ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
(ICAI)

Trabajo de Fin de Grado

DISEÑO E IMPLANTACIÓN DEL SISTEMA DE CONTROL DE UN CUADRICÓPTERO AUTÓNOMO

Autor
Álvaro Maestroarena Navamuel

Dirigido por
Juan Luis Zamora

Madrid
Junio 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título:

DISEÑO E IMPLANTACIÓN DEL SISTEMA DE CONTROL DE UN CUADRICÓPTERO AUTÓNOMO

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/2025 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Álvaro Maestroarena Navamuel Fecha: 10/07/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Luis Zamora Fecha: 11/07/2025

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
(ICAI)

Trabajo de Fin de Grado

DISEÑO E IMPLANTACIÓN DEL SISTEMA DE CONTROL DE UN CUADRICÓPTERO AUTÓNOMO

Autor
Álvaro Maestroarena Navamuel

Dirigido por
Juan Luis Zamora

Madrid
Junio 2025

Resumen Ejecutivo

Título: Diseño e Implantación del Sistema de Control de un Cuadricóptero Autónomo

Autor: Álvaro Maestroarena Navamuel

Entidad colaboradora: Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas

Resumen

Este Trabajo de Fin de Grado presenta el diseño e implantación de un sistema de control autónomo para un cuadricóptero que opera en interiores, sin necesidad de sistemas de posicionamiento global como el GPS. Se ha adoptado una configuración en "X" y se ha desarrollado un modelo dinámico detallado que incluye los empujes de los motores, los pares de reacción, el retardo en los actuadores y la dinámica completa del sistema.

El controlador implementado es un regulador lineal cuadrático (LQR), basado en un modelo linealizado y desacoplado, que ha sido extendido con retardos e integración del error para mejorar la estabilidad y el seguimiento de trayectorias mediante la técnica de la realimentación de estado.

La solución se ha desarrollado en MATLAB/Simulink, empleando bloques personalizados y la UAV Toolbox for PX4 Autopilots, y se ha validado en un entorno de simulación que reproduce el comportamiento físico del dron. Finalmente, el sistema se ha desplegado sobre hardware real con el controlador CubePilot Cube Orange+, sensores inerciales, LIDAR, motores BLDC y comunicación vía ESP32-C3.

El resultado es un sistema de control robusto, capaz de gestionar de forma autónoma la actitud y navegación del dron, estableciendo una base sólida para futuras pruebas en vuelo real.

Introducción

Los UAVs (vehículos aéreos no tripulados) han ganado protagonismo en múltiples sectores gracias a sus capacidades de navegación autónoma y bajo coste relativo. No obstante, su uso en interiores plantea importantes retos técnicos, especialmente en la estimación de estados y el control sin señales externas. Este proyecto nace con la motivación de desarrollar una plataforma capaz de volar de forma autónoma en interiores, como paso previo a aplicaciones industriales, educativas o científicas.

Motivación y objetivos

El desarrollo de UAVs autónomos supone un área de gran proyección dentro de la ingeniería, especialmente en aplicaciones donde se requiere precisión y autonomía sin apoyo de sistemas externos como el GPS. Entre las distintas plataformas disponibles, el cuadricóptero destaca por su simplicidad mecánica y su capacidad de maniobra, lo que lo convierte en una excelente base para el estudio de técnicas avanzadas de control.

El objetivo principal de este trabajo es diseñar e implementar un sistema de control autónomo para un cuadricóptero que opere en interiores. Para ello, se plantean los siguientes objetivos específicos: modelar dinámicamente el sistema, diseñar un controlador óptimo tipo LQR, validar su comportamiento en simulación usando MATLAB/Simulink, integrarlo en hardware real mediante el autopiloto Cube Orange+, y evaluar su respuesta frente a trayectorias de referencia.

Metodología y desarrollo

El proyecto se ha desarrollado siguiendo una metodología estructurada, combinando fases de modelado, control, simulación e implementación. Tras una primera etapa de colaboración en el montaje y configuración del dron, se elaboró un modelo dinámico completo que incluye efectos como empuje, par de reacción y retardos de actuadores. Este modelo fue linealizado y discretizado para el diseño de un controlador LQR con integración del error.

La implementación se realizó en MATLAB/Simulink, empleando bloques personalizados y la UAV Toolbox for PX4 Autopilots para establecer la conexión con el

firmware PX4. Se desarrolló una arquitectura jerárquica dividida en subsistemas de control, simulación, hardware y monitorización. Una vez validado el sistema en simulación, se integró sobre el controlador Cube Orange+ junto a sensores inerciales, un LIDAR TFmini-S y motores BLDC. El sistema fue monitorizado mediante QGroundControl a través de comunicación UDP.

Resultados

A lo largo del desarrollo del proyecto se ha implementado una arquitectura completa de control autónomo para un cuadricóptero, incluyendo el modelado dinámico de la planta, el diseño de un controlador LQR, la creación de una máquina de estados y la integración completa en Simulink del sistema de simulación, control y comunicaciones.

Se han realizado múltiples pruebas en entorno de simulación, verificando el comportamiento del controlador ante diferentes escenarios y referencias. Asimismo, se han validado por separado los principales componentes físicos: sensores inerciales (IMU), sensor de altura LiDAR TFmini-S, emisora RC y módulo ESP32-C3. Todos estos elementos han sido integrados con éxito en el sistema embarcado del autopiloto Cube Orange+, incluyendo la configuración de comunicaciones mediante MAVLink y la generación de firmware compatible.

Aunque no se ha alcanzado la fase de vuelo real, el sistema queda preparado para su validación experimental una vez completada la etapa de integración final y puesta a punto del hardware. El trabajo realizado constituye una base sólida para continuar con ensayos de vuelo en el futuro.

Referencias más relevantes

1 MathWorks, "UAV Toolbox Support Package for PX4 Autopilots," 2025.

2 PX4 Autopilot, "CubePilot Cube Orange Flight Controller," 2025.

23 L. Martin, et al., "Linear quadratic regulator for trajectory tracking of a quadrotor," *Control Engineering Practice*, 2019.

24 P. Saraf, et al., "A comparative study between a classical and optimal controller for a quadrotor," *arXiv*, 2020.

Executive Summary

Title: Design and Implementation of the Control System of an Autonomous Quadcopter

Author: Álvaro Maestroarena Navamuel

Collaborating Institution: ICAI School of Engineering, Comillas Pontifical University

Abstract

This Bachelor's Thesis presents the design and implementation of an autonomous control system for a quadcopter operating indoors, without the need for global positioning systems such as GPS. An "X" configuration has been adopted, and a detailed dynamic model has been developed, including motor thrust, reaction torques, actuator delays, and the complete system dynamics.

The implemented controller is a Linear Quadratic Regulator (LQR), based on a linearized and decoupled model, extended with delays and error integration to improve stability and trajectory tracking through state feedback techniques.

The solution has been developed in MATLAB/Simulink, using custom blocks and the UAV Toolbox for PX4 Autopilots, and has been validated in a simulation environment that replicates the physical behavior of the drone. Finally, the system has been deployed on real hardware using the CubePilot Cube Orange+ controller, inertial sensors, LiDAR, BLDC motors, and communication via ESP32-C3.

The result is a robust control system, capable of autonomously managing the drone's attitude and navigation, providing a solid foundation for future real-flight testing.

Introduction

UAVs (Unmanned Aerial Vehicles) have gained prominence in multiple sectors due to their autonomous navigation capabilities and relatively low cost. However, their

use indoors presents significant technical challenges, especially in state estimation and control without external signals.

This project is driven by the motivation to develop a platform capable of flying autonomously indoors, as a preliminary step toward industrial, educational, or scientific applications.

Motivation and Objectives

The development of autonomous UAVs represents a highly promising field within engineering, especially in applications requiring precision and autonomy without relying on external systems such as GPS. Among the various available platforms, the quadcopter stands out due to its mechanical simplicity and maneuverability, making it an excellent base for the study of advanced control techniques.

The main objective of this project is to design and implement an autonomous control system for a quadcopter operating indoors. To achieve this, the following specific objectives are proposed: to model the system dynamics, design an optimal LQR controller, validate its performance in simulation using MATLAB/Simulink, implement it on real hardware using the Cube Orange+ autopilot, and evaluate its response to reference trajectories.

Methodology and Development

The project has been developed following a structured methodology, combining modeling, control, simulation, and implementation phases. After an initial stage of collaboration in assembling and configuring the drone, a complete dynamic model was built, including effects such as thrust, reaction torques, and actuator delays. This model was then linearized and discretized for the design of an LQR controller with error integration.

The implementation was carried out in MATLAB/Simulink, using custom blocks and the UAV Toolbox for PX4 Autopilots to interface with the PX4 firmware. A hierarchical architecture was developed, divided into control, simulation, hardware, and monitoring subsystems. Once validated in simulation, the system was integrated on the Cube Orange+ controller, along with inertial sensors, a TFmini-S LiDAR, and BLDC motors. The system was monitored using QGroundControl via UDP communication.

Results

Throughout the development of the project, a complete autonomous control architecture for a quadcopter has been implemented, including dynamic modeling of the plant, LQR controller design, the creation of a state machine, and full integration in Simulink of the simulation, control, and communication system.

Multiple tests have been conducted in the simulation environment, verifying the controller's behavior under different scenarios and reference trajectories. Additionally, the main physical components have been individually validated: inertial sensors (IMU), the TFmini-S LiDAR height sensor, RC transmitter, and ESP32-C3 module. All these elements have been successfully integrated into the Cube Orange+ autopilot embedded system, including communication setup via MAVLink and generation of compatible firmware.

Although real flight has not yet been reached, the system is ready for experimental validation once the final integration and hardware tuning are completed. The work carried out provides a solid foundation for continuing with flight tests in the future.

Most Relevant References

1 MathWorks, "UAV Toolbox Support Package for PX4 Autopilots," 2025.

2 PX4 Autopilot, "CubePilot Cube Orange Flight Controller," 2025.

23 L. Martin, et al., "Linear quadratic regulator for trajectory tracking of a quadrotor," *Control Engineering Practice*, 2019.

24 P. Saraf, et al., "A comparative study between a classical and optimal controller for a quadrotor," *arXiv*, 2020.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este Trabajo de Fin de Grado.

En primer lugar, al profesor Juan Luis Zamora. Gracias por ofrecerme este proyecto y por brindarme la oportunidad de trabajar en un reto tan enriquecedor y en el que he aprendido tanto.

A mi compañera Clara Lucena Vicente, con quien he compartido este proyecto a lo largo de todo el año. Gracias por tu implicación, paciencia, perseverancia y por haber formado un equipo en el que nos hemos complementado perfectamente.

Y, por supuesto, a mi familia. En especial, a mis padres. Gracias por vuestro apoyo incondicional y por estar siempre pendientes de mí. Gracias por preguntarme cada día cómo iba el proyecto, por animarme incluso en los momentos complicados y por hacerme sentir acompañado durante todo el camino.

Junio 22, 2025

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Acrónimos

BLDC	Brushless Direct Current
DCL	Diagrama de Cuerpo Libre
EKF	Extended Kalman Filter
ESC	Electronic Speed Controllers
FS-A8S	FlySky FS-A8S (Receptor RC)
FS-i6S	FlySky FS-i6S (Emisora RC)
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
ICAI	Instituto Católico de Artes e Industrias
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
LPF	Low-Pass Filter
LQR	Linear-Quadratic Regulator
MAVLink	Micro Air Vehicle Link
MCS	Motion Capture System
PDB	Power Distribution Board
PID	Proporcional-Integral-Derivativo
PWM	Pulse Width Modulation
RC	Remote Control
SFC	State Feedback Control
SISO	Single Input Single Output
TFG	Trabajo de Fin de Grado
TFmini-S	LiDAR TFmini-S Sensor
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
VSCode	Visual Studio Code
ZOH	Zero Order Hold

1. Introducción

En la última década, el uso de UAVs (Vehículos Aéreos no Tripulados, por sus siglas en inglés), comúnmente conocidos como drones, ha experimentado mucho crecimiento en múltiples sectores como la industria, la logística, la vigilancia o la investigación científica. Este desarrollo ha sido posible gracias al avance en tecnologías de navegación, control y autonomía, permitiendo que los UAVs realicen tareas cada vez más complejas con una menor intervención humana.

En este proyecto se ha implementado un sistema de control para un cuadricóptero autónomo de cuatro rotores, con capacidad de operar en interiores. El controlador seleccionado es un regulador lineal cuadrático (LQR, Linear Quadratic Regulator), una técnica de control óptimo adecuada para sistemas dinámicos lineales, que permite equilibrar el rendimiento del sistema con el esfuerzo de control. El objetivo es garantizar la estabilidad del dron y un comportamiento preciso en vuelo, especialmente en tareas de actitud y mantenimiento de posición.

El desarrollo se realiza utilizando el entorno Matlab/Simulink, integrando el código con el microcontrolador CubePilot Cube Orange+. Además, se utiliza QGroundControl como herramienta de monitorización y telemetría. Este trabajo se lleva a cabo en colaboración con Clara Lucena Vicente, responsable del diseño y ensamblaje del hardware, dentro del entorno académico de ICAI, en coordinación con el equipo mecánico y en el marco de proyectos previos realizados en la misma institución.

1.1. Motivación

El desarrollo de UAVs autónomos representa uno de los campos con mayor proyección dentro de la ingeniería, dado su impacto potencial en sectores donde se requiere precisión, autonomía y capacidad de adaptación. Entre las distintas configuraciones posibles, el cuadricóptero destaca por su simplicidad estructural y por su capacidad de hacer maniobras, lo que lo convierte en una plataforma ideal para el estudio de técnicas avanzadas de control.

La implementación de un sistema de control sobre un dron de tipo cuadricóptero aborda múltiples desafíos reales, como la estabilidad en vuelo, la gestión de perturbaciones y la navegación sin dependencia de señales externas. Esto habilita su

uso en entornos interiores o limitados, donde es necesario un control preciso.

Además, este tipo de desarrollo permite aplicar conocimientos en áreas clave de la ingeniería, como el diseño y el ajuste de controladores, la simulación de sistemas dinámicos o la implementación práctica sobre hardware de vuelo. También ofrece la oportunidad de validar estos conceptos en condiciones reales, lo cual es esencial para su aplicación en contextos industriales o de investigación.

1.2. Objetivos

El objetivo principal de este trabajo esdiseñar e implementar un sistema de control autónomo para un cuadricóptero para que sea capaz de operar en entornos interiores, sin depender de sistemas de posicionamiento global. Este sistema debe garantizar la estabilidad y precisión del vehículo durante el vuelo, mediante el uso de técnicas avanzadas de control lineal.

Como objetivos específicos, se plantean:

- Desarrollar un modelo dinámico del cuadricóptero que permita comprender su comportamiento y sirva como base para el diseño del controlador.
- Diseñar un controlador óptimo basado en la técnica LQR (Linear Quadratic Regulator), que permita estabilizar el dron y controlar su actitud.
- Implementar dicho sistema de control en un entorno de simulación realista utilizando *Matlab/Simulink*, validando su comportamiento antes de trasladarlo a hardware real.
- Integrar el sistema de control en la arquitectura del autopiloto *Cube Orange+*, asegurando la comunicación con sensores y módulos de actuación.
- Evaluar la respuesta del sistema en pruebas de simulación y establecer criterios de rendimiento y estabilidad para el vuelo autónomo.

1.3. Metodología

El desarrollo de este trabajo ha seguido una metodología estructurada que combina fases teóricas, prácticas y de validación. Desde el inicio, se ha trabajado en coordinación con Clara Lucena Vicente, responsable del diseño y ensamblaje del hardware. En esta primera etapa, he colaborado también en el montaje y en la

configuración del cuadricóptero, a pesar de que el foco principal del presente TFG reside en el diseño del sistema de control.

Una parte fundamental del trabajo ha sido el estudio detallado de la *UAV Toolbox Support Package for PX4 Autopilots* [1], una librería que permite integrar modelos y bloques ya desarrollados en *Matlab/Simulink* con el firmware PX4.

Posteriormente, se han llevado a cabo distintos ensayos y simulaciones con el objetivo de verificar el comportamiento del sistema en distintas condiciones. Estas simulaciones han permitido modelar la planta del dron y ajustar los parámetros del controlador LQR antes de su aplicación práctica.

Durante el desarrollo del proyecto, se han identificado y corregido múltiples errores en la implementación, llevando a cabo procesos iterativos de depuración y validación, utilizando las herramientas que la propia librería ofrece.

Esta metodología ha permitido desarrollar un sistema de control autónomo robusto, con una base sólida para su futura validación sobre el dron real.

1.4. Recursos

En esta sección se describen todos los componentes físicos y los programas y aplicaciones utilizados para el desarrollo del proyecto, tanto los facilitados por la universidad como los adquiridos específicamente para el proyecto.

1.4.1. Elementos Físicos (Hardware)

CubePilot Cube Orange+

Para el control del dron, utilizaremos el CubePilot Cube Orange+ (véase Figura 1.1), un controlador diseñado para usarse como una placa base específica para reducir el cableado, mejorar la fiabilidad y facilitar el montaje. Se utilizará por su aplicación de vuelo avanzado que permite la integración de diversos sensores y actuadores.

El Cube Orange+ forma parte de la familia de controladores descrita en la documentación oficial de CubePilot [2].

Figura 1.1: CubePilot Cube Orange+ (Fuente: RobotShop)

Dicho controlador está plenamente documentado en la página oficial del proyecto PX4 [3].

Sensor de Altura – TFmini-S

Para una mejor medición de la altura, empleamos el TFmini-S, un sensor de altura LiDAR compacto que ofrece mediciones de distancia con alta precisión (véase Figura 1.2). El propio controlador en principio puede calcular o estimar la altura a partir de las medidas de los giroscopios y acelerómetros, pero se ha añadido para obtener una mejor medida.

Figura 1.2: Sensor de altura - TFmini-S (Fuente: Mouser Electronics)

Batería

Para alimentar al controlador, se dispone de varias baterías Zeee 3S 1500mAh 120C (Figura 1.3). Estas baterías proporcionan una buena relación entre capacidad, peso y descarga instantánea, adecuada para la demanda de corriente de los motores y el controlador.

Figura 1.3: Batería Zeee 3S 1500mAh 120C (Fuente: Amazon)

Hélices tripala

La propulsión del dron se logra mediante hélices tripalas 5x3.1x3.5 de 5 pulgadas, las cuales proporcionan un buen equilibrio entre control y eficiencia (Figura 1.4). Estas hélices han sido seleccionadas por su rendimiento en condiciones de vuelo en interiores, donde se prioriza la maniobrabilidad y la respuesta rápida frente al empuje máximo.

Figura 1.4: Hélices tripalas 5x3x3 5 pulgadas (Fuente: Amazon)

Motores

Hemos reutilizado algunos de los motores disponibles en el laboratorio: Motores CrazePony DX2205 2300KV y Racerstar BR2205 2300KV (Motores BLDC, Figura 1.5).

Hicimos pruebas con 7 motores, medimos la fuerza que ejercen en función de la velocidad de giro y escogimos los más parecidos. Este proceso se realizó para asegurar que el comportamiento físico del dron sea lo más fiel posible al modelo matemático utilizado en el desarrollo del sistema de control, donde se asume que todos los motores son idénticos.

Figura 1.5: Motor Racerstar BR2205 2300KV (Fuente: Amazon)

ESC BLHeli

Para controlar la velocidad de los motores brushless, se utilizan ESCs (Electronic Speed Controllers, Figura 1.6) BLHeli, los cuales reciben las señales del CubePilot Cube Orange+ y modulan la potencia enviada a cada motor.

Figura 1.6: Electronic Speed Controllers - BLHeli (Fuente: RC_Drone.top)

PDB con conector XT60

El sistema de distribución de energía se basa en una Power Distribution Board (PDB) equipada con un conector XT60, desde la cual se alimentan los ESCs y, por ende, los motores. Véase la Figura 1.7.

Figura 1.7: PDB con conector XT60 (Fuente: Amazon)

Emisora FS-i6S y receptor FS-A8S

Para el control manual del dron, se utiliza una emisora FlySky FS-i6S junto con el receptor FS-A8S v2 (Figura 1.8). Este sistema de radiocontrol permite manejar el vehículo en modo manual durante las pruebas, así como actuar como sistema de respaldo en caso de fallo del control autónomo. El receptor está conectado al Cube Orange+ mediