velocidades angulares x 4 sensores).

2. stimación del nivel de amplitud del ruido, para lo cual se calcula la desviación típica con respecto a su valor medio. Este valor de dispersión será empleado posteriormente para crear la matriz de covarianzas que forma parte del algoritmo de Kalman que estime *roll* y *pitch*.

A continuación, se realiza una exposición de los resultados obtenidos en la calibración del bloque y del barco.

4.2.1 Bloque

Para explicar los resultados obtenidos tras la calibración del bloque se utilizarán los datos correspondientes al sensor 2 (S2), así como una visión global de los resultados obtenidos.

En primer lugar, en la Figura 4.1 se pueden observar los datos "*raw*" (sin aplicar ningún filtro) obtenidos del acelerómetro del sensor en los 3 ejes durante el tiempo en el que se tomaron las muestras. En el eje vertical se representa la aceleración en $^{m}/_{s^2}$ y en el eje horizontal el tiempo.

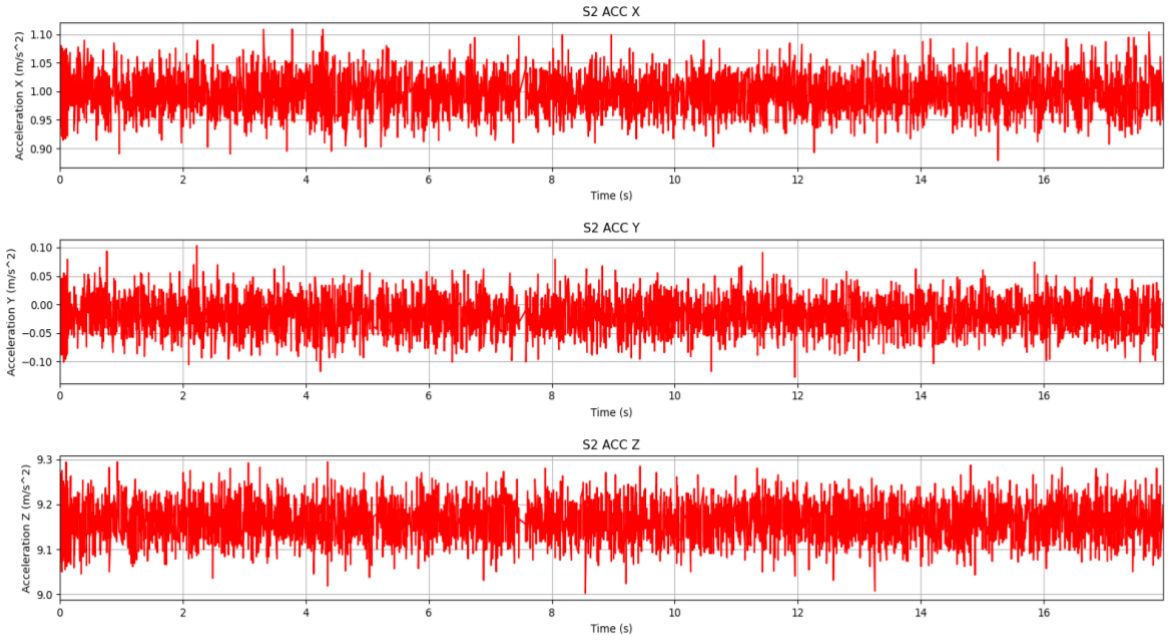

Figura 4-1: Calibración bloque. Aceleraciones sin filtrar. S2. [Fuente Propia]

Al observar los datos acelerómetro tanto en el eje X como en el eje Z, se puede apreciar que existe un desvío u offset en la salida de la señal promedio, incluso estando en reposo. Esto es lo que se conoce como "*offset*" o "*sensor bias*" y podríamos obtenerlo como un valor medio aproximado de 1 y en 9,15 respectivamente. El valor del eje Z tendría que ser cercano a la aceleración de la gravedad en positivo debido a la posición de la IMU y el resto de los ejes deberían dar valores nulos. Los sensores presentan gran sensibilidad a las agitaciones externas a pesar del ruido, lo que impide que podamos distinguir cuál es la forma de la señal con claridad, pero, si embargo, podemos acotar el ruido con una amplitud de aproximadamente 0,2 $^{m}/_{s^2}$ en cada uno de los ejes.

La siguiente gráfica que pasamos a analizar es la relacionada con las velocidades angulares del *roll*, *pitch* y *yaw* (definidos en el apartado 2.1.2 de la memoria), donde las unidades del eje vertical representan º/s y el eje horizontal representa el tiempo transcurrido durante la toma de datos.

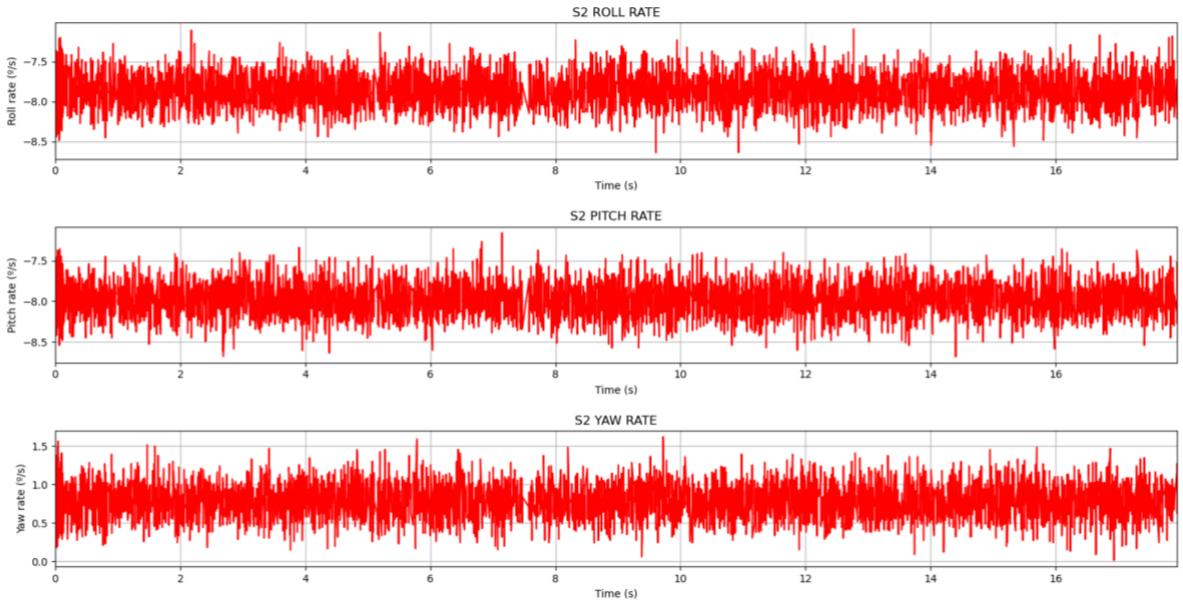

Figura 4-2: Calibración bloque. Velocidades angulares del *roll, pitch y yaw. S2.* [Fuente Propia]

En esta gráfica, hay que destacar que el valor medio diferente de cero no significa que el sensor se mueva a dicha velocidad, sino que debido a que el sensor no esta colocado perfectamente se genera un *offset* en las velocidades angulares debido a la proyección del vector de la aceleración de la gravedad sobre los ejes no alineados con el sistema de referencia del objeto, por lo que de no realizar la corrección podríamos pensar que el objeto se encuentra girando en torno a sus ejes, cuando no es así. Se puede apreciar nuevamente el ruido que se genera en las señales obtenidas de los giróscopos, de forma que al igual que ocurría con las aceleraciones, se emplearán sus propiedades estadísticas para definir *offsets* y matrices de covarianza.

Por último, antes de finalizar el análisis de los datos "*raw*" obtenidos del sensor 2, en la Figura 4.3 se pueden apreciar una serie de histogramas tanto de las aceleraciones como de las velocidades angulares.

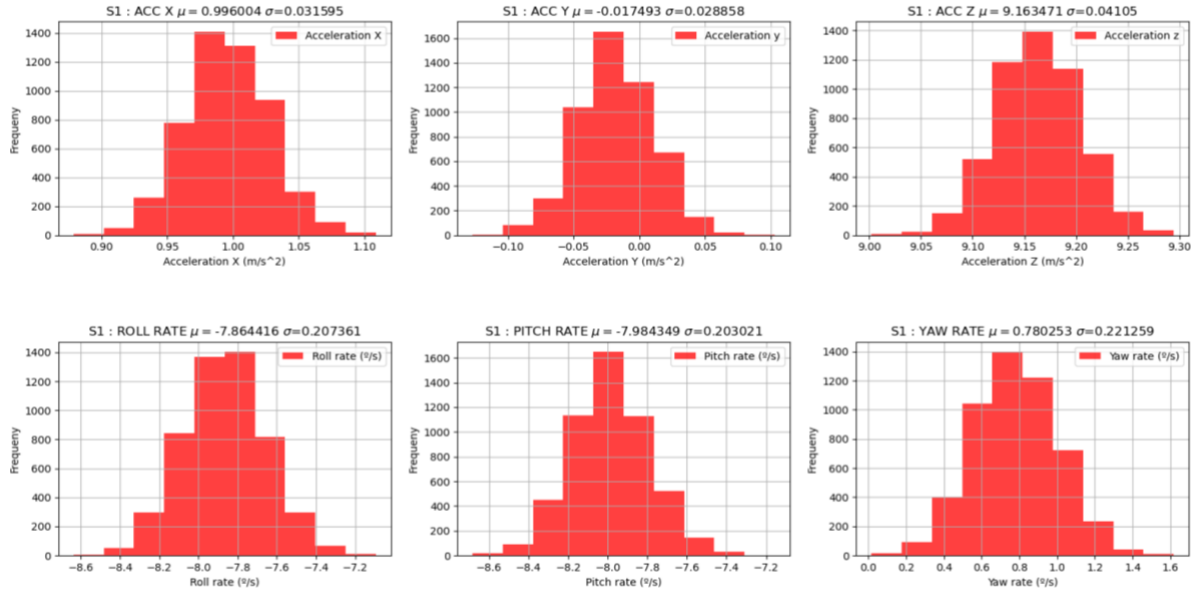

Figura 4-3: Calibración bloque. Histogramas de aceleraciones y velocidades angulares. S2. [Fuente Propia]

Los ejes horizontales representan las magnitudes de las variables obtenidas (aceleraciones y velocidades angulares) y el eje vertical representa la frecuencia. Esta frecuencia nos indica el número de veces que el sensor ha tomado cierto valor de aceleración o de velocidad angular durante el tiempo de calibración. En la parte superior, vemos que se indica el valor medio calculado μ, que nos dará el *bias* y el *offset*, así como la desviación típica (σ) que nos da el valor medio del ruido que tiene la señal y que utilizaremos para generar el filtro de Kalman.

Debe tenerse en cuenta que los histogramas presentados son funciones de distribución discretas (se representan mediante diagrama de barras) que no han sido normalizados.

Para tener una visión global del *bias* de todos los ejes al mismo tiempo, se adjunta a continuación una gráfica (Figura 4-4) con los histogramas de todos los sensores. Estos histogramas presentan una forma cuasi-gaussiana unimodal, lo cual justifica el uso de un filtro de Kalman para realizar estimaciones de los ángulos de roll y pitch a partir de estas señales, ya que recordemos que una de las necesidades para aplicar un estimador de este tipo consiste en que el ruido de los modelos de movimiento y de observación deben estar sujetos a ruido Gaussiano. Se puede apreciar que el *bias* de cada sensor es totalmente diferente respecto al resto, lo cual justifica la necesidad de realizar la calibración cada vez que un sensor es modificado o reinstalado.

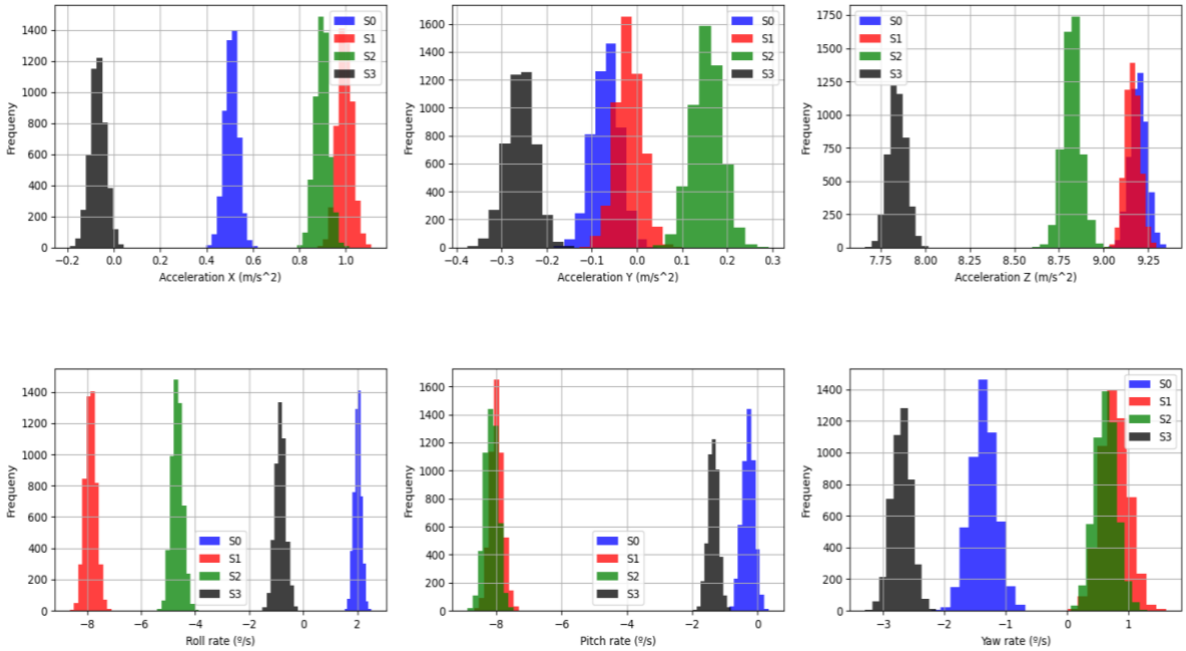

Figura 4-4: Calibración bloque. Histogramas de aceleraciones y velocidades angulares. S1, S2, S3 y S4. [Fuente Propia]

Una vez analizada la información de calibración de cada sensor, se procede a realizar la corrección o filtrado de las señales *raw* obtenidas, para generar las señales de entrada definitivas que serán empleadas en los cálculos posteriores. Para ello vamos a aplicar un filtro GH para reducir el nivel de ruido de la señal y además de aplicará el *offset* obtenido para corregir el error causado por este problema. Cabe destacar que no se ha considerado aplicar otro tipo de filtrado más avanzado debido a que las señales *raw* obtenidas tiene la suficiente calidad como para conseguir buenos resultados con un filtro sencillo de primer orden como el propuesto.

Como ejemplo, se muestra en las figuras 4-5, 4-6 y 4-7 el resultado obtenido tras aplicar el procedimiento indicado, siendo los valores aplicados al filtro GH: g=0.2 y h=0.02 para todas las señales (aceleraciones y velocidades angulares).

