Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

Sistema Integrado de Navegación con IMU y GPS para Vehículos Autónomos Basado en el Filtro de Kalman

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **ALEXANDER ABRICA LARA**

Director **DR. LUIS ENRIQUE GONZÁLEZ JIMÉNEZ**
Codirector **DR. RAÚL ARTURO GARCÍA HUERTA**

Tlaquepaque, Jalisco. 15 de junio de 2026.

Dedicado a mi padre.

Resumen

El uso de vehículos aéreos no tripulados ha aumentado en las últimas décadas debido a los diversos servicios que ofrecen. Pero la mayoría de las personas utiliza estas aeronaves sin conocer las ciencias aplicadas necesarias para desarrollar un modelo propio. El objetivo es implementar un prototipo de bajo costo que permita a cualquier persona, sin importar que no tenga conocimientos ni experiencia previa con drones, adquirir las bases necesarias para desempeñarse en aplicaciones de vehículos aéreos no tripulados. El documento expone las teorías fundamentales sobre las transformaciones de cuerpos rígidos, su traslación y rotación, facilitando así la comprensión matemática y la interpretación en modelos lineales para estimar valores y contrastarlos con mediciones ruidosas. Asimismo, se presentan ejemplos de sensores y módulos básicos para su implementación, el procesamiento de datos y su reinterpretación para su integración en el diseño del vehículo autónomo. Antes de abordar el diseño, se guía al lector en la obtención de los valores de covarianza de cada sensor y módulo, con el fin de comprender cómo reducir los errores de medición y de proceso en el sistema lineal. Una vez identificadas todas las variables, se procede al diseño e implementación de los sistemas de observación de estado que permiten estimar la orientación y la posición del vehículo. Finalmente, se analizan los resultados obtenidos en distintos experimentos que pondrán a prueba si las estimaciones de orientación y posición se aproximan a la verdad fundamental propuesta.

Contenido

Introducción

Los vehículos aéreos no tripulados, o drones, se han convertido en herramientas que pueden integrarse en diversas aplicaciones y disciplinas gracias a su capacidad de desplazarse y comunicarse en zonas de difícil acceso para las personas. Sin embargo, muchas personas desconocen o ignoran los conceptos indispensables para desarrollar un dron. Por esta razón, el presente escrito tiene como objetivo proporcionar los conocimientos básicos necesarios para desarrollar un vehículo no tripulado, abarcando los conceptos de control, diseño embebido y programación de los diversos módulos que lo conforman.

En el primer capítulo se presentan algunas de las posibles aplicaciones de los vehículos aéreos no tripulados. Además de presentar casos reales en los que los drones han demostrado ser una herramienta valiosa.

El segundo capítulo está dedicado a la exposición de temas fundamentales de la mecánica clásica, como la traslación y la rotación de cuerpos rígidos. También aborda la teoría de control empleada en el desarrollo de modelos matemáticos de sistemas dinámicos, en particular el filtro de Kalman, cuyo objetivo es obtener estimaciones que corrijan los valores de ruido externo. En este mismo capítulo se describen los sensores que integran la unidad de medición inercial (IMU) y el módulo del sistema de navegación por satélite (GNSS).

El tercer capítulo se centra en los sensores y módulos que se utilizarán en el prototipo de prueba, y expone las características del ruido obtenidas durante las mediciones. Posteriormente, esos valores se convierten en medidas estadísticas que describen sus variaciones para el diseño del sistema dinámico. Al mismo tiempo, se definen las características del software de cada módulo para establecer la frecuencia de trabajo que mejor se ajuste a las necesidades del sistema. El objetivo es diseñar sistemas dinámicos que prevean y corrijan las estimaciones de posición y trayectoria del vehículo aéreo.

El capítulo cuatro presenta los resultados finales de los diferentes experimentos enfocados en la orientación y la trayectoria del prototipo de vehículo aéreo no tripulado. Las pruebas de orientación se llevaron a cabo en un ambiente controlado mediante la integración del prototipo en

un brazo robótico, mientras que las pruebas de trayectoria se realizaron a campo abierto en una cancha de fútbol.

En el capítulo final se exponen las resoluciones de los experimentos, en las que se presentan los aciertos, las fallas y las limitaciones. Además, se proponen alternativas y puntos de mejora con el fin de alentar a los lectores a diseñar su propio prototipo y alcanzar el éxito en su propia versión.

1. Sistemas de navegación para vehículos autónomos

Los vehículos aéreos no tripulados (VANT), también denominados drones, han contado con una amplia variedad de usos desde sus inicios. Inicialmente concebidos para fines militares, en la actualidad abarcan diversas aplicaciones en la vida diaria, como la seguridad, la agricultura, el mapeo, la investigación ambiental y la logística.

La versatilidad de estos dispositivos para acceder a lugares de difícil acceso, sus bajos costos de construcción y su adaptabilidad tecnológica han hecho posible que los VANT sean considerados una alternativa a otros medios de transporte convencionales.

El desarrollo de los VANT integra diversas disciplinas de la ingeniería, como la electrónica, la mecatrónica, la programación embebida, la teoría de control y la sensórica, lo que convierte a estos sistemas en plataformas ideales para la investigación y la aplicación de tecnologías avanzadas de automatización.

1.1. Planteamiento del problema

En regiones de difícil acceso, como áreas montañosas, comunidades rurales aisladas o zonas afectadas por desastres naturales, la entrega de suministros médicos y alimenticios representa un reto logístico. Las condiciones del terreno, la falta de infraestructuras y el tiempo de traslado dificultan o incluso imposibilitan el uso de medios de transporte convencionales.

Los VANT surgen como una alternativa ideal por su capacidad de control en vuelo y en el aterrizaje sin necesidad de intervención humana.

Sin embargo, para que estas soluciones puedan implementarse de manera efectiva, es indispensable desarrollar sistemas embebidos de instrumentación y control robustos, capaces de integrar sensores y algoritmos de navegación.

1.2. Antecedentes

En la última década, el desarrollo de los drones ha experimentado un crecimiento acelerado tanto en el ámbito comercial como en el académico. Empresas y organizaciones internacionales han invertido en la investigación y el diseño de sistemas aéreos autónomos capaces de realizar tareas que tradicionalmente dependían de la presencia humana.

En el ámbito académico y científico, numerosos estudios han abordado la optimización de algoritmos de control, el desarrollo de sistemas embebidos especializados y la fusión sensorial para mejorar la estabilidad y la precisión del vuelo. Por ejemplo, en 2021, la Universidad Nacional de Investigación de Tecnología Electrónica de Moscú (MIET) diseñó un vehículo autónomo no tripulado que proporciona un control de precisión para la navegación y la orientación mediante módulos del sistema global de navegación por satélite (GNSS) y una unidad de medición inercial (IMU). El objetivo de aplicación de este vehículo es transportar cargas de más de 200 kg, ya que se basa en un sistema autónomo (SDS) para la plataforma de carga en paracaídas (PLP), lo que implica un alto costo de desarrollo en comparación con modelos de drones enfocados en el uso civil [1].

Por otro lado, las empresas privadas han generado un impacto social al implementar drones para la distribución de suministros médicos y víveres. Un ejemplo es la empresa estadounidense Zipline, dedicada al diseño, fabricación y operación de drones de reparto. Desde 2018, Zipline ha establecido acuerdos con los gobiernos de Ruanda y Ghana para construir centros de distribución destinados a la entrega de suministros médicos en zonas rurales e inaccesibles, demostrando el impacto positivo de estos VANT al reducir significativamente los tiempos de entrega de sangre, medicamentos y vacunas [2].

1.3. Objetivos

1.3.1 General

Realizar una implementación embebida de algoritmos de instrumentación y control robustos que integren sensores y la fusión de sus señales, incluyendo algoritmos de navegación.

1.3.2 Específicos

- Investigar y seleccionar los sensores adecuados para el control de la orientación y la navegación del VANT.
- Desarrollar algoritmos de control e instrumentación basados en la teoría de sistemas dinámicos y en la fusión de sensores.
- Implementar un sistema de orientación y navegación basado en IMU y GNSS.
- Realizar pruebas experimentales para validar el desempeño del sistema en condiciones reales o simuladas.

2. Marco teórico

Para desarrollo un vehículo aéreo no tripulado desde cero es importante contar con conocimientos básicos de los fundamentos de la rotación, los sistemas de navegación, la fusión de sensores y los algoritmos de teoría de control.

2.1. Transformaciones de cuerpo rígido

Un cuerpo rígido libre en el espacio tiene seis grados de libertad, lo que permite moverse y rotar de manera tridimensional. Estos grados consisten en dos categorías. Los grados de libertad de traslación, cuyo propósito es el movimiento en los 3 ejes (X, Y, Z); y los grados de libertad de rotación de los ejes (α, β, γ).

La orientación de un objeto se describe con respecto a un sistema de coordenadas de referencia conocido. Por lo tanto, se deben definir dos cosas:

1. Un sistema de coordenadas de referencia en el cuerpo se mueve en el espacio.
2. Especificar la rotación de dichas coordenadas mediante un método de representación.

2.1.1 Traslaciones

Hay dos formas de representación mediante razonamiento geométrico: sintética y analítica. La primera utiliza entidades geométricas (por ejemplo, puntos y líneas), mientras que la otra se representa mediante manipulación algebraica (coordenadas o ecuaciones) [3].

En la Fig. 2-1, muestra un punto p con respecto a dos referencias coordenadas, o_0,x_0,y_0 y o_1,x_1,y_1. Donde el vector d_0^1 expresa la distancia desplazada con respecto al origen o_0,x_0,y_0 hacia el marco de referencia o_1,x_1,y_1. Mientras que el punto p puede presentar sus coordenadas con base en cualquiera de sus dos referencias o_0,x_0,y_0 y o_1,x_1,y_1 como los vectores p^0 y p^1, respectivamente. Al final se obtiene la representación de uno de los puntos, en este caso p^0, de la siguiente manera:

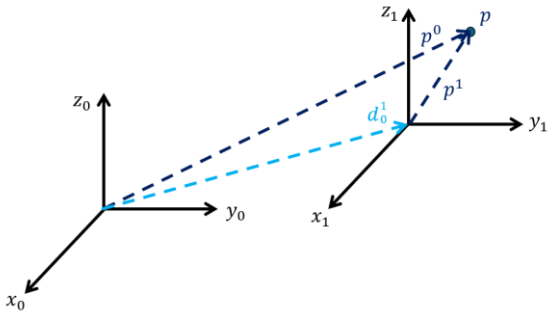

Fig. 2-1 Dos referencias de coordenadas sobre un punto p.

$$\boldsymbol{p^0 = d_0^1 + p^1} \tag{2-1}$$

Esta operación se denomina mapeo de vector de un sistema de referencia

2.1.2 Rotaciones

Para representar la posición relativa y la orientación de un cuerpo rígido con respecto a otro, se unirán rígidamente las coordenadas de referencia de ambos cuerpos y se especificarán las relaciones geométricas entre ellas [3].

Al proyectar los ejes de un sistema de coordenadas (o_0,x_0,y_0) sobre los ejes de otro sistema de coordenadas (o_1,x_1,y_1), el producto de ambos es igual al mostrado en la Fig. 2-2.

Una manera de representar la orientación relativa de estos dos sistemas es especificar las coordenadas vectoriales de los ejes del sistema de referencia o_1,x_1,y_1 con respecto al sistema o_0,x_0,y_0. Esto permite representar la orientación de una matriz de rotación (2-5).

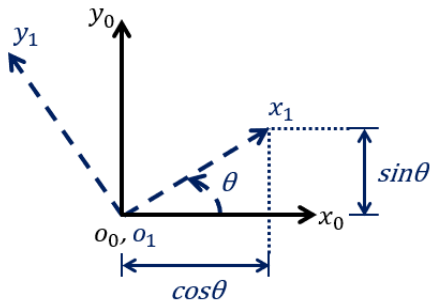

Fig. 2-2 Proyección del sistema de coordenadas o_0,x_0,y_0 a o_1,x_1,y_1 al sistema de coordenadas o_0,x_0,y_0.

$$\boldsymbol{R_1^0} = \left[\boldsymbol{x_1^0}\middle|\boldsymbol{y_1^0}\right] \tag{2-2}$$

$$\boldsymbol{x_1^0} = \begin{bmatrix} cos\,\theta \\ sin\,\theta \end{bmatrix} \tag{2-3}$$

$$\boldsymbol{y_1^0} = \begin{bmatrix} -sin\,\theta \\ cos\,\theta \end{bmatrix} \tag{2-4}$$

$$\boldsymbol{R_1^0} = \begin{bmatrix} cos\,\theta & -sin\,\theta \\ sin\,\theta & cos\,\theta \end{bmatrix} \tag{2-5}$$

Otra alternativa para representarlo consiste en crear la matriz rotacional al proyectar los ejes del sistema o_1,x_1,y_1 sobre los ejes de coordenadas del sistema o_0,x_0,y_0.

$$\boldsymbol{x_1^0} = \begin{bmatrix} \boldsymbol{x_1} * \boldsymbol{x_0} \\ \boldsymbol{x_1} * \boldsymbol{y_0} \end{bmatrix} \tag{2-6}$$

$$\boldsymbol{y_1^0} = \begin{bmatrix} \boldsymbol{y_1} * \boldsymbol{x_0} \\ \boldsymbol{y_1} * \boldsymbol{y_0} \end{bmatrix} \tag{2-7}$$

$$\boldsymbol{R_1^0} = \begin{bmatrix} \boldsymbol{x_1} * \boldsymbol{x_0} & \boldsymbol{x_1} * \boldsymbol{y_0} \\ \boldsymbol{x_1} * \boldsymbol{y_0} & \boldsymbol{y_1} * \boldsymbol{y_0} \end{bmatrix} \tag{2-8}$$

En el caso de proyectar en tres dimensiones, cada eje de las coordenadas o_1,x_1,z_1 es proyectado sobre un sistema de coordenadas o_0,x_0,z_0. Por lo tanto:

$$\boldsymbol{R_1^0} = \begin{bmatrix} \boldsymbol{x_1} * \boldsymbol{x_0} & \boldsymbol{y_1} * \boldsymbol{x_0} & \boldsymbol{z_1} * \boldsymbol{x_0} \\ \boldsymbol{x_1} * \boldsymbol{y_0} & \boldsymbol{y_1} * \boldsymbol{y_0} & \boldsymbol{z_1} * \boldsymbol{y_0} \\ \boldsymbol{x_1} * \boldsymbol{z_0} & \boldsymbol{y_1} * \boldsymbol{z_0} & \boldsymbol{z_1} * \boldsymbol{z_0} \end{bmatrix} \tag{2-9}$$

Este es el caso en el que las matrices son ortogonales.

Al consideraremos un eje z_0, mientras los demás productos escalares sean cero, como se observa en la Fig. 2-3, la matriz rotacional es la siguiente:

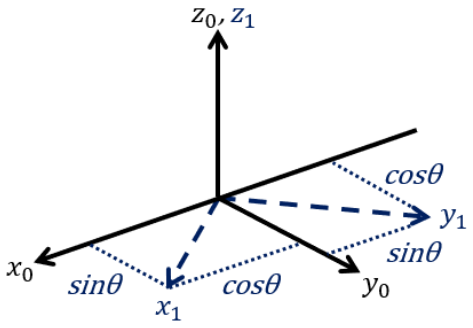

Fig. 2-3 Proyección del sistema de coordenadas en tres dimensiones.

$$\boldsymbol{R_1^0} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-10}$$

Después se busca la matriz de rotación del eje Z.

$$\boldsymbol{R_{z,0}} = \boldsymbol{I} \tag{2-11}$$

$$\boldsymbol{R_{z,\theta}R_{z,\phi}} = \boldsymbol{R_{z,\theta+\phi}} \tag{2-12}$$

Similarmente, la representación básica de las matrices de rotación sobre los ejes X (2-6), Y (2-7) son:

$$\boldsymbol{R_{x,\theta}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2-13}$$

$$\boldsymbol{R_{y,\theta}} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2-14}$$

2.1.3 Ángulos de Euler

Existen distintos métodos para especificar la matriz de rotación de forma independientes; uno de ellos es el uso de los ángulos de Euler. Los ángulos de Euler se representan mediante tres coordenadas angulares que describen la orientación tridimensional de un cuerpo rígido a través de una secuencia de rotación en los planos $x_0y_0z_0$ hacia $x_1y_1z_1$ como punto de origen, cuya intersección se le denomina línea de nodos [3].

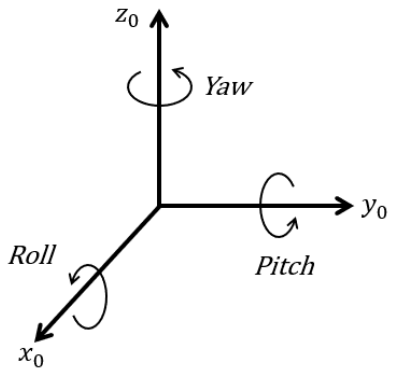

Fig. 2-4 Proyección de los ángulos de Euler.

En la Fig. 2-4, los ángulos de Euler se definen de la siguiente manera:

- α es el ángulo entre el eje x_0 y la línea de nodos, cuyo movimiento alrededor de un cuerpo rígido se denomina balanceo (Roll).
- β es el ángulo entre el eje y_0 y el eje y_1, cuyo movimiento alrededor de un cuerpo rígido se denomina cabeceo (Pitch).
- γ es el ángulo entre la línea de nodos y el eje x_1, cuyo movimiento alrededor de un cuerpo rígido se denomina guiñado (Yaw).

En términos de rotación básica, la transformación de rotación puede describirse como un producto de rotación sucesiva sobre la coordenada principal de los ejes x_0,y_0,z_0; como se muestra en el siguiente producto:

$$\boldsymbol{R_{XYZ}} = \boldsymbol{R_{x,\alpha}}\boldsymbol{R_{y,\beta}}\boldsymbol{R_{z,\gamma}} \tag{2-15}$$

$$\boldsymbol{R_{XYZ}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sin\,\alpha \\ 0 & sin\,\alpha & cos\,\alpha \end{bmatrix}\begin{bmatrix} cos\,\beta & 0 & sin\,\beta \\ 0 & 1 & 0 \\ -sin\,\beta & 0 & cos\,\beta \end{bmatrix}\begin{bmatrix} cos\,\gamma & -sin\,\gamma & 0 \\ sin\,\gamma & cos\,\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-16}$$

$$\boldsymbol{R_{XYZ}} = \begin{bmatrix} cos\,\beta\,cos\,\gamma & -cos\,\beta\,sin\,\gamma & sin\,\beta \\ sin\,\alpha\,sin\,\beta\,cos\,\gamma + cos\,\alpha\,sin\,\gamma & -sin\,\alpha\,sins\,\beta\,sin\,\gamma + cos\,\alpha\,cos\,\gamma & -sin\,\alpha\,cos\,\beta \\ -cos\,\alpha\,sin\,\beta\,cos\,\gamma + sin\,\alpha\,sin\,\gamma & cos\,\alpha\,sin\,\beta\,sin\,\gamma + sin\,\alpha\,cos\,\gamma & cos\,\alpha\,cos\,\beta \end{bmatrix} \tag{2-17}$$

$$\boldsymbol{R_{XYZ}} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2-18}$$

Mediante la representación de una matriz de rotación $\boldsymbol{R}$ (2-18), se obtienen los ángulos de Euler γ, β y α al igualar cada elemento de R con su correspondiente elemento en el producto matricial $\mathbf{R_{XYZ}}$ (2-17).

$$R_{13} = sin\,\beta \tag{2-19}$$

$$\beta = \sin^{-1}(R_{13}) \tag{2-20}$$

$$R_{23} = -sin\,\alpha\,cos\,\beta \tag{2-21}$$

$$R_{33} = cos\,\alpha\,cos\,\beta \tag{2-22}$$

$$\frac{-R_{23}}{R_{33}} = \tan\alpha \tag{2-23}$$

$$\alpha = \tan^{-1}\left(\frac{-R_{23}}{R_{33}}\right) \tag{2-24}$$

$$R_{12} = cos\,\beta\,cos\,\gamma \tag{2-25}$$

$$R_{11} = -cos\,\beta\,sin\,\gamma \tag{2-26}$$

$$\frac{-R_{12}}{R_{11}} = \tan\gamma \tag{2-27}$$

$$\gamma = \tan^{-1}\left(\frac{-R_{12}}{R_{11}}\right) \tag{2-28}$$

En el caso de que $cos\,\beta = 0$, es decir, que el ángulo pitch esté cercano a ±90°, las entradas de la matriz $\mathbf{R_{XYZ}}$ (2-17) se vuelven indistinguibles cuando dos ejes se posicionan en paralelo, lo que bloquea el sistema de rotación en un espacio bidimensional y no en uno tridimensional [4]. Esto provoca que los ángulos roll y yaw (α, γ) se alineen entre sí y pierdan un grado de libertad, lo que puede conllevar la pérdida de control sobre la orientación del objeto. Este fenómeno se denomina bloqueo de cardán o gimbal.

Una solución para este fenómeno consiste en implementar un límite de rango para los ángulos α, γ. Otra opción es utilizar un algoritmo de compensación. Si bien no evita por completo el bloqueo de cardán, permite obtener una salida consistente.

En el caso que $\beta = \frac{\pi}{2}$:

$$R_{21} = sin\,\alpha\,cos\,\gamma + cos\,\alpha\,sin\,\gamma = sin(\alpha+\gamma) \tag{2-29}$$

$$R_{31} = -cos\,\alpha\,cos\,\gamma + sin\,\alpha\,sin\,\gamma = -cos(\alpha+\gamma) \tag{2-30}$$

$$R_{22} = -sin\,\alpha\,sin\,\gamma + cos\,\alpha\,cos\,\gamma = cos(\alpha+\gamma) = -R_{31} \tag{2-31}$$

$$R_{32} = cos\,\alpha\,sin\,\gamma + sin\,\alpha\,cos\,\gamma = sin(\alpha+\gamma) = R_{21} \tag{2-32}$$

$$(\alpha+\gamma) = \tan^{-1}\left(\frac{R_{21}}{-R_{31}}\right) \tag{2-33}$$

$$\gamma = \tan^{-1}\left(\frac{R_{21}}{-R_{31}}\right) - \alpha \tag{2-34}$$

Y mientras el caso que $\beta = -\frac{\pi}{2}$:

$$R_{21} = -sin\,\alpha\,cos\,\gamma + cos\,\alpha\,sin\,\gamma = -sin(\alpha-\gamma) \tag{2-35}$$

$$R_{31} = cos\,\alpha\,cos\,\gamma + sin\,\alpha\,sin\,\gamma = cos(\alpha-\gamma) \tag{2-36}$$

$$R_{22} = sin\,\alpha\,sin\,\gamma + cos\,\alpha\,cos\,\gamma = cos(\alpha - \gamma) = R_{31} \tag{2-37}$$

$$R_{32} = -cos\,\alpha\,sin\,\gamma + sin\,\alpha\,cos\,\gamma = -sin(\alpha - \gamma) = R_{21} \tag{2-38}$$

$$(\alpha - \gamma) = \tan^{-1}\left(\frac{R_{21}}{R_{31}}\right) \tag{2-39}$$

$$\gamma = \tan^{-1}\left(\frac{R_{21}}{R_{31}}\right) + \alpha \tag{2-40}$$

Al final, los ángulos de Euler pueden interpretarse dependiendo del valor de β, que es igual a $\pm\frac{\pi}{2}$. Pero, utilizando el elemento R_{13} (2-19) de la matriz de rotación, se definen condiciones que pueden aplicarse en un algoritmo.

Mientras $R_{13} \neq \pm 1$, los ángulos de Euler se representan mediante las ecuaciones (2-20), (2-24) y (2-28).

En los casos en que $R_{13} = 1$, el ángulo γ se representa en la ecuación (2-34); y en el caso que $R_{13} = -1$, la representación de yaw es igual a la ecuación (2-40).

Mientras que el ángulo α, es conveniente establecer que $\alpha = 0$ para mitigar el fenómeno de bloqueo de cardán.

Existen diferentes maneras de obtener los ángulos de rotación, ya sea cambiando la secuencia de rotación; sin embargo, siempre se obtendrán los tres ángulos principales de rotación de un objeto libre en el espacio [3].

2.1.4 Transformaciones homogéneas

Al considerar dos sistemas de referencia x_0,y_0,z_0 y x_1,y_1,z_1 cuyos orígenes difieren por desplazamientos y orientaciones específicas, como se muestra en la Fig. 2-5.

La relación de ambos sistemas incluye la matriz de rotación $\boldsymbol{R_0^1}$ y el vector de traslación $\boldsymbol{d_0^1}$ permite representarse de la siguiente manera:

$$\boldsymbol{p^0 = d_0^1 + R_0^1 p^1} \tag{2-41}$$

En (2-41) se incluye el término de $\boldsymbol{R_0^1 p^1}$, que puede volverse intangible tras una larga secuencia de movimientos rígidos. Por lo tanto, la forma de compactar su representación es a través

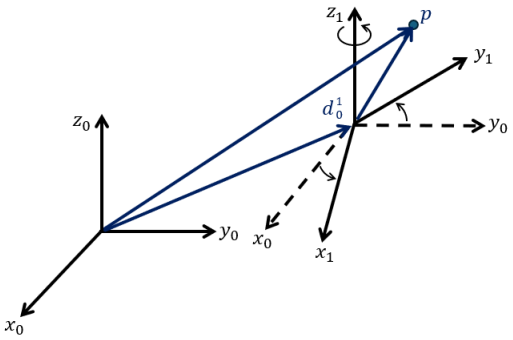

Fig. 2-5 Proyección de dos sistemas de coordenadas $o_0x_0y_0z_0$, y $o_1x_1y_1z_1$.

de una transformación homogénea, que reduce su expresión a una matriz que represente tanto la traslación como su orientación.

$$\boldsymbol{H_0^1} = \begin{bmatrix} \boldsymbol{R_0^1} & \boldsymbol{d_0^1} \\ 0 & 1 \end{bmatrix}; R \in SO(3), d \in \mathbb{R}^3 \tag{2-42}$$

Para poder expresar la transformación (2-41) en una matriz, se agrega a los vectores $\boldsymbol{p^0}$ y $\boldsymbol{p^1}$ un componente de uno, obteniendo así las representaciones homogéneas $\boldsymbol{P^0}$ y $\boldsymbol{P^1}$:

$$\boldsymbol{P^0} = \begin{bmatrix} \boldsymbol{p^0} \\ 1 \end{bmatrix} \tag{2-43}$$

$$\boldsymbol{P^1} = \begin{bmatrix} \boldsymbol{p^1} \\ 1 \end{bmatrix} \tag{2-44}$$

Por lo tanto, la transformación resulta equivalente a la representación mediante la siguiente matriz homogénea:

$$\boldsymbol{P^0} = \boldsymbol{H_0^1} + \boldsymbol{P^1} \tag{2-45}$$

Al final, las matrices de transformación homogénea pueden tener diversas interpretaciones con respecto a los ejes X, Y, Z.

$$\boldsymbol{Trans_{x,a}} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \ \boldsymbol{R_{x,\phi}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\phi & -sin\,\phi & 0 \\ 0 & sin\,\phi & cos\,\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-46}$$

$$\boldsymbol{Trans_{y,b}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ \boldsymbol{R_{y,\theta}} = \begin{bmatrix} cos\,\theta & 0 & sin\,\phi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\,\theta & 0 & cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}47)$$

$$\boldsymbol{Trans_{z,c}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix};\ \boldsymbol{R_{y,\psi}} = \begin{bmatrix} cos\,\psi & -sin\,\psi & 0 & 0 \\ sin\,\psi & cos\,\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}48)$$

2.2. Modelo en Espacio de Estados

El modelo en espacio de estados es un enfoque matemático que utiliza variables de estado para representar un sistema mediante un conjunto de ecuaciones diferenciales de cualquier orden en el dominio del tiempo. Si el conjunto de ecuaciones diferenciales es lineal en las variables de estado y sus entradas, el modelo se denomina espacio de estados lineal [5].

La estructura del modelo de estados lineal es ideal para estimaciones rápidas, ya que requieren únicamente un parámetro: el orden del modelo *n*.

Una forma de representar el espacio de estados de un sistema lineal en tiempo continuo, como se muestra en la Fig. 2-6, es la siguiente:

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu} \quad (2\text{-}49)$$

$$\boldsymbol{y} = \boldsymbol{Cx} \quad (2\text{-}50)$$

donde $\boldsymbol{x}$ es el vector de estado, $\boldsymbol{y}$ es el vector de salidas, $\boldsymbol{u}$ es el vector de entradas, $\boldsymbol{A}$ es la matriz de estados, $\boldsymbol{B}$ es la matriz de entrada y $\boldsymbol{C}$ es la matriz de salida.

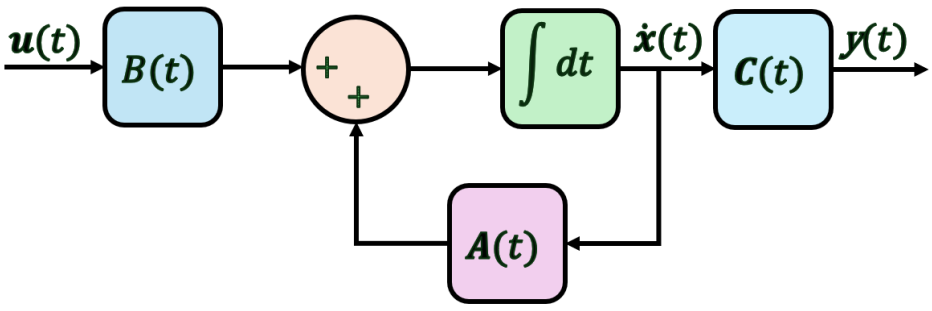

Fig. 2-6 Modelo de espacio de estado típico.

2.2.1 Observadores de Estados

En teoría de control, la observación de estado es un sistema que proporciona una estimación del espacio de estados de un sistema real a partir de mediciones de sus entradas y salidas.

El modelo matemático del observador es similar al del sistema, con la excepción de que incluye un término adicional que incorpora el error de estimación para compensar las imprecisiones en las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$, así como la ausencia de un error inicial. El error de estimación corresponde a la diferencia entre la salida medida y la salida estimada, mientras que el error inicial es la diferencia entre el estado inicial y el estado estimado inicial, como se muestra en la Fig. 2-7 [6].

Considerando el sistema definido en (2-49) y (2-50), el modelo matemático del observador se expresa de la siguiente manera:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{K}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{2-51}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{C}\hat{\boldsymbol{x}} \tag{2-52}$$

donde $\hat{x}$ es el vector del estado estimado, $\hat{y}$ es el vector de la salida estimada y $\boldsymbol{K}$ es la matriz de ganancia del observador.

Desafortunadamente, no existen modelos matemáticos perfectos en la vida real. Por lo que se requiere un estado de estimación que elimine la diferencia entre el valor de la medición y el de estimación de los estados internos. En términos simples, se implementan un sistema de retroalimentación que intenta controlar el error entre la medición y la estimación mediante un controlador denominado $\boldsymbol{K}$.

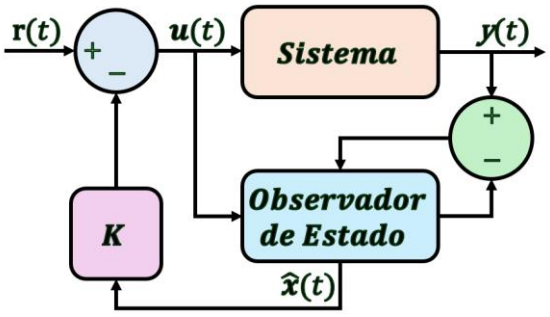

Fig. 2-7 Modelo representativo de un espacio de estado con observador de estado.

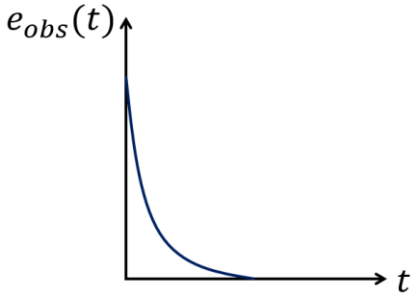

Fig. 2-8 Función exponencial de los errores dinámicos de los estados de medición y de observación

Si se sustraen las ecuaciones del estado de medición y del estado de observación, se obtienen los errores dinámicos, que se representan en las funciones exponenciales (2-53) y (2-54). Si el error de observación es 0, el valor de la estimación interna es igual al valor interno del bloque de medición, lo que permite determinar cuándo confiar en la medición y cuándo en la estimación [7].

$$\boldsymbol{e_{obs} = x - \hat{x}} \tag{2-53}$$

$$\boldsymbol{y - \hat{y} = Ce_{obs}} \tag{2-54}$$

$$\boldsymbol{\dot{e}_{obs} = (Ax - KC)e_{obs}} \tag{2-55}$$

$$\boldsymbol{e_{obs}}(t) = e^{(\boldsymbol{A-KC})t}\boldsymbol{e_{obs}}(0) \tag{2-56}$$

Si $(\boldsymbol{Ax - KC}) < 0$, entonces $e_{obs} \to 0$ como $t \to \infty$, como se representa en la Fig. 2-8. Así, $\hat{x} \to x$.

2.2.2 Filtro de Kalman

El filtro de Kalman es una técnica matemática que permite estimar el estado de un sistema dinámico a partir de mediciones incompletas o imprecisas. Este filtro fue desarrollado por Rudolf Kalman en la década de 1960 y se basa en la teoría de la probabilidad y en la teoría de control. Es especialmente útil en sistemas complejos donde hay ruido y errores en las mediciones, lo que dificulta obtener una estimación precisa del estado del sistema [7].

2.MARCO TEÓRICO

El filtro de Kalman utiliza un modelo matemático del sistema que se está monitoreando, incluyendo los parámetros de medición para estimar el estado del sistema. El algoritmo del filtro consiste en dos pasos: primero predice y luego corrige el estado del sistema.

La etapa de predicción consiste en definir el sistema adecuado para reducir el error de ruido en las mediciones. En este caso de un sistema lineal:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{w}(t) \tag{2-57}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{v}(t) \tag{2-58}$$

donde $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ son las matrices del sistema, $\boldsymbol{w}$ es el vector del ruido gaussiano con media 0 y covarianza Q, y $\boldsymbol{v}$ es el vector del ruido gaussiano con media 0 y covarianza R

En el caso de que se desee trabajar con fusiones de sensores, es necesario cambiar del sistema continuo a uno discreto, ya que únicamente se puede trabajar con muestras actuales y pasadas de las variables, no con muestras futuras. Para realizar este cambio al sistema discreto, se aproxima la derivada usando su definición $\dot{\boldsymbol{x}}(\boldsymbol{t}) \approx \frac{\boldsymbol{x_k} - \boldsymbol{x_{k-1}}}{\boldsymbol{T}}$, resultando en:

$$\frac{\boldsymbol{x_k} - \boldsymbol{x_{k-1}}}{\boldsymbol{T}} = \boldsymbol{A}\boldsymbol{x_{k-1}} + \boldsymbol{B}\boldsymbol{u_{k-1}} + \boldsymbol{w_{k-1}} \tag{2-59}$$

$$\boldsymbol{y_k} = \boldsymbol{C}\boldsymbol{x_k} + \boldsymbol{v_k} \tag{2-60}$$

Al despejar los vectores de estado y de salida se obtiene:

$$\boldsymbol{x_k} = (\boldsymbol{AT} + \boldsymbol{I_3})\boldsymbol{x_{k-1}} + \boldsymbol{BT}\boldsymbol{u_{k-1}} + \boldsymbol{T}\boldsymbol{w_{k-1}} \tag{2-61}$$

$$\boldsymbol{y_k} = \boldsymbol{C}\boldsymbol{x_k} + \boldsymbol{v_k} \tag{2-62}$$

Los siguientes términos pueden definirse de la siguiente manera: $\boldsymbol{A_d} = (\boldsymbol{AT} + \boldsymbol{I_3}), \boldsymbol{B_d} = \boldsymbol{BT}, \boldsymbol{C_d} = \boldsymbol{C}$, lo que resulta en el sistema equivalente:

$$\boldsymbol{x_k} = \boldsymbol{A_d}\boldsymbol{x_{k-1}} + \boldsymbol{B_d}\boldsymbol{u_{k-1}} + \boldsymbol{w_{k-1}} \tag{2-63}$$

$$\boldsymbol{y_k} = \boldsymbol{C_d}\boldsymbol{x_k} + \boldsymbol{v_k} \tag{2-64}$$

El filtro de Kalman emplea las mediciones recientes para predecir el estado futuro del sistema y, posteriormente, ajusta dicha predicción con base en las últimas mediciones, con el objetivo de mejorar la precisión de la estimación.

Primero se obtiene una actualización de la covarianza de la estimación:

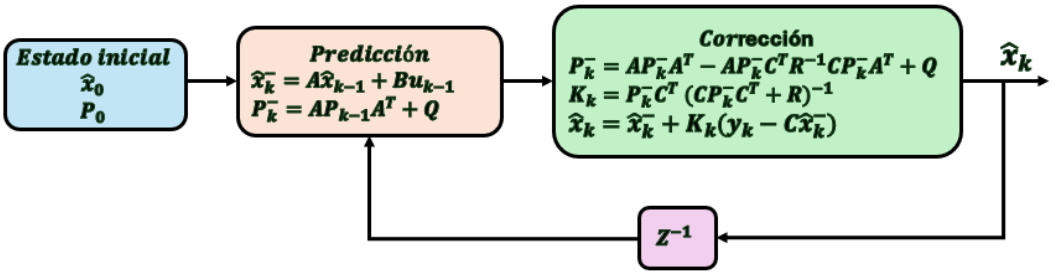

Fig. 2-9 Diagrama a bloques del filtro de Kalman.

$$\boldsymbol{P_k = A_d P_k^- A_d^T - A_d P_k^- C_d^T R^{-1} C_d P_k^- A_d^T + Q} \tag{2-65}$$

A continuación, se calcula la ganancia de control del observador, denominada ganancia de Kalman:

$$\boldsymbol{K_k = P_k^- C_d \left(C_d P_k^- C_d^T + R\right)^{-1}} \tag{2-66}$$

Finalmente, se obtiene la actualización de la estimación con la medida:

$$\boldsymbol{\hat{x}_k = \hat{x}_k^- + K_k(y_k - C_d \hat{x}_k^-)} \tag{2-67}$$

Una vez definidos los elementos anteriores, el filtro de Kalman en un sistema discreto se representa en la Fig. 2-9.

2.3. Sensores de navegación

Esta sección describe los tipos de sensores y el sistema de navegación empleados en el proyecto. Además, se explican los métodos de fusión utilizados para obtener datos de orientación y rotación, y se detalla la teoría de control aplicada en el diseño del VANT.

2.3.1 IMU

El sensor de medición inercial, conocido como IMU (Inertial Measurement Unit), es un dispositivo electrónico compuesto por varios sensores; cada uno con tres ejes de movimiento (x, y, z).

Existe un tipo de IMU compuesto por un acelerómetro, un giroscopio y un magnetómetro, que controla todos los tres ejes para obtener mediciones de nueve grados de libertad (9-DOF).

2.3.2 Magnetómetro

El magnetómetro es un sensor que mide el campo magnético terrestre. De manera informal, se asemeja a una brújula. Su funcionamiento se basa en el uso de magnetorresistencias, que varían su resistencia en función del campo magnético que las atraviesa, mediante sensores AMR (Anisotropic Magneto Resistance) [8]. Estos cambios de resistencia provocan variaciones de voltaje, lo que permite al dispositivo determinar el vector orientado hacia el norte de la Tierra.

2.3.3 Acelerómetro

El acelerómetro es un dispositivo electromecánico que mide la fuerza de aceleración producida por el movimiento o la vibración del propio dispositivo. Los acelerómetros se componen de celdas capacitivas, formadas por placas fijas y placas móviles unidas a una masa, que provocan cambios en la capacitancia o capacidad eléctrica. Cuando el dispositivo sufre una aceleración, aparece una fuerza en sentido contrario al movimiento, lo que provoca el desplazamiento de todo el sistema. La masa opone resistencia al cambio de posición, generando un movimiento relativo entre ella y las placas fijas [9].

2.3.4 Giroscopio

El giroscopio es un aparato electromecánico que permite medir la velocidad angular de un cuerpo mediante el efecto de Coriolis. Este fenómeno se observa en un sistema en rotación cuando un cuerpo se encuentra en movimiento respecto a dicho sistema y consiste en una aceleración relativa del cuerpo en el sistema, perpendicular al eje de rotación y a la velocidad del cuerpo [9]. Los giroscopios tienen una configuración denominada "horquilla vibrante", que consiste en dos masas que oscilan y se mueven constantemente en direcciones opuestas.

2.3.5 Sistema de coordenadas NED (North-East-Down)

El sensor IMU permite determinar la orientación a partir de la aceleración de un cuerpo y la dirección del norte con base en la gravedad de la Tierra, según el hemisferio en el que se encuentre. Por estas razones, la representación del cuerpo en el que se instala la IMU se realiza en el sistema de coordenadas NED.

El sistema NED es un marco de referencia local que se origina en un punto específico de la superficie terrestre y consta de tres ejes ortogonales: Norte, Este y Down (abajo). El eje Norte apunta hacia el norte geodésico, siguiendo la tangente a la superficie de la Tierra; el eje Este apunta en la dirección perpendicular al norte y está alineado con la tangente a la superficie terrestre; finalmente, el eje Down apunta perpendicular al plano tangente, es decir, hacia el centro de la Tierra (dirección local de la gravedad) [10].

La IMU permite obtener una representación vectorial del sistema NED. Inicialmente, se obtiene Down, que se basa en la dirección del vector del acelerómetro respecto a la gravedad.

$$\left|a_{xyz}\right| = \sqrt{(a_x)^2 + \left(a_y\right)^2 + (a_z)^2} \tag{2-68}$$

$$\vec{D}_{x,y,z} = \frac{a_{x,y,z}}{\left|a_{xyz}\right|} \tag{2-69}$$

donde a_x, a_y, a_z son los vectores de aceleración de los ejes X, Y, Z en base a la posición de la IMU; $\left|a_{xyz}\right|$ es la magnitud vectorial de aceleración de los ejes X, Y, Z en base a la posición de la IMU; y $\vec{D}_{x,y,z}$ es el vector de la dirección de Down de los ejes X, Y, Z en base a la posición de la IMU.

Luego, se obtienen los vectores direccionales del Este mediante el producto vectorial entre los vectores direccionales de Down y del campo magnético, medidos con el magnetómetro.

$$\left|T_{xyz}\right| = \sqrt{(T_x)^2 + \left(T_y\right)^2 + (T_z)^2} \tag{2-70}$$

$$\vec{T}_{x,y,z} = \frac{T_{x,y,z}}{\left|T_{xyz}\right|} \tag{2-71}$$

$$\vec{E}_{x,y,z} = \vec{D}_{x,y,z} \times \vec{T}_{x,y,z} \tag{2-72}$$

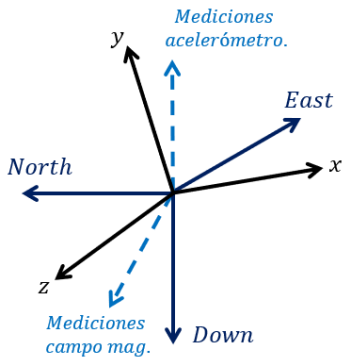

Fig. 2-10 Representación de la proyección del sistema de coordenadas Norte, Este y "Abajo".

Donde T_x, T_y, T_z son los vectores de la fuerza del campo magnético de los ejes X, Y, Z en base a la posición de la IMU; $|T_{xyz}|$ es la magnitud vectorial de la fuerza del campo magnético de los ejes X, Y, Z en base a la posición de la IMU; $\vec{T}_{x,y,z}$ es el vector de las direcciones de las fuerzas del campo magnético de los ejes X, Y, Z en base a la posición de la IMU; y $\vec{E}_{x,y,z}$ es el vector de la dirección de Este de los ejes X, Y, Z en base a la posición de la IMU.

Finalmente, se obtiene Norte como el producto de los vectores direccionales de Este y Down, lo que da como resultado su representación en un marco de referencia local, como se muestra en la Fig. 2-10.

$$\vec{N}_{x,y,z} = \vec{E}_{x,y,z} \times \vec{D}_{x,y,z} \tag{2-73}$$

Una vez obtenidos los tres vectores del sistema NED, es posible obtener los ángulos de Euler al sustituir la representación de los vectores del sistema local NED en la matriz de rotación (2-74).

$$\boldsymbol{Rxyz} = \begin{bmatrix} \vec{N}_x & \vec{E}_x & \vec{D}_x \\ \vec{N}_y & \vec{E}_y & \vec{D}_y \\ \vec{N}_z & \vec{E}_z & \vec{D}_z \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2-74}$$

2.3.6 GNSS

El GNSS (Global Navigation Satellite System) es un conjunto de sistemas de navegación por satélite que permite determinar la ubicación de un objeto en la Tierra mediante una red de satélites que orbitan el planeta. Cada satélite envía mensajes constantes a la Tierra, donde el

receptor GNSS los decodifica para obtener sus datos y luego calcula el tiempo que transcurre entre los mensajes desde la órbita hasta la superficie [11]. Esto permite determinar el tiempo, la velocidad de transmisión y la distancia entre el receptor y el satélite. Sin embargo, al comunicarse con un solo satélite, no es posible obtener una posición precisa. Por lo tanto, el receptor debe comunicarse con múltiples satélites que analicen los puntos en común y determinen una ubicación precisa.

Los satélites se clasifican en grupos denominados constelaciones satelitales, que operan como sistemas que proporcionan cobertura total o parcial de la Tierra. Si un dispositivo GNSS opera con más de una constelación satelital, los resultados son más precisos. Además, si un sistema de satélites no está disponible, el receptor puede utilizar señales de otros sistemas para seguir funcionando [12]. En la TABLA I se muestran las constelaciones de satélite más utilizadas para la navegación global.

El planeta Tierra no es una figura regular debido a su composición mineral interna y a las distintas densidades, por lo que presenta una forma denominada geoide. Esto implica que cada punto de la superficie terrestre se encuentra a una distancia diferente del centro de la Tierra al punto del geoide, distancia a la cual ninguno de los modelos geométricos logra adaptarse por completo debido a las irregularidades del geoide, excepto en ciertas zonas. Para asignar coordenadas geográficas a los distintos puntos de la superficie terrestre, es necesario representar el geoide mediante un elipsoide [10].

TABLA I

CONSTELACIONES SATELITALES DE NAVEGACIÓN

Sistema GNSS	Origen	Región de cobertura
GPS	EUA	Global
GLONASS	Rusia	Global
Galileo	Unión Europea	Global
BeiDou	China	Global
QZSS	Japón	Asía-Oceanía

2.MARCO TEÓRICO

El elipsoide funciona como un "Punto Fundamental" que permite establecer un sistema de ejes coordenados. La referencia geodésica más utilizada en los sistemas GNSS es el Sistema Geodésico Mundial 1984 (WGS84), ya que su "Punto Fundamental" se ubica lo más cerca posible del centro de gravedad de la Tierra.

3. Sistema Integrado de Navegación con IMU y GNSS Basado en el Filtro de Kalman

El sistema aéreo desarrollado para la entrega de carga ligera se compone de una serie de dispositivos electrónicos, sensores, actuadores y módulos de comunicación que, en conjunto, constituyen una arquitectura embebida robusta y adaptable. Cada componente fue cuidadosamente seleccionado en función de sus características técnicas, de su compatibilidad con el sistema y del equilibrio entre el rendimiento y el consumo energético. A continuación, se detallan los principales dispositivos que integran el VANT, tal como se describe en la Fig. 3-1.

3.1. Hardware

3.1.1 Microcontrolador

El microcontrolador ESP32 (Fig. 3-2) es un procesador de doble núcleo con arquitectura Xtensa que opera a 240 MHz y ofrece capacidades avanzadas para sistemas en tiempo real [13]. El ESP32 cuenta con conectividad inalámbrica integrada, Wi-Fi y Bluetooth, lo que permite la comunicación con estaciones remotas sin necesidad de hardware adicional.

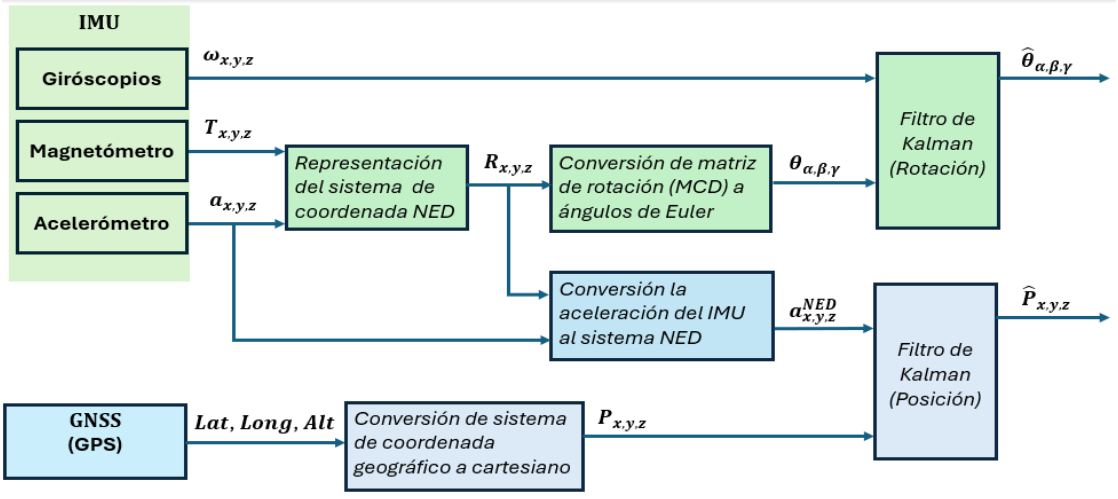

Fig. 3-1 Diagrama general del prototipo del VANT.

Fig. 3-2 Placa de desarrollo ESP32

Gracias a su amplio conjunto de periféricos digitales, como los puertos I2C, SPI y UART, así como las entradas analógicas, el ESP32 permite la comunicación simultánea con múltiples sensores y actuadores. Además, su bajo consumo de energía y su compatibilidad con la programación en C++ y MicroPython lo convierten en una tarjeta de desarrollo ideal para aplicaciones embebidas en drones de pequeño y mediano tamaño.

3.1.1 BNO055

El BNO055 (Fig. 3-3) es un módulo IMU fabricado por Bosch que integra tres sensores: acelerómetro, giroscopio y magnetómetro. Este módulo puede comunicarse mediante los protocolos I2C o UART, lo que facilita su integración con el microcontrolador.

Esta combinación de sensores permite que el módulo actúe como un sensor de 9 grados de libertad (Degrees of Freedom, o DOF por sus siglas). Los sensores de 9-DOF son ideales para el cálculo de la orientación en un estado dinámico al aire libre, cuando hay menores errores de dispersión (drift errors) [14]. Sin embargo, presentan la desventaja de que las mediciones del magnetómetro se ven fácilmente afectadas por alteraciones de los campos magnéticos en ambientes cerrados; además, se recomienda evitar la proximidad de objetos ferromagnéticos que puedan provocar mediciones erróneas.

Además, el BNO055 incorpora un microprocesador interno que permite ejecutar un algoritmo de fusión para obtener otros datos, como la orientación absoluta, ya sea en ángulos de Euler o en cuaterniones.

Fig. 3-3 Módulo BNO055

3.1.2 GNSS NEO-M8N

El GNSS NEO-M8N es un módulo receptor basado en el chip u-blox que permite la comunicación con múltiples constelaciones satelitales, incluyendo GPS, GLONASS, Galileo y BeiDou, lo que mejora considerablemente la velocidad de adquisición y la precisión de la ubicación. Puede proporcionar actualizaciones de posición a una frecuencia de 1 a 10 Hz, con una precisión horizontal aproximada de 2.5 metros en condiciones normales [15].

Puede comunicarse por UART; su bajo consumo de energía lo convierte en un dispositivo confiable para aplicaciones aéreas, especialmente en entornos abiertos. Además, es posible integrar distintos tipos de antena según las necesidades del usuario (Fig. 3-4).

3.1.3 Caracterización de Sensores

En el uso de aplicaciones de logística, cualquier error puede conllevar un resultado ineficiente y, dado que cada sensor de la IMU y del módulo GNSS produce datos con cierto grado de error, estos pueden proporcionar orientación y posición erróneas.

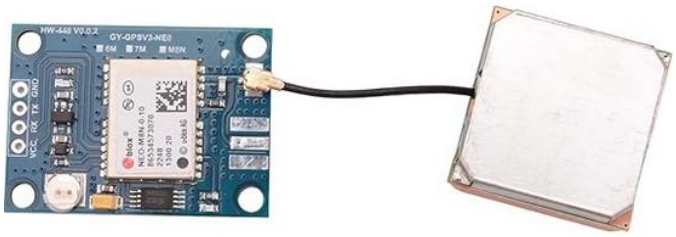

Fig. 3-4 Módulo receptor GNSS NEO-8M con antena cerámica.

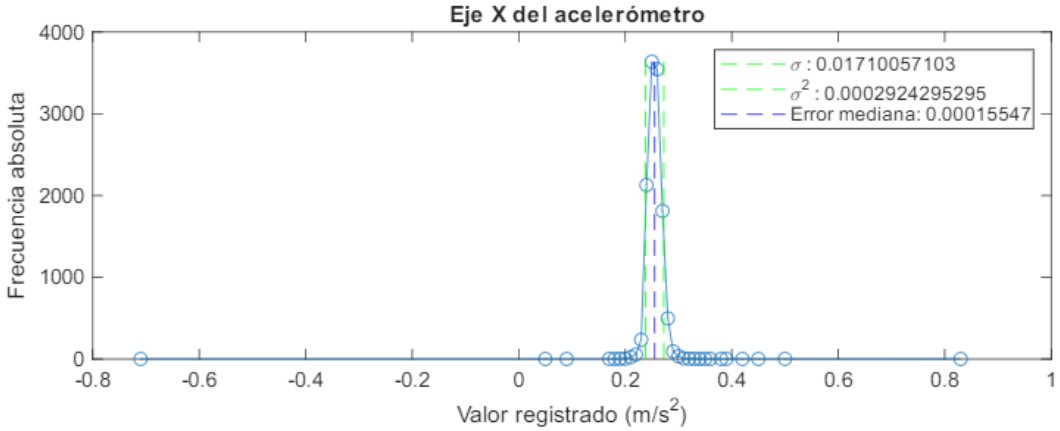

Fig. 3-5 Distribución normal del eje X del acelerómetro.

Para corregirlo, el diseño del algoritmo de fusión de sensores debe incorporar datos de covarianza. La covarianza ayuda a cuantificar los errores tanto de medición de los sensores como de los del proceso del sistema de control; esto permite obtener una estimación más precisa, robusta y fiable de los datos procesados.

Para determinar las covarianzas, primero se deben poner a prueba los tres sensores del módulo IMU. En este caso, se registra la información colocando el módulo en una posición estable e inerte durante un periodo de tiempo, con el fin de conocer la distribución de los datos, su desviación estándar y obtener la covarianza de cada eje de los tres sensores.

En la Fig. 3-5 se observa la distribución normal en un muestreo de mediciones del eje X del acelerómetro.

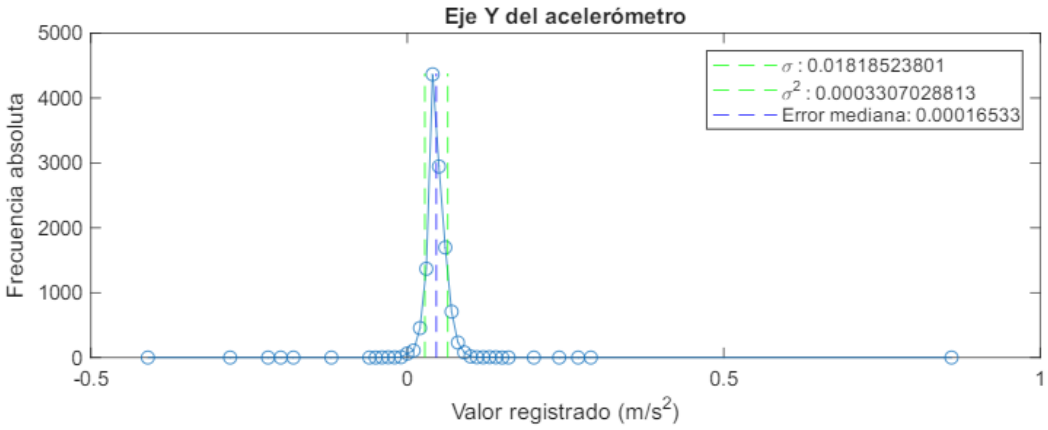

Fig. 3-6 Distribución normal del eje Y del acelerómetro.

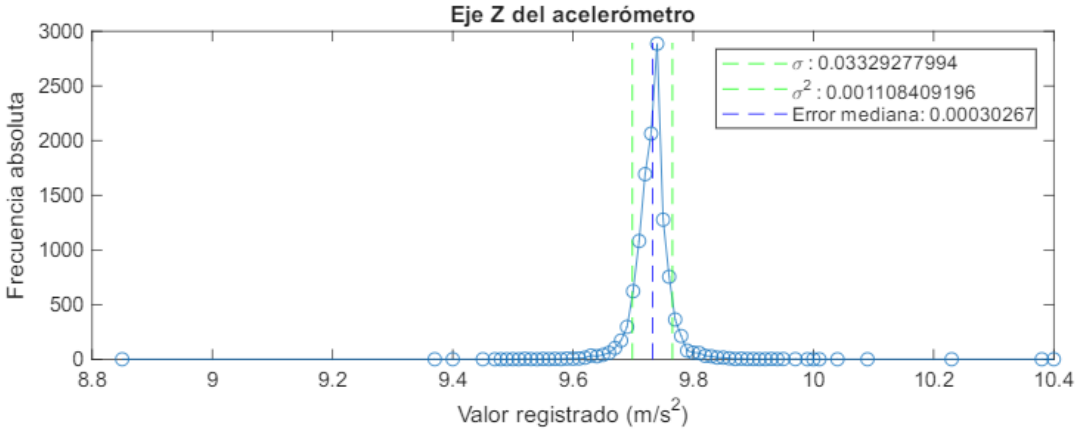

Fig. 3-7 Distribución normal del eje Z del acelerómetro.

En la Fig. 3-6 se observa la distribución normal en un muestreo de mediciones del eje Y del acelerómetro.

En la Fig. 3-7 se observa la distribución normal en un muestreo de mediciones del eje Z del acelerómetro.

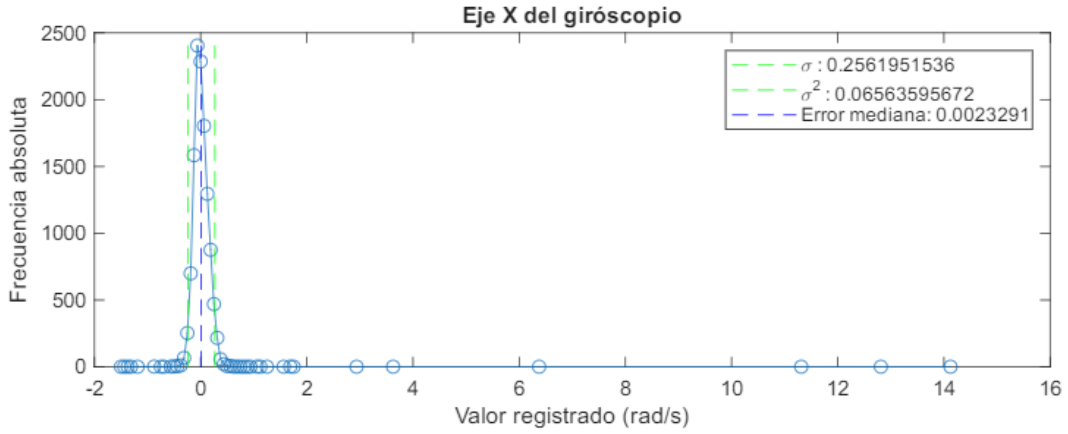

Fig. 3-8 Distribución normal del eje X del giroscopio.

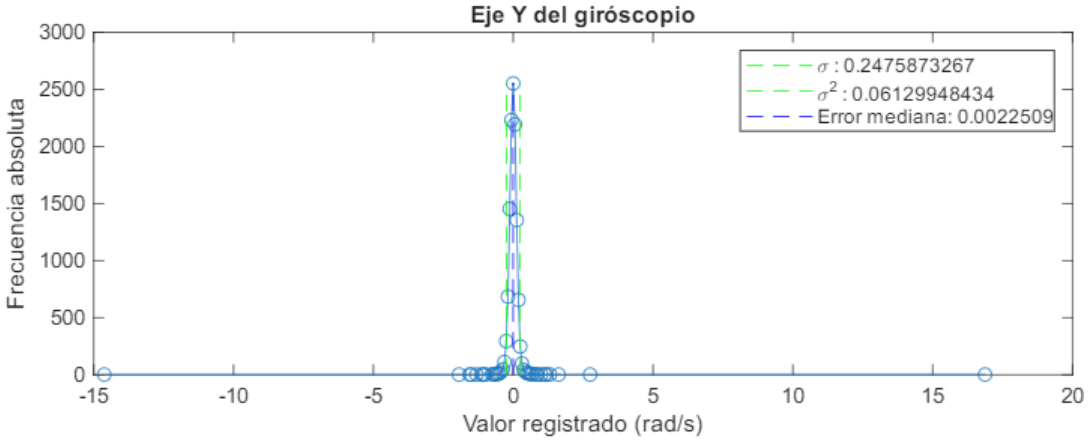

Fig. 3-9 Distribución normal del eje Y del giroscopio.

En la Fig. 3-8 se observa la distribución normal en un muestreo de mediciones del eje X del giroscopio.

En la Fig. 3-9 se observa la distribución normal en un muestreo de mediciones del eje Y del giroscopio.

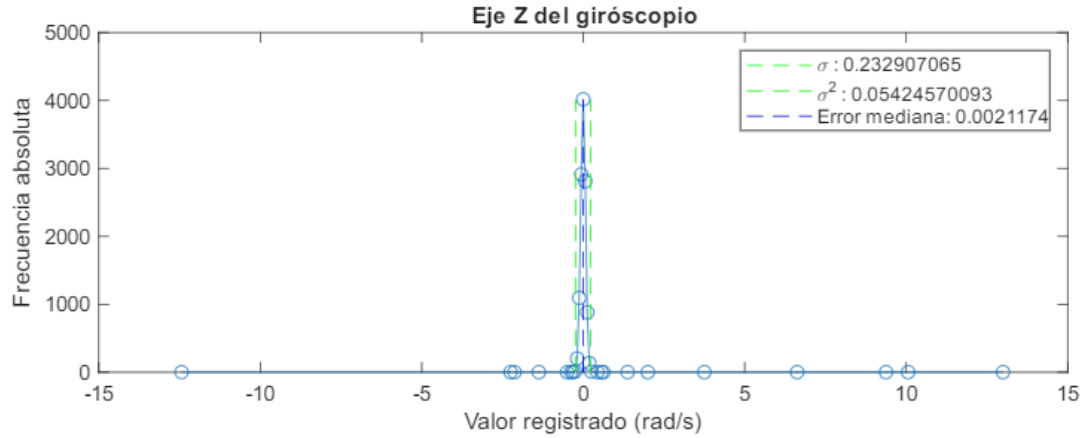

Fig. 3-10 Distribución normal del eje Z del giroscopio.

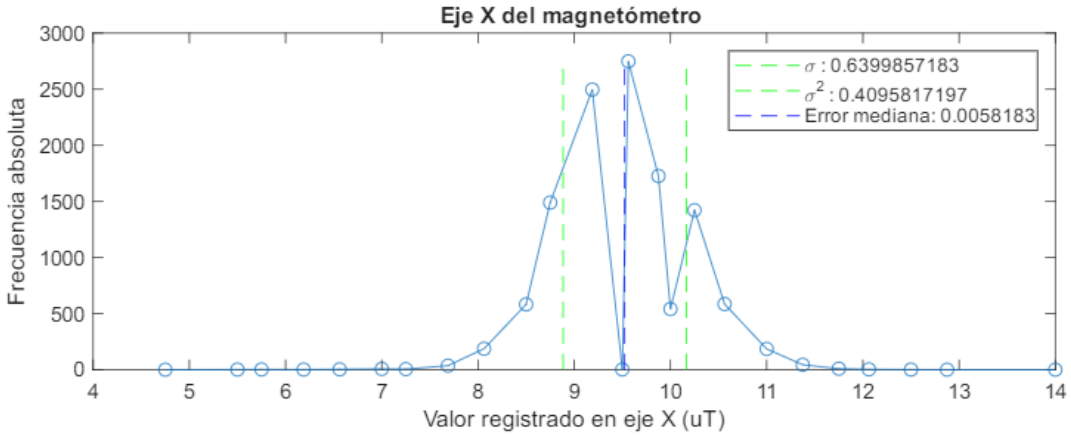

Fig. 3-11 Distribución normal del eje X del magnetómetro.

En la Fig. 3-10 se observa la distribución normal en un muestreo de mediciones del eje Z del giroscopio.

En la Fig. 3-11 se observa la distribución normal en un muestreo de mediciones del eje X del magnetómetro.

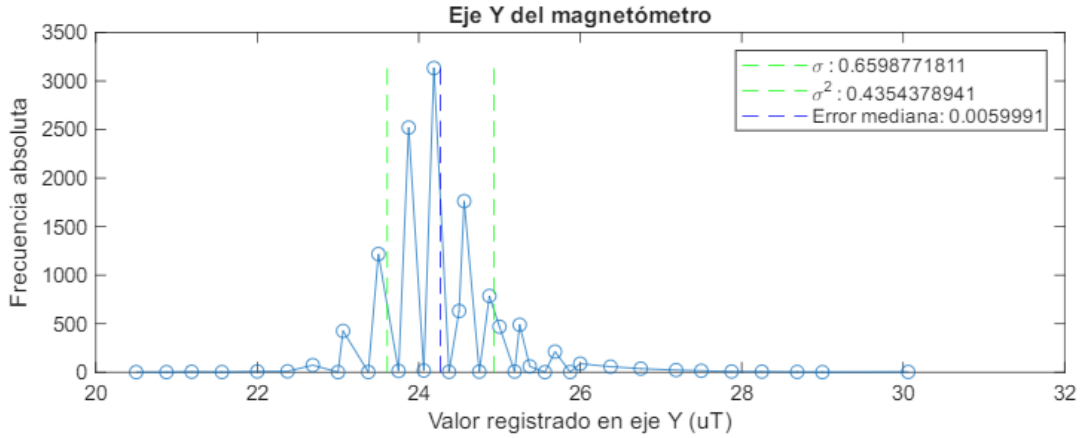

Fig. 3-12 Distribución normal del eje Y del magnetómetro

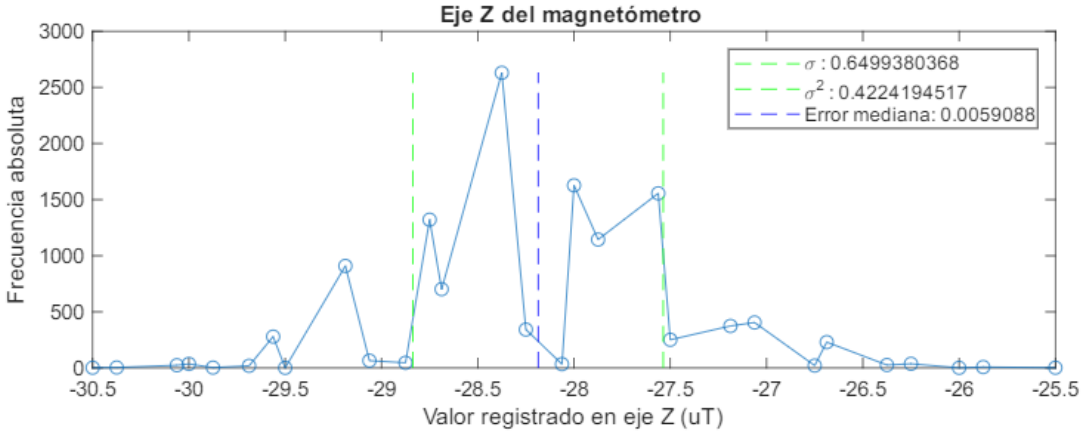

Fig. 3-13 Distribución normal del eje Z del magnetómetro.

En la Fig. 3-12 se observa la distribución normal en un muestreo de mediciones del eje Y del magnetómetro.

En la Fig. 3-13 se observa la distribución normal en un muestreo de mediciones del eje Z del magnetómetro.

Al final, la TABLA II muestra los datos de distribución de los sensores de la IMU. Estos mismos datos se utilizarán en el diseño del filtro de Kalman para la orientación (sección 3.2.1.).

TABLA II
DATOS ESTADÍSTICOS DE LOS SENSORES IMU

Sensor	Eje	Desviación estándar σ	Varianza σ^2	Error de mediana
Acelerómetro	X	0.01710057103	0.0002924295295	0.00015547
	Y	0.01818523801	0.0003307028813	0.00016533
	Z	0.03329277994	0.001108409196	0.00030267
Giroscopio	X	0.2561951536	0.06563595672	0.0023291
	Y	0.2475873267	0.06129948434	0.0022509
	Z	0.232907065	0.05424570093	0.0021174
Magnetómetro	X	0.6399857183	0.4095817197	0.0058183
	Y	0.6598771811	0.4354378941	0.0059991
	Z	0.6499380368	0.4224194517	0.0059088

Información del proyecto	Sistema de coordenadas	
Nombre:	Nombre:	UTM
Tamaño:	Datum:	ITRF 2005
Modificado/a:	Zona:	13 North (105W)
Número de referencia:	Geoide:	MEXICO97 (Mexico)
Descripción:	Datum vertical:	

Lista de puntos

ID	Latitud (Local)	Longitud (Local)	Altura (Local) (Metro)
BASE-COMUNA	N20°40'09.79978"	O103°22'19.47438"	1560.706
ITESO-01	N20°36'24.10626"	O103°25'04.32286"	1584.849
ITESO-02	N20°36'24.04983"	O103°25'02.26791"	1585.738

Fig. 3-14 Puntos de referencia geográficos dentro del plantel del ITESO.

De igual forma, el módulo GNSS se somete a prueba en una posición estable e inerte durante un periodo determinado para evaluar la distribución de los datos. Para obtener esos datos estadísticos, se realizan pruebas de campo en un punto geográfico de referencia conocido, con el fin de observar la variabilidad al colocar el sensor en ese punto.

El punto de referencia seleccionado fue el ITESO-02 (Fig. 3-14), ubicado dentro del plantel, cerca de la entrada sur, entre el edificio H y el edificio de servicio médico y enfermería.

En la Fig. 3-15 se observa el mapeo de los datos de las coordenadas del módulo GNSS en el punto de referencia ITESO-02.

Fig. 3-15 Prueba del módulo GNSS en punto de referencia ITESO-02.

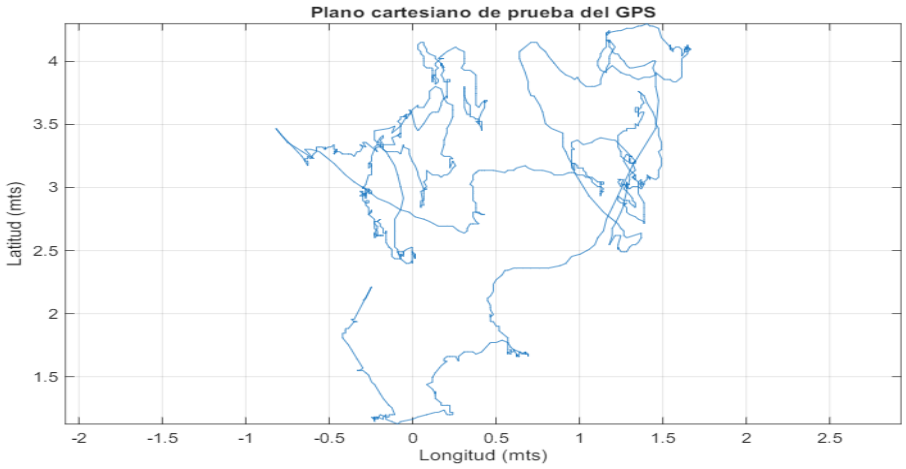

Fig. 3-16 Representación cartesiana de la prueba del módulo GNSS en ITESO-02.

En la Fig. 3-16 se representan las coordenadas geográficas en un plano cartesiano, con el origen del plano en el punto de referencia geográfico ITESO-02. Al graficar los resultados, se analiza la dispersión respecto del punto de origen y de todas las ubicaciones obtenidas en el plano cartesiano, con el objetivo de observar la variación del receptor GNSS desde el inicio del registro de su ubicación geográfica hasta que logra calibrarse.

El análisis de los datos de la TABLA III muestra que el receptor tiende a variar en 3 metros en la latitud (eje Y) desde el punto de referencia establecido como origen del plano, mientras que no se identifica una variación significativa en la longitud (eje X). Por lo tanto, las variaciones en las longitudes obtenidas tienden a afectar la posición del receptor respecto de la esperada.

Para la fusión GNSS+IMU, es importante ajustar los parámetros de ambos sensores para el procesamiento de datos en el filtro de Kalman. La IMU proporciona los cambios de movimiento

TABLA III

TABLA DE LOS DATOS DE DISPERSIÓN DE LA PRUEBA DEL GNSS

Parámetros	Promedio	Media	Desviación estándar σ	Varianza σ^2
X	0.4987	0.3624	0.7722	0.5963
Y	3.0519	3.1275	0.6551	0.4292
Hipotenusa	3.1525	3.2741	0.8057	0.6491

(aceleración y velocidad angular) en un marco de referencia local y relativo, mientras que el módulo GNSS proporciona datos de posición en un marco de referencia global. Para que el proceso de fusión de ambos sensores pueda realizarse, es necesario alinear ambos dispositivos con un sistema de referencia; en este caso, con el sistema NED.

La aceleración se considera el dato de entrada del modelo lineal de espacio de estados para el diseño de la fusión GNSS+IMU. Pero, como su marco de referencia local se basa en la posición de su cuerpo, se emplean los ángulos de Euler para la conversión. Como los ángulos $\theta_{\alpha,\beta,\gamma}$ se representan con respecto a la Tierra (NED), la aceleración se representa en el sistema NED al realizar el producto de la matriz de rotación (2-17) por los vectores de aceleración X, Y, Z;

$$\begin{bmatrix} a_x^{NED}(t) \\ a_y^{NED}(t) \\ a_z^{NED}(t) \end{bmatrix} = \boldsymbol{R_{XYZ}} * \begin{bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{bmatrix} \tag{3-1}$$

donde $a_x^{NED}(t)$, $a_y^{NED}(t)$, $a_z^{NED}(t)$ son las aceleraciones de los ejes X, Y, Z en el sistema de referencia NED; $\boldsymbol{R_{XYZ}}$ es la matriz de rotación relacionado con los ángulos de Euler; y $a_x(t)$, $a_y(t)$, $a_z(t)$ son las aceleraciones de los ejes X, Y, Z con respecto a la posición de la IMU.

En el caso de los parámetros de salida del sistema de fusión de sensores, se procesan la posición y la velocidad de los cuerpos, de igual forma, con base en el sistema de referencia NED.

Para obtener los valores de posición, primero se deberán convertir los valores del módulo GNSS, expresados en grados, a radianes.

$$\phi_n = \measuredangle\phi_n * \left(\frac{\pi}{180°}\right) \tag{3-2}$$

$$\lambda_n = \measuredangle\lambda_n * \left(\frac{\pi}{180°}\right) \tag{3-3}$$

Donde ϕ_n es la latitud (en radianes) obtenida por el módulo GNSS, y λ_n es la longitud (en radianes) obtenida del módulo GNSS.

El siguiente paso es obtener los valores de las posiciones X, Y, Z en base al marco de referencia NED por medio del radio de la Tierra:

$$X_n^{NED} = Radio_{Tierra} * (\phi_n - \phi_0) \tag{3-4}$$

$$Y_n^{NED} = Radio_{Tierra} * \cos(\phi_n) * (\lambda_{n_n} - \lambda_{n_0}) \tag{3-5}$$

$$Z_n^{NED} = Alt_0 - Alt_n \tag{3-6}$$

donde X_n^{NED},Y_n^{NED},Z_n^{NED} son las posiciones de los ejes X, Y, Z en base al sistema NED; Alt_n es la altitud (en metros) obtenida por el módulo GNSS; y $Radio_{Tierra} = 6{,}371\ km$.

Con los datos de posición obtenidos, se pueden obtener los datos de velocidad que permiten complementar el filtro de Kalman para la fusión de sensores:

$$v_{x,n}^{NED} = \left(X_n^{NED} - X_{n-1}^{NED}\right) \Big/ \Delta t_{GPS} \tag{3-7}$$

$$v_{y,n}^{NED} = \left(Y_n^{NED} - Y_{n-1}^{NED}\right) \Big/ \Delta t_{GPS} \tag{3-8}$$

$$v_{z,n}^{NED} = \left(Z_k^{NED} - Z_{n-1}^{NED}\right) \Big/ \Delta t_{GPS} \tag{3-9}$$

donde $v_{x,n}^{NED}$, $v_{y,n}^{NED}$, $v_{z,n}^{NED}$ son las velocidades de los ejes X, Y, Z en base al sistema NED, y Δt_{GPS} es el tiempo de muestreo del módulo GNSS para actualizar sus datos. El módulo está configurado a un muestreo de tiempo de 0.5 segundos.

3.1.4 Interconexión de Componentes

El montaje del hardware para la obtención de datos durante el experimento se realizó según el esquema mostrado de la Fig. 3-17, donde el módulo IMU se colocó en el centro del prototipo de prueba. Al situarse en un punto de referencia centrado, se obtienen datos de orientación lo más precisos posible, lo que permite reducir errores de medición como las vibraciones y las fuerzas desproporcionadas.

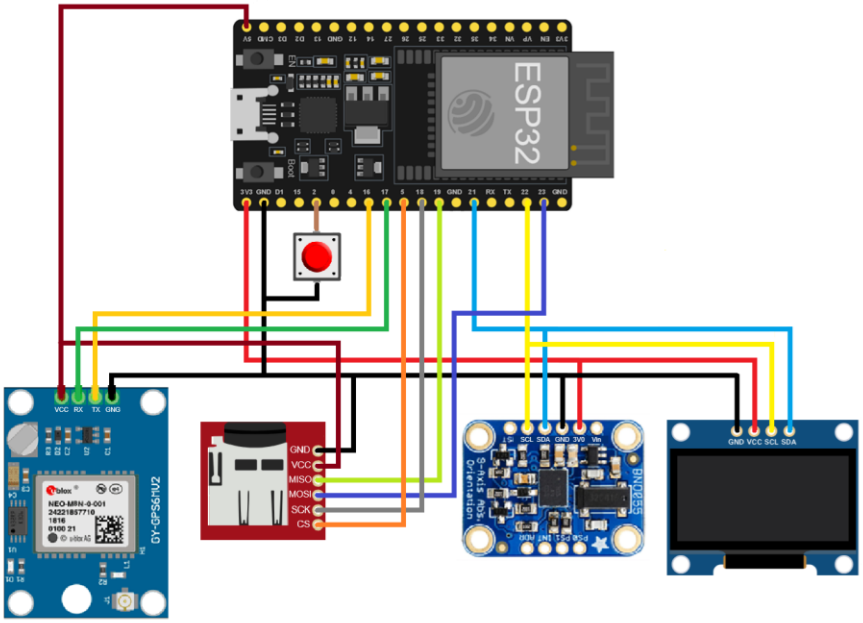

Fig. 3-17 Interconexión del prototipo VANT.

Además, se incorporó un display OLED para monitorear la calibración y los resultados de las pruebas de campo. También se añadió un botón push al microcontrolador, el cual, al ser presionado, inicia la ejecución del programa y el registro de los resultados en la memoria microSD externa.

3.2. Firmware

3.2.1 Períodos de muestreo del microcontrolador

Esta experimentación previa consiste en obtener los tiempos de muestreo de cada sensor y dispositivo que se emplearán en el sistema embebido. El propósito es definir períodos de muestreo en el microcontrolador, dado que todos los datos se almacenarán en una unidad de memoria SD externa, la cual permite conservar una gran cantidad de mediciones, además de los valores procesados y filtrados según el sistema implementado en el VANT.

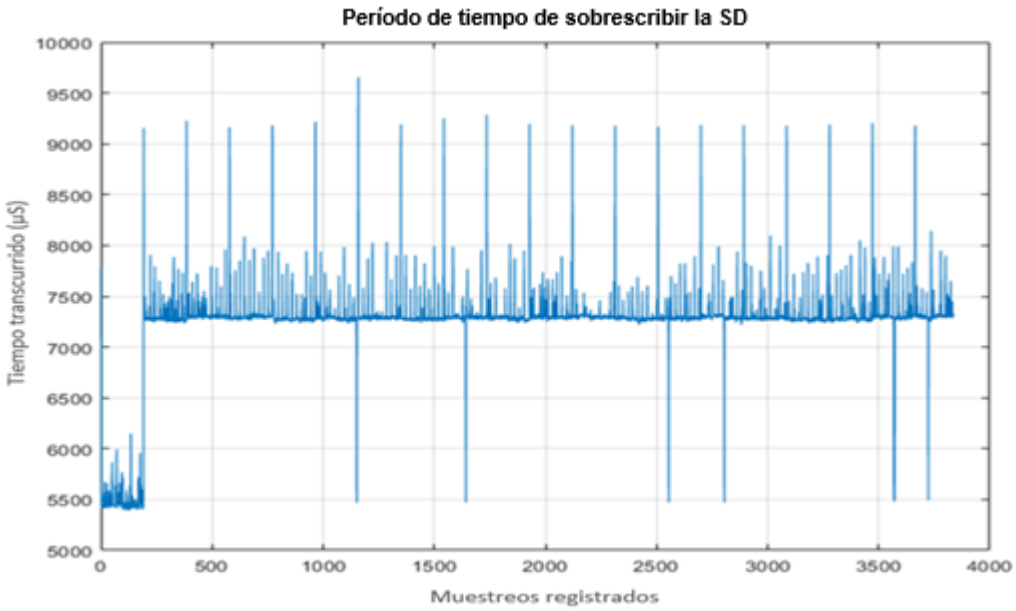

Fig. 3-18 Registro de tiempo en la sobrescritura en la memoria SD.

Si bien se contempla un tiempo de muestreo del módulo IMU, también debe considerarse el tiempo que tarda el algoritmo del microcontrolador en abrir y sobrescribir el registro de datos. Por lo tanto, se midieron los tiempos de sobrescritura en la memoria SD, como se muestra en la Fig. 3-18 y en la TABLA IV.

TABLA IV
PERÍODOS DE TIEMPO DE SOBRESCRITURA EN LA MEMORIA SD

Dispositivo	Tiempo mínimo	Tiempo máximo	Tiempo mediana	Tiempo promedio
Sobreescritura de SD	5.39 ms	9.65 ms	7.29 ms	7.23 ms

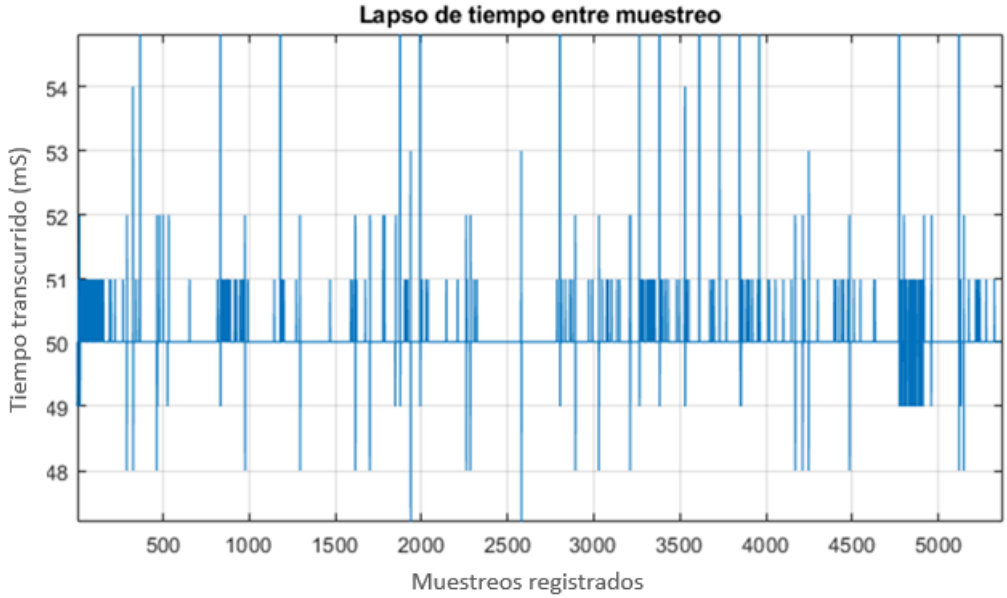

Fig. 3-19 Tiempos de muestreo del sensor IMU en la ESP32.

Considerando el período máximo de cada dispositivo, se estableció que el tiempo de muestreo y de sobrescritura del registro fuera de al menos 20 mS. Sin embargo, después de varias pruebas, se optó por utilizar 50 mS para el período de las tareas de muestreo del módulo BNO085 y la sobrescritura en la memoria SD, al ser el período de tiempo más corto obtenido de forma estable.

El análisis de la Fig. 3-19 indica períodos superiores a 50 mS, aunque estos solo se presentan aproximadamente 15 veces de 5000 muestras capturadas. Además, en la TABLA V, tanto el promedio como la mediana del tiempo se aproxima a los 50 mS.

TABLA V

REGISTRO DE TIEMPOS DEL TASK SCHEDULER PARA LA IMU EN LA ESP32

Dispositivo	Tiempo mínimo	Tiempo máximo	Tiempo mediana	Tiempo promedio
Microcontrolador	47 mS	157 mS	50 mS	50.26 mS

3.2.1 Implementación de Filtro de Kalman para Estimación de Orientación

En primer lugar, se identifica el modelo del espacio de estados de la IMU.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{w}(t) \tag{3-10}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{v}(t) \tag{3-11}$$

Donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son las matrices del sistema; $\boldsymbol{w}(t)$ el ruido gaussiano con media (μ) igual a 0 y covarianza Q; $\boldsymbol{v}(t)$ el ruido gaussiano con media (μ) igual a 0 y covarianza R; $\boldsymbol{x}(t)$ es el valor del estado del modelo; y $\boldsymbol{y}(t)$ es el valor medido con magnetómetros y acelerómetros

Posteriormente, se definen las variables del sistema.

$$\boldsymbol{A} = \boldsymbol{0_3} \tag{3-12}$$

$$\boldsymbol{B} = \boldsymbol{I_3} \tag{3-13}$$

$$\boldsymbol{C} = \boldsymbol{I_3} \tag{3-14}$$

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} \theta_\alpha(t) \\ \theta_\beta(t) \\ \theta_\gamma(t) \end{bmatrix} \tag{3-15}$$

$$\boldsymbol{u}(t) = \begin{bmatrix} \omega_x(t) \\ \omega_y(t) \\ \omega_z(t) \end{bmatrix} \tag{3-16}$$

$$\boldsymbol{y}(t) = \begin{bmatrix} \theta_\alpha(t) \\ \theta_\beta(t) \\ \theta_\gamma(t) \end{bmatrix} \tag{3-17}$$

Donde $\theta_{\alpha,\beta,\gamma}$ son los ángulos de Euler α, β, γ del dispositivo IMU; y $\omega_{x,y,z}$ son las velocidades angulares de los ejes X, Y, Z de la IMU, respectivamente.

El modelo continuo se convierte en discreto.

$$\boldsymbol{x_k} = \boldsymbol{A_d}\boldsymbol{x_{k-1}} + \boldsymbol{B_d}\boldsymbol{u_{k-1}} + \boldsymbol{w_{k-1}} \tag{3-18}$$

$$\boldsymbol{y_k} = \boldsymbol{C_d}\boldsymbol{x_k} + \boldsymbol{v_k} \tag{3-19}$$

$$\boldsymbol{A_d} = (\boldsymbol{A}T + \boldsymbol{I_3}) = \boldsymbol{I_3} \tag{3-20}$$

$$\boldsymbol{B_d} = \boldsymbol{B}T = \begin{bmatrix} \boldsymbol{0_3} \\ \boldsymbol{TI_3} \end{bmatrix} \tag{3-21}$$

$$\boldsymbol{C_d = C = I_3} \tag{3-22}$$

Donde T es el período de muestreo de la IMU, que definimos como 50 mS.

A continuación, se obtiene la etapa de predicción.

$$\boldsymbol{\hat{x}_k^- = A_d \hat{x}_{k-1} + B_d u_{k-1}} \tag{3-23}$$

$$\boldsymbol{P_k^- = A_d P_{k-1} A_d^T + Q} \tag{3-24}$$

Donde $\boldsymbol{u_{k-1}}$ es la velocidad angular X, Y, Z de la IMU definido como:

$$\boldsymbol{u_{k-1}} = \begin{bmatrix} \omega_{x,k-1} \\ \omega_{y,k-1} \\ \omega_{z,k-1} \end{bmatrix} \tag{3-25}$$

Q es la covarianza del ruido de medición:

$$\boldsymbol{Q} = \begin{bmatrix} 0.065636 & 0 & 0 \\ 0 & 0.061299 & 0 \\ 0 & 0 & 0.054245 \end{bmatrix} \tag{3-26}$$

$\boldsymbol{\hat{x}_0^-}$ es el valor inicial del estado del modelo $\boldsymbol{\hat{x}_k}$ para los ángulos de Euler:

$$\boldsymbol{\hat{x}_0^-} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3-27}$$

$\boldsymbol{P_0}$ es el valor inicial del modelo de predicción P_k para los ángulos de Euler con la forma

$$\boldsymbol{P_0 = I_3} \tag{3-28}$$

Finalmente, se realiza la etapa de corrección, comenzando con el cálculo de la ganancia de Kalman:

$$\boldsymbol{K_k = P_k^- C_d \left(C_d P_k^- C_d^T + R \right)^{-1}} \tag{3-29}$$

Actualizando la covarianza de error:

$$\boldsymbol{P_k = A_d P_k^- A_d^T - A_d P_k^- C_d^T R^{-1} C_d P_k^- A_d^T + Q} \text{ (Libros)} \tag{3-30}$$

$$\boldsymbol{P_k = (I_3 - K_k C_d) P_k^-} \text{ (Matlab)} \tag{3-31}$$

Y finalmente actualizando la estimación con la medida:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}} = \widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} + \boldsymbol{K}_{\boldsymbol{k}}(\boldsymbol{y}_{\boldsymbol{k}} - \boldsymbol{C}_{\boldsymbol{d}}\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}) \tag{3-32}$$

Donde $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$ es predicción de los ángulos de Euler α, β, γ basada únicamente en el modelo de la IMU:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \begin{bmatrix} \hat{\theta}_{\alpha,k}^{-} \\ \hat{\theta}_{\beta,k}^{-} \\ \hat{\theta}_{\gamma,k}^{-} \end{bmatrix} \tag{3-33}$$

$\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ es la estimación de los ángulos de Euler α, β, γ:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}} = \begin{bmatrix} \hat{\theta}_{\alpha,k} \\ \hat{\theta}_{\beta,k} \\ \hat{\theta}_{\gamma,k} \end{bmatrix} \tag{3-34}$$

$\boldsymbol{y}_{\boldsymbol{k}}$ son los ángulos de Euler α, β, γ obtenidos:

$$\boldsymbol{y}_{\boldsymbol{k}} = \begin{bmatrix} \boldsymbol{\theta}_{\alpha,k} \\ \boldsymbol{\theta}_{\beta,k} \\ \boldsymbol{\theta}_{\gamma,k} \end{bmatrix} \tag{3-35}$$

R es la covarianza del ruido del proceso:

$$\boldsymbol{R} = \begin{bmatrix} 0.409581 & 0 & 0 \\ 0 & 0.435438 & 0 \\ 0 & 0 & 0.422419 \end{bmatrix} \tag{3-36}$$

3.2.2 Implementación de Filtro de Kalman para Estimación de Posición

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{A}\boldsymbol{X}(t) + \boldsymbol{B}\tilde{\boldsymbol{u}}(t) \tag{3-37}$$

$$\tilde{\boldsymbol{y}}(t) = \boldsymbol{C}\boldsymbol{X}(t) + \boldsymbol{n}_{\boldsymbol{R}} \tag{3-38}$$

Donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son las matrices del sistema; $\tilde{\boldsymbol{u}}(\boldsymbol{t})$ es la aceleración en formato NED de la IMU más el ruido gaussiano con media (μ) igual a 0 y covarianza **Q**; $\boldsymbol{n}_{\boldsymbol{R}}$ el ruido gaussiano con media (μ) igual a 0 y covarianza **R** del módulo GNSS; $\boldsymbol{X}(t)$ es el valor del estado del modelo; y $\tilde{\boldsymbol{y}}(t)$ es el valor medido del GNSS con ruido.

Se define el modelo en espacio de estados de la IMU + GNSS.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0_3} & \boldsymbol{I_3} \\ \boldsymbol{0_3} & \boldsymbol{0_3} \end{bmatrix} \tag{3-39}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0_3} \\ \boldsymbol{I_3} \end{bmatrix} \tag{3-40}$$

$$\boldsymbol{C} = [\boldsymbol{I_3} \quad \boldsymbol{0_3}] \tag{3-41}$$

$$\dot{\boldsymbol{X}}(\boldsymbol{t}) = \begin{bmatrix} X^{NED}(t) \\ Y^{NED}(t) \\ Z^{NED}(t) \\ v_x^{NED}(t) \\ v_y^{NED}(t) \\ v_z^{NED}(t) \end{bmatrix} \tag{3-42}$$

$$\tilde{\boldsymbol{u}}(\boldsymbol{t}) = \begin{bmatrix} a_x^{NED}(t) \\ a_y^{NED}(t) \\ a_z^{NED}(t) \end{bmatrix} \tag{3-43}$$

Donde $X^{NED}(t)$, $Y^{NED}(t)$, $Z^{NED}(t)$ son las posiciones de los ejes X, Y, Z en base a las coordenadas NED; $v_x^{NED}(t)$, $v_y^{NED}(t)$, $v_z^{NED}(t)$ son las velocidades de los ejes X, Y, Z en base a las coordenadas NED; y a_x^{NED}, a_y^{NED}, a_z^{NED} son las aceleraciones de los ejes X, Y, Z en base a las coordenadas NED.

El modelo continuo se convierte en discreto.

$$\boldsymbol{X_{k+1}} = \boldsymbol{A_d X_k} + \boldsymbol{B_d \tilde{u}_k} \tag{3-44}$$

$$\tilde{\boldsymbol{y}}_{\boldsymbol{k}} = \boldsymbol{C_d X_k} + \boldsymbol{n_R} \tag{3-45}$$

$$\boldsymbol{A_d} = (\boldsymbol{AT} + \boldsymbol{I_6}) = \begin{bmatrix} \boldsymbol{I_3} & \boldsymbol{TI_3} \\ \boldsymbol{0_3} & \boldsymbol{I_3} \end{bmatrix} \tag{3-46}$$

$$\boldsymbol{B_d} = \boldsymbol{BT} = \begin{bmatrix} \boldsymbol{0_3} \\ \boldsymbol{TI_3} \end{bmatrix} \tag{3-47}$$

$$\boldsymbol{C_d} = \boldsymbol{C} = [\boldsymbol{I_3} \quad \boldsymbol{0_3}] \tag{3-48}$$

Con T como el período de muestreo del acelerómetro, definido como 50 mS.

A continuación, se obtienen la etapa de predicción, cuyo modelo resulta en:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \boldsymbol{A_d}\hat{\boldsymbol{x}}_{\boldsymbol{k-1}} + \boldsymbol{B_d}\tilde{\boldsymbol{u}}_{\boldsymbol{k}} \tag{3-49}$$

$$\boldsymbol{P_k^-} = \boldsymbol{A_d P_{k-1} A_d^T} + \boldsymbol{Q} \tag{3-50}$$

Donde $\tilde{\boldsymbol{u}}_{\boldsymbol{k}}$ es la aceleración en formato NED de la IMU:

$$\tilde{\boldsymbol{u}}_{\boldsymbol{k}} = \begin{bmatrix} a_{x,k}^{NED} \\ a_{y,k}^{NED} \\ a_{z,k}^{NED} \end{bmatrix} \tag{3-51}$$

Q es la covarianza del ruido de medición del acelerómetro:

$$\boldsymbol{Q} = \begin{bmatrix} 17.1\text{e}-03 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18.18\text{e}-03 & 0 & 0 & 0 & 0 \\ 0 & 0 & 33.29\text{e}-03 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17.1\text{e}-03 & 0 & 0 \\ 0 & 0 & 0 & 0 & 18.18\text{e}-03 & 0 \\ 0 & 0 & 0 & 0 & 0 & 33.29\text{e}-03 \end{bmatrix} \tag{3-52}$$

$\hat{\boldsymbol{x}}_{\boldsymbol{0}}^{-}$ es el valor inicial del estado del modelo $\hat{x}_k$ para las posiciones y velocidades en base a las coordenadas del GNSS:

$$\hat{\boldsymbol{x}}_{\boldsymbol{0}}^{-} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3-53}$$

$\boldsymbol{P_0}$ es el valor inicial del modelo de predicción $\boldsymbol{P_k}$ para las posiciones y velocidades:

$$\boldsymbol{P_0} = \boldsymbol{I_6} \tag{3-54}$$

Finalmente, se realiza la etapa de corrección, comenzando con el cálculo de la ganancia de Kalman:

$$\boldsymbol{K_k} = \boldsymbol{P_k^-}\boldsymbol{C_d}(\boldsymbol{C_d}\boldsymbol{P_k^-}\boldsymbol{C_d^T} + \boldsymbol{R})^{-1} \tag{3-55}$$

Fusión:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} + \boldsymbol{K_k}(\tilde{\boldsymbol{y}}_{\boldsymbol{k}} - \boldsymbol{C_d}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}) \tag{3-56}$$

Recursividad:

$$\boldsymbol{P_k} = (\boldsymbol{I_6} - \boldsymbol{K_k}\boldsymbol{C_d})\boldsymbol{P_k^-} \tag{3-57}$$

Donde $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$ es la predicción de las posiciones y velocidades de los ejes X, Y, Z en base a las coordenadas NED:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \begin{bmatrix} \hat{X}_k^{-NED} \\ \hat{Y}_k^{-NED} \\ \hat{Z}_k^{-NED} \\ \hat{v}_{x,k}^{-NED} \\ \hat{v}_{y,k}^{-NED} \\ \hat{v}_{z,k}^{-NED} \end{bmatrix} \tag{3-58}$$

$\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ es la estimación de las posiciones y velocidades de los ejes X, Y, Z en base a las coordenadas NED:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}} = \begin{bmatrix} \hat{X}_k^{NED} \\ \hat{Y}_k^{NED} \\ \hat{Z}_k^{NED} \\ \hat{v}_{x,k}^{NED} \\ \hat{v}_{y,k}^{NED} \\ \hat{v}_{z,k}^{NED} \end{bmatrix} \tag{3-59}$$

$\boldsymbol{y}_{\boldsymbol{k}}$ es el conjunto de las posiciones y velocidades de los ejes X, Y, Z en base a las coordenadas NED obtenidos:

$$\boldsymbol{y}_{\boldsymbol{k}} = \begin{bmatrix} X_k^{NED} \\ Y_k^{NED} \\ Z_k^{NED} \\ v_{x,k}^{NED} \\ v_{y,k}^{NED} \\ v_{z,k}^{NED} \end{bmatrix} \tag{3-60}$$

R es la covarianza del ruido del proceso:

$$\boldsymbol{R} = \begin{bmatrix} 0.6012 & 0 & 0 \\ 0 & 0.4276 & 0 \\ 0 & 0 & 15.0659 \end{bmatrix} \tag{3-61}$$

4. Resultados Experimentales

Antes de implementar el sistema completo de control y navegación del VANT, se lleva a cabo una etapa de evaluación individual de los sensores y de los componentes electrónicos. Esta fase tiene el objetivo de conocer el comportamiento real de cada dispositivo, así como su nivel de ruido, precisión, tiempo de respuesta y compatibilidad, con el fin de su posterior integración en un sistema de fusión sensorial.

Este enfoque permite anticipar errores, ajustar los parámetros de muestreo y comprender las limitaciones prácticas de cada sensor o módulo, lo cual resulta fundamental para la estabilidad y la eficiencia del sistema embebido de control.

4.1. Diseño de filtro de Kalman de la IMU

4.1.1 Prueba de filtro de Kalman

Para verificar el funcionamiento del filtro de Kalman desarrollado para el ESP32, se realiza un experimento integrando el prototipo del VANT en un brazo robótico. El objetivo consistió en mover el brazo del robot en los tres ejes del sensor IMU a distintos ángulos, con el fin de analizar la precisión y las variaciones en comparación con los ángulos preprogramados.

No solo se comparan los resultados esperados con los proporcionados por el filtro del microcontrolador, sino que también se contrastaran con la respuesta de salida del IMU mediante su propia librería. Además, se incluyó un filtro de Kalman diseñado y procesado en un entorno de programación diferente al del ESP32, en este caso, Matlab. Todo esto para determinar qué diseño de filtro de Kalman se aproxima mejor a la realidad y es menos propenso a errores.

Con la finalidad de obtener mejores resultados, se realizaron tres experimentos con distintas secuencias de movimiento del brazo para verificar si existían anomalías o diferencias en la respuesta entre los filtros.

4.RESULTADOS EXPERIMENTALES

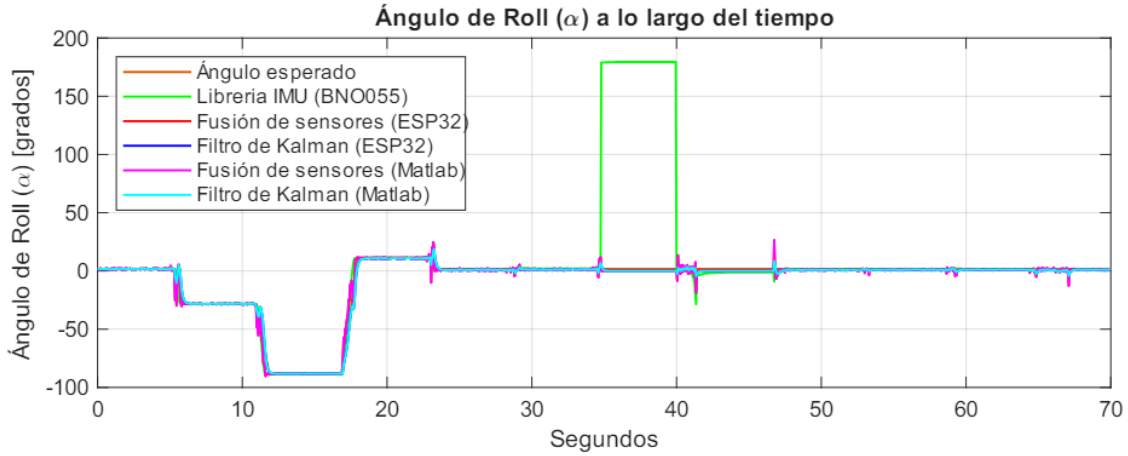

Fig. 4-1 Graficación del experimento 1 para la comparación de resultados del ángulo α.

En la Fig. 4-1 se muestra la comparación del ángulo α esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 1.

En la Fig. 4-2 se muestra la comparación del ángulo β esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 1.

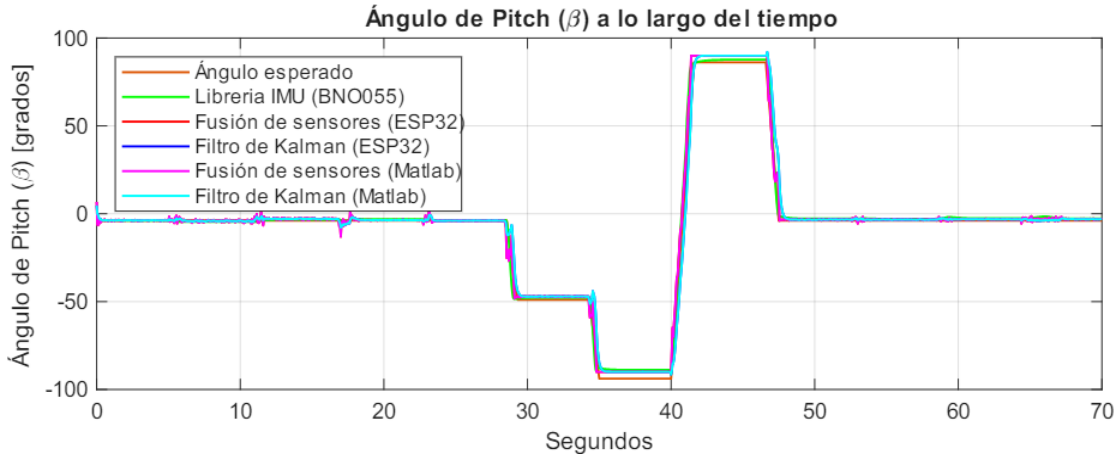

Fig. 4-2 Graficación del experimento 1 para la comparación de resultados del ángulo β.

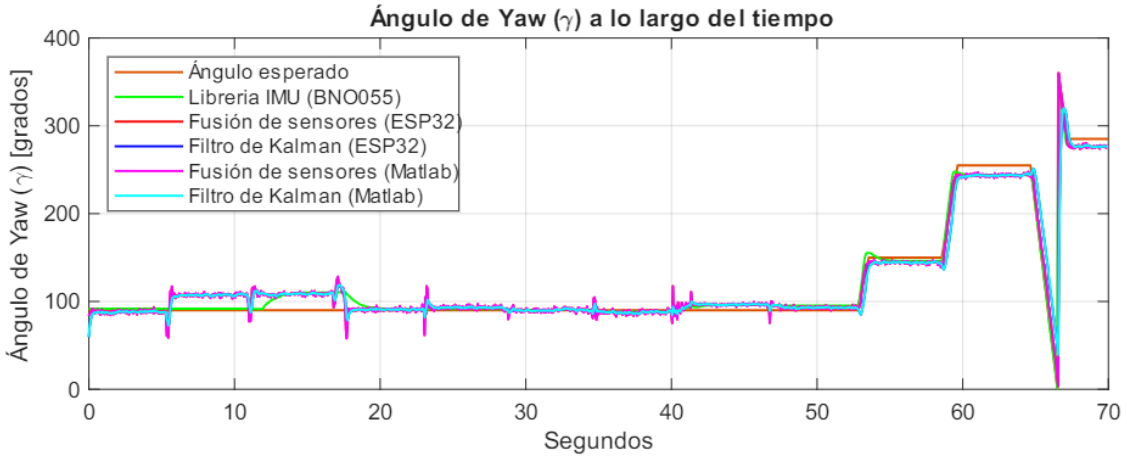

Fig. 4-3 Graficación del experimento 1 para la comparación de resultados del ángulo γ.

En la Fig. 4-3 se muestra la comparación del ángulo γ esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 1.

TABLA VI

DATOS DE DISTRIBUCIÓN D DEL EXPERIMENTO CON IMU 1

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
Librería BNO055	α	14.22299353981	0.375	46.37819106258
	β	1.60473386356	1.0625	2.66940909593
	γ	5.03401473061	2	5.38263401008
Filtro de Kalman (ESP32)	α	1.17029631215	0.7418	1.93630053105
	β	1.92426181299	0.7506	3.86637277984
	γ	8.59136329069	5.198	13.93846250773
Filtro de Kalman (Matlab)	α	1.17192505062	0.74659440931	1.93511001344
	β	1.93352171668	0.69780052285	3. 89077238959
	γ	8.60351769831	5.18519988215	13.96817613202

4.RESULTADOS EXPERIMENTALES

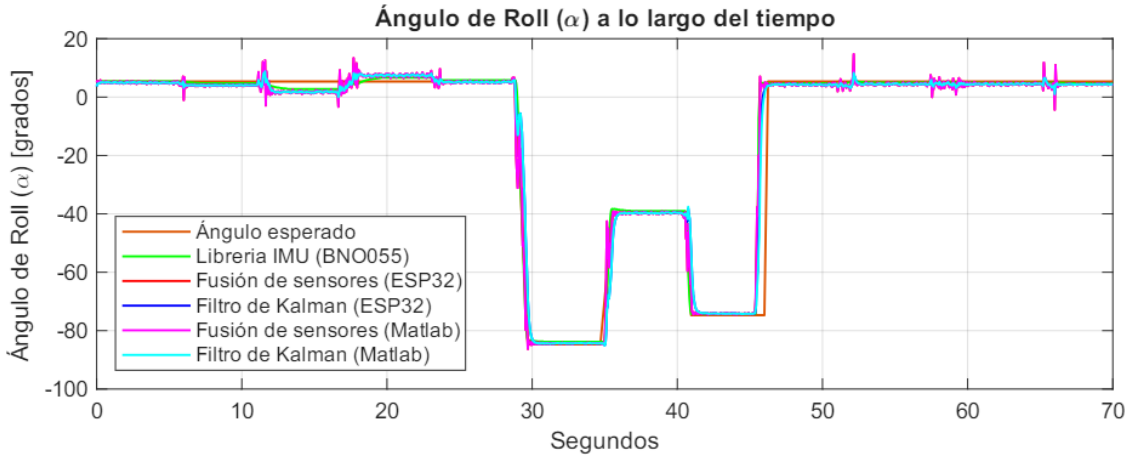

Fig. 4-4 Graficación del experimento 2 para la comparación de resultados del ángulo α.

En la Fig. 4-4 se muestra la comparación del ángulo α esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 2.

En la Fig. 4-5 se muestra la comparación del ángulo β esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 2.

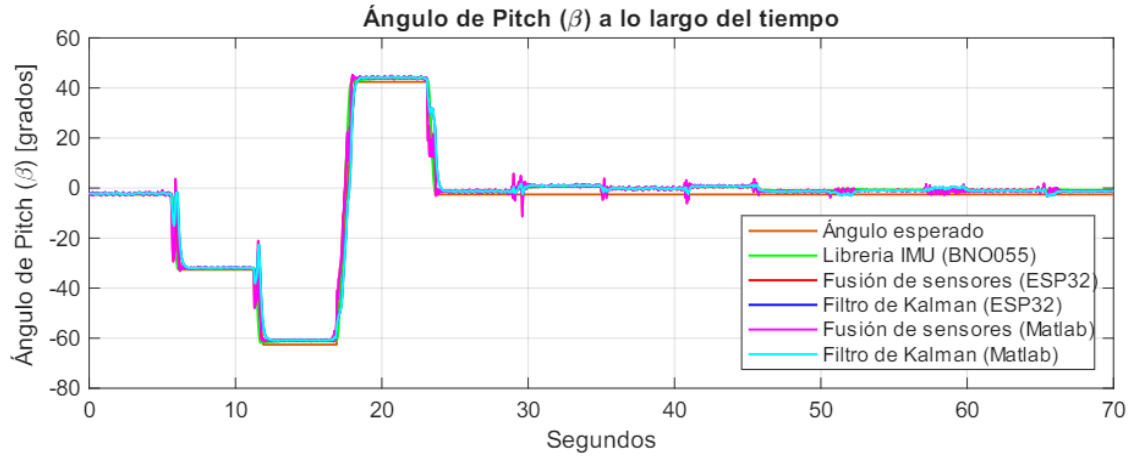

Fig. 4-5 Graficación del experimento 2 para la comparación de resultados del ángulo β.

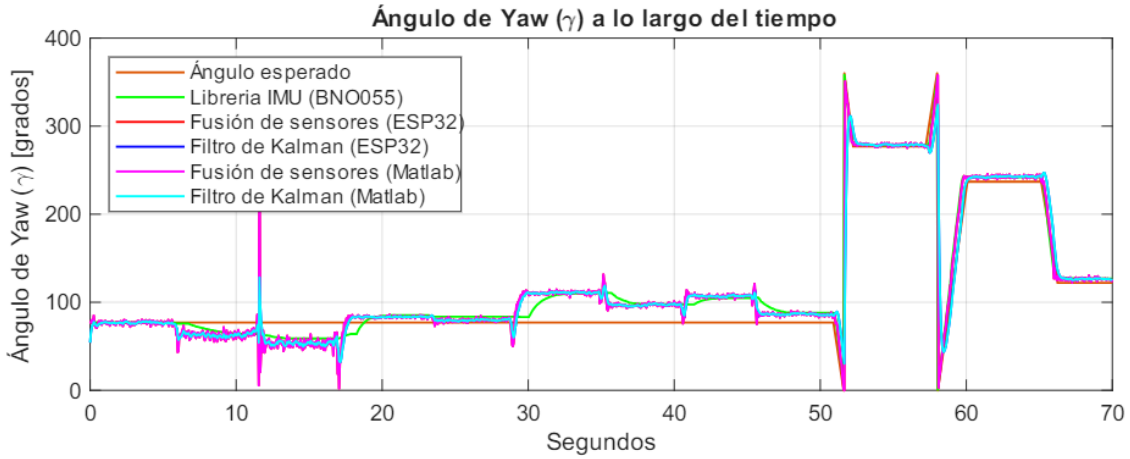

Fig. 4-6 Graficación del experimento 2 para la comparación de resultados del ángulo γ.

En la Fig. 4-6 se muestra la comparación del ángulo γ esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 2.

En la Fig. 4-7 se muestra la comparación del ángulo α esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 3.

TABLA VII

DATOS DE DISTRIBUCIÓN DEL EXPERIMENTO CON IMU 2

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
IMU BNO055	α	1.483779968090021	0.5625	6.725446351569812
	β	1.928821446219731	1.75	2.085683999392282
	γ	12.24924161313347	8.25	10.014799719631421
Filtro de Kalman (ESP32)	α	2.078424117227192	0.9051	6.375466349827902
	β	2.096504815241862	1.5715	2.418673449120325
	γ	14.71088803525936	9.1010	19.324807894725861
Filtro de Kalman (Matlab)	α	2.019658713624774	0.902944390216001	5.923960979420517
	β	2.103904879897313	1.580282178399243	2.418684498964716
	γ	14.74432414988180	9.352518706653996	19.307168004188973

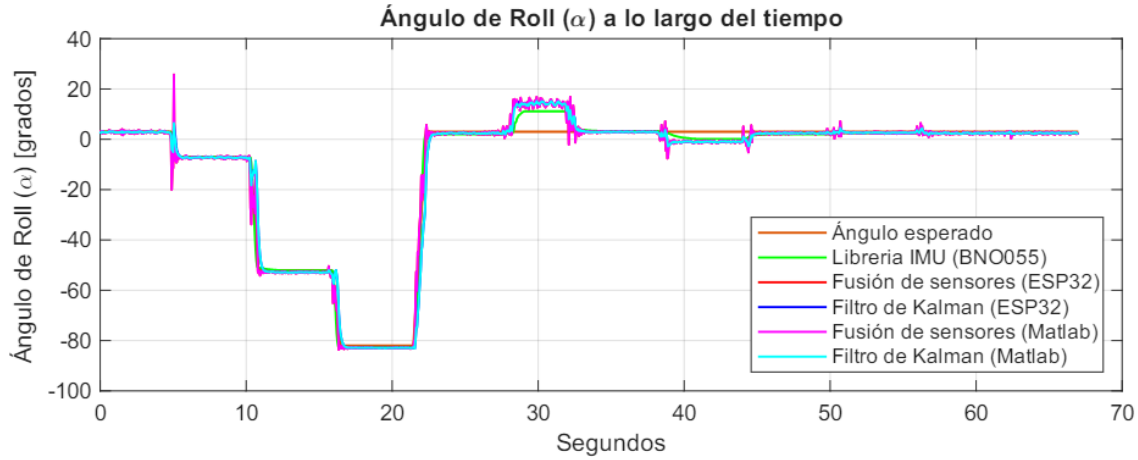

Fig. 4-7 Graficación del experimento 3 para la comparación de resultados del ángulo α.

En la Fig. 4-8 se muestra la comparación del ángulo β esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 3.

En la Fig. 4-9 se muestra la comparación del ángulo γ esperado con los ángulos obtenidos por los filtros de Kalman en el microcontrolador y por los de Matlab en el experimento 3.

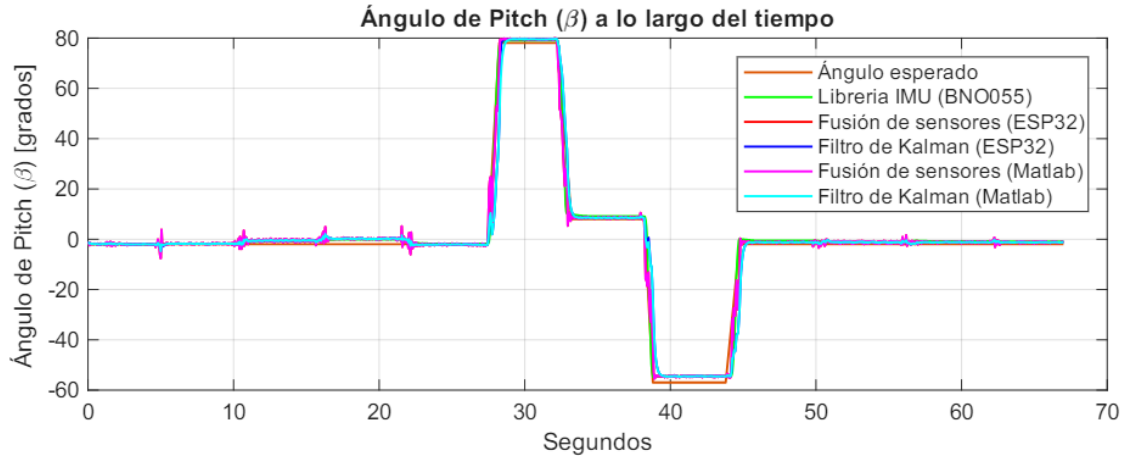

Fig. 4-8 Graficación del experimento 3 para la comparación de resultados del ángulo β.

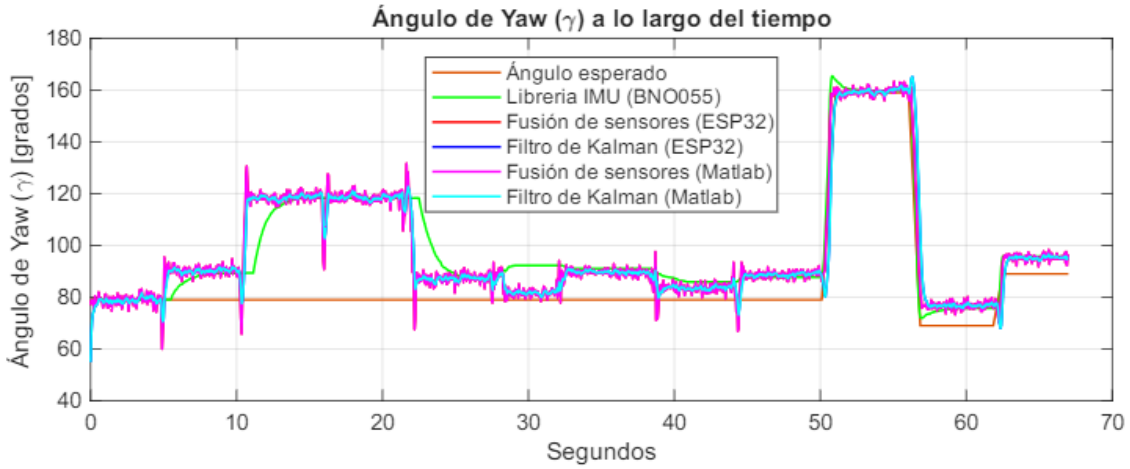

Fig. 4-9 Graficación del experimento 3 para la comparación de los resultados del ángulo γ.

Al comparar los datos de distribución de los tres experimentos en TABLA VI, TABLA VII y TABLA VIII, se observa que en todos los experimentos existen variaciones y diferencias en los ángulos esperados en los ejes α, β y γ, ya sea cuando uno de estos dos ejes cambia de valor. Por lo tanto, todas las señales de la IMU tienden a sufrir los mismos efectos del bloqueo cardán, aun cuando tengan implementada una limitación o condición en sus algoritmos.

TABLA VIII

DATOS DE DISTRIBUCIÓN DEL EXPERIMENTO CON IMU 3

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
BNO055	α	1.285607755406413	0.625	2.004655514391535
	β	1.363473350602456	1.0625	2.23605091306951
	γ	12.88786711983021	8.75	12.192838354934031
Filtro de Kalman (ESP32)	α	1.910317151379570	0. 5902	3.578126989996663
	β	1.808107602339183	0.7506	4.480645754751921
	γ	12.55343771431677	8.3639	13.078433117742303
Filtro de Kalman (Matlab)	α	1.900148069948604	0.592333615081541	3.550411022216109
	β	1.831708848644370	0.749108766847883	4.522776213576350
	γ	12.56902455385197	8.359969080691059	13.094154163431684

4.2. Diseño de filtro de Kalman de la fusión GNSS+IMU

4.2.1 Prueba de filtro de Kalman

Para verificar el funcionamiento del filtro de Kalman GNSS+IMU, se implementa un experimento en campo abierto. El objetivo es realizar una trayectoria en una cancha de fútbol al aire libre para examinar la veracidad y las variaciones de la posición. En este caso, el filtro de Kalman se aplica mediante Matlab, donde el microcontrolador se limitará a obtener los datos del módulo GNSS y de la IMU, y a calcular los ángulos de Euler y los valores del filtro de Kalman de la IMU.

En la prueba, el prototipo realizo dos vueltas alrededor de las líneas de banda para obtener una referencia con la que comparar los datos observados con los esperados. Dichos resultados se reflejan en la Fig. 4-10.

En la Fig. 4-10 se muestra la imagen satelital del recorrido del prototipo en la cancha de fútbol. Por otro lado, en la Fig. 4-11 se presenta su representación gráfica en un plano cartesiano con las medidas correctas de la cancha.

Fig. 4-10 Imagen satelital del experimento de trayectoria en la cancha de fútbol.

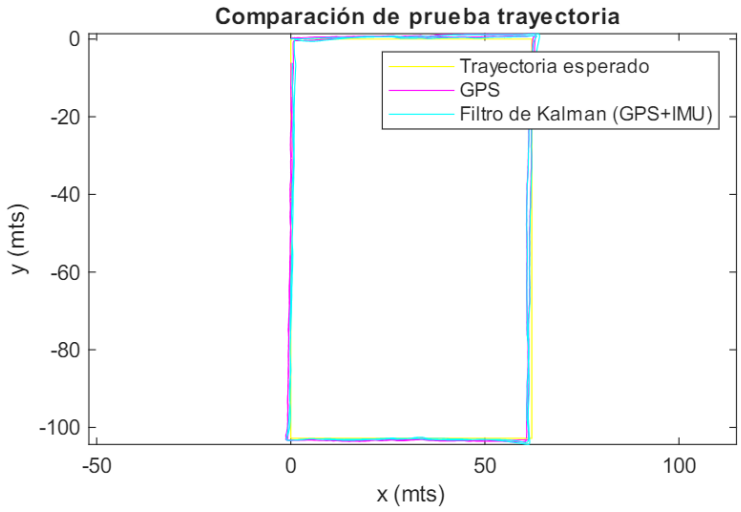

Fig. 4-11 Gráfico en dos dimensiones del experimento de trayectoria en la cancha de fútbol.

En la Fig. 4-12 se muestra la comparación del ángulo alfa con el proporcionado por la librería del módulo IMU, así como la fusión de sensores y el filtro de Kalman durante la experimentación en la cancha de fútbol.

En la Fig. 4-13 se muestra la comparación del ángulo beta con el proporcionado por la librería del módulo IMU, así como la fusión de sensores y el filtro de Kalman durante la experimentación en la cancha de fútbol.

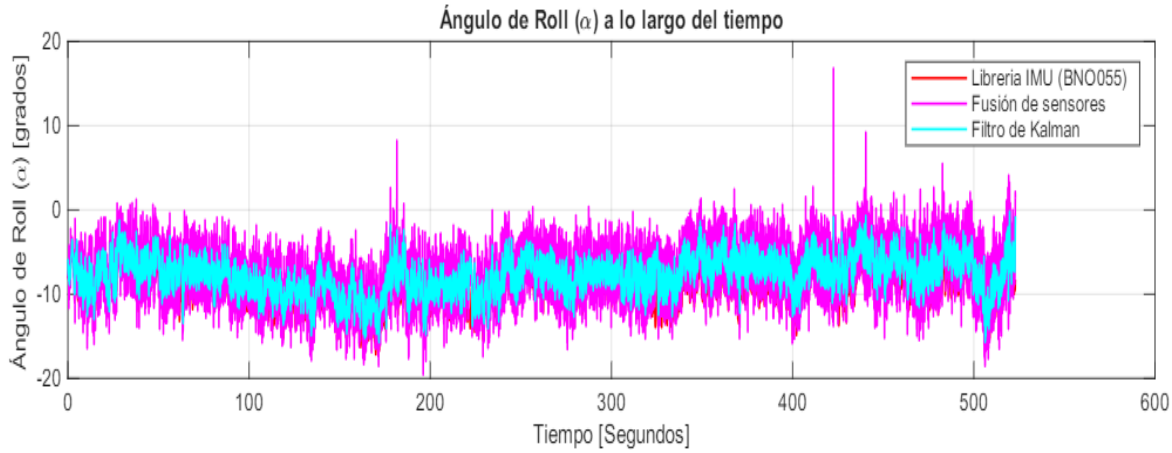

Fig. 4-12 Graficación del ángulo α del experimento de la cancha de fútbol.

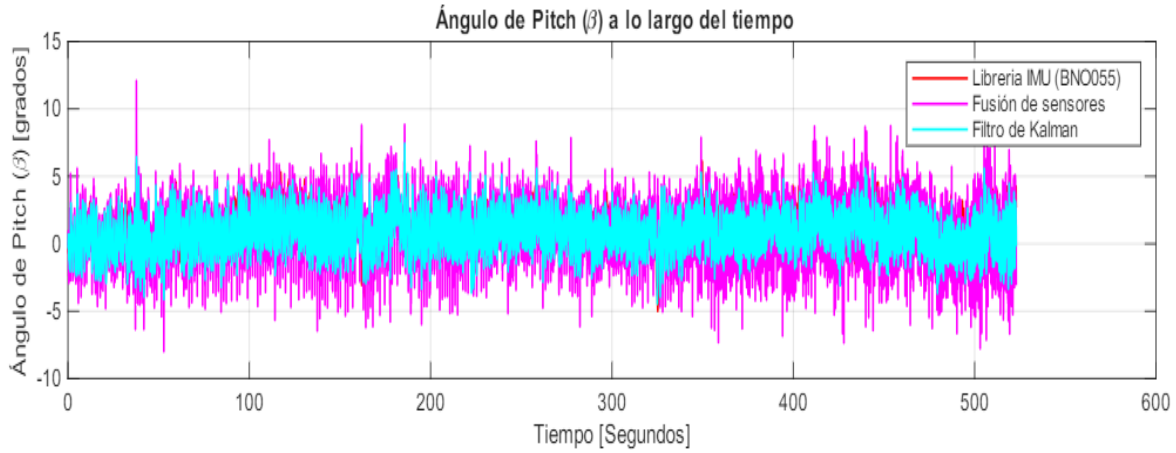

Fig. 4-13 Graficación del ángulo β del experimento de la cancha de fútbol.

En la Fig. 4-14 se muestra la comparación del ángulo gama con el proporcionado por la librería del módulo IMU, así como la fusión de sensores y el filtro de Kalman durante la experimentación en la cancha de fútbol. Donde es el único de los tres ángulos que cambia, ya que el prototipo se trasladó únicamente mediante rotación alrededor del eje del guiñado.

En la Fig. 4-15 se muestra la comparación de la trayectoria de la posición X en la experimentación realizada en la cancha de fútbol.

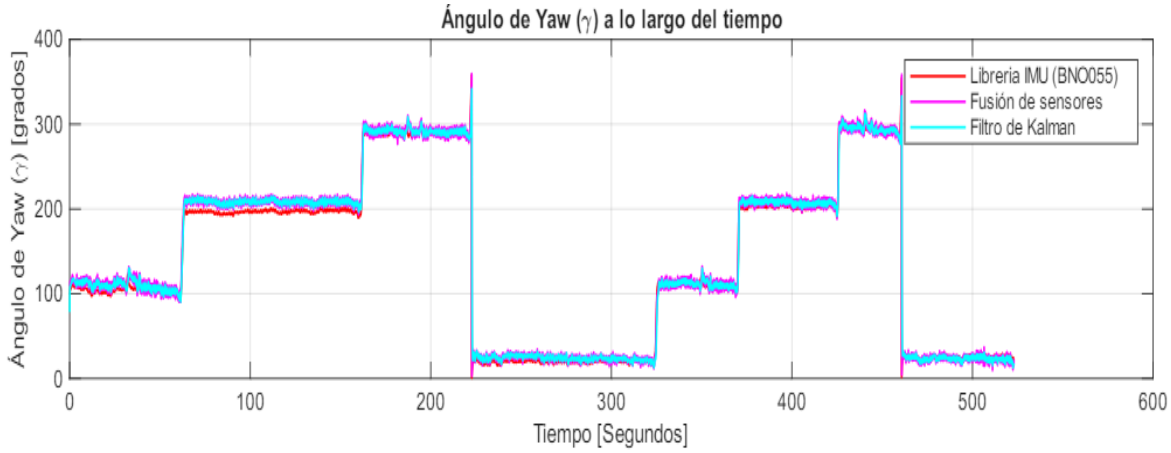

Fig. 4-14 Graficación del ángulo γ del experimento de la cancha de fútbol.

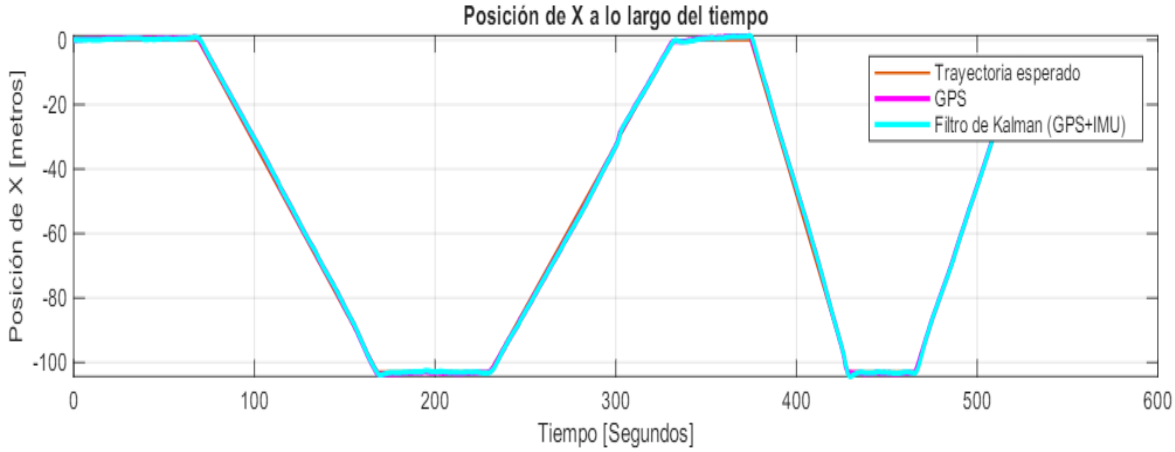

Fig. 4-15 Graficación de la posición en *X* del experimento de la cancha de fútbol.

En la Fig. 4-16 se muestra la comparación de la trayectoria de la posición Y en la experimentación realizada en la cancha de fútbol.

En la Fig. 4-17 se muestra la comparación de la trayectoria de la posición Z en la experimentación realizada en la cancha de fútbol.

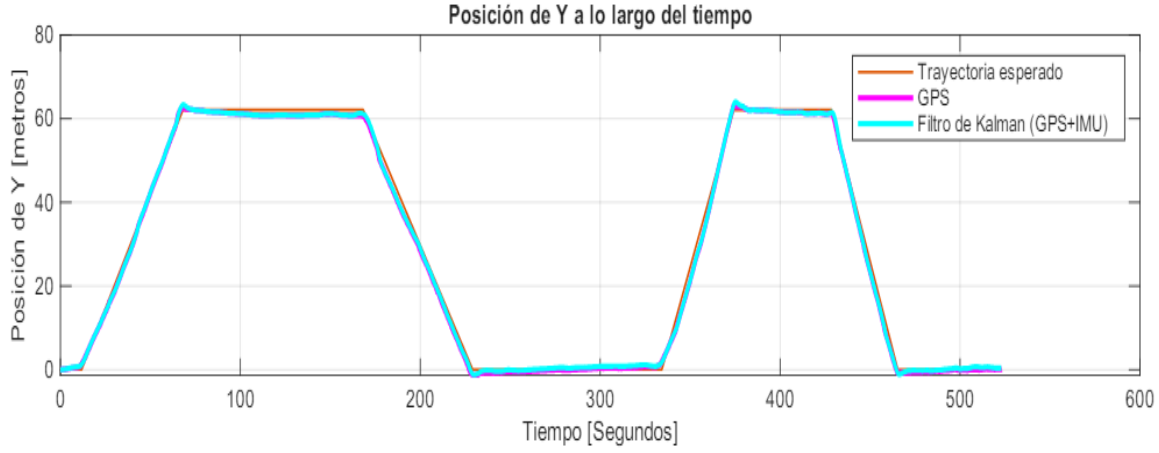

Fig. 4-16 Graficación de la posición en Y del experimento de la cancha de fútbol.

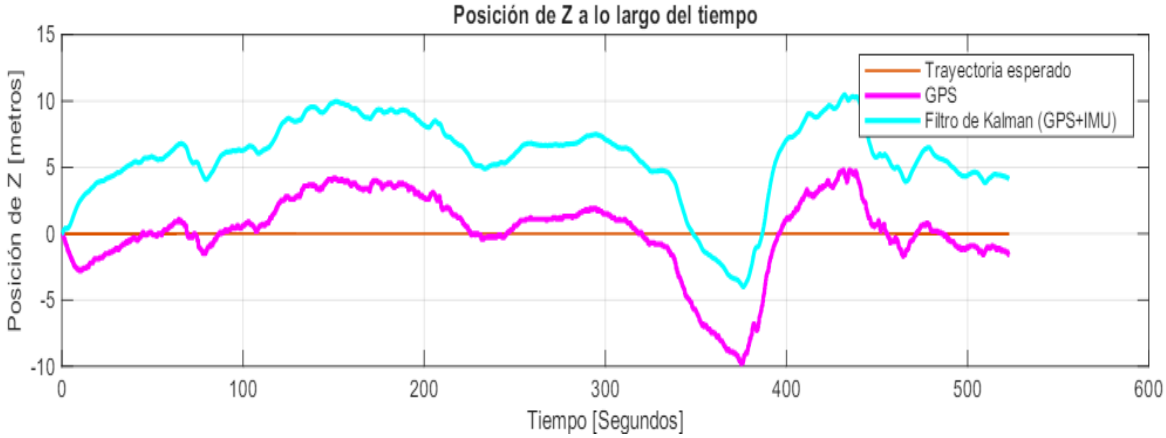

Fig. 4-17 Graficación de la posición en Z del experimento de la cancha de fútbol.

A partir de los resultados mostrados en la TABLA IX, se observa que las posiciones X y Y del filtro de Kalman se encuentran en el entorno de la medida, salvo una distorsión armónica cuando el valor de la posición varía considerablemente. Sin embargo, el resultado de la posición Z presenta una variación significativa en su amplitud tanto en el valor esperado como en el obtenido en las mediciones. Esta diferencia puede deberse a que la medición de altitud obtenida por el módulo GNSS tiende a variar considerablemente, por lo que su covarianza de error es mucho mayor que las de las covarianzas de las posiciones de longitud y latitud.

TABLA IX

DATOS DE DISTRIBUCIÓN DEL EXPERIMENTO DE FUSIÓN DE SENSORES

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
GNSS	X	0.9262	0.7257	0.6188
	Y	0.9519	0.7467	0.7991
	Z	2.1087	1.3	2.0728
Filtro de Kalman	X	0.9252	0.6943	0.7234
	Y	0.92	0.8115	0.7254
	Z	6.0701	6.1424	2.3392

Fig. 4-18 Imagen satelital del experimento de trayectoria en la cancha de fútbol con $\boldsymbol{R} = 0.5$

Para ajustar los valores de corrección del filtro de Kalman, se estudia su comportamiento con una matriz de covarianza del ruido $\boldsymbol{R}$ de distintos valores.

En el caso de que la covarianza $\boldsymbol{R}$ sea 0.5:

$$\boldsymbol{R} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \tag{4-1}$$

En la Fig. 4-18 se muestra la imagen satelital del recorrido del prototipo en la cancha de fútbol, con una covarianza del ruido de proceso de 0.5.

En la Fig. 4-19 se muestra un fragmento de la comparación de la trayectoria de la posición X en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.5.

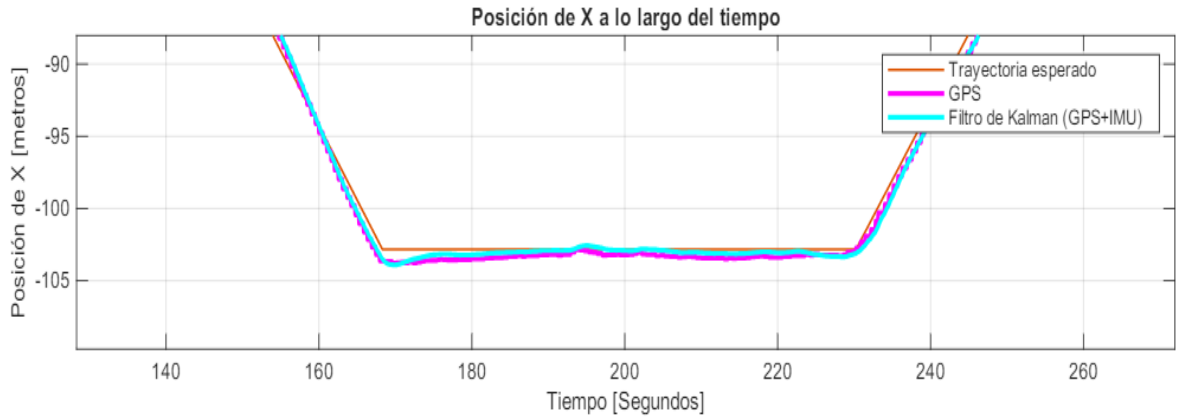

Fig. 4-19 Graficación de la posición en X con la covarianza de ruido $\boldsymbol{R} = 0.5$.

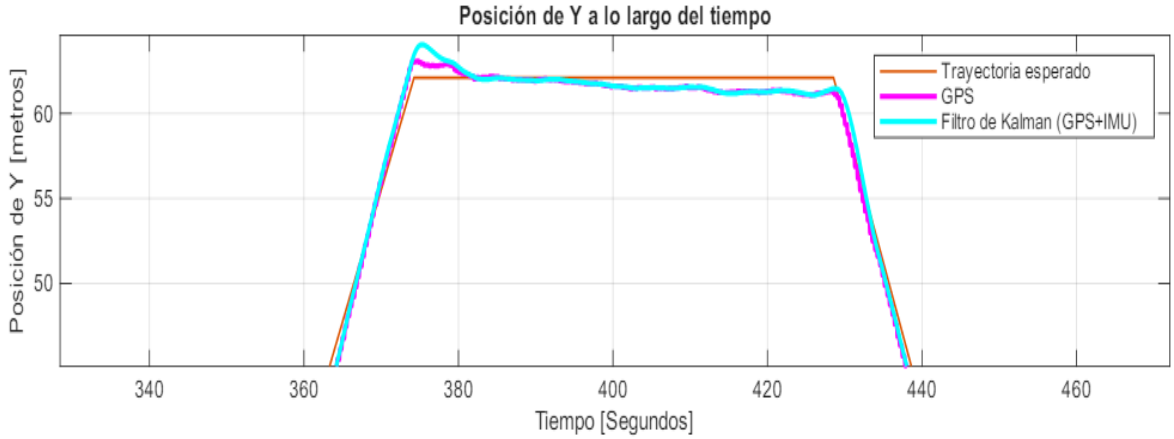

Fig. 4-20 Graficación de la posición en Y con la covarianza de ruido $\boldsymbol{R} = 0.5$.

En la Fig. 4-20 se muestra un fragmento de la comparación de la trayectoria de la posición Y en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.5.

En la Fig. 4-21 se muestra un fragmento de la comparación de la trayectoria de la posición Z en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.5.

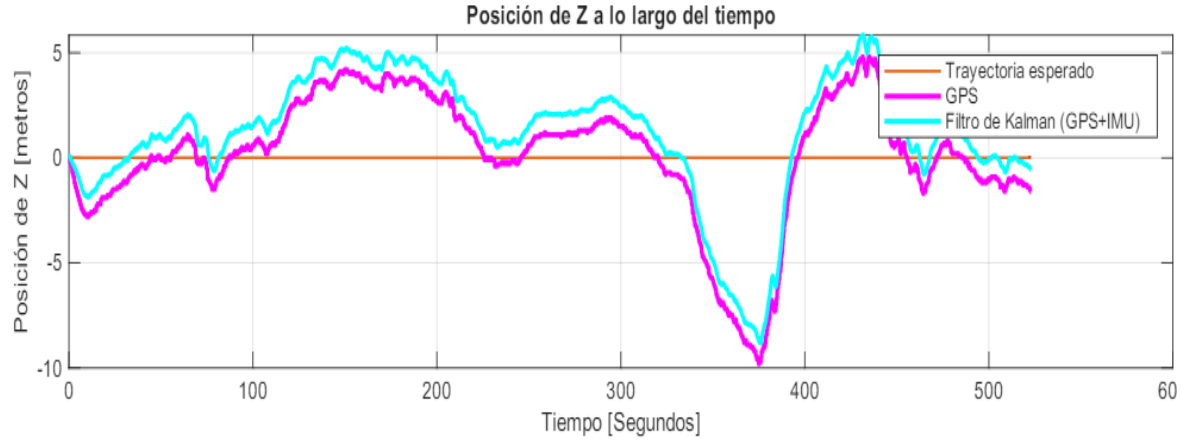

Fig. 4-21 Graficación de posición en Z con la covarianza de ruido $\boldsymbol{R} = 0.5$.

TABLA X

DATOS DE DISTRIBUCIÓN DEL EXPERIMENTO DE FUSIÓN DE SENSORES ($\boldsymbol{R} = 0.5$)

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
Filtro de Kalman	X	0.9246	0.6936	0.7107
	Y	0.9207	0.8248	0.7216
	Z	2.4766	1.9593	1.993

Como se muestra en la TABLA X, la covarianza del error de medición es aproximadamente igual a las covarianzas obtenidas de las longitudes y las latitudes, por lo que no hay un cambio significativo en los parámetros de las posiciones X y Y. Lo único que se ve afectado fueron los resultados de la posición Z, ya que su valor de covarianza cambió. Sin embargo, aún persiste una diferencia en las amplitudes de los valores obtenidos en las mediciones del módulo GNSS.

Por último, se considera el caso de que la covarianza de ruido $\boldsymbol{R}$ es 0.04:

$$\boldsymbol{R} = \begin{bmatrix} 0.04 & 0 & 0 \\ 0 & 0.04 & 0 \\ 0 & 0 & 0.04 \end{bmatrix} \tag{4-2}$$

En la Fig. 4-22 se muestra la imagen satelital del recorrido del prototipo en la cancha de fútbol, con una covarianza del ruido de proceso de 0.04.

Fig. 4-22 Imagen satelital del experimento de trayectoria en la cancha de fútbol con $\boldsymbol{R} = 0.04$.

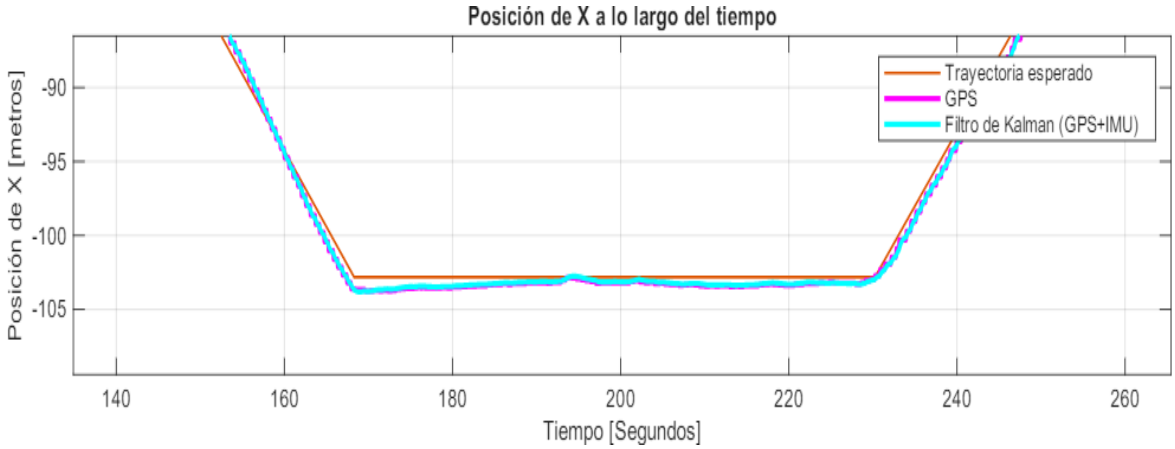

Fig. 4-23 Graficación de la posición en X con la covarianza de ruido $R = 0.04$.

En la Fig. 4-23 se muestra un fragmento de la comparación de la trayectoria de la posición X en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.04.

En la Fig. 4-24 se muestra un fragmento de la comparación de la trayectoria de la posición Y en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.04.

Fig. 4-24 Graficación de la posición en Y con la covarianza de ruido $R = 0.04$.

Fig. 4-25 Graficación de la posición en Z con la covarianza de ruido $R = 0.04$.

En la Fig. 4-25 se muestra un fragmento de la comparación de la trayectoria de la posición Z en la experimentación en la cancha de fútbol, con una covarianza de ruido de proceso de 0.04.

Para esta condición, los datos de la TABLA XI muestran que la covarianza del error de medición es mucho menor que en el experimento anterior. En esta ocasión, el cambio es visible en todos los resultados, donde la distorsión armónica causada por el cambio de valor disminuye y se aproxima más al valor obtenido en las mediciones. Además, se observa que la señal del filtro de posición Z se aproxima a la obtenida en la medición del módulo del dispositivo.

TABLA XI

DATOS DE DISTRIBUCIÓN DEL EXPERIMENTO DE FUSIÓN DE SENSORES (R=0.04)

		Error absoluto medio	Error absoluto mediana	Desviación estándar (σ)
Filtro de Kalman	X	0.9236	0.7097	0.6331
	Y	0.9298	0.724	0.776
	Z	2.1737	1.3711	2.0478

Conclusiones

Este estudio ha intentado mostrar las herramientas básicas para el desarrollo de un vehículo no tripulado mediante filtros de Kalman para estimar sus estados, y ha proporcionado resultados que se aproximan a lo observado en los experimentos. Sin embargo, se evidencian muchos puntos de mejora para quienes deseen un prototipo más eficiente.

El diseño del filtro de Kalman utilizado para la fusión de GNSS e IMU ofrece mejores resultados en la estimación de la posición en dos dimensiones (X y Y). Para obtener una estimación más estable de la posición en Z, se recomienda integrar un sensor de presión barométrica. Al obtener la presión atmosférica, es posible estimar indirectamente la altura sobre el nivel del mar en vez de depender únicamente de la recepción satelital, como se observó durante la experimentación.

Para optimizar un nuevo prototipo, sería ideal replantear el desarrollo del algoritmo para el microcontrolador. El código implementado para el almacenamiento de datos en la memoria SD presentaba un periodo de ejecución más elevado que el del módulo IMU, por lo que este último debía muestrear sus datos a la misma frecuencia con la que se guardaban los parámetros en la SD. Afortunadamente, se comprobó que la implementación de las operaciones matriciales y del filtrado mediante codificación en tiempo real para la estimación de la orientación fue exitosa, por lo que se deberá mejorar la recopilación de datos a alta frecuencia para optimizar el programa y evitar demoras en la estimación. Otra propuesta sería buscar una alternativa a los ángulos de Euler para reducir el efecto del bloqueo de cardán en los ángulos α y γ, por lo que se recomienda obtener los cuaterniones.

Otro punto ausente en la experimentación fue no aplicar el filtrado de Kalman para estimar la posición en tiempo real en el microcontrolador. Si bien logramos procesar la estimación al recopilar los datos obtenidos de los módulos por código en Matlab, también sería idóneo implementar un diseño más enfocado en trabajos aéreos, como un filtro de Kalman extendido.

Bibliografía

[1] A. V. Tulush, P. V. Erkin, V. V. Puzikov, A. S. Timoshenkov, y E. V. Lagunov, «Development of the Self-driving System for Parachute-loading Platform», *2021 IEEE Conf. Russ. Young Res. Electr. Electron. Eng. ElConRus*, pp. 2760-2763, 2021, doi: 10.1109/ElConRus51938.2021.9396478.

[2] J. Demuyakor, «Ghana Go Digital Agenda: The impact of Zipline Drone Technology on Digital Emergency Health Delivery in Ghana», *Shanlax Int. J. Arts Sci. Humanit.*, vol. 8, n.º 1, pp. 242-253, jul. 2020, doi: 10.34293/sijash.v8i1.3301.

[3] M. W. Spong, S. Hutchinson, y M. Vidyasagar, *Robot Modeling and Control*, 1.ª ed. Wiley, 2005.

[4] K. Shoemake, «Animating Rotation with Quaternion Curves», en *ACM SIGGRAPH*, San Francisco, 1985, pp. 245-254.

[5] MATLAB, «What Are State-Space Models?» [En línea]. Disponible en: https://la.mathworks.com/help/ident/ug/what-are-state-space-models.html

[6] K. Ogata, *Ingeniería de control moderna*, 5.ª ed. Madrid, España: Pearson Educación, 2010.

[7] G. Welch y G. Bishop, «An Introduction to the Kalman Filter», *Dep. Comput. Sci. Univ. N. C. Chap. Hill*, 1997.

[8] N. Hadjigeorgiou, «Circuit and systems architectures and performance evaluation of AMR magnetometers», School of Electrical and Computer Engineering, National Technical University of Athens, Athens , Greece, 2024. doi: 10.12681/eadd/57031.

[9] I. Arun Faisal, T. Waluyo Purboyo, y A. Siswo Raharjo Ansori, «A Review of Accelerometer Sensor and Gyroscope Sensor in IMU Sensors on Motion Capture», *J. Eng. Appl. Sci.*, vol. 15, n.º 3, pp. 826-829, nov. 2019, doi: 10.36478/jeasci.2020.826.829.

[10] A. Noureldin, T. B. Karamat, y J. Georgy, *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-30466-8.

[11] International Civil Aviation Organization, *World Geodetic System – 1984 (WGS-84) Manual*, 9674, 2002.

[12] B. Hofmann-Wellenhof, H. Lichtenegger, y E. Wasle, *GNSS — Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. en SpringerLink Bücher. Vienna: Springer-Verlag Wien, 2008. doi: 10.1007/978-3-211-73017-1.

[13] Espressif, «ESP32 Series Datasheet Version 5.2», Espressif Systems, Shanghai, China, Datasheet, 2025. [En línea]. Disponible en: httphttps://documentation.espressif.com/esp32_datasheet_en.pdf

[14] Bosch, «BNO055 Intelligent 9-axis absolute orientation sensor», Bosch Sensortec, Reutlingen, Germany, Datasheet BST-BNO055-DS000-18, oct. 2021. [En línea]. Disponible en: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf

[15] u-blox, «NEO-M8 u-blox M8 concurrent GNSS modules», u-blox AG, Thalwil, Switzerland, Datasheet UBX-15031086-R14, sep. 2025. [En línea]. Disponible en: www.u-blox.com

Apéndices

A. CÓDIGO DE LA FUNCIÓN PRINCIPAL DEL PROTOTIPO DEL VANT

Este apéndice incluye el programa en C++ que utiliza el framework de Arduino para la inicialización y configuración del prototipo de vehículo aéreo no tripulado.

```
#define _TASK_MICRO_RES
#include <TaskScheduler.h>
#include "BNO055_config.h"
#include "SD_config.h"
#include <esp_task_wdt.h>
#include "GPS_config.h"
#include <Wire.h>
#include "TinyGPS++.h"
#include <OLED_config.h>
#define LED 2    // define appropriate pin for your board
TaskHandle_t task1_handle = NULL;
TaskHandle_t task2_handle = NULL;
TaskHandle_t task3_handle = NULL;
unsigned long previousMillis = 0;
volatile long startMillis, endMillis, delayMillis,
             startMillis2, endMillis2, delayMillis2,
             startMillis3, endMillis3, delayMillis3;
struct Button {
   const uint8_t PIN;
   bool pressed;
};
Button button1 = {15, false};
```

```
unsigned long button_time = 0;
unsigned long last_button_time = 0;
void IRAM_ATTR isr(){
button_time = millis();
 if (button_time - last_button_time > 350){
    button1.pressed = true;
    last_button_time = button_time;
  }
}

/* Leer IMU */
void task1(void * parameters){
  for(;;){
    volatile long startMillis = millis();
    Read_imu_data();
    volatile long delayMillis = millis()-startMillis;
      if (delayMillis < 50){
        vTaskResume(task2_handle);
        vTaskDelay((50-delayMillis) / portTICK_PERIOD_MS);
    }
  }
}

/* Escribir dato en la SD */
void task2(void * parameters){
  for(;;){
    volatile long startMillis2 = millis();
    logSDCard();
    vTaskSuspend(task2_handle);
    volatile long delayMillis2 = millis()-startMillis2;
```

```
  }
}
/* Guardar en memoria SD */
void task3(void * parameters){
  for(;;){
    volatile long startMillis3 = millis();
    Read_gps();
    volatile long delayMillis3 = millis()-startMillis3;
    if (delayMillis3 < 500){
      vTaskDelay((500-delayMillis3) / portTICK_PERIOD_MS);
      }
  }
}
void setup(){
  Serial.begin(9600);
  pinMode(LED , OUTPUT);
  digitalWrite(LED, LOW);
  pinMode(button1.PIN, INPUT_PULLUP);
  attachInterrupt(button1.PIN, isr, FALLING);
  Serial.println("Scheduler TEST");
  while (inti_SD()==false){
    delay(1000);
  }
  digitalWrite(LED, HIGH);
  Init_OLED();
  imu_init();
  check_calib();
  delay(3000);
  digitalWrite(LED, LOW);
  while(button1.pressed==false){
```

```
  delay(1000);
 }
 digitalWrite(LED, HIGH);
 xTaskCreatePinnedToCore(task1, // Function to implement the task //
                         "Task1", // Name of the task //
                         10000,  // Stack size in words //
                         NULL,  // parameter of the task //
                         1,  // Priority of the task //
                         &task1_handle,  // Task handle. //
                         1); // Core where the task should run //
 xTaskCreatePinnedToCore(task2, // Function to implement the task
                         "Task2", // Name of the task
                         10000,  // Stack size in words
                         NULL,  // parameter of the task
                         1,  // Priority of the task //
                         &task2_handle,  // Task handle. //
                         1); // Core where the task should run //
 vTaskSuspend(task2_handle);
 /*xTaskCreatePinnedToCore(task3, // Function to implement the task
                         "Task3", // Name of the task
                         10000,  // Stack size in words
                         NULL,  // parameter of the task
                         1,  // Priority of the task
                         &task3_handle,  // Task handle.
                         0); // Core where the task should run */
}
void loop(){
 delay(1000);
 print_speed();
}
```

B. LECTURA DEL MODULO IMU

Este apéndice incluye el programa en C++ que utiliza el framework de Arduino para la inicialización y lectura de los sensores del módulo IMU. También se procesan esos datos para obtener los ángulos de Euler.

```
#include "BNO055_config.h"
#include <OLED_config.h>
#include <utility/imumaths.h>

// Dirección I2C del dispositivo
Adafruit_BNO055 bno = Adafruit_BNO055(-1, 0x28, &Wire);

imuType sensor3;
adafruit_bno055_offsets_t calibrationData;
adafruit_bno055_offsets_t newCalib;

void imu_init(){
  if(!bno.begin()){
    Serial.print("Ooops, no se detectada el BNO055...");
    while(1);
  }
  delay(1000);
  //bno.setAxisRemap(Adafruit_BNO055::REMAP_CONFIG_P7);
  //bno.setAxisSign(Adafruit_BNO055::REMAP_SIGN_P7);
  bno.setExtCrystalUse(true);
  Serial.println("Valores de estatus de calibración: 0=descalibrado, 3=calibrado");
}
```

```
void check_calib(void){
  int eeAddress = 0;
  long bnoID;
  bool foundCalib = false;

  EEPROM.get(eeAddress, bnoID);
  sensor_t sensor;
  /*
  *  Look for the sensor's unique ID at the beginning oF EEPROM.
  */
  bno.getSensor(&sensor);
  if (bnoID != sensor.sensor_id)
  {
     Serial.println("\nDatos de calibración no encontrados en EEPROM");
     delay(500);
  }
  else
  {
   Serial.println("\nDatos de calibración encontrados en EEPROM");
   eeAddress += sizeof(long);
   EEPROM.get(eeAddress, calibrationData);
   Serial.println("\n\nRestaurando datos de calibración al EEPROM...");
   bno.setSensorOffsets(calibrationData);
   Serial.println("\n\nDatos de calibración cargados a la EEPROM");
   foundCalib = true;
  }
  delay(1000);

  sensors_event_t event;
  bno.getEvent(&event);
```

```
if (foundCalib){
  Serial.println("Mueva el sensor ligeramente para calibrar el magnetómetro");
  while (!bno.isFullyCalibrated())
  {
    bno.getEvent(&event);
    delay(BNO055_SAMPLERATE_DELAY_MS);
  }
}
else
{
  Serial.println("Iniciar calibración de la IMU: ");
  init_calib();
}

Serial.println("\nCalibración Completado");
Serial.println("--------------------------------");
Serial.println("Resultado de la calibración: ");
bno.getSensorOffsets(newCalib);

Serial.println("\n\nGuadar datos de calibración al EEPROM...");

eeAddress = 0;
bno.getSensor(&sensor);
bnoID = sensor.sensor_id;

EEPROM.put(eeAddress, bnoID);

eeAddress += sizeof(long);
EEPROM.put(eeAddress, newCalib);
Serial.println("Guadar datos");
```

```
  Serial.println("\n--------------------------------\n");
  delay(500);
}

void init_calib(){
  bool calb_flag = false;
  Serial.println("Mueva el sensor ligeramente para calibrar sus sensores");
  /* Espera hasta que se calibren todos los modulos/sensores del IMU */
  while(!calb_flag)
  {
    running_calib();
    if((sensor3.calb_system == 3) && (sensor3.calb_gyro==3) && (sensor3.calb_accel==3) &&
(sensor3.calb_mag==3))
      calb_flag = true;
    else
      calb_flag = false;
  }
  init_Kalman_Filter();
}

void running_calib(){
      bno.getCalibration(&sensor3.calb_system,      &sensor3.calb_gyro,      &sensor3.calb_accel,
&sensor3.calb_mag);
  Serial.print("CALIBRATION: Sys=");
  Serial.print(sensor3.calb_system, DEC);
  Serial.print(" Gyro=");
  Serial.print(sensor3.calb_gyro, DEC);
  Serial.print(" Accel=");
  Serial.print(sensor3.calb_accel, DEC);
```

```
  Serial.print(" Mag=");
  Serial.println(sensor3.calb_mag, DEC);
  print_imu(sensor3.calb_gyro, sensor3.calb_accel, sensor3.calb_mag, sensor3.calb_system);
  delay(BNO055_SAMPLERATE_DELAY_MS);
  uint8_t system_status, self_test_results, system_error;
  system_status = self_test_results = system_error = 0;
  bno.getSystemStatus(&system_status, &self_test_results, &system_error);
}

void Read_imu_temp(){
  sensor3.temp = bno.getTemp();
}

void Read_imu_data(){
  /* Vectores de los sensores de la IMU */
  sensor3.acceler = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER); //Accel (m/s^2)
   sensor3.gyrosc = bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);          //Gyro (rad/s)
                                                sensor3.magnet                                                =
bno.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);   //Magnet (uT)
  sensor3.euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);          //Euler (°)

  /* Calculo de la magnitud del acelerometro */
  sensor3.magnitud_acceler = sqrtl((sensor3.acceler.x()*sensor3.acceler.x()) +
                                   (sensor3.acceler.y()*sensor3.acceler.y()) +
                                   (sensor3.acceler.z()*sensor3.acceler.z()));
  if(sensor3.magnitud_acceler == 0)
    sensor3.magnitud_acceler = 0.00001;
  sensor3.magnitud_magnet = sqrtl((sensor3.magnet.x())*(sensor3.magnet.x()) +
```

```
                    (sensor3.magnet.y())*(sensor3.magnet.y()) +
                    (sensor3.magnet.z())*(sensor3.magnet.z()));
  if(sensor3.magnitud_magnet == 0)
    sensor3.magnitud_magnet = 0.00001;
  /* Calculo de los vectores de Down */
  sensor3.dir_acceler[0] = sensor3.acceler.x()/sensor3.magnitud_acceler;  //sensor3.down[0]
  sensor3.dir_acceler[1] = sensor3.acceler.y()/sensor3.magnitud_acceler;  //sensor3.down[1]
  sensor3.dir_acceler[2] = sensor3.acceler.z()/sensor3.magnitud_acceler;  //sensor3.down[2]
  /* Calculo de los vectores de North y East */
  sensor3.dir_magnet[0] = sensor3.magnet.x()/sensor3.magnitud_magnet;
  sensor3.dir_magnet[1] = sensor3.magnet.y()/sensor3.magnitud_magnet;
  sensor3.dir_magnet[2] = sensor3.magnet.z()/sensor3.magnitud_magnet;
          sensor3.east[0]     =     (sensor3.dir_acceler[1]*sensor3.dir_magnet[2])-
(sensor3.dir_acceler[2]*sensor3.dir_magnet[1]);
          sensor3.east[1]     =     (sensor3.dir_acceler[2]*sensor3.dir_magnet[0])-
(sensor3.dir_acceler[0]*sensor3.dir_magnet[2]);
          sensor3.east[2]     =     (sensor3.dir_acceler[0]*sensor3.dir_magnet[1])-
(sensor3.dir_acceler[1]*sensor3.dir_magnet[0]);
            sensor3.north[0]     =     (sensor3.east[1]*sensor3.dir_acceler[2])-
(sensor3.east[2]*sensor3.dir_acceler[1]);
            sensor3.north[1]     =     (sensor3.east[2]*sensor3.dir_acceler[0])-
(sensor3.east[0]*sensor3.dir_acceler[2]);
            sensor3.north[2]     =     (sensor3.east[0]*sensor3.dir_acceler[1])-
(sensor3.east[1]*sensor3.dir_acceler[0]);
  /* Calculo de los ángulos de Euler */
  if((sensor3.dir_acceler[0]>-0.99)&&(sensor3.dir_acceler[0]<0.99)){
    sensor3.angles[1] = asinl(sensor3.dir_acceler[0]) * RAD_TO_DEG;
        sensor3.angles[0]  =  atan2l((-1)*sensor3.dir_acceler[1],sensor3.dir_acceler[2])  *
RAD_TO_DEG;
    sensor3.angles[2] = (atan2l((-1)*sensor3.east[0],sensor3.north[0]) - (PI/2))* RAD_TO_DEG;
```

```
  }
  else{
    sensor3.angles[0] = 0;
    if(sensor3.dir_acceler[0] <= -0.99){
      sensor3.angles[1] = (-PI/2) * RAD_TO_DEG;
      sensor3.angles[2] = (((atan2l(sensor3.north[1],sensor3.north[2])) - (PI/2))*RAD_TO_DEG);
    }
    else if(sensor3.dir_acceler[0] >= 0.99){
      sensor3.angles[1] = (PI/2) * RAD_TO_DEG;
                  sensor3.angles[2]     =     (((atan2l(sensor3.north[1],(-1)*sensor3.north[2]))     -
(PI/2))*RAD_TO_DEG);
    }
  }
  if (sensor3.angles[2] > 360)
    sensor3.angles[2] = sensor3.angles[2] - 360;
  else if(sensor3.angles[2] < 0)
    sensor3.angles[2] = sensor3.angles[2] + 360;
  /* Aplicar filtro de Kalman a los ángulos de Euler */
   IMU_Kalman_Filter(filter.I3,  filter.I3,  filter.I3,  filter.xhat_m3,  sensor3.angles,  filter.P_m3,
filter.K3,filter.P3,filter.x3);
}
```

C. LECTURA DEL MODULO GNSS

Este apéndice incluye el programa en C++ que utiliza el framework de Arduino, la inicialización, configuración y lectura de los datos obtenidos en el módulo GNSS.

```
#include "GPS_config.h"
#include <OLED_config.h>
#include <ESP32Time.h>
#include <Wire.h>
#include "TinyGPS++.h"

HardwareSerial gpshw(1);
ESP32Time rtc_gps(-21600);  // offset GMT-6

bool ggaFound = false;      // Bandera de mensaje GGA
bool rmcFound = false;      // Bandera de mensaje RMC
bool gsaFound = false;      // Bandera de mensaje GSA
bool gsvFound = false;      // Bandera de mensaje GSV
bool ChecksumTerm = false;  // CheckSum de mensajes del modulo Neo8mGPS
bool messagetype = false;   // Bandera para indicar si se encontró un mensaje GGA
uint8_t parity, messcount = 0;
char incomingbyte;
String gpsData, allmesage, parstring, datarmc[12], datagga[15], datagsa[18], datagsv[20];
rmcType gps1;
ggaType gps2;
gsaType gps3;
gsvType gps4;

void gps_init(void){
```

```
  bool gpsCalibrated = false;
  volatile uint8_t i = 0;
  /* Comunicación serial del Neo8mGPS */
  gpshw.begin(9600, SERIAL_8N1, RXD2, TXD2);
  GNSS_config(0);
  delay(100);
 for (unsigned long start = millis(); millis() - start < 1000;){
  while (!gpsCalibrated) {
    while (gpshw.available()){
      incomingbyte=gpshw.read();

      /* Si el caracter recibido indica inicio del mensaje */
      if (incomingbyte == '$') {
        gpsData = "";
        rmcFound = false;
        allmesage = "";
        parstring = "";
        parity=0;
        ChecksumTerm = false;
      }
      /* Si el caracter recibido indica si es un nuevo atributo en el mensaje */
      else if (incomingbyte == ',' && (rmcFound)){
        if(rmcFound==true)
          datarmc[i++] = gpsData;
        parity ^= (uint8_t)incomingbyte;
        gpsData = "";
      }
      /* Si el caracter recibido indica fin del mensaje e inicio el valor del CheckSum */
      else if (incomingbyte == '*'){
        if(rmcFound==true)
```

```
      datarmc[i] = gpsData;
    ChecksumTerm = true;
    gpsData = "";
   }
   /* Si el caracter recibido indica el final de la linea del mensaje */
   else if((incomingbyte == '\r')&& (ChecksumTerm)){
    parstring = String(parity, HEX);
    parstring.toUpperCase();
    /* Si el valor de paridad del bloque del mensaje es igual al CheckSum prupuesto */
    if(gpsData == parstring){
     if(rmcFound){
      conver_values();
      messcount++;
      rtc_gps.setTime(gps1.seconds.toInt(),gps1.minutes.toInt(),gps1.hours.toInt(),
                gps1.day.toInt(),gps1.mounth.toInt(),gps1.year.toInt(),gps1.microseconds.toIn
t());
      gpsCalibrated = true;
     }
    }
    i=0;
    /* Limpiar la variable y comenzar a capturar el nuevo mensaje */
    gpsData = "";
   }
   /* Si recibimos es un caracter cualquiera */
   else{
    /* Añadir el byte recibido a la variable gpsData */
    gpsData += incomingbyte;
    if(!ChecksumTerm)
     parity ^= (uint8_t)incomingbyte;
   }
```

```
      /* Clasifique el tipo de mensaje GNSS */
       if((gpsData == "GNRMC,")||(gpsData == "GPRMC,")||(gpsData == "GLRMC,")||(gpsData
== "GARMC,")){
        rmcFound = true;
        messagetype = true;
        gpsData = "";
       }
     }
   }
  }
}

void gps_setup(void)
{
  // Enable $PUBX,40,GLL,0,1,0,0*5D
  //gpshw.println("$PUBX,40,GLL,0,1,0,0*5D");
   // disable $PUBX,40,GLL,0,0,0,0*5C
  gpshw.println("$PUBX,40,GLL,0,0,0,0*5C");
  delay(100);

  // Enable $PUBX,40,RMC,0,1,0,0*46
  gpshw.println("$PUBX,40,RMC,0,1,0,0*46");
    // disable $PUBX,40,RMC,0,0,0,0*47
 // gpshw.println("$PUBX,40,RMC,0,1,0,0*46"); on
  delay(100);

  // Enable $PUBX,40,VTG,0,1,0,0*5F
  //gpshw.println("$PUBX,40,VTG,0,1,0,0*5F");
  // disable $PUBX,40,VTG,0,0,0,0*5E
  gpshw.println("$PUBX,40,VTG,0,0,0,0*5E");
```

```
  delay(100);

  // Enable $PUBX,40,GGA,0,1,0,0*5B
  gpshw.println("$PUBX,40,GGA,0,1,0,0*5B");
  // disable $PUBX,40,GGA,0,1,0,0*5A
  //gpshw.println("$PUBX,40,GGA,0,0,0,0*5A"); //ojo
  delay(100);

  // Enable $PUBX,40,GSA,0,1,0,0*4F
  gpshw.println("$PUBX,40,GSA,0,1,0,0*4F");
  // disable $PUBX,40,GSA,0,0,0,0*4E
  //gpshw.println("$PUBX,40,GSA,0,0,0,0*4E");
  delay(100);

  // Enable $PUBX,40,GSV,0,1,0,0*58
  gpshw.println("$PUBX,40,GSV,0,1,0,0*58");
  // disable $PUBX,40,GSV,0,0,0,0*59
  //gpshw.println("$PUBX,40,GSV,0,0,0,0*59");
  delay(100);

  SetUp2hZ();
}

void Read_gps(void){
  volatile uint8_t i = 0;
  volatile uint16_t parint, parmssg;
  bool gsablock = false, gsvblock = false;
  messcount = 0;
  while(messcount < 4)
  {
```

```
if (gpshw.available())
{
 incomingbyte=gpshw.read();
 /* Si el caracter recibido indica inicio del mensaje */
 if (incomingbyte == '$') {
  gpsData = "";  // Limpiar la variable y comenzar a capturar el nuevo mensaje
  ggaFound = rmcFound = gsaFound = false; // Reiniciar la bandera de mensaje encontrado
  gsvFound = false;
  allmesage = "";
  parstring = "";
  parity=0;
  ChecksumTerm = false;
 }
 /* Si el caracter recibido indica si es un nuevo atributo en el mensaje
    y si no se repite el mismo tipo de mensaje GNSS */
 else if (incomingbyte == ',' && (ggaFound || rmcFound || gsaFound || gsvFound)){
  if(ggaFound==true)
    datagga[i++] = gpsData;
  else if(rmcFound==true)
    datarmc[i++] = gpsData;
  else if(gsaFound==true)
    datagsa[i++] = gpsData;
  else if(gsvFound==true)
    datagsv[i++] = gpsData;
  parity ^= (uint8_t)incomingbyte;
  gpsData = "";  // Limpiar la variable y comenzar a capturar el nuevo dato
 }
 /* Si el caracter recibido indica fin del mensaje e inicio el valor del CheckSum */
 else if (incomingbyte == '*'){
  if(ggaFound==true)
```

```
    datagga[i] = gpsData;
  else if(rmcFound==true)
    datarmc[i] = gpsData;
  else if(gsaFound==true)
    datagsa[i] = gpsData;
  else if(gsvFound==true)
    datagsv[i] = gpsData;
  ChecksumTerm = true;
  gpsData = "";  // Limpiar la variable y comenzar a capturar el nuevo dato
}
/* Si el caracter recibido indica el final de la linea del mensaje */
else if((incomingbyte == '\r')&& (ChecksumTerm))
{
  parstring = String(parity, HEX);
  parmssg = gpsData.toInt();
  parint = parstring.toInt();
  if(parmssg == parint){
    if(rmcFound){
      conver_values();
      messcount++;
    }
    else if(ggaFound){
      conver_values();
      messcount++;
    }
    else if(gsaFound){
      conver_values();
      messcount++;
      gsablock = true;
    }
```

```
        else if(gsvFound){
          conver_values();
          messcount++;
          gsvblock = true;
         }
       }
       i=0;
       gpsData = "";  // Limpiar la variable y comenzar a capturar el nuevo mensaje
     }
     /* Si recibimos es un caracter cualquiera */
     else{
       gpsData += incomingbyte;  // Añadir el byte recibido a la variable gpsData
       if(!ChecksumTerm)
       parity ^= (uint8_t)incomingbyte;
     }
     if((gpsData == "GNGGA,")||(gpsData == "GPGGA,")||(gpsData == "GLGGA,")||(gpsData ==
"GAGGA,")){
       ggaFound = true;
       messagetype = true;
       gpsData = "";
     }
     if((gpsData == "GNRMC,")||(gpsData == "GPRMC,")||(gpsData == "GLRMC,")||(gpsData ==
"GARMC,")){
       rmcFound = true;
       messagetype = true;
       gpsData = "";
     }
     if(((gpsData == "GPGSA,")||(gpsData == "GNGSA,")||(gpsData == "$GLGSA,")||(gpsData ==
"GAGSA,")) && (!gsablock)){ //&& (!gsablock)
       gsaFound = true;
```

```
      messagetype = true;
      gpsData = "";
     }
    if(((gpsData == "GPGSV,")||(gpsData == "GNGSV,")||(gpsData == "$GLGSV,")||(gpsData ==
"GAGSV,")) && (!gsvblock)){
      gsvFound = true;
      messagetype = true;
      gpsData = "";
     }
   }
  }
}

void conver_values()
{
  volatile int8_t hours;
  bool day_correction = false, timeStamp_check = false;
  if(rmcFound)
  {
   /* Ajustar a la zona horaria de México (UTC-6) */
   hours = (datarmc[0].toInt()/10000) - 6;
   /* Asegurarse de que la hora esté en el rango correcto (0-23) */
   if (hours < 0) {
    hours += 24;
    day_correction = true;
    //gps1.day--;
   }

   if((hours) < 10)
    gps1.hours = "0" + String(hours);
```

```
else
  gps1.hours = String(hours);

if(((datarmc[0].toInt()/100)%100) < 10)
  gps1.minutes = "0" + String((datarmc[0].toInt()/100)%100);
else
  gps1.minutes = String((datarmc[0].toInt()/100)%100);

if((datarmc[0].toInt()%100) < 10)
  gps1.seconds = "0" + String(datarmc[0].toInt()%100);
else
  gps1.seconds = String(datarmc[0].toInt()%100);
gps1.microseconds = String(datarmc[0].substring(7,9).toInt());

if (day_correction){
  gps1.day = String((datarmc[8].toInt()/10000)-1);
  day_correction=false;
}
else
  gps1.day = String((datarmc[8].toInt()/10000));
if((gps1.day.toInt()) < 10)
  gps1.day = "0" + String((datarmc[8].toInt()/10000));
else
  gps1.day = String(gps1.day.toInt());
if(((datarmc[8].toInt()/100)%100) < 10)
  gps1.mounth = "0" + String((datarmc[8].toInt()/100)%100);
else
  gps1.mounth = String((datarmc[8].toInt()/100)%100);

if(((datarmc[8].toInt()%100) + 2000) < 10)
```

```
  gps1.year = "0" + String((datarmc[8].toInt()%100) + 2000);
else
  gps1.year = String((datarmc[8].toInt()%100) + 2000);

if(!timeStamp_check)
{
  //gps1.utctime = String(rtc.getTime("%H:%M:%S")+":"+rtc.getMillis()/10);
  gps1.utcdate = String(rtc_gps.getDate());
}

gps1.utctime = datarmc[0];
gps1.utctime2 = datarmc[0].toFloat();

if (datarmc[1] == "A")
{
  /* Conversion de los grados,minutos y segundos (sistema sexagesimal) */
  gps1.latitud = (float)datarmc[2].substring(0,2).toInt() +
                 (datarmc[2].substring(2,4).toFloat() +
                 (datarmc[2].substring(5,10).toFloat())/100000)/60;
  if (datarmc[3] == "S")
    gps1.latitud = -gps1.latitud;

  gps1.longitud = (float)datarmc[4].substring(0,3).toInt() +
                 (datarmc[4].substring(3,5).toFloat() +
                 (datarmc[4].substring(6,11).toFloat())/100000)/60;
  if (datarmc[5] == "W")
    gps1.longitud = -gps1.longitud;
  /* Obtner velocidad (en Nudos) y cursor sobre "suelo" (en grados) */
  gps1.speed = (float)datarmc[6].toInt();
  gps1.course = (float)datarmc[7].toInt();
```

```
  }

  else if(datarmc[1] == "V")
  {
    //gps1.latitud = 0;
    //gps1.longitud = 0;
  }
}

if(ggaFound)
{
  /* Ajustar a la zona horaria de México (UTC-6) */
  hours = (datagga[0].toInt()/10000) - 6;
  /* Asegurarse de que la hora esté en el rango correcto (0-23) */
  if (hours < 0) {
    hours += 24;
    //gps1.day--;
  }

  if((hours) < 10)
    gps2.hours = "0" + String(hours);
  else
    gps2.hours = String(hours);

  if(((datagga[0].toInt()/100)%100) < 10)
    gps2.minutes = "0" + String((datagga[0].toInt()/100)%100);
  else
    gps2.minutes = String((datagga[0].toInt()/100)%100);

  if((datagga[0].toInt()%100) < 10)
```

```
  gps2.seconds = "0" + String(datagga[0].toInt()%100);
 else
  gps2.seconds = String(datagga[0].toInt()%100);
 gps2.microseconds = String(datagga[0].substring(7,9).toInt());

 if ((datagga[5] == "1") || (datagga[5] == "2"))
 {
  /* Conversion de los grados,minutos y segundos (sistema sexagesimal) */
  gps2.latitud = (float)datagga[1].substring(0,2).toInt() +
              (datagga[1].substring(2,4).toDouble() +
              (datagga[1].substring(5,10).toDouble())/100000)/60;
  if (datagga[2] == "S")
    gps2.latitud = -gps2.latitud;

  gps2.longitud = (float)datagga[3].substring(0,3).toInt() +
                (datagga[3].substring(3,5).toDouble() +
                (datagga[3].substring(6,11).toDouble())/100000)/60;
  if (datagga[4] == "W")
    gps2.longitud = -gps2.longitud;
 }
 else
 {
  //gps2.latitud = 0;
  //gps2.longitud = 0;
 }
 gps2.sat = datagga[6].toInt();
 gps2.hpod = datagga[7].toFloat();
 gps2.altitud = datagga[8].toFloat();
}
```

```
  if(gsaFound)
  {
    gps3.pdop = datagsa[14].toFloat();
    gps3.hdop = datagsa[15].toFloat();
    gps3.vdop = datagsa[16].toFloat();
  }

  if(gsvFound)
  {
    gps4.sv = datagsa[2].toInt();
  }
}

/* Configuracion de velocidad de muestreo del módulo GNSS */
/* Muestro de 1Hz*/
void SetUp1hZ()
{
  byte packet[] = {
    0xB5, //
    0x62, //
    0x06, //
    0x08, //
    0x06, // length
    0x00, //
    0xE8, // measRate, hex 0X03E8 = dec 1000 ms
    0x03, //
    0x01, // navRate, always =1
    0x00, //
    0x01, // timeRef, stick to GPS time (=1)
    0x00, //
```

```
    0x01, // CK_A
    0x39, // CK_B
  };
 for (byte i = 0; i < 14; i++){
  gpshw.write(packet[i]); // GPS is HardwareSerial
 }
}
/* Muestro de 2Hz*/
void SetUp2hZ()
{
  byte packet[] = {
    0xB5, //
    0x62, //
    0x06, //
    0x08, //
    0x06, // length
    0x00, //
    0xF4, // measRate, hex 0X01F4 = dec 500 ms
    0x01, //
    0x01, // navRate, always =1
    0x00, //
    0x01, // timeRef, stick to GPS time (=1)
    0x00, //
    0x0B, // CK_A
    0x77, // CK_B
  };
 for (byte i = 0; i < 14; i++){
  gpshw.write(packet[i]); // GPS is HardwareSerial
 }
}
```

```
/* Muestro de 4Hz*/
void SetUp4hZ()
{
  byte packet[] = {
    0xB5, //
    0x62, //
    0x06, //
    0x08, //
    0x06, // length
    0x00, //
    0xFA, // measRate, hex 0X00FA = dec 250 ms
    0x00, //
    0x01, // navRate, always =1
    0x00, //
    0x01, // timeRef, stick to GPS time (=1)
    0x00, //
    0x10, // CK_A
    0x96, // CK_B
  };
 for (byte i = 0; i < 14; i++){
  gpshw.write(packet[i]); // GPS is HardwareSerial
 }
}

/* Configuración constelaciones en el módulo GNSS */
void GNSS_config(uint8_t constelaciones)
{
    /* GPS + GLONASS + SBAS + QZSS */
    if(constelaciones==0){
        byte packet[] = {
```

```
      0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
      0x00, 0x00, 0x20, 0x07,
      0x00, 0x08, 0x10, 0x00, 0x01, 0x00, 0x01, 0x01, //GPS +
      0x01, 0x01, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //SBAS +
      0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
      0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
      0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
      0x05, 0x00, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //QZSS +
      0x06, 0x08, 0x0E, 0x00, 0x01, 0x00, 0x01, 0x01, //GLONASS +
      0x2F, 0x89,
   };
   for (byte i = 0; i < 68; i++){
    gpshw.write(packet[i]); // GPS is HardwareSerial
    }
}
/* GPS */
else if(constelaciones==1){
   byte packet[] = {
      0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
      0x00, 0x00, 0x20, 0x07,
      0x00, 0x08, 0x10, 0x00, 0x01, 0x00, 0x01, 0x01, //GPS +
      0x01, 0x01, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //SBAS +
      0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
      0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
      0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
      0x05, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //QZSS +
      0x06, 0x08, 0x0E, 0x00, 0x00, 0x00, 0x01, 0x01, //GLONASS +
      0x2C, 0x4D,
   };
   for (byte i = 0; i < 68; i++){
```

```
      gpshw.write(packet[i]); // GPS is HardwareSerial
      }
}
/* GPS + SBAS + QZSS */
else if(constelaciones==2){
   byte packet[] = {
      0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
      0x00, 0x00, 0x20, 0x07,
      0x00, 0x08, 0x10, 0x00, 0x01, 0x00, 0x01, 0x01, //GPS +
      0x01, 0x01, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //SBAS +
      0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
      0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
      0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
      0x05, 0x00, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //QZSS +
      0x06, 0x08, 0x0E, 0x00, 0x00, 0x00, 0x01, 0x01, //GLONASS +
      0x2E, 0x85,
   };
   for (byte i = 0; i < 68; i++){
     gpshw.write(packet[i]); // GPS is HardwareSerial
     }
}
/* GLONASS */
else if(constelaciones==3){
   byte packet[] = {
      0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
      0x00, 0x00, 0x20, 0x07,
      0x00, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //GPS +
      0x01, 0x01, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //SBAS +
      0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
      0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
```

```
      0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
      0x05, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //QZSS +
      0x06, 0x08, 0x0E, 0x00, 0x01, 0x00, 0x01, 0x01, //GLONASS +
      0x2C, 0x1D,
   };
   for (byte i = 0; i < 68; i++){
    gpshw.write(packet[i]); // GPS is HardwareSerial
    }
}
/* GLONASS + SBAS + QZSS */
else if(constelaciones==4){
   byte packet[] = {
      0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
      0x00, 0x00, 0x20, 0x07,
      0x00, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //GPS +
      0x01, 0x01, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //SBAS +
      0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
      0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
      0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
      0x05, 0x00, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //QZSS +
      0x06, 0x08, 0x0E, 0x00, 0x01, 0x00, 0x01, 0x01, //GLONASS + 56+18=4
      0x2E, 0x55,
   };
   for (byte i = 0; i < 68; i++){
    gpshw.write(packet[i]); // GPS is HardwareSerial
    }
}

/* GPS + GLONASS */
else if(constelaciones==5){
```

```
    byte packet[] = {
        0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
        0x00, 0x00, 0x20, 0x07,
        0x00, 0x08, 0x10, 0x00, 0x01, 0x00, 0x01, 0x01, //GPS +
        0x01, 0x01, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //SBAS +
        0x02, 0x04, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //GALILEO
        0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
        0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
        0x05, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x01, //QZSS +
        0x06, 0x08, 0x0E, 0x00, 0x01, 0x00, 0x01, 0x01, //GLONASS +
        0x2D, 0x51,
    };
    for (byte i = 0; i < 68; i++){
      gpshw.write(packet[i]); // GPS is HardwareSerial
      }
}

/* GPS + GLONASS + SBAS + QZSS + GALILEO */
else if(constelaciones==6){
    byte packet[] = {
        0xB5, 0x62, 0x06, 0x3E, 0x3C, 0x00,
        0x00, 0x00, 0x20, 0x07,
        0x00, 0x08, 0x10, 0x00, 0x01, 0x00, 0x01, 0x01, //GPS +
        0x01, 0x01, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //SBAS +
        0x02, 0x04, 0x08, 0x00, 0x01, 0x00, 0x01, 0x01, //GALILEO
        0x03, 0x08, 0x10, 0x00, 0x00, 0x00, 0x01, 0x01, //BEIDOU
        0x04, 0x00, 0x08, 0x00, 0x00, 0x00, 0x01, 0x01, //IMES
        0x05, 0x00, 0x03, 0x00, 0x01, 0x00, 0x01, 0x01, //QZSS +
        0x06, 0x08, 0x0E, 0x00, 0x01, 0x00, 0x01, 0x01, //GLONASS +
        0x30, 0xAD,
```

```
      };
      for (byte i = 0; i < 68; i++){
        gpshw.write(packet[i]); // GPS is HardwareSerial
        }
   }
}
```

D. CÓDIGO DEL FILTRO DE KALMAN PARA LA ORIENTACIÓN

Este apéndice incluye el programa en C++ que utiliza el framework de Arduino para procesar y obtener estimaciones de los ángulos de Euler mediante el filtro de Kalman.

```
#include "KF.h"
#include <iostream>
#include <BNO055_config.h>

// Variables para el Filtro de Kalman
double dt_IMU = 0.05; // Intervalo de tiempo (ajustar según la frecuencia de muestreo)
double dt_GPS = 0.5; // Intervalo de tiempo (ajustar según la frecuencia de muestreo)

double R3[3][3] = {{0.409581719659758, 0, 0}, {0, 0.435437894075242, 0}, {0, 0, 0.422419451721678}}; //
double Q3[3][3] = {{0.065635956715436, 0, 0}, {0, 0.061299484337187, 0}, {0, 0, 0.054245700928569}}; //

kfType filter;

void init_Kalman_Filter(void){
   for (int row = 0; row < 3; row++) {
      for (int col = 0; col < 3; col++) {
         if (row == col){
            filter.P3[row][col] = 1;
            filter.x3[col] = 0;
         }
         else{
            filter.P3[row][col] = 0;
         }
```

```
      }
    }
}

void IMU_Kalman_Filter(double A[3][3], double B[3][3], double C[3][3],
    long double xhat_min[3][3], long double xhat[3],
    long double P_min[3][3],long double K[3][3],long double P[3][3],long double X[3]){
    /* Predicción */
    xhat_min[0][0] = (A[0][0]*X[0]) + (dt_IMU*B[0][0]*sensor3.gyrosc.x());
    xhat_min[1][1] = (A[1][1]*X[1]) + (dt_IMU*B[1][1]*sensor3.gyrosc.y());
    xhat_min[2][2] = (A[2][2]*X[2]) + (dt_IMU*B[2][2]*sensor3.gyrosc.z());


    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            if (row == col){
                P_min[row][col] = A[row][col]*P[row][col]*A[col][row]+Q3[row][col];
                /* Correción */
                 K[row][col]=(P_min[row][col]*C[col][row])/(C[row][col]*P_min[row][col]*C[col][ro
w]+R3[row][col]);
                P[row][col] = (filter.I3[row][col]-K[row][col]*C[row][col])*P_min[row][col];
                X[row]=xhat_min[row][col]+K[row][col]*(xhat[row]-
            C[row][col]*xhat_min[row][col]);
            }
        }
    }
}
```

E. CÓDIGO DEL FILTRO DE KALMAN PARA LA POSICIÓN

Este apéndice incluye el archivo MAT para procesar y obtener estimaciones de ángulos y posición mediante el filtro de Kalman. También se grafican y se comparan con los resultados obtenidos del prototipo de la VANT.

```
clear;
filename = 'C:\Users\Alex\Desktop\sd_1_junio\16102024.txt';
data = dlmread(filename, ',', 1, 0);
original_data = data;
original_data = original_data(:, 1:20);

% Datos obtenidos del sensor IMU (acelerómetro, giroscopio, magnetómetro)
accel = original_data(:,2:4); gyro = original_data(:,5:7);
mag = original_data(:,8:10); euler = original_data(:,11:13) ;

%% Datosdelta de modulo NEOm8 (GPS)
gps_data = [original_data(:,1), original_data(:,14), original_data(:,15), original_data(:,20), original_data(:,16), original_data(:,17), original_data(:,18), original_data(:,19)];

%Grafico de datos de modulo NEOm8
timestamp = gps_data(:,1);
latitud = gps_data(:,2);
longitud = gps_data(:,3);
alt = gps_data(:,4);
pdop = gps_data(:,5);
hdop = gps_data(:,6);
vdop = gps_data(:,7);
sat = gps_data(:,8);
```

```
% Radio de la Tierra en metros
Radio = 6378137;  % Aproximadamente 6371 km
refloc = [20.607209333333331, -103.413280000000007, 1604];

lat1_rad = deg2rad(latitud(1));
lon1_rad = deg2rad(longitud(1));
for ii=1:size(alt,1)
   % Convertir latitudes y longitudes de grados a radianes
   lat2_rad = deg2rad(latitud(ii));
   lon2_rad = deg2rad(longitud(ii));
   % Diferencias de latitud y longitud
   delta_lat = lat2_rad - lat1_rad;
   delta_lon = lon2_rad - lon1_rad;
   % Calcular la distancia en la dirección Norte (N)
   N = Radio * delta_lat;
   % Calcular la distancia en la dirección Este (E), ajustada por la latitud
   E = Radio * cos(lat1_rad) * delta_lon;
   % Calcular la distancia en la dirección Down (D) 1608.20
   D = alt(1) - alt(ii);  % Diferencia de altitud (positiva hacia abajo)
   pos_gps(ii,:) = [N,E,D];
end

%% Calculo de matriz de rotación y aceleración por NED
for ii=1:size(euler_data,1)
rmat = [
       cosd(euler(ii,2)).*cosd(euler(ii,1)),...
      -cosd(euler(ii,2)).*sind(euler(ii,1)),...
       sind(euler(ii,2))
       sind(euler(ii,3)).*sind(euler(ii,2)).*cosd(euler(ii,1))+cosd(euler(ii,3)).*sind(euler(ii,1)) ,...
```

```
    -sind(euler(ii,3)).*sind(euler(ii,2)).*sind(euler(ii,1))+cosd(euler(ii,3)).*cosd(euler(ii,1)) ,...
    -sind(euler(ii,3)).*cosd(euler(ii,2));...
    -cosd(euler(ii,3)).*sind(euler(ii,2)).*cosd(euler(ii,1))+sind(euler(ii,3)).*sind(euler(ii,1)) ,...
     cosd(euler(ii,3)).*sind(euler(ii,2)).*sind(euler(ii,1))+sind(euler(ii,3)).*cosd(euler(ii,1)) ,...
     cosd(euler(ii,3)).*cosd(euler(ii,2));...
];

  aceel_NED(ii,:) = rmat*((accel(ii,:)/9.8)');

  if (ii == 1 || ii == size(pos_gps,1))
     gps_vel(ii,1) = 0;
     gps_vel(ii,2) = 0;
     gps_vel(ii,3) = 0;
  else
     gps_vel_prevN = (pos_gps(ii+1,1)-pos_gps(ii,1))/0.5;
     gps_vel_prevE = (pos_gps(ii+1,2)-pos_gps(ii,2))/0.5;
     gps_vel_prevD = (pos_gps(ii+1,3)-pos_gps(ii,3))/0.5;

     if(gps_vel_prevN ~= 0)
        gps_vel(ii,1) = gps_vel_prevN;
     else
        gps_vel(ii,1) = gps_vel(ii-1,1);
     end
     if(gps_vel_prevE ~= 0)
        gps_vel(ii,2) = gps_vel_prevE;
     else
        gps_vel(ii,2) = gps_vel(ii-1,2);
     end
     if(gps_vel_prevD ~= 0)
        gps_vel(ii,3) = gps_vel_prevD;
```

```
        else
            gps_vel(ii,3) = gps_vel(ii-1,3);
        end
    end
end

%% Matrices de estado de FK GPS+IMU
deltat_gps = 0.5;
deltat = 0.05;
A = [eye(3) deltat*eye(3); zeros(3) eye(3)];
B = [zeros(3); deltat*eye(3)];
C = [eye(3) zeros(3)];
z = [pos_gps];
% Covarianza de medición obtenida por experimentación
R = [(0.775369184638881)^2 0 0 ;0 (0.653883683413194)^2 0 ; 0 0 (3.881485988578624)^2];
% Covarianza de procesamiento
Q = diag([((0.01818523801)^2), ((0.01818523801)^2), ((0.03329277994)^2), ...
          ((0.01818523801)^2), ((0.01818523801)^2), ((0.03329277994)^2)]);
XK = [0; 0; 0; 0; 0; 0];
u = [aceel_NED(:,1) aceel_NED(:,2) aceel_NED(:,3)];
n = size(pos_gps,1);
P = eye(6);

j=0;
% Filtro de kalman por código
for i=1:n;
    % Predición
    Xmenos = (A*XK)+B*u(i,:)';
    Pmenos = A*P*A'+Q;
    % Correción
```

```
    K = (Pmenos* C') / (C*Pmenos*C' + R);
    XK = Xmenos + K*(z(i,:)'-C*Xmenos);
    P = (eye(6)-K*C)*Pmenos;
    pos_estimado(i,:) = [XK(1) XK(2) XK(3)];
    vel_estimado(i,1) =XK(4);
    vel_estimado(i,2) =XK(5);
    vel_estimado(i,3) =XK(6);
end

%% Graficación del experimento
X = 0:0.05:522.95;
segmentosX0 = [
    0.00      0    68.65     0;
    68.65     0    168.30 -102.82;
    168.30 -102.82 230.25 -102.82;
    230.25 -102.82 331.90    0;
    331.90    0    374.40    0;
    374.40    0    428.65 -102.82;
    428.65 -102.82 465.70 -102.82;
    465.70 -102.82 522.95   -6;
    ];
    % Rellenar Y con un bucle
    for k = 1:size(segmentosX0,1)
        x1 = segmentosX0(k,1);
        y1 = segmentosX0(k,2);
        x2 = segmentosX0(k,3);
        y2 = segmentosX0(k,4);
        idx = X >= x1 & X <= x2;
        if y1 == y2
            % --- Segmento constante ---
```

```
        Yx(idx) = y1;
    else
        % --- Segmento con pendiente ---
        Yx(idx) = y1 + (y2 - y1) / (x2 - x1) * (X(idx) - x1);
    end
end

segmentosY0 = [
    0.00      0    11.75   0;
    11.75      0   68.10  62.1;
    68.10   62.1  168.3  62.1;
    168.3  62.1  229.15   0;
    229.15    0   334.25   0;
    334.25    0   374.25  62.1;
    374.25  62.1  428.55  62.1;
    428.55  62.1  465.40   0;
    465.40    0   522.95   0;
    ];
    % Rellenar Y con un bucle
    for k = 1:size(segmentosY0,1)
        x1 = segmentosY0(k,1);
        y1 = segmentosY0(k,2);
        x2 = segmentosY0(k,3);
        y2 = segmentosY0(k,4);
        idx = X >= x1 & X <= x2;
        if y1 == y2
            % --- Segmento constante ---
            Yy(idx) = y1;
        else
            % --- Segmento con pendiente ---
```

```
        Yy(idx) = y1 + (y2 - y1) / (x2 - x1) * (X(idx) - x1);
    end
end

segmentosZ0 = [
    0.00  0  522.95  0;    % constante Y=0
];
% Rellenar Y con un bucle
for k = 1:size(segmentosZ0,1)
    x1 = segmentosZ0(k,1);
    y1 = segmentosZ0(k,2);
    x2 = segmentosZ0(k,3);
    y2 = segmentosZ0(k,4);
    idx = X >= x1 & X <= x2;
    if y1 == y2
        % --- Segmento constante ---
        Yz(idx) = y1;
    else
        % --- Segmento con pendiente ---
        Yz(idx) = y1 + (y2 - y1) / (x2 - x1) * (X(idx) - x1);
    end
end
x_cancha = [0, 62.1, 62.1, 0, 0];
y_cancha = [0, 0, -102.82, -102.82, 0];

figure(1);
plot(x_cancha,y_cancha,'y')
hold on;
plot(pos_gps(:,2),pos_gps(:,1),'m')
hold on;
```

```
xlabel('x (mts)')
ylabel('y (mts)')
title('Comparación de prueba trayectoria');
plot(pos_estimado(:,2),pos_estimado(:,1), 'c')
hold on;
legend("Trayectoria esperado", "GPS", "Filtro de Kalman (GPS+IMU)");
hold on;
axis('equal');

t = (1:size(pos_gps, 1));
figure(2);
subplot(3,1,1);
plot(X, Yx, 'y', 'LineWidth', 1);
hold on;
plot(X, pos_gps(:,1), 'm', 'LineWidth', 2);
hold on;
plot(X, pos_estimado(:,1)', 'c', 'LineWidth', 2);
xlabel('Tiempo [Segundos]');
ylabel('Posición de X [metros]');
title('Posición de X a lo largo del tiempo');
legend("Trayectoria esperado", "GPS", "Filtro de Kalman (GPS+IMU)");
grid on;

subplot(3,1,2);
plot(X, Yy, 'y', 'LineWidth', 1);
hold on;
plot(X, pos_gps(:,2), 'm', 'LineWidth', 2);
hold on;
plot(X, pos_estimado(:,2)', 'c', 'LineWidth', 2);
```

```
hold on;
xlabel('Tiempo [Segundos]');
ylabel('Posición de Y [metros]');
title('Posición de Y a lo largo del tiempo');
legend("Trayectoria esperado", "GPS", "Filtro de Kalman (GPS+IMU)");
grid on;

subplot(3,1,3);
plot(X, Yz, 'y', 'LineWidth', 1);
hold on;
plot(X, pos_gps(:,3), 'm', 'LineWidth', 2);
hold on;
plot(X, pos_estimado(:,3)', 'c', 'LineWidth', 2);
hold on;
xlabel('Tiempo [Segundos]');
ylabel('Posición de Z [metros]');
title('Posición de Z a lo largo del tiempo');
legend("Trayectoria esperado", "GPS", "Filtro de Kalman (GPS+IMU)");
grid on;

figure(3);
geoplot(trazolat,trazolon,'c');
hold on;
geoplot(gpslat,gpslon,'m');
hold on;
geoplot(kalmanlat,kalmanlon,'b');
hold on;
title('Imagen satelital de prueba de trayectoria');
legend("Trayectoria esperado", "GPS", "Filtro de Kalman (GPS+IMU)");
geobasemap satellite
```

```
subplot(3,1,2);
plot(X, gps_vel(:,2), 'm', 'LineWidth', 2);
hold on;
plot(X, vel_estimado(:,2), 'b', 'LineWidth', 2);
hold on;
xlabel('Tiempo [Segundos]');
ylabel('Velocidad de Y [m/seg]');
title('Velocidad de Y a lo largo del tiempo');
legend("GPS", "Filtro de Kalman (GPS+IMU)");
grid on;

subplot(3,1,3);
plot(X, gps_vel(:,3), 'm', 'LineWidth', 2);
hold on;
plot(X, vel_estimado(:,3), 'b', 'LineWidth', 2);
hold on;
xlabel('Tiempo [Segundos]');
ylabel('Velocidad de Z [m/seg]');
title('Velocidad de Z a lo largo del tiempo');
legend("GPS", "Filtro de Kalman (GPS+IMU)");
grid on;
```

F. REPOSITORIO DEL PROTOTIPO

Este apéndice contiene la dirección del repositorio en GitHub para que los lectores puedan tener acceso al proyecto y ayudarlos a tener las bases necesarias para desarrollar su propio prototipo de VANT.

https://github.com/AlexAbrica12/Prototipo-VANT.git

Índice