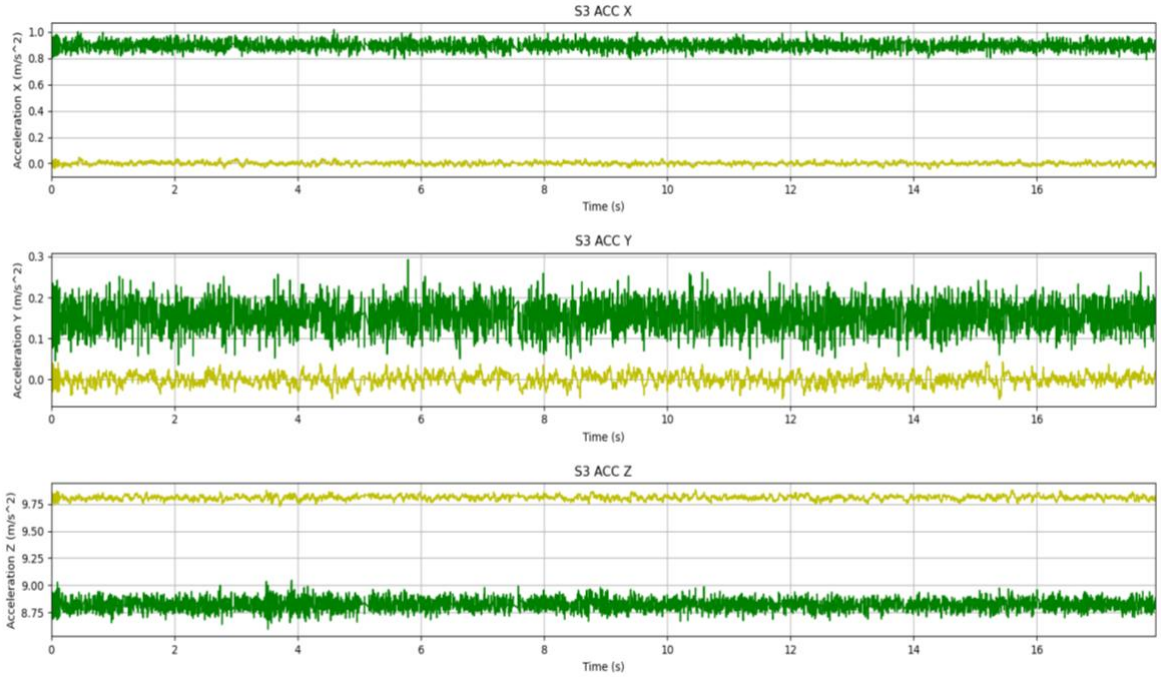

Figura 4-5: Calibración bloque. Aplicación filtro GH y offset para aceleraciones. S3. [Fuente Propia]

En color verde se representa la señal original, o *raw*, y en color amarillo el resultado tras reducir el ruido con el filtro GH y tras haber prácticamente eliminado el *bias* aplicando el *offset* a la señal, es decir, restándole el valor que tenía inicialmente para poder entrarlo en 0 o en un valor cercano a la aceleración de la gravedad en el caso del acelerómetro en el eje Z. Lo mismo ocurre para las velocidades angulares:

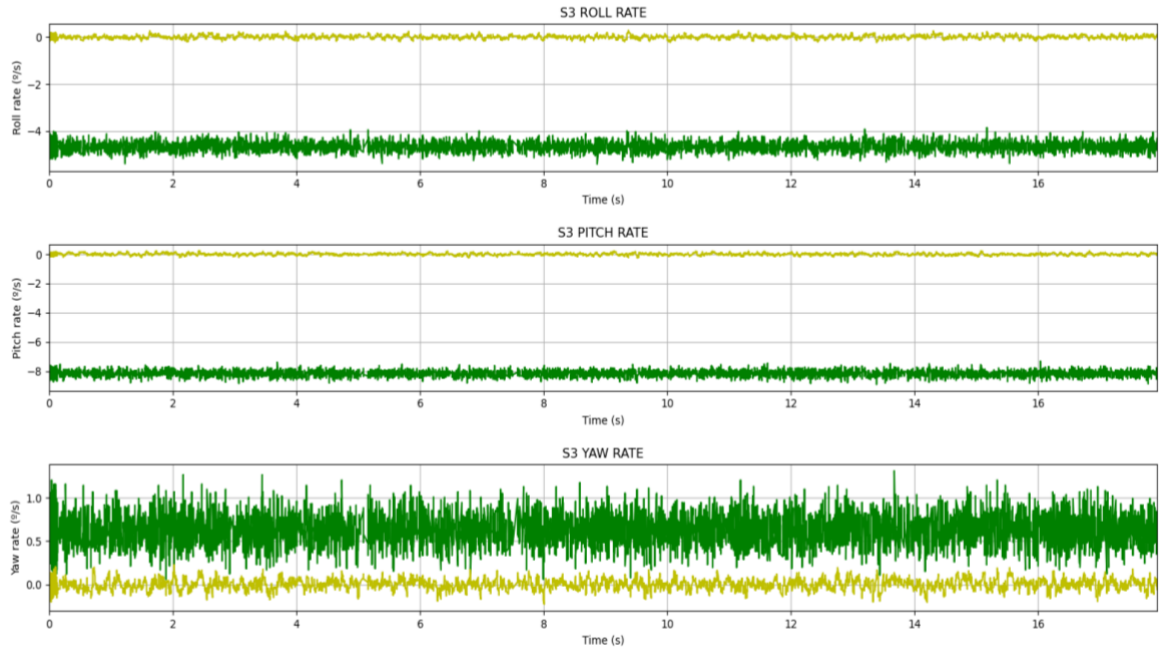

Figura 4-6: Calibración bloque. Aplicación filtro GH y offset para velocidades angulares. S3. [Fuente Propia]

En la Figura 4-7 podemos observar los diferentes histogramas obtenidos del sensor 3 (color verde) y el resultado tras haber aplicado el *offset* y el filtro GH (color amarillo). Con el procedimiento aplicado conseguimos centrar el valor medio de cada uno de los ejes al valor deseado en la posición estática y reducir considerablemente los niveles de ruido de la señal. Para los ejes X e Y de aceleraciones y los ejes X, Y, Z en las velocidades angulares salen valores próximos a 0; para el eje Z del acelerómetro se obtiene un valor medio cercano a la aceleración de la gravedad. También, se puede distinguir una gran diferencia en el ancho de la señal *raw* con respecto a la señal filtrada y corregida, lo que significa que se ha reducido los niveles de ruido obtenidos en el test de calibración.

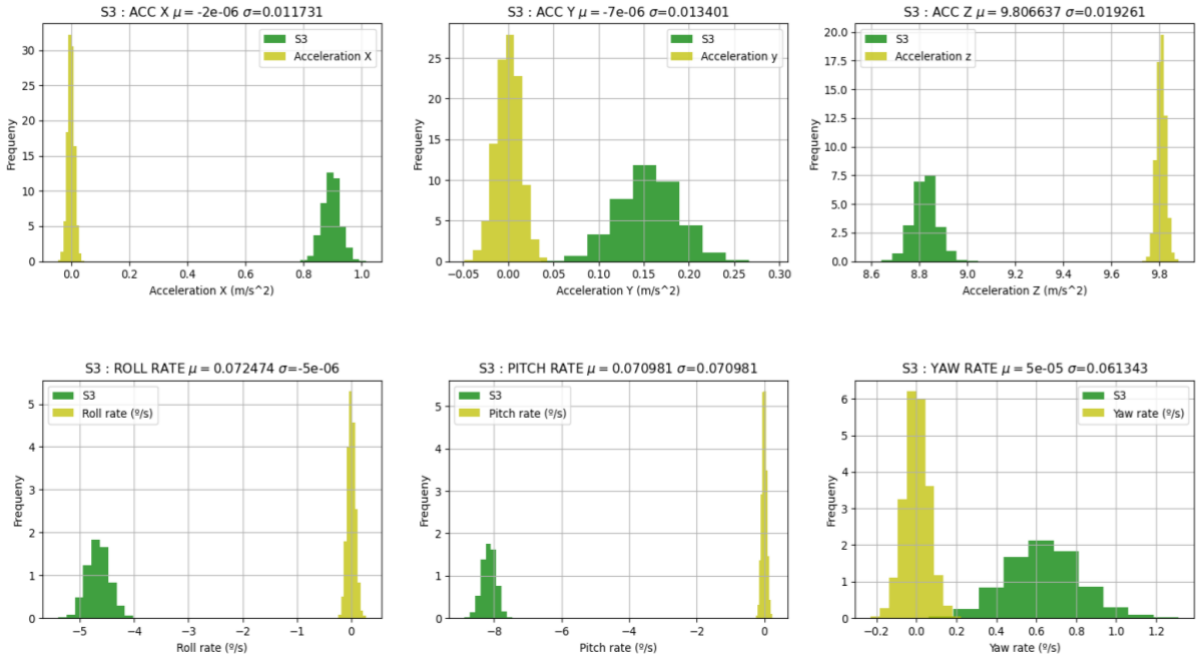

Figura 4-7: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S3. [Fuente Propia]

Por último, se adjunta a continuación dos tablas que contienen los *bias* de cada uno de los sensores en su estado inicial, así como la varianza asociada al ruido de la señal en estado estacionario.

BIAS	ACC X	ACC Y	ACC Z	ROLL	PITCH	YAW
S1	0.506559	-0.068776	9.196736- gravity	2.007802	-0.29176	-1.369547
S2	0.996004	-0.017493	9.163471 - gravity	-7.864416	-7.984349	0.780253
S3	0.896217,	0.154769	8.822748 - gravity	-4.659935	-8.154066	0.629115
S4	-0.068339	-0.257984	7.842505 - gravity	-0.848788	-1.358859	-2.704832

Tabla 2: Calibración bloque. Parámetros de offset obtenidos. [Fuente Propia]

NOISE	ACC X	ACC Y	ACC Z	ROLL	PITCH	YAW
S1	0.030387	0.030614	0.047509	0.136699	0.185545	0.208693
S2	0.031595	0.028858	0.04105	0.207361	0.203021	0.221259
S3	0.030249	0.032747	0.050443	0.205358	0.207641	0.182644
S4	0.033905	0.032539	0.049021	0.201353	0.160368	0.160808

Tabla 3: Calibración bloque. Parámetros de dispersión del ruido. [Fuente Propia]

4.2.2 Barco

Tal y como vimos en el apartado anterior del bloque, a continuación, se muestran las gráficas asociadas al proceso de calibración de las IMUs instaladas en el barco. Cabe destacar se calibraron las IMUS previamente a ser instalarlas en el barco, por lo que se espera que los offsets finales que sean aplicados no sean totalmente correctos. Se decidió actuar de esta forma, y no realizar una nueva toma de datos con las IMUs montadas en el barco, para analizar cómo el proceso de montaje puede alterar los resultados en las estimaciones y generar, por tanto, errores en el proceso de cálculos de la posición del elemento flotante.

En primer lugar, en la Figura 4-8 e Figura 4-9, observamos los datos *raw* y filtrados del sensor 1 sensor 1, en donde, tras haberle aplicado el filtro GH con los mismos parámetros que el que utilizamos en el bloque, se consigue reducir en gran medida el ruido generado por el sensor. Así mismo, se aplica el *offset* para centrar los datos de aceleraciones y de velocidades angulares en torno a un valor medio nulo, excepto el valor en el eje Z del acelerómetro que tendría que ser cercano a la aceleración de la gravedad.

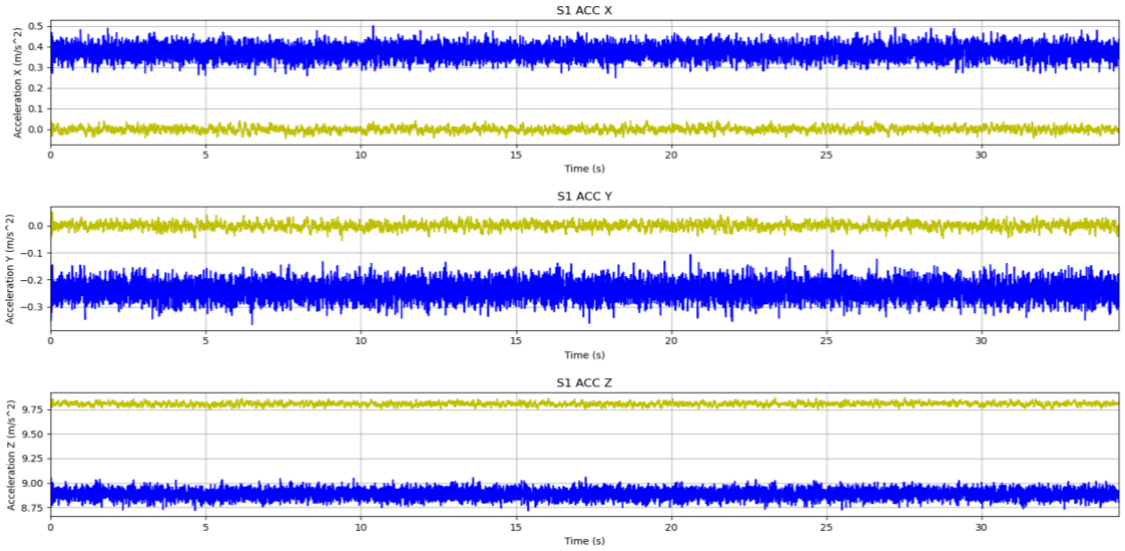

Figura 4-8: Calibración bloque. Aplicación filtro GH y offset para aceleraciones. S1. [Fuente Propia]

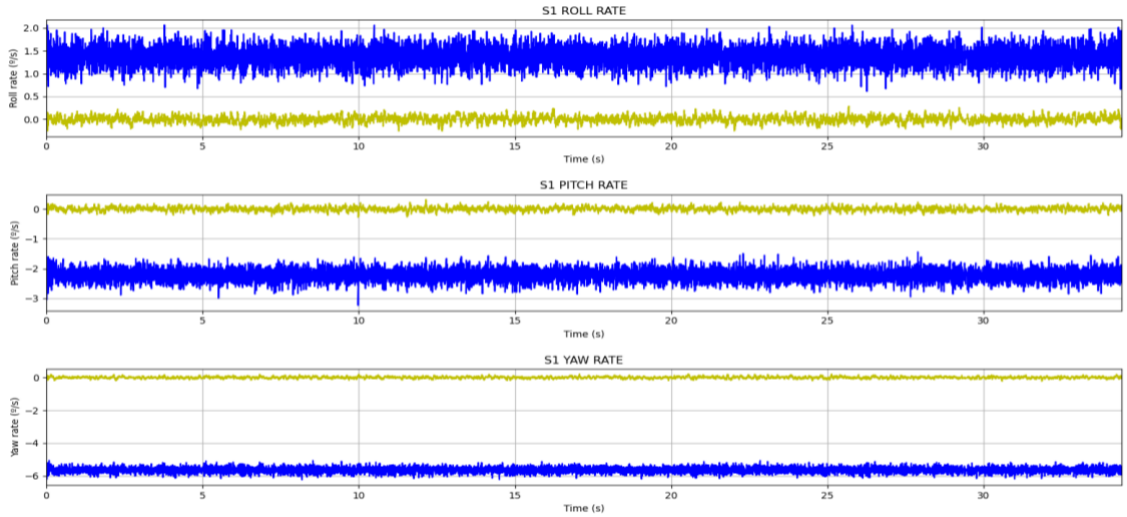

Figura 4-9: Calibración bloque. Aplicación filtro GH y offset para velocidades angulares. S1. [Fuente Propia]

Por último, se adjuntan los diferentes histogramas encuadrados en las figuras 4-10 y 4-11 para ambos sensores instalados en el barco. Podemos comprobar y comparar la forma cuasi-gaussiana unimodal que poseen tanto los datos *raw* como corregidos. Esto justificará el uso del filtro de Kalman debido a la necesidad de estimar el *roll* y el *pitch* del barco, que son principalmente los encargados de la estabilización en el agua debido a las condiciones donde realizaremos los tests posteriores.

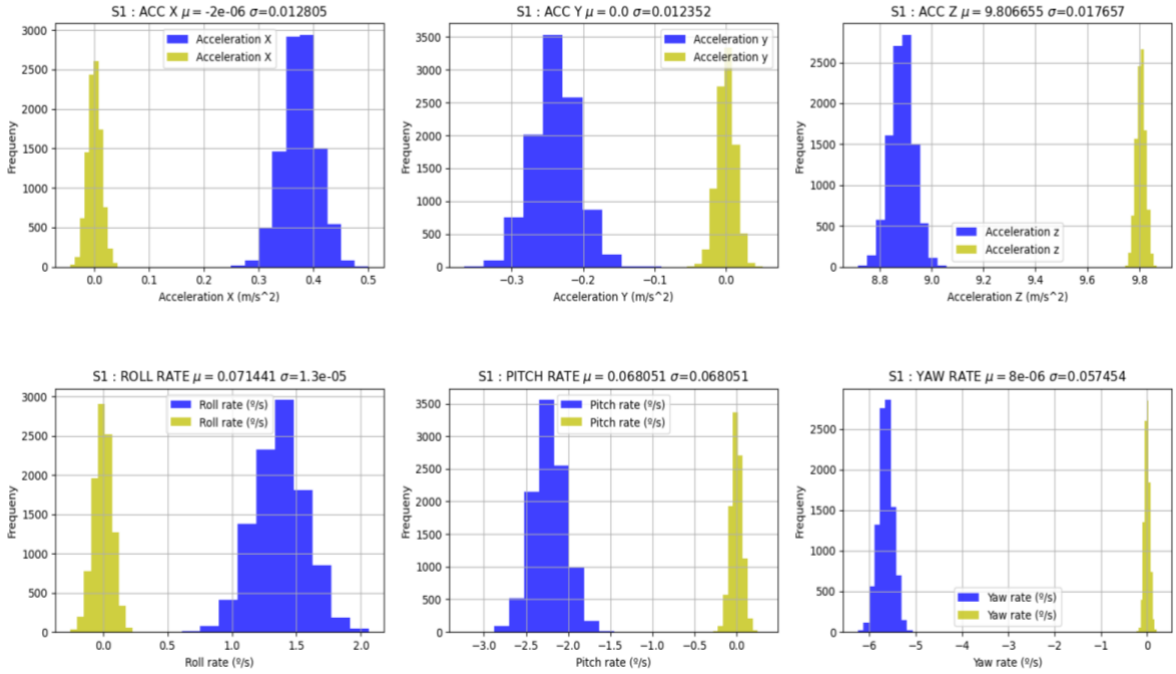

Figura 4-10: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S1. [Fuente Propia]

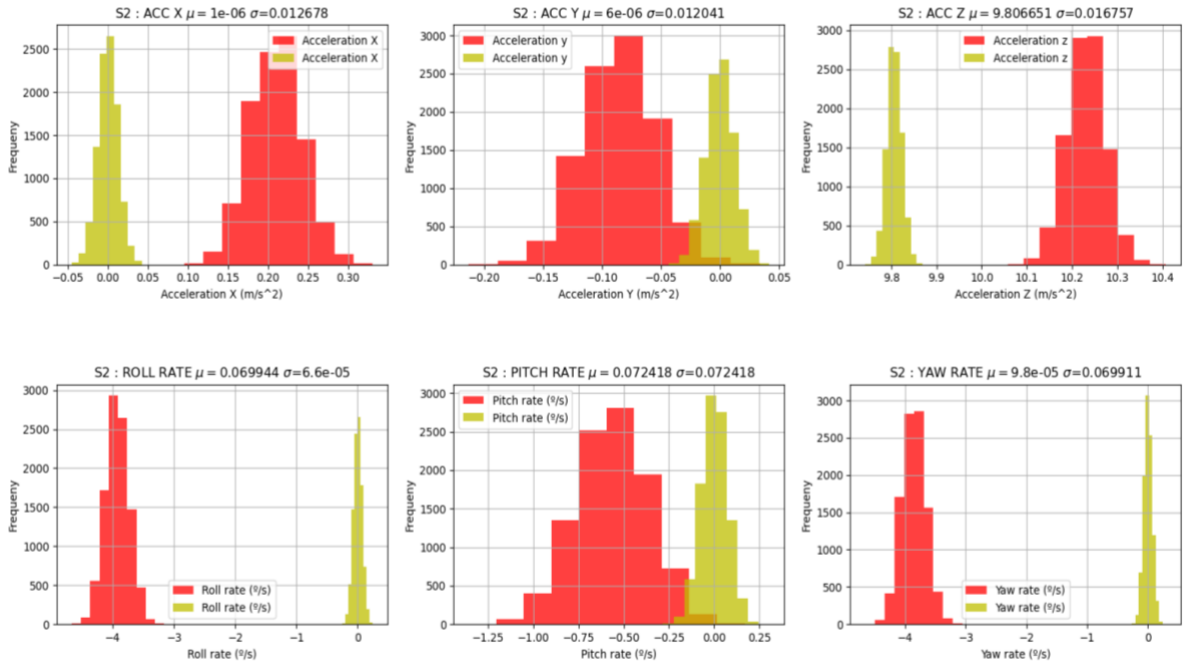

Figura 4-11: Calibración bloque. Comparativa de histogramas entre datos raw y filtrados. S2. [Fuente Propia]

4.3 Estimación de posición

Una vez se han corregido tanto el ruido como la desviación que poseían los sensores debido a razones de montaje, se procede a elaborar un algoritmo que nos permita estimar el *roll* y el *pitch* de las estructuras flotantes (tanto barco como bloque). Para realizar esta tarea se utilizará un filtro de Kalman, definido teóricamente en el punto 0, y que en este apartado se empleará desde un enfoque práctico.

El filtro de Kalman también se conoce como estimador cuadrático lineal. Es un tipo de observador o estimador de estado que es óptimo en el sentido de que intenta minimizar una función de coste cuadrática. Además, el filtro de Kalman no solo tendrá en cuenta las medidas de nuestros sensores, sino también la dinámica subyacente del sistema y anulará cíclicamente la media y varianza del resultado. El modo de actuar puede resumirse en la Figura 4-12 y se describe comúnmente como "el bucle de filtro que sigue y sigue". Es decir, dado que los valores medidos (en actualización) se ajustan relativamente bien a los estimados (por predicción), el filtro mejora paso a paso para asegurar que la estimación reduzca su error, aunque los valores obtenidos contengan ruido gaussiano.

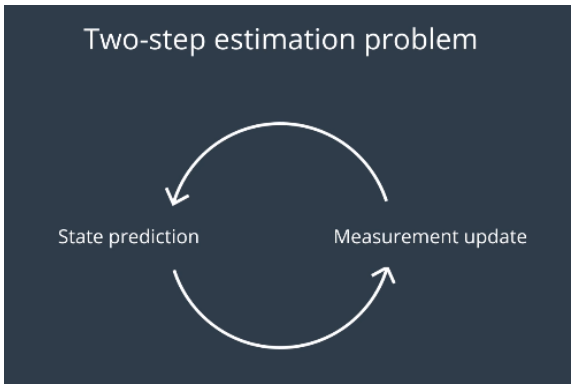

Figura 4-12: Kalman Filter [30]

En este trabajo, las dinámicas del bloque y del barco van a ser definidas empleando un modelo dinámico lineal e invariante en el tiempo, cuya expresión es la siguiente:

$$\vec{x}_{t+1} = \mathbf{A} \cdot \vec{x}_t + \mathbf{B} \cdot \vec{u}_t + \vec{w}_t$$

$$\vec{y}_{t+1} = \mathbf{C} \cdot \vec{x}_{t+1} + \vec{v}_{t+1}$$

Donde el vector de estado del sistema en un instante *t* está representado por x_t y el vector entrada como u_t. A u B son las matrices del sistema que relacionan de forma lineal la evolución temporal con respecto al estado y la entrada actuales, y C relaciona los estados del sistema con respecto a las observaciones obtenidas. Es decir, A y B representan el modelo de movimiento empleado en le filtro, y C representa el modelo de observación. Por último, los vectores w_t y v_t representarán, respectivamente, el ruido asociado a procesos aditivos y el obtenido a través de mediciones.

En primer lugar, para realizar el cálculo de manera óptima a pesar del ruido de la medición, se debe conocer la "intensidad" del ruido. Este parámetro se conoce como la varianza de la distribución normal, que se indica en las gráficas expuestas en el apartado anterior. Una vez determinado para cada uno de los sensores que se está utilizando en ambos cuerpos flotantes (tanto bloque como barco) se utilizará este valor como "incertidumbre" para el cálculo de las matrices de covarianza que emplea el proceso de filtrado recursivo.

Debido a que vamos a conocer en todo momento la posición de las IMUs instaladas en los diferentes elementos flotantes que estudiaremos, así como el *bias* de nuestros sensores que hemos corregido para obtener una estimación actualizada de nuestra posición angular, podemos definir nuestros vectores estado, entrada y de medición como:

$$\vec{x}_t = \begin{bmatrix} \hat{\phi}_t \\ b_{\hat{\phi}_t} \\ \hat{\theta}_t \\ b_{\hat{\theta}_t} \end{bmatrix}$$

$$\vec{u}_t = \begin{bmatrix} \dot{\phi}_{G_t} \\ \dot{\theta}_{G_t} \end{bmatrix}$$

$$\vec{z}_t = \begin{bmatrix} \hat{\phi}_{Acc_t} \\ \hat{\theta}_{Acc_t} \end{bmatrix}$$

Donde $b_{\phi t}$ representa el bias asociado al estimador del ángulo ϕ *(roll)* y $b_{\theta t}$ el asociado al estimador del ángulo θ *(pitch).* La ecuación dinámica asociada el modelo de movimiento del estado del sistema a estimar se obtiene como:

$$\vec{x}_{t+1} = \begin{bmatrix} 1 & -\Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}_t + \begin{bmatrix} \Delta t & 0 \\ 0 & 0 \\ 0 & \Delta t \\ 0 & 0 \end{bmatrix} \vec{u}_t + \vec{w}_t$$

$$\vec{y}_{t+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \vec{x}_{t+1} + \vec{v}_t$$

Tras definir la ecuación de estado general que seguirán cada uno de nuestros sistemas de estudio, pasamos a definir las ecuaciones que utilizaremos para el filtro de Kalman. Estas ecuaciones pueden dividirse en función de la fase del proceso de filtrado:

- **Predicción:**

$$\vec{x}_{t+1} = \mathbf{A} \cdot \vec{x}_t + \mathbf{B} \cdot \vec{u}_t$$
$$\mathbf{P} = \mathbf{A} \cdot \mathbf{P} \cdot \mathbf{A}^T + \mathbf{Q}$$

- **Actualización:**

$$\vec{\tilde{y}}_{t+1} = \vec{z}_{t+1} - \mathbf{C} \cdot \vec{x}_{t+1}$$
$$\mathbf{S} = \mathbf{C} \cdot \mathbf{P} \cdot \mathbf{C}^T + \mathbf{R}$$
$$\mathbf{K} = \mathbf{P} \cdot \mathbf{C}^T \cdot \mathbf{S}^{-1}$$
$$\vec{x}_{t+1} = \vec{x}_{t+1} + \mathbf{K} \cdot \vec{\tilde{y}}_{t+1}$$
$$\mathbf{P} = (\mathbf{I} - \mathbf{K} \cdot \mathbf{C}) \cdot \mathbf{P}$$

En este caso, **K** representará la ganancia de Kalman que se elige de manera óptima dependiendo de lo que el filtro cree que proporciona una estimación más confiable del estado: la medición o el modelo dinámico del sistema. **P** es matriz de covarianza de error, que estableceremos nosotros en primera instancia con los resultados obtenidos de las gráficas tras haber aplicado los diferentes *offsets*, pero será actualizada por el Filtro de Kalman a medida que vaya procesando los datos. **Q** y **R** son las matrices de covarianza del ruido de proceso y el medido.

La matriz Q nos informa sobre el filtro y cómo el estado del sistema puede "saltar" de un paso al siguiente. Es decir, imaginando que el bloque o el barco pudiese ser perturbado por una ráfaga de viento o golpes en la cubeta, esto tendría un efecto de fuerza en el sistema. Mover con la mano el bloque y provocar un cambio de velocidad es también una aceleración que actúa sobre los sensores. Si una aceleración afectase por tanto al estado del sistema, entonces la dependencia física que tendría sería la matriz Q. Por último, R indica la confianza que nos dan los sensores a la hora de medir las aceleraciones y velocidades angulares. Valores altos de R significa altos niveles de ruido, y por lo tanto se genera un filtro que “desconfía” de las nuevas medidas y “confía” más en sus propias predicciones en base a la información anteriormente recogida.

Para implementar el filtro de Kalman, haremos uso de la librería Numpy dentro de Python para realizar todas las operaciones matriciales. Inicializaremos en primer lugar el vector de estimación de estado $x_{t=0}$ y la matriz de covarianza de error $\mathbf{P}_{t=0}$, determinaremos las matrices de varianza del ruido **Q** y **R** y finalmente actualizaremos nuestra estimación de estado siempre que haya una medición disponible usando las ecuaciones dadas anteriormente.

4.3.1 Bloque

Una vez diseñado el filtro, en este apartado se describen una serie de tests que se van a realizar desde distintos lugares de la cubeta para así analizar las aceleraciones y velocidades angulares obtenidas por las IMUs y poder obtener estimaciones del estado del sistema.

4.3.1.1 Test lateral 1

En este test se procede a introducir perturbaciones laterales con respecto a la posición del elemento en la cubeta.

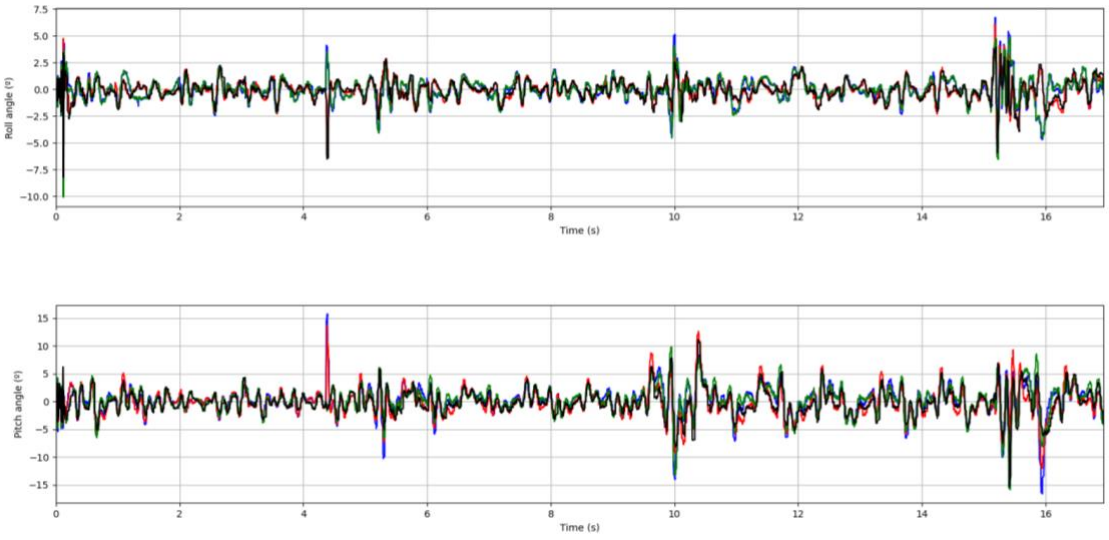

Figura 4-13: Pitch y roll estimados mediante el filtro de Kalman-Test Lateral 1. [Fuente Propia]

En la Figura 4-13 se muestra la estimación de roll y pitch para cada uno de los sensores. Puede observarse que la estimación tiende a divergir cuando se produce un evento impulsivo que, tal y como veremos en el siguiente punto, se asocia a instantes en donde el objeto choca con una de las superficies laterales de la cubeta.

El efecto de estos golpes puede apreciarse nuevamente en la gráfica que muestra las aceleraciones del sensor 1, en donde se puede ver visualizar claramente el efecto impulsivo que generan estos eventos en las señales obtenidas.

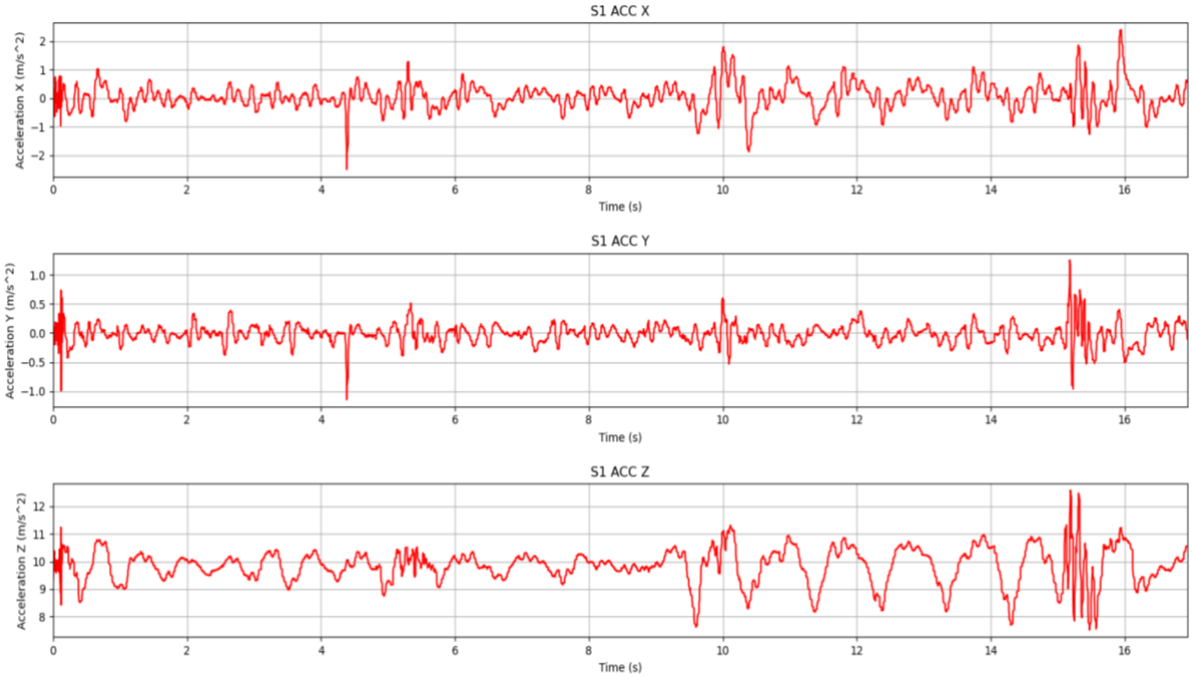

Figura 4-14: Aceleraciones Sensor 1. [Fuente Propia]

4.3.1.2 Test lateral 2

En este caso repetimos las condiciones de test anterior para evaluar si el comportamiento de los resultados obtenidos muestra algún tipo de inconsistencia.

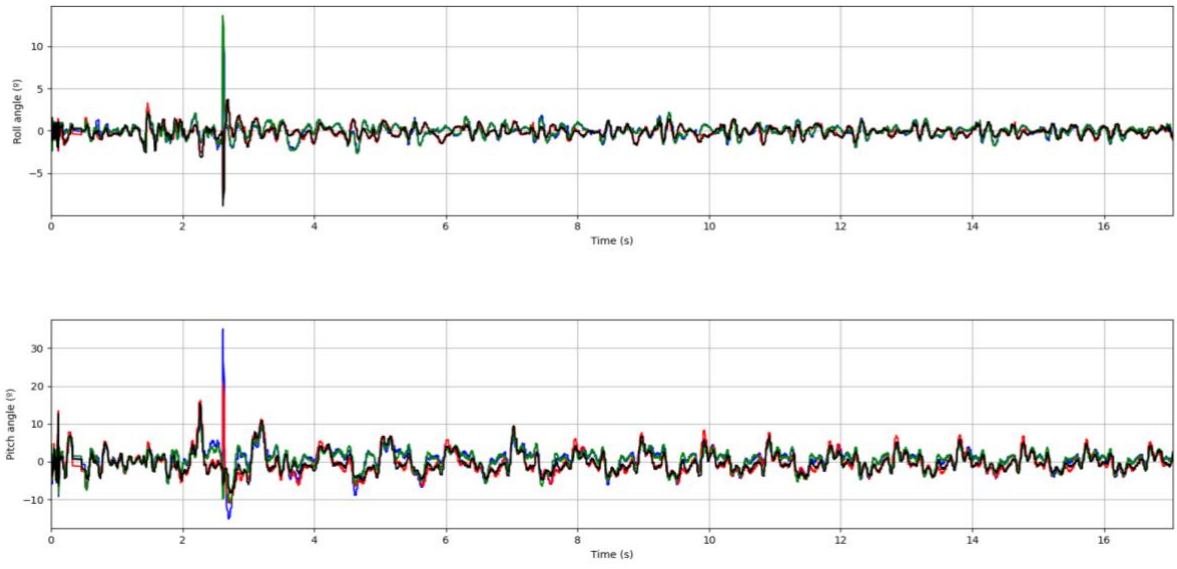

Figura 4-15: Pitch y roll estimados mediante el filtro de Kalman-Test Lateral 2. [Fuente Propia]

En la Figura 4-15 se muestra la estimación de roll y pitch para cada uno de los sensores, en donde podemos apreciar que el comportamiento es similar al test anterior, y en donde nuevamente es posible apreciar los instantes en donde se producen golpes. A falta de un sistema exterior que permita validar los resultados obtenidos, la repetición de varios tests permiten comprobar si en algún momento el algoritmo presenta un comportamiento inestable o con demasiadas diferencias entre diferentes pruebas.

4.3.1.3 Test longitudinal 1

En este caso se procede a introducir perturbaciones en el sentido longitudinal del bloque. La muestra el resultado obtenido en la estimación de los ángulos de posición, en donde no solo es posible apreciar los instantes en los que se producen choques, sino que, además, empleando los vídeos grabados durante los tests, es posible intuir que la localización de la posición del golpe genera patrones en las respuestas de las IMUs que pueden ser empleados para realizar una estimación de la zona en donde se ha recibido el impacto.

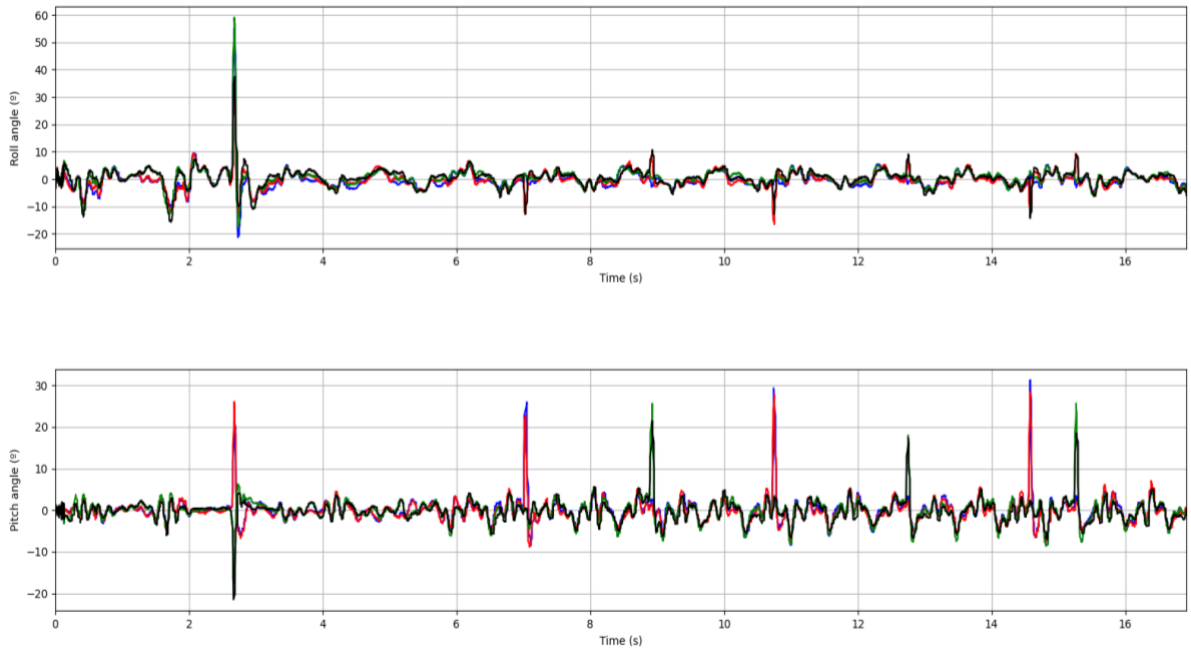

Figura 4-16: Pitch y roll estimados mediante el filtro de Kalman-Test Longitudinal 1. [Fuente Propia]

4.3.1.4 Test longitudinal 2

Al igual que en el caso anterior repetimos el tipo de test para comprobar la consistencia de los resultados. La Figura 4-17 muestra la estimación del pitch y el roll obtenidos en este caso.

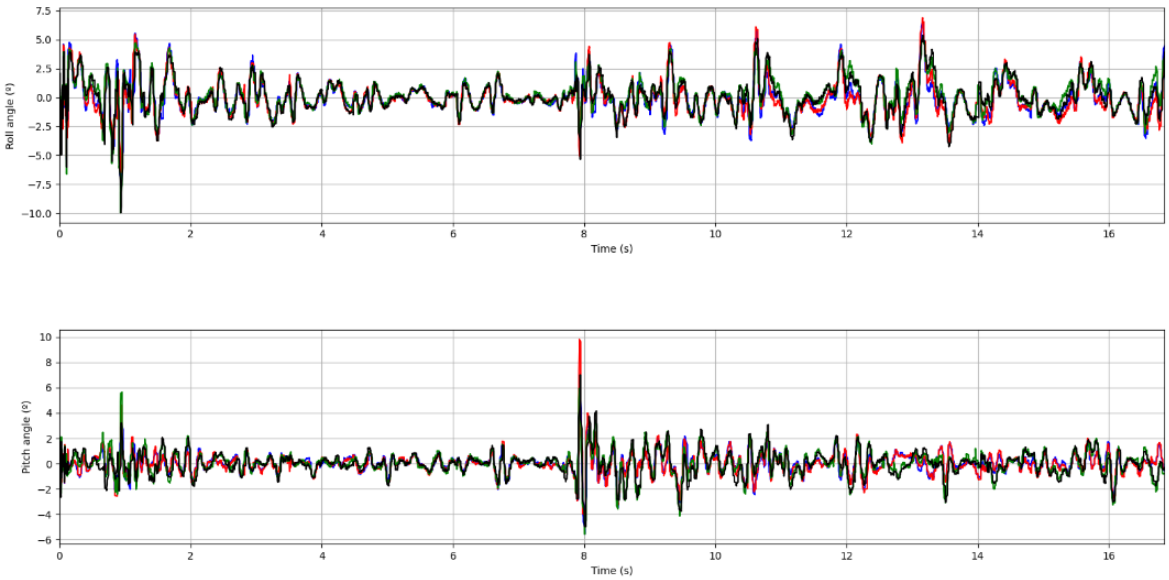

Figura 4-17: Pitch y roll estimados mediante el filtro de Kalman-Test Longitudinal 2. [Fuente Propia]

4.3.2 Barco:

En el caso del barco se repitieron test similares a los empleados con el bloque, pero en este caso debemos tener en cuenta que únicamente se han empleado dos IMUs. Las siguientes gráficas muestran los resultados obtenidos.

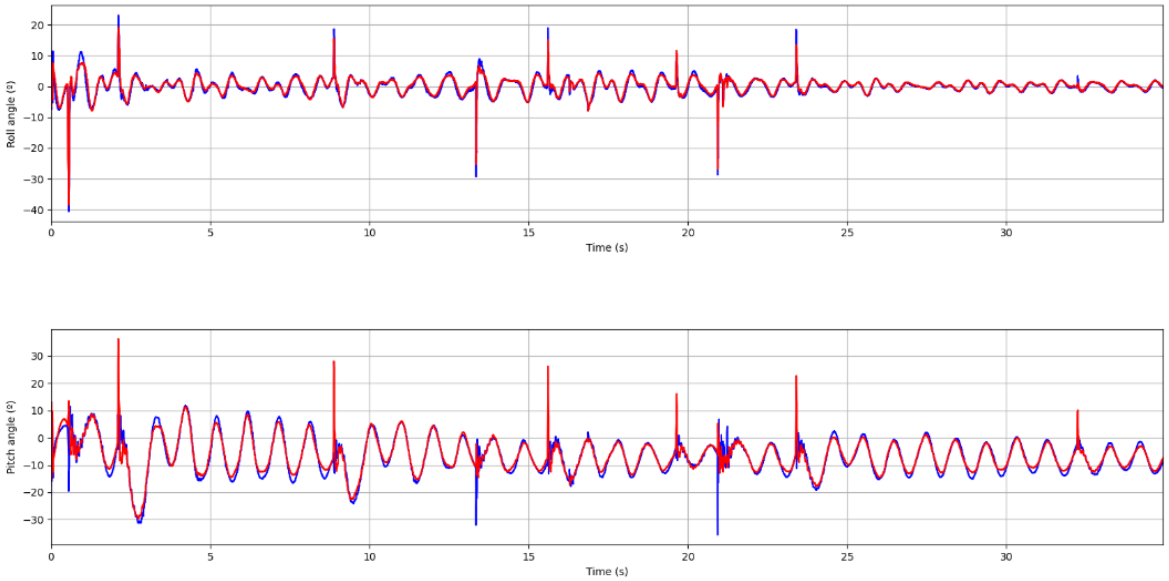

Figura 4-18: Barco. Test lateral 2. Estimación del pitch y el roll. [Fuente Propia]

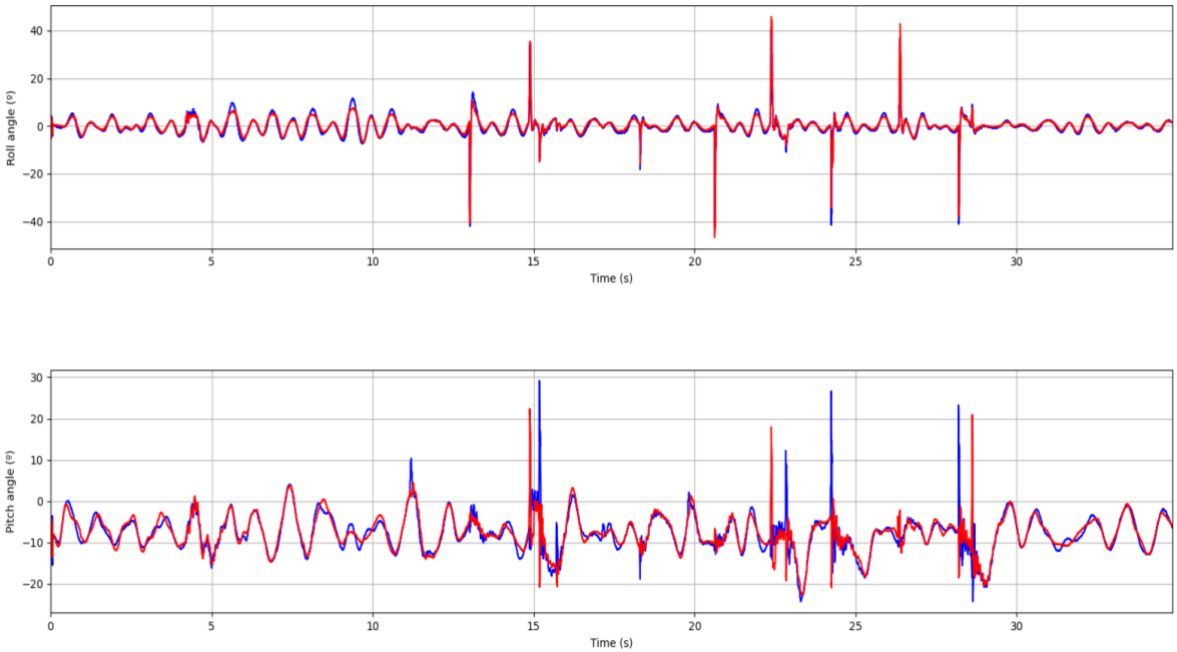

Figura 4-19: Barco. Test lateral 1. Estimación del pitch y el roll. [Fuente Propia]

De forma similar a lo que ocurría en el caso del bloque, es posible apreciar los instantes de los choques, así como una divergencia temporal entre los dos estimadores, que ocurre en función de la localización del golpe (proa o popa). En este caso, debido a la situación de las IMUs y la propia estructura del barco, incluso es posible apreciar un cierto retardo en la reacción de uno de los sensores con respecto al otro en el instante de la detección de los golpes. En la Figura 4-20 se muestra la evolución de las aceleraciones, en donde se aprecia la alta sensibilidad de las señales de aceleraciones X e Y con respecto a la respuesta en la aceleración del eje Z, que, obviamente, se ve menos afectada por los impactos laterales recibidos.

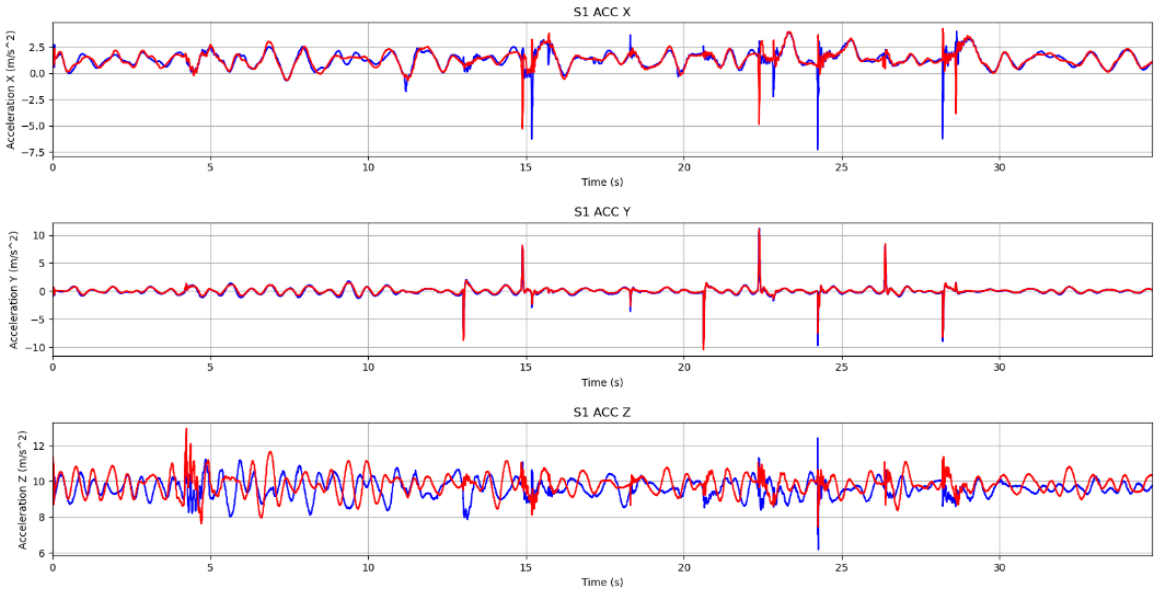

Figura 4-20: Barco. Test Lateral 1. Aceleraciones. [Fuente Propia]

4.4 Identificación de choques

En esta sección vamos a introducir una solución simple para generar un sistema de detección de choques del objeto flotante. El algoritmo empleado para detectar estos eventos es el siguiente:

En cada instante se mantiene una ventana temporal que contiene los datos anteriores de la señal. En los test se ha empleado un tamaño de ventana igual a 300

En cada instante se obtiene el valor medio de los datos de la ventana temporal, y se almacena el valor: | acceleration – valor medio ventana |. Siendo *acceleration* el valor actual de la señal de aceleración.

Por último, se normaliza el vector obtenido y se genera una distribución de probabilidad que representa la posibilidad de detectar un choque. A partir de los datos obtenidos por diferentes pruebas es posible diseñar un algoritmo del tipo *Double Thershold* que establezca la lógica definitiva de la detección.

A continuación, vamos a visualizar los resultados obtenidos tras aplicar el algoritmo propuesto.

4.4.1 Bloque

4.4.1.1 Test Longitudinal 1

Durante este test (existe un video en donde se puede ver la evolución de la prueba) el bloque golpea en varias ocasiones con los bordes de la cubeta. En la Figura 4-21 podemos ver la respuesta de las aceleraciones obtenidas (una vez filtradas y con el offset corregido) así como el valor calculado (|acceleration – valor medio ventana|). En la siguiente figura se muestra la probabilidad de detección de cada sensor, en donde es posible apreciar que, mediante un proceso de selección de considiciones límite, es posible seleccionar valores que permitan establecer la lógica asociada al algoritmo del tipo *Double Thershold.*

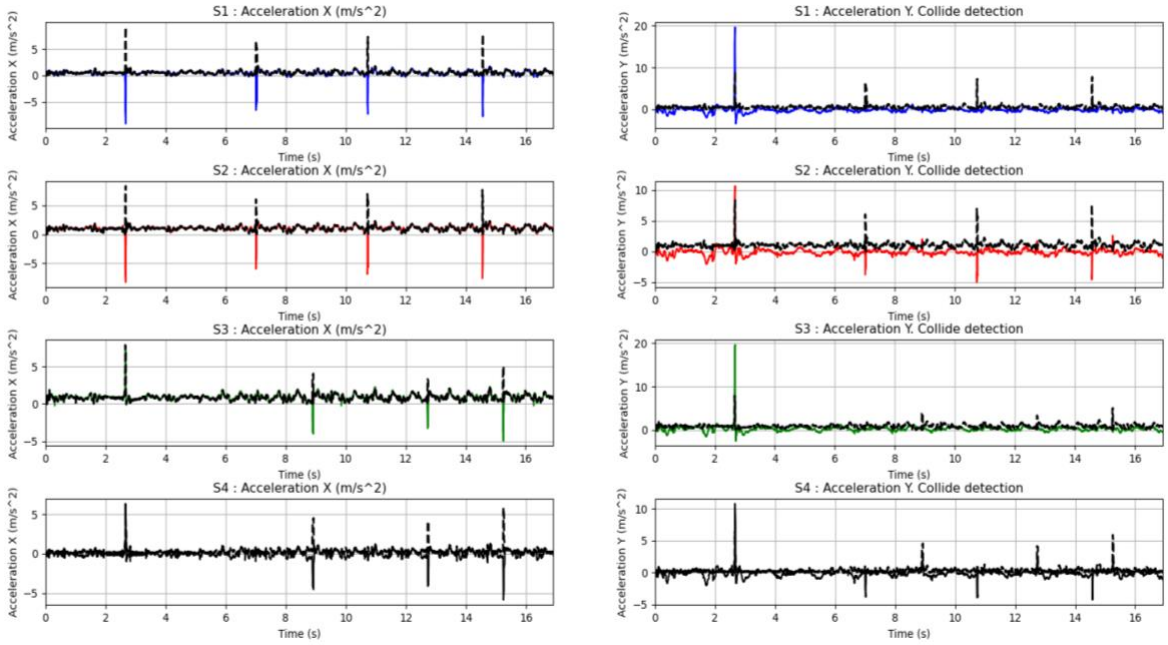

Figura 4-21: Bloque. Detección de colisiones en todos los sensores. [Fuente Propia]

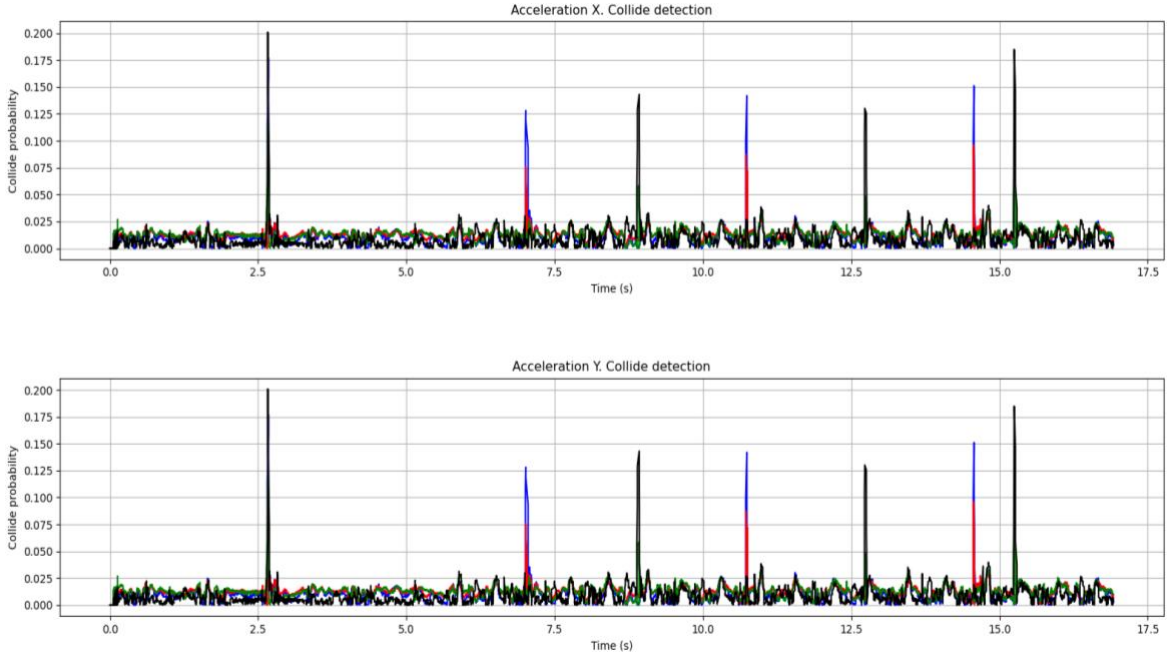

Figura 4-22: Bloque. Probabilidad de colisión obtenida en cada sensor. [Fuente Propia]

4.4.2 Barco

4.4.2.1 Test Lateral 2

En este caso vamos a presentar los resultados del sistema propuesto de detección de choques para el barco en un test en donde se producen varios golpes en proa y popa.

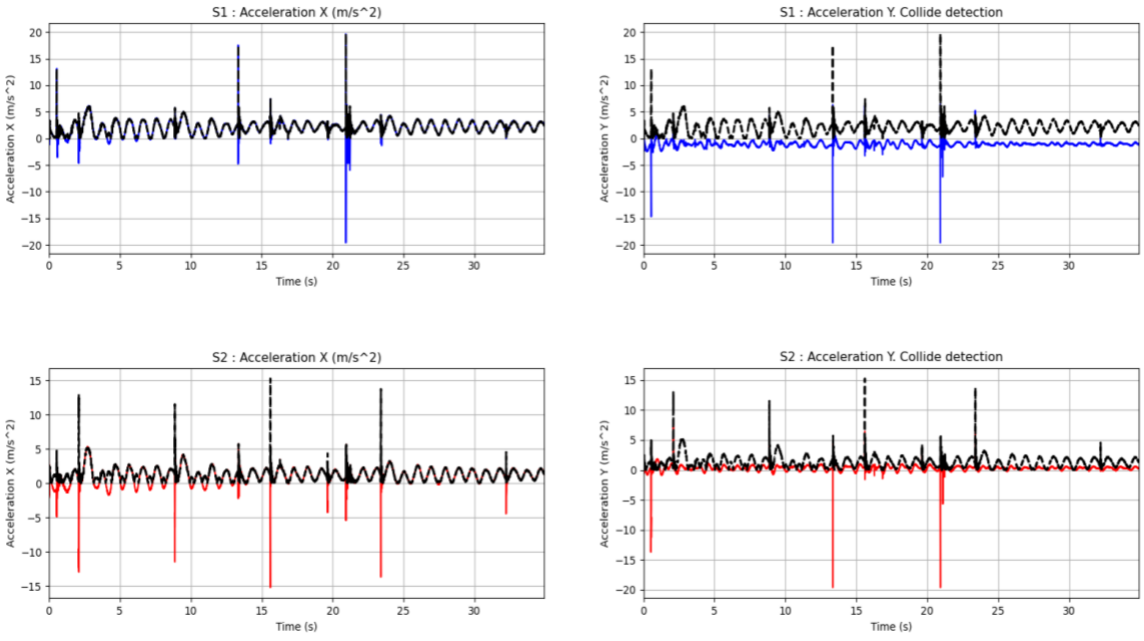

Figura 4-23: Barco. Detección de colisiones en todos los sensores. [Fuente Propia]

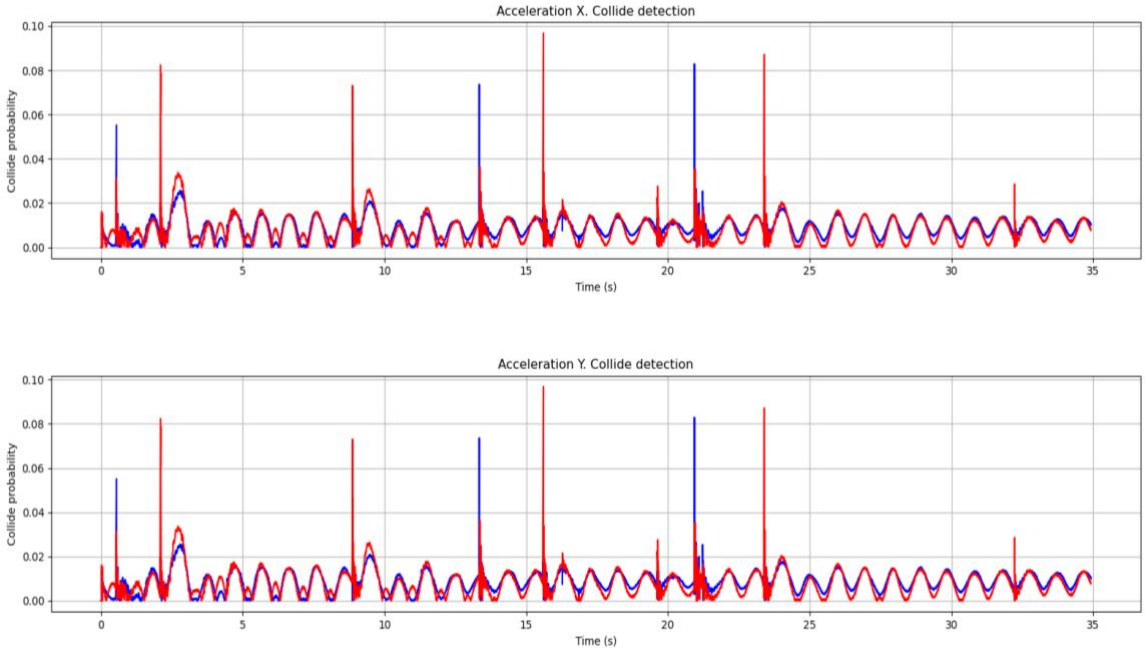

Figura 4-24: Barco. Probabilidad de colisión obtenida en cada sensor. [Fuente Propia]

A partir del concepto mostrado es posible crear un sistema que estime, no solo la presencia de un evento de choque, sino también la situación del golpe recibido a partir de los patrones de respuesta que dichos eventos impulsivos generan en la respuesta de las IMUs. Sin embargo, el desarrollo y la validación de este tipo de soluciones requiere de la toma de una serie de datos suficiente que capture con un mínimo de fiabilidad las diferentes dinámicas (continuas e impulsivas) que pueden llegar a presentarse, así como la aplicación de algoritmos más complejos, tales como estimación mediante modelos no lineales o la aplicación de técnicas de *machine learning*.

CONCLUSIONES Y LÍNEAS FUTURAS

Conclusiones y propuestas de mejora

Tras los experimentos realizados con el fin de estimar la posición de la estructura flotante a partir de sensores de bajo coste, hemos podido comprobar que los datos que nos proporcionaron las IMU tiene la calidad suficiente como para, por medio de un proceso final de fusión, generar unas estimaciones robustas. Por otra parte, es posible crear una respuesta tolerante a fallos, ya que si un sensor falla el resto de los sensores de la red pueden seguir manteniendo operativa la respuesta, y además es posible detectar el fallo del sensor al realizar análisis estadísticos comparando sus resultados con los valores obtenidos en otros sensores. Además, se hace más fácil la sustitución de cualquiera de ellos en caso de error de algún tipo.

El servidor de mensajes MQTT dio problemas en el momento en que los cuatro sensores a la vez mandaban información, ya que a veces dejábamos de ver alguno de ellos o simplemente el búfer se llenaba muy pronto y colapsaba la interfaz obligándonos a cerrarla y empezar de nuevo el proceso. Con más tiempo, se podría haber solicitado un servidor de mensajes privado y dedicado a nuestro equipo como CloudMQTT, que al ser de pago nos proporciona mayor control de los mensajes y un usuario único. Por último, podríamos haber hecho uso de otros servidores como Mosca, Aedes, HBMQTT, EMQTT, entre otros y así comprobar cuál es el más adecuado en nuestro proyecto en cuanto a prestaciones.

El software de Arduino nos agilizó el proceso de comprobar si funcionaban los sensores, puesto que con pocas líneas y de manera intuitiva nos mostraba por pantalla si estos se conectaban al WiFi y al servidor MQTT. Sin embargo, al utilizar sensores de bajo coste, debíamos conectar al ordenador cualquier sensor que fallase para comprobar que mandaba datos, proceso lento que demoraba el tiempo para realizar los experimentos. Así mismo, Python nos facilitó la legibilidad del código y en pocas líneas podíamos conseguir que los sensores realizasen una calibración, que nos sirvió también para realizar las medidas en la cubeta.

A la hora de realizar los experimentos, la cubeta no era lo suficientemente grande para que tanto la estructura elaborada con los 4 sensores como el velero pudiesen realizar los movimientos de balance, cabezada o guiñada sin interrupciones de ningún tipo. Como propuesta de mejora, se podría realizar el estudio de las propiedades dinámicas de cualquier estructura en un espacio amplio donde poder además modificar el oleaje, viento, etc.

Líneas futuras

Este trabajo significa un comienzo realmente importante en lo que puede ser un futuro totalmente diferente al que conocemos hoy en día en cuanto a sistemas de navegación. El uso de sensores repartidos por todo el buque en lugares estratégicos, así como de cámaras por donde visualizar los puntos conocidos dentro del espacio de estudio (cubeta, piscina o, en un futuro, puntos de costa) pueden suponer un gran cambio en la forma de situarnos mientras navegamos.

Como continuación más próxima a este TFG, el prototipo construido no cumplía del todo con las características que necesitábamos, ya que al no tener quilla ni otros elementos que podemos encontrar en el casco de un barco no nos proporcionaba la estabilidad que esperábamos. Es por ello, que podría construirse un prototipo en 3D en el que se incluyan los huecos necesarios para incluir los sensores. Así mismo, la sustitución de los sensores utilizados en este proyecto (Véase 3.2.1) por microprocesadores que ocupen apenas unos centímetros, facilitaría su colocación sin suponer un gran impacto económico y se conseguirían resultados más exactos.

Tras esto, sería de gran utilidad una piscina a gran escala donde poder situar el casco del barco creado para poder someterlo a situaciones reales tanto de lluvia como de viento, oleaje, entre otros. Además, podría cambiarse el agua dulce por agua salada para comprobar tanto la resistencia del casco en sí como las diferencias de flotabilidad y el envío de datos por parte de los sensores.

Por otra parte, podríamos integrar el TFG-12 que estudia el posicionamiento de objetos mediante ultrasonidos para poder situar a la embarcación dentro del área donde vamos a estudiar su comportamiento y así poder tener una forma de medir con precisión el error de posición que se obtiene mediante señores inerciales. Como se comentó al inicio de este apartado, la integración con otro tipo de sensores como cámaras, que nos sitúen respecto a una serie de puntos fijos de un entorno, o magnetómetros, que nos revelen la orientación del yaw, nos ayudará a la hora de mejorar las estimaciones relativas a la posición del barco sensorizado.

Sería deseable que este tipo de experimentos generase, a lo largo de los años, proyectos con embarcaciones de mayor tamaño y escenarios mas realistas, pudiendo, por ejemplo, culminar con la experimentación en las Lanchas de Instrucción que se emplean en la Escuela Naval. Comenzando con la toma de imágenes por parte de las cámaras de puntos fijos de la costa, así como los datos proporcionados por los sensores, nos permitirían realizar los cálculos debidos para poder situar las Lanchas y, en un futuro algo más lejano, navegar solo con este sistema de navegación o complementando al sistema de navegación por GPS.

5 BIBLIOGRAFÍA

[1] T. I. Fossen, Handbook of Marine Craft Hydrodynamics and Motion Control, First Edition., John Wiley & Sons Ltd, 2011.

[2] I. Britannica Group, "Enciclopedia Británica," [Online]. Available: https://www.britannica.com. [Accessed 12 Febrero 2021].

[3] O. M. Faltinsen, Hydrodynamics of High-Speed Marine Vehicles, Cambridge University Press, 2005.

[4] NTNU, «NTNU,» [En línea]. Available: https://www.ntnu.edu. [Último acceso: 6 Febrero 2021].

[5] W. Alcocer, «Research Gate,» [En línea]. Available: https://www.researchgate.net/figure/Notacion-para-los-movimientos-de-un-buque_fig1_331906829. [Último acceso: 16 Enero 2021].

[6] SNAME, The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin, 1950.

[7] O. M. Faltinsen, «SpringerLink,» 28 Agosto 2008. [En línea]. Available: https://link.springer.com/article/10.1007/s00773-008-0025-2. [Último acceso: 27 Febrero 2021].

[8] G. RIzzi, Relativity in rotating frames: relativistic physics in rotating reference frames, Londres: Kluwer Academic Publishers, 2004.

[9] G. F. Mínguez, «Integración Kalman de sensores inerciales INS con GPS en un UAV,» 2009.

[10] A. Gudmundsson, «Definiciones,» de *Prácticas de seguridad relativas a la estabilidad de buques pesqueros pequeños*, Roma, FAO, 2009.

[11] E. Weisstein, Euler Angles, MathWorld, 2005.

[12] T. I. Fossen, «Lecture Notes TTK 4190 Guidance and Control of Vehicles,» de *Handbook of Marine Craft Hydrodynamics and Motion Control*, John Wiley & Sons Ltd, 2011, p. 42.

[13] T. D. Science, «Alpha beta filter,» 15 Agosto 2020. [En línea]. Available: https://towardsdatascience.com/alpha-beta-filter-21d3276cf35e. [Último acceso: 4 Marzo 2021].

[14] M. C. M. Raga, «Filtro de Kalman y sus aplicaciones,» Barcelona, 2018.

[15] A. Becker, «KalmanFilter.NET,» 2018. [En línea]. Available: https://www.kalmanfilter.net/ES/default_es.aspx. [Último acceso: 3 Marzo 2021].

[16] P. Salmony, «IMU Attitude Estimation,» [En línea]. Available: http://philsal.co.uk/projects/imu-attitude-estimation. [Último acceso: 6 Marzo 2021].

[17] U. Electronics, «Unit Electronics,» [En línea]. Available: https://uelectronics.com/producto/imu-mpu6050-6-grados-de-libertad/. [Último acceso: 20 Diciembre 2020].

[18] Universidad Santo Tomás, Ingenio Magno 8, Colombia: Jotamar Ltda., 2017.

[19] V. García, Septiembre 2015. [En línea]. Available: https://www.diarioelectronicohoy.com/blog/sensor-mpu6050. [Último acceso: 14 Enero 2021].

[20] M. Rico, «Research Gate,» 2019. [En línea]. Available: https://www.researchgate.net/figure/Figura-12-Esquema-para-determinar-orientacion-y-posicion-de-una-IMU-usando-solo-los_fig1_334635865. [Último acceso: 14 Febrero 2021].

[21] D. S., «JuegosRobotica.es,» [En línea]. Available: https://juegosrobotica.es/alarma-por-correo-electronico/. [Último acceso: 13 Enero 2021].

[22] Arduino, «Arduino,» [En línea]. Available: https://www.arduino.cc. [Último acceso: 15 Enero 2021].

[23] Conda, «Conda,» 2017. [En línea]. Available: https://conda.io/en/latest/. [Último acceso: 27 Enero 2021].

[24] Á. Robledano, «OpenWebinars,» 23 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-python/. [Último acceso: 18 Enero 2021].

[25] J. M. López, «Blogthinkbig.com,» 3 Diciembre 2019. [En línea]. Available: https://blogthinkbig.com/razones-aprender-python. [Último acceso: 2020 Noviembre 25].

[26] VTK, "VTK," [Online]. Available: https://vtk.org. [Accessed 15 Febrero 2021].

[27] PyQtGraph, «PyQtGraph-Scientific Graphics and GUI Library for Python,» [En línea]. Available: http://www.pyqtgraph.org. [Último acceso: 22 Febrero 2021].

[28] L. Llamas, «Luis Llamas,» 17 Abril 2019. [En línea]. Available: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/. [Último acceso: 1 Marzo 2021].

[29] C. Mazzeo, «¿Cómo se garantiza la seguridad de los aviones?,» [En línea]. Available: https://www.revistadeacuerdo.org/2018/02/18/como-se-garantiza-la-seguridad-de-los-aviones/. [Último acceso: 22 Febrero 2021].

[30] S. Srinivasan, "Towards Data Science," 18 Abril 2018. [Online]. Available: https://towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e. [Accessed 1 Marzo 2021].

ANEXO I: CÓDIGO C (ARDUINO)

Sensor inercial en módulo ESP8266

```
    ////////////////////////// LIBRARIES LOAD //////////////////////////

    #include <ESP8266WiFi.h>  // Wifi library
    #include <PubSubClient.h>  // MQTT library to publish messages
    #include <Arduino.h>
    #include <TinyMPU6050.h>

    ////////////////////////// SENSOR CONFIGURATION //////////////////////////

    const char* sensor_id = "1"; // Sensor unique identification
uint16_t sensor_loop_delay = 0; // Time between imu data send in miliseconds

    // 0-> Listening at MQTT topic TFG/PROTOCOL waiting for one instruction. No IMU reading
or data transmision.

    // 1-> Listening at MQTT topic TFG/PROTOCOL waiting for one instruction. IMU reading and
sending data to MQTT topic TFG/IMUDATA.

    // TFG/PROTOCOL: [0]-> change to state 0; [1]-> Change to state 1; [5, parameter, value]-
>Set parameter value

    uint8_t sensor_state = 0;
    uint8_t  sensor_state1_mqtt_interval  =  500;  //  Time  between  loops  for  reading  MQTT
messages in state 1


    ////////////////////////// AUXILIAR CONFIGURATION //////////////////////////

    uint32_t start_timer;
    uint32_t interval_timer;
    uint32_t max_uint32_t = 4294967295;

    ////////////////////////// WIFI CONFIGURATION //////////////////////////

    // Wifi network name
    const char* wifi_ssid = "Rafa";
     // Wifi password
    const char* wifi_password = "rrrrrrrr";

    ////////////////////////// MQTT CONFIGURATION //////////////////////////

    char mqtt_hostname[] ="172.20.10.3"; // MQTT IP address of machine
    const char* mqtt_data_topic = "TFG/IMU"; // MQTT token to publish IMU data
    const char* mqtt_protocol_topic = "TFG/PROTOCOL"; // MQTT token to read and publish IMU
protocol
    int mqtt_port = 1883; // Default MQTT port
    String mqtt_message = ""; // MQTT message to be send o received

    ////////////////////////// IMU CONFIGURATION //////////////////////////
```

```
    // Number of decimals of imu data send to MQTT server
    const uint8_t imu_decimals = 3;



    ////////////////////////// PROJECT OBJECTS //////////////////////////

    WiFiClient wifiClient;
    PubSubClient MQTTClient(wifiClient);
    MPU6050 mpu (Wire);

    ////////////////////////SETUP AND LOOP MAIN FUNCTIONS ///////////////////////
    void setup()
    {
      // Initialize serial communication
      Serial.begin(115200);

      // Connect to MQTT server
      MQTTClient.setServer(mqtt_hostname, mqtt_port);
      MQTTClient.setCallback(onMQTTMessageReceived);
      mqttConnect();
       // Initialize IMU
      mpu.Initialize();
      }

    void loop()
    {
       if (sensor_state == 0){ // 0-> Listening at MQTT topic TFG/PROTOCOL waiting for one
instruction. No sensor reading or data transmision.
          MQTTClient.loop(); // Check in a MQTT protocol message has arrived
             }else if (sensor_state == 1){ // 1-> Listening at MQTT topic TFG/PROTOCOL
             waiting for one instruction. Sensor reading and sending data to MQTT topic
             TFG/DATA.
          mpu.Execute();
          interval_timer = computeIntervalMicroSenconds(start_timer, micros());
          mqtt_message = String(sensor_id)+ "," + String(interval_timer/1e6, 3) + "," +
String(mpu.GetAccX(), imu_decimals) + "," + String(mpu.GetAccY(), imu_decimals);
          mqtt_message = mqtt_message + "," + String(mpu.GetAccZ(), imu_decimals);
          mqtt_message = mqtt_message + "," + String(mpu.GetGyroX(), imu_decimals) + "," +
String(mpu.GetGyroY(), imu_decimals);
          mqtt_message = mqtt_message + "," + String(mpu.GetGyroZ(), imu_decimals);
          mqtt_message = mqtt_message + "," + String(mpu.GetAngX(), imu_decimals) + "," +
String(mpu.GetAngY(), imu_decimals)+ "," + String(mpu.GetAngZ(), imu_decimals);
          //Serial.println(mqtt_message);
          mqttSendMessage();
          MQTTClient.loop();
          delay(50);
             }else if (sensor_state == 2){
        Serial.println("====================================");
        Serial.println("Starting calibration...");
        mpu.Calibrate();
        Serial.println("Calibration complete!");
        // Shows offsets
        Serial.println("--- Offsets:");
        Serial.print("GyroX Offset = ");
        Serial.println(mpu.GetGyroXOffset());
        Serial.print("GyroY Offset = ");
        Serial.println(mpu.GetGyroYOffset());
        Serial.print("GyroZ Offset = ");
        Serial.println(mpu.GetGyroZOffset());
```

```
        mqtt_message = String(sensor_id)+ "," + String(interval_timer/1e6, 3) + "," +
String(mpu.GetGyroXOffset(),   imu_decimals)   +   ","   +   String(mpu.GetGyroYOffset(),
imu_decimals);
        mqtt_message = mqtt_message + "," + String(mpu.GetGyroZOffset(), imu_decimals);
        mqttSendMessage();
        sensor_state = 0;
              }
      if (sensor_loop_delay > 0){
        delay(sensor_loop_delay);
       }
    }

    ////////////////////////// COMPUTE FUNCTIONS //////////////////////////

    uint32_t computeIntervalMicroSenconds(uint32_t start_timer, uint32_t stop_timer){
        uint32_t interval_timer;
        if (stop_timer > start_timer){
          interval_timer = stop_timer - start_timer;
         }else{
          interval_timer = max_uint32_t - start_timer + stop_timer;
                    }
        return interval_timer;
    }

    ////////////////////////// MQTT FUNCTIONS //////////////////////////

    void onMQTTMessageReceived(char* topic, byte* payload, unsigned int length) {
      Serial.println("..................................................");
      Serial.print("Message arrived [");
      Serial.print(topic);
      Serial.print("] ");
      for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
      }
      Serial.println();
      sensor_state = (uint8_t) payload[0] - '0';
      Serial.println("sensor_state:" + String(sensor_state));
      if (sensor_state == 1) {
        start_timer = micros();
      }
    }
     void mqttSendMessage()
    {
      char attributes[1000];
      mqtt_message.toCharArray( attributes, 200 );
      MQTTClient.publish(mqtt_data_topic, attributes);
    }
     void mqttConnect()
    {
      while (!MQTTClient.connected()) {
        if ( WiFi.status() != WL_CONNECTED) {
          Serial.println("Connecting to Wifi");
          WiFi.begin(wifi_ssid, wifi_password);
          while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
          }
          Serial.println(".");
          Serial.println("Connected to Wifi !!");
```

```
      }
      Serial.println("Connecting to MQTT server");
      if (MQTTClient.connect(sensor_id, mqtt_data_topic, NULL) )
      {
        Serial.println("Connected to MQTT server !!" );
        MQTTClient.subscribe(mqtt_protocol_topic);
      }
      else {
        Serial.println( " ... retrying in 1 second ... " );
        delay( 1000 );
      }
    }
  }
```

ANEXO II: CÓDIGO PYTHON

Script de toma de datos de calibración y tests dinámicos

```
# -*- coding: utf-8 -*-
import traceback, time, sys
sys.path.insert(0, 'C:\\Users\\jagprieto\\Desktop\\JUNE')
from MQTT import MQTT
import numpy as np

# Ground truth
ground_acc_x = 0.0
ground_acc_y = 0.0
ground_acc_z = 9.80665
ground_gyro_x = 0.0
ground_gyro_y = 0.0
ground_gyro_z = 0.0
imu_data_size = 20000

file_data_name = "2021_03_02_ship_calibration_ground.csv"
print('file_data_name=', file_data_name)
calibration_imu_data = np.zeros(shape=(imu_data_size, 14), dtype=float) # id, time, acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z, ground_acc_x, ground_acc_y, ground_acc_z, ground_gyro_x, ground_gyro_y,ground_gyro_z

MQTTObj =  MQTT()
MQTTObj.connect('192.168.43.164', 1883)
MQTTObj.subscribe([("TFG/IMU",1), ("TFG/PROTOCOL",1), ("TFG/ENVIROMENT",1)])
MQTTObj.publish("TFG/PROTOCOL", "1")

counter = 0
start_time = time.time()
while counter < imu_data_size:
   imu_data_txt = MQTTObj.read()
   if not imu_data_txt is None:
      imu_data  = imu_data_txt.split(",")
      if len(imu_data) == 11:
         if (counter % 100) ==  0:
            print ('counter=', counter)
         data_time = time.time() - start_time

         calibration_imu_data[counter, 0] = float(imu_data[0])
         calibration_imu_data[counter, 1] = float(data_time)
         calibration_imu_data[counter, 2] = float(imu_data[2]) # g
         calibration_imu_data[counter, 3] = float(imu_data[3])
         calibration_imu_data[counter, 4] = float(imu_data[4])
         calibration_imu_data[counter, 5] = float(imu_data[5]) # º/s
         calibration_imu_data[counter, 6] = float(imu_data[6])
         calibration_imu_data[counter, 7] = float(imu_data[7])

         calibration_imu_data[counter, 8] = ground_acc_x
         calibration_imu_data[counter, 9] = ground_acc_y
         calibration_imu_data[counter, 10] = ground_acc_z
         calibration_imu_data[counter, 11] = ground_gyro_x
         calibration_imu_data[counter, 12] = ground_gyro_y
         calibration_imu_data[counter, 13] = ground_gyro_z

         counter +=1
```

```
MQTTObj.publish("TFG/PROTOCOL", "0")
np.savetxt(file_data_name, calibration_imu_data, delimiter=",")
```

Script de toma de análisis, filtrado y generación de gráficas de resultados

```
import numpy as np
import matplotlib.pyplot as plt
from readFileData import  readFileData
from filterSensorData import filterSensorData
from estimateAttitude import estimateAttitude
from plotSensorAttitude import  plotSensorAttitude
from plotSensorHistograms import  plotSensorHistograms
from showCalibrationData import  showCalibrationData
from plotSensorsData import  plotSensorsData
from plotGlobalHistograms import plotGlobalHistograms
from plotSensorAccelerations import plotSensorAccelerations
from plotSensorGyros import plotSensorGyros
from plotSensorsAttitudes import  plotSensorsAttitudes
from filterSensorsData import  filterSensorsData
from detectCollides import  detectCollides

plot_global_histograms = False
plot_unfiltered_data = True
plot_filtered_data = True
show_calibration_data = False

simulation_type = 0 # 0:ship, 1:block
file_data_name = "2021_03_02_ship_calibration_ground.csv"


print('file_data_name=', file_data_name)

sensor_colors = ['b', 'r', 'g', 'k']
sensor_colors_filtered = ['y', 'y', 'y', 'y']
gravity = 9.80665
filter_data = {'acc_x_g': 0.2, 'acc_x_h': 0.02, 'acc_y_g': 0.2, 'acc_y_h': 0.02, 'acc_z_g': 0.2, 'acc_z_h': 0.02,
          'gyro_x_g': 0.2, 'gyro_x_h': 0.02, 'gyro_y_g': 0.2, 'gyro_y_h': 0.02, 'gyro_z_g': 0.2, 'gyro_z_h': 0.02}
block_calibration_bias_data = []
block_calibration_bias_data.append([0.506559, -0.068776,  9.196736 - gravity, 2.007802, -0.29176,  -1.369547])
block_calibration_bias_data.append([0.996004, -0.017493,  9.163471 - gravity, -7.864416, -7.984349,  0.780253])
block_calibration_bias_data.append([0.896217,  0.154769,  8.822748 - gravity, -4.659935, -8.154066,  0.629115])
block_calibration_bias_data.append([-0.068339, -0.257984,  7.842505 - gravity, -0.848788, -1.358859, -2.704832])
block_calibration_noise_data = []
block_calibration_noise_data.append([0.030387, 0.030614, 0.047509, 0.136699, 0.185545, 0.208693])
block_calibration_noise_data.append([0.031595, 0.028858, 0.04105,  0.207361, 0.203021, 0.221259])
block_calibration_noise_data.append([0.030249, 0.032747, 0.050443, 0.205358, 0.207641, 0.182644])
block_calibration_noise_data.append([0.033905, 0.032539, 0.049021, 0.201353, 0.160368, 0.160808])

ship_calibration_bias_data = []
ship_calibration_bias_data.append([0.374999, -0.239635,  8.885915 - gravity,  1.3755,   -2.224935, -5.643531])
ship_calibration_bias_data.append([0.209884, -0.085617, 10.2304 - gravity,   -3.927265, -0.555641, -3.853688])
ship_calibration_noise_data = []
ship_calibration_noise_data.append([0.032443, 0.032042, 0.04604,  0.200847, 0.19774,  0.160153])
ship_calibration_noise_data.append([0.032458, 0.03084,  0.042666, 0.196109, 0.20157,  0.196449])

if simulation_type == 1:
   calibration_bias_data = block_calibration_bias_data
   calibration_noise_data = block_calibration_noise_data
else:
   calibration_bias_data = ship_calibration_bias_data
   calibration_noise_data = ship_calibration_noise_data
```

```
(sensor_data_dict, statistics_data) = readFileData(file_data_name, gravity)
num_sensors = len(sensor_data_dict)



##################### SHOW CALIBRATION DATA BIAS ####################
if show_calibration_data:
   showCalibrationData(sensor_data_dict, statistics_data)

#################### PLOT UNFILTERED DATA FOR EACH SENSOR ####################
if plot_unfiltered_data:
   plotSensorsData(sensor_data_dict, statistics_data, sensor_colors)

#################### PLOT UNFILTERED DATA FOR EACH SENSOR ####################
if plot_filtered_data:
   (sensor_data_dict_filtered, statistics_data_dict_filtered) = filterSensorsData(sensor_data_dict, filter_data, calibration_bias_data)
   if plot_unfiltered_data:
      plotSensorsData(sensor_data_dict_filtered, statistics_data_dict_filtered, sensor_colors_filtered)
   else:
      plotSensorsData(sensor_data_dict_filtered, statistics_data_dict_filtered, sensor_colors)

#################### PLOT GLOBAL HISTROGAMS ####################
if plot_global_histograms:
   plotGlobalHistograms(sensor_data_dict, statistics_data, sensor_colors)
plt.show()
```

Funciones auxiliares empleadas por el script de análisis de resultados

```
import numpy as np

def detectCollides(sensor_id, sensor_data):
    sensor_data_size = sensor_data.shape[0]
    time_data = sensor_data[:,1]
    acc_x_collides = np.zeros(shape=(1,sensor_data_size))
    acc_y_collides = np.zeros(shape=(1,sensor_data_size))
    acc_x_window_data = np.zeros(shape=(1,sensor_data_size))
    acc_y_window_data = np.zeros(shape=(1,sensor_data_size))
    windows_size = 250
    for index in range(sensor_data_size):
        if index < sensor_data_size - 1:
            dt = time_data[index+1] - time_data[index]

        if index < windows_size:
            acc_x_window_data[0, index] = sensor_data[index, 2]
            acc_y_window_data[0, index] = sensor_data[index, 3]
        else:
            acc_x_window_data[0, 0:windows_size-1] = acc_x_window_data[0, 1:windows_size]
            acc_x_window_data[0, windows_size-1] = sensor_data[index, 2]
            acc_y_window_data[0, 0:windows_size-1] = acc_y_window_data[0, 1:windows_size]
            acc_y_window_data[0, windows_size-1] = sensor_data[index, 3]

            acc_x_window_mean = np.mean(acc_x_window_data[0, :])
            acc_x_window_std = np.std(acc_x_window_data[0, :])
            acc_y_window_mean = np.mean(acc_y_window_data[0, :])
            acc_y_window_std = np.std(acc_y_window_data[0, :])

    return (acc_x_collides, acc_y_collides)
```

```
import numpy as np
from filterpy.gh import GHFilter

def estimateAttitude(sensor_data, noise_data):
    sensor_data_size = sensor_data.shape[0]
    time_data = sensor_data[:,1]

    # Initialise matrices and variables
    C = np.array([[1, 0, 0, 0], [0, 0, 1, 0]])
    P = np.eye(4)
    Q = np.eye(4)
    R = np.eye(2)
    R [0,0] = noise_data[3]
    R [1,1] = noise_data[4]
    state_estimate = np.array([[0], [0], [0], [0]])
    roll_hat = 0.0 # Roll
    pitch_hat = 0.0 # Pitch
    roll_offset = 0.0
    pitch_offset = 0.0
    state_estimated_data = np.zeros(shape=(sensor_data_size,2)) # Roll, Pitch
    for index in range(sensor_data_size):
        if index < sensor_data_size - 1:
            dt = time_data[index+1] - time_data[index]
        ax = sensor_data[index,2]
        ay = sensor_data[index,3]
        az = sensor_data[index,4]

        # Read angular velocities (rad/s)
        p = sensor_data[index,5] * np.pi / 180.0
        q = sensor_data[index,6] * np.pi / 180.0
        r = sensor_data[index,7] * np.pi / 180.0

        # Calculate aceleration angles
        roll_acc = np.arctan2(ay, np.sqrt(ax ** 2.0 + az ** 2.0))
        pitch_acc = np.arctan2(-ax, np.sqrt(ay ** 2.0 + az ** 2.0))
        roll_acc -= roll_offset
        pitch_acc -= pitch_offset

        # Calculate Euler angles derivaties
        roll_dot = p + np.sin(roll_hat) * np.tan(pitch_hat) * q + np.cos(roll_hat) * np.tan(pitch_hat) * r
        pitch_dot = np.cos(roll_hat) * q - np.sin(roll_hat) * r

        # Kalman filter
        A = np.array([[1, -dt, 0, 0], [0, 1, 0, 0], [0, 0, 1, -dt], [0, 0, 0, 1]])
        B = np.array([[dt, 0], [0, 0], [0, dt], [0, 0]])
        gyro_input = np.array([[roll_dot], [pitch_dot]])
        state_estimate = A.dot(state_estimate) + B.dot(gyro_input)
        P = A.dot(P.dot(np.transpose(A))) + Q
        measurement = np.array([[roll_acc], [pitch_acc]])
        y_tilde = measurement - C.dot(state_estimate)
        S = R + C.dot(P.dot(np.transpose(C)))
        K = P.dot(np.transpose(C).dot(np.linalg.inv(S)))
        state_estimate = state_estimate + K.dot(y_tilde)
        P = (np.eye(4) - K.dot(C)).dot(P)

        roll_hat = state_estimate[0][0]
        pitch_hat = state_estimate[2][0]

        state_estimated_data[index, 0] = roll_hat * 180.0 / np.pi
        state_estimated_data[index, 1] = pitch_hat * 180.0 / np.pi
```

```
    return state_estimated_data
def filterGH(time, data, x0, dx, dt, g, h):
    x_est = x0
    results = []
    index = 0
    len_data = len(data)
    for z in data:
        # prediction step
        x_pred = x_est + (dx*dt)

        # update step
        residual = z - x_pred
        dx = dx + h * (residual) / dt
        x_est = x_pred + g * residual
        results.append(x_est)
    return np.array(results)
```

```
import numpy as np
from filterpy.gh import GHFilter

def filterSensorData(sensor_data, filter_data, bias_data = None):

    sensor_data_size = sensor_data.shape[0]
    time_data = sensor_data[:,1]

    ############################## GH FILTER ##############################
    sensor_data_filtered_gh = np.copy(sensor_data)

    dt = np.mean(time_data[1:]-time_data[0:-1])
    acc_x_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,2], dx=0.0, dt=dt, g=filter_data['acc_x_g'], h=filter_data['acc_x_h'])
    acc_y_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,3], dx=0.0, dt=dt, g=filter_data['acc_y_g'], h=filter_data['acc_z_h'])
    acc_z_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,4], dx=0.0, dt=dt, g=filter_data['acc_z_g'], h=filter_data['acc_z_h'])
    gyro_x_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,5], dx=0.0, dt=dt, g=filter_data['gyro_x_g'], h=filter_data['gyro_x_h'])
    gyro_y_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,6], dx=0.0, dt=dt, g=filter_data['gyro_y_g'], h=filter_data['gyro_z_h'])
    gyro_z_filter_gh = GHFilter(x= sensor_data_filtered_gh[0,7], dx=0.0, dt=dt, g=filter_data['gyro_z_g'], h=filter_data['gyro_z_h'])

    for index in range(sensor_data_size):
        if index < sensor_data_size - 1:
            dt = time_data[index+1] - time_data[index]
        acc_x_filter_gh.update(z=sensor_data[index,2])
        sensor_data_filtered_gh[index,2] = acc_x_filter_gh.x
        acc_y_filter_gh.update(z=sensor_data[index,3])
        sensor_data_filtered_gh[index,3] = acc_y_filter_gh.x
        acc_z_filter_gh.update(z=sensor_data[index,4])
        sensor_data_filtered_gh[index,4] = acc_z_filter_gh.x
        gyro_x_filter_gh.update(z=sensor_data[index,5])
        sensor_data_filtered_gh[index,5] = gyro_x_filter_gh.x
        gyro_y_filter_gh.update(z=sensor_data[index,6])
        sensor_data_filtered_gh[index,6] = gyro_y_filter_gh.x
        gyro_z_filter_gh.update(z=sensor_data[index,7])
        sensor_data_filtered_gh[index,7] = gyro_z_filter_gh.x

    if not bias_data is None:
        sensor_data_filtered_gh[:,2] -= bias_data[0]
        sensor_data_filtered_gh[:,3] -= bias_data[1]
        sensor_data_filtered_gh[:,4] -= bias_data[2]
        sensor_data_filtered_gh[:,5] -= bias_data[3]
        sensor_data_filtered_gh[:,6] -= bias_data[4]
        sensor_data_filtered_gh[:,7] -= bias_data[5]
    statistics_data_filtered_gh = np.zeros(shape=(1,12))

    statistics_data_filtered_gh[0, 0] = round(np.mean(sensor_data_filtered_gh[:,2]), 6) # acc_x mean (m/s^2)
    statistics_data_filtered_gh[0, 1] = round(np.std(sensor_data_filtered_gh[:,2]), 6) # acc_x std (m/s^2)
    statistics_data_filtered_gh[0, 2] = round(np.mean(sensor_data_filtered_gh[:,3]), 6) # acc_y mean (m/s^2)
    statistics_data_filtered_gh[0, 3] = round(np.std(sensor_data_filtered_gh[:,3]), 6) # acc_y std (m/s^2)
    statistics_data_filtered_gh[0, 4] = round(np.mean(sensor_data_filtered_gh[:,4]), 6) # acc_z mean (m/s^2)
    statistics_data_filtered_gh[0, 5] = round(np.std(sensor_data_filtered_gh[:,4]), 6) # acc_z std (m/s^2)
    statistics_data_filtered_gh[0, 6] = round(np.std(sensor_data_filtered_gh[:,5]), 6) # gyro_x mean (¬∫/s)
    statistics_data_filtered_gh[0, 7] = round(np.mean(sensor_data_filtered_gh[:,5]), 6) # gyro_x std (¬∫/s)
    statistics_data_filtered_gh[0, 8] = round(np.std(sensor_data_filtered_gh[:,6]), 6) # gyro_y mean (¬∫/s)
    statistics_data_filtered_gh[0, 9] = round(np.std(sensor_data_filtered_gh[:,6]), 6) # gyro_y std (¬∫/s)
```

```
statistics_data_filtered_gh[0, 10] = round(np.mean(sensor_data_filtered_gh[:,7]), 6) # gyro_z mean (¬∫/s)
statistics_data_filtered_gh[0, 11] = round(np.std(sensor_data_filtered_gh[:,7]), 6) # gyro_z std (¬∫/s)

return (sensor_data_filtered_gh, statistics_data_filtered_gh)
```

```
import numpy as np
from filterSensorData import filterSensorData

def filterSensorsData(sensor_data_dict, filter_data, calibration_bias_data):
    num_sensors = len(sensor_data_dict)
    sensor_data_dict_filtered = {}

    statistics_data_dict_filtered = np.zeros(shape = (4,12))
    calibration_bias_data
    for sensor_id in range(num_sensors):
        (sensor_data_filtered_gh, statistics_data_filtered_gh) = filterSensorData(sensor_data = sensor_data_dict[sensor_id], filter_data = filter_data, bias_data = calibration_bias_data[sensor_id])
        sensor_data_dict_filtered[sensor_id] = sensor_data_filtered_gh
        statistics_data_dict_filtered[sensor_id,:] = statistics_data_filtered_gh

    return (sensor_data_dict_filtered, statistics_data_dict_filtered)
```

```
import numpy as np
from plotSensorHistograms import  plotSensorHistograms

def plotGlobalHistograms(sensor_data_dict, statistics_data, sensor_colors):
    num_sensors = len(sensor_data_dict)
    figure_id = 200
    for sensor_id in range(num_sensors):
        plotSensorHistograms(sensor_id=sensor_id, sensor_data = sensor_data_dict[sensor_id], statistics_data = statistics_data[sensor_id,:],
                             figure_id = figure_id, color = sensor_colors[sensor_id], show_statistics_data = False)
```

```
import numpy as np
import matplotlib.pyplot as plt

def plotSensorAccelerations(sensor_id, sensor_data, statistics_data, figure_id, color):
    max_time = np.max(sensor_data[:,1])
    plt.figure(figure_id)
    plt.subplot(311)
    plt.plot(sensor_data[:,1], sensor_data[:,2],  str('-' + color), label='Acceleration X')
    plt.title(str('S' + str(sensor_id+1)) + ' ACC X')
    plt.xlabel('Time (s)')
    plt.ylabel('Acceleration X (m/s^2)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
    plt.subplot(312)
    plt.plot(sensor_data[:,1], sensor_data[:,3],  str('-' + color), label='Acceleration Y')
    plt.title(str('S' + str(sensor_id+1)) + ' ACC Y')
    plt.xlabel('Time (s)')
    plt.ylabel('Acceleration Y (m/s^2)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
    plt.subplot(313)
    plt.plot(sensor_data[:,1], sensor_data[:,4],  str('-' + color), label='Acceleration Z')
    plt.title(str('S' + str(sensor_id+1)) + ' ACC Z')
    plt.xlabel('Time (s)')
    plt.ylabel('Acceleration Z (m/s^2)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
```

```
import numpy as np
import matplotlib.pyplot as plt

def plotSensorAttitude(sensor_id, time_data, attitude_data, figure_id, color, show_title = True):
    max_time = np.max(time_data)
    plt.figure(figure_id)
    plt.subplot(211)
    plt.plot(time_data, attitude_data[:,0],  str('-' + color), label='Roll angle')
    if show_title:
        plt.title(str('S' + str(sensor_id)) + ' ROLL angle')
    plt.xlabel('Time (s)')
    plt.ylabel('Roll angle (¬∫)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
    plt.subplot(212)
    plt.plot(time_data, attitude_data[:,1],  str('-' + color), label='Pitch angle')
    if show_title:
        plt.title(str('S' + str(sensor_id)) + ' PITCH angle')
    plt.xlabel('Time (s)')
    plt.ylabel('Pitch angle (¬∫)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
```

```
import numpy as np
import matplotlib.pyplot as plt

def plotSensorGyros(sensor_id, sensor_data, statistics_data, figure_id, color):
    max_time = np.max(sensor_data[:,1])
    plt.figure(figure_id)
    plt.subplot(311)
    plt.plot(sensor_data[:,1], sensor_data[:,5],  str('-' + color), label='Roll rate')
    plt.title(str('S' + str(sensor_id+1)) + ' ROLL RATE')
    plt.xlabel('Time (s)')
    plt.ylabel('Roll rate (¬∫/s)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
    plt.subplot(312)
    plt.plot(sensor_data[:,1], sensor_data[:,6],  str('-' + color), label='Pitch rate')
    plt.title(str('S' + str(sensor_id+1)) + ' PITCH RATE')
    plt.xlabel('Time (s)')
    plt.ylabel('Pitch rate (¬∫/s)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
    plt.subplot(313)
    plt.plot(sensor_data[:,1], sensor_data[:,7],  str('-' + color), label='Yaw rate')
    plt.title(str('S' + str(sensor_id+1)) + ' YAW RATE')
    plt.xlabel('Time (s)')
    plt.ylabel('Yaw rate (¬∫/s)')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
    plt.xlim([0, max_time])
```

```
import numpy as np
import matplotlib.pyplot as plt

def plotSensorHistograms(sensor_id, sensor_data, statistics_data, figure_id, color, show_statistics_data = True, sensor_label = None):
    if sensor_label is None:
        sensor_label = str('S' + str(sensor_id))
    plt.figure(figure_id)
    plt.subplot(231)

    if show_statistics_data:
        plt.title(str('S' + str(sensor_id+1) + ' : ACC X ' + r'$\mu=$' + str(statistics_data[0]) + r' $\sigma$=' + str(statistics_data[1])))
        plt.hist(sensor_data[:,2],  facecolor=color, alpha=0.75, label='Acceleration X') # acc_x
    else:
        plt.hist(sensor_data[:,2],  facecolor=color, alpha=0.75, label=sensor_label) # Si
    plt.legend()
    plt.xlabel('Acceleration X (m/s^2)')
    plt.ylabel('Frequeny')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)

    plt.subplot(232)
    if show_statistics_data:
        plt.title(str('S' + str(sensor_id+1) + ' : ACC Y ' + r'$\mu=$' + str(statistics_data[2]) + r' $\sigma$=' + str(statistics_data[3])))
        plt.hist(sensor_data[:,3],  facecolor=color, alpha=0.75, label='Acceleration y') # acc_Y
    else:
        plt.hist(sensor_data[:,3],  facecolor=color, alpha=0.75, label=sensor_label) # Si
    plt.legend()
    plt.xlabel('Acceleration Y (m/s^2)')
    plt.ylabel('Frequeny')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)

    plt.subplot(233)
    if show_statistics_data:
        plt.title(str('S' + str(sensor_id+1) + ' : ACC Z ' + r'$\mu=$' + str(statistics_data[4]) + r' $\sigma$=' + str(statistics_data[5])))
        plt.hist(sensor_data[:,4],  facecolor=color, alpha=0.75, label='Acceleration z') # acc_Y
    else:
        plt.hist(sensor_data[:,4],  facecolor=color, alpha=0.75, label=sensor_label) # Si
    plt.legend()
    plt.xlabel('Acceleration Z (m/s^2)')
    plt.ylabel('Frequeny')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)


    plt.subplot(234)
    if show_statistics_data:
        plt.title(str('S' + str(sensor_id+1) + ' : ROLL RATE ' + r'$\mu=$' + str(statistics_data[6]) + r' $\sigma$=' + str(statistics_data[7])))
        plt.hist(sensor_data[:,5],  facecolor=color, alpha=0.75, label='Roll rate (¬∫/s)') # gyro_x
    else:
        plt.hist(sensor_data[:,5],  facecolor=color, alpha=0.75, label=sensor_label) # Si
    plt.legend()
    plt.xlabel('Roll rate (¬∫/s)')
    plt.ylabel('Frequeny')
    plt.grid(True)
    plt.subplots_adjust(hspace=0.5)
```

```
        plt.subplot(235)
        if show_statistics_data:
            plt.title(str('S' + str(sensor_id+1) + ' : PITCH RATE ' + r'$\mu=$' + str(statistics_data[8]) + r' $\sigma$=' + str(statistics_data[9])))
            plt.hist(sensor_data[:,6],   facecolor=color, alpha=0.75, label='Pitch rate (¬∫/s)') # gyro_x
        else:
            plt.hist(sensor_data[:,6],   facecolor=color, alpha=0.75, label=sensor_label) # Si
        plt.legend()
        plt.xlabel('Pitch rate (¬∫/s)')
        plt.ylabel('Frequeny')
        plt.grid(True)
        plt.subplots_adjust(hspace=0.5)

        plt.subplot(236)
        if show_statistics_data:
            plt.title(str('S' + str(sensor_id+1) + ' : YAW RATE ' + r'$\mu=$' + str(statistics_data[10]) + r' $\sigma$=' + str(statistics_data[11])))
            plt.hist(sensor_data[:,7],   facecolor=color, alpha=0.75, label='Yaw rate (¬∫/s)') # gyro_x
        else:
            plt.hist(sensor_data[:,7],   facecolor=color, alpha=0.75, label=sensor_label)
        plt.legend()
        plt.xlabel('Yaw rate (¬∫/s)')
        plt.ylabel('Frequeny')
        plt.grid(True)
        plt.subplots_adjust(hspace=0.5)
```

```
import numpy as np
from plotSensorAttitude import  plotSensorAttitude
from estimateAttitude import estimateAttitude
from filterSensorData import filterSensorData

def plotSensorsAttitudes(sensor_data_dict, calibration_bias_data, calibration_noise_data, filter_data, sensor_colors):
    num_sensors = len(sensor_data_dict)
    figure_id = 500
    for sensor_id in range(num_sensors):
        sensor_data =  sensor_data_dict[sensor_id]
        time_data = sensor_data[:,1]
        (sensor_data_filtered_gh, statistics_data_filtered_gh) = filterSensorData(sensor_data = sensor_data, filter_data = filter_data, bias_data = calibration_bias_data[sensor_id])
        sensor_data_attitude = estimateAttitude(sensor_data = sensor_data_filtered_gh, noise_data = calibration_noise_data[sensor_id])
        plotSensorAttitude(sensor_id, time_data, sensor_data_attitude, figure_id, sensor_colors[sensor_id], False)
```

```
import numpy as np
from plotSensorHistograms import  plotSensorHistograms
from plotSensorAccelerations import plotSensorAccelerations
from plotSensorGyros import plotSensorGyros

def plotSensorsData(sensor_data_dict, statistics_data, sensor_colors):
    num_sensors = len(sensor_data_dict)
    figure_id = 50
    for sensor_id in range(num_sensors):
        plotSensorHistograms(sensor_id = sensor_id, sensor_data = sensor_data_dict[sensor_id], statistics_data = statistics_data[sensor_id,:],
                             figure_id = figure_id, color = sensor_colors[sensor_id], show_statistics_data = True)
        figure_id += 1
        plotSensorAccelerations(sensor_id = sensor_id, sensor_data = sensor_data_dict[sensor_id], statistics_data = statistics_data[sensor_id,:],
                                figure_id = figure_id, color = sensor_colors[sensor_id])
        figure_id += 1
        plotSensorGyros(sensor_id = sensor_id, sensor_data = sensor_data_dict[sensor_id], statistics_data = statistics_data[sensor_id,:],
                             figure_id = figure_id, color = sensor_colors[sensor_id])
        figure_id += 1

import numpy as np
from numpy import genfromtxt

def readFileData(file_data_name, gravity):
    sensors_data = genfromtxt(file_data_name, delimiter=',')
    sensors_id = np.unique(sensors_data[:,0])
    num_sensors = len(sensors_id)
    sensor_data_dict = {}
    statistics_data = np.zeros(shape = (4,12))
    for i in range(num_sensors):
        sensor_data = sensors_data[sensors_data[:,0] == i+1]
        sensor_data[:,2] = sensor_data[:,2]*gravity
        acc_x = sensor_data[:,2]
        col = 0
        statistics_data[i,col] =  round(np.mean(acc_x), 6) # acc_x mean (m/s^2)
        col = 1
        statistics_data[i,col] =  round(np.std(acc_x), 6) # acc_x std (m/s^2)
        sensor_data[:,3] = sensor_data[:,3]*gravity

        acc_y = sensor_data[:,3]
        col = 2
        statistics_data[i,col] =  round(np.mean(acc_y), 6) # acc_y mean (m/s^2)
        col = 3
        statistics_data[i,col] = round(np.std(acc_y), 6) # acc_y std (m/s^2)
        sensor_data[:,4] = sensor_data[:,4]*gravity

        acc_z = sensor_data[:,4]
        col = 4
        statistics_data[i,col] =  round(np.mean(acc_z), 6) # acc_z mean (m/s^2)
        col = 5
        statistics_data[i,col] =  round(np.std(acc_z), 6) # acc_z std (m/s^2)

        gyro_x = sensor_data[:,5]
        col = 6
        statistics_data[i,col] =  round(np.mean(gyro_x), 6       col = 7
        statistics_data[i,col] =  round(np.std(gyro_x), 6)
        gyro_y = sensor_data[:,6]
        col = 8
```

```
        statistics_data[i,col] =  round(np.mean(gyro_y), 6)       col = 9
        statistics_data[i,col] =  round(np.std(gyro_y), 6)
        gyro_z = sensor_data[:,7]
        col = 10
        statistics_data[i,col] =  round(np.mean(gyro_z), 6)       col = 11
        statistics_data[i,col] =  round(np.std(gyro_z), 6)
        sensor_data_dict[i] = sensor_data
    return (sensor_data_dict, statistics_data)
```

```
import numpy as np

def showCalibrationData(sensor_data_dict, statistics_data):
    num_sensors = len(sensor_data_dict)
    calibration_bias_data = np.zeros(shape=(num_sensors,6)) # acc_x_bias, acc_y_bias, acc_z_bias, gyro_x_bias,
gyro_y_bias, gyro_z_bias
    calibration_noise_data = np.zeros(shape=(num_sensors,6)) # acc_x_std, acc_y_std, acc_z_std, gyro_x_std,
gyro_y_std, gyro_z_bias
    for sensor_id in range(num_sensors):
        sensor_data =  sensor_data_dict[sensor_id]
        calibration_bias_data[sensor_id, 0] =  statistics_data[sensor_id, 0]
        calibration_bias_data[sensor_id, 1] =  statistics_data[sensor_id, 2]
        calibration_bias_data[sensor_id, 2] =  statistics_data[sensor_id, 4]
        calibration_bias_data[sensor_id, 3] =  statistics_data[sensor_id, 6]
        calibration_bias_data[sensor_id, 4] =  statistics_data[sensor_id, 8]
        calibration_bias_data[sensor_id, 5] =  statistics_data[sensor_id, 10]


        calibration_noise_data[sensor_id, 0] =  statistics_data[sensor_id, 1]
        calibration_noise_data[sensor_id, 1] =  statistics_data[sensor_id, 3]
        calibration_noise_data[sensor_id, 2] =  statistics_data[sensor_id, 5]
        calibration_noise_data[sensor_id, 3] =  statistics_data[sensor_id, 7]
        calibration_noise_data[sensor_id, 4] =  statistics_data[sensor_id, 9]
        calibration_noise_data[sensor_id, 5] =  statistics_data[sensor_id, 11]


print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%')
    print('calibration_bias_data=',calibration_bias_data)
    print('calibration_noise_data=',calibration_noise_data)

print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%')
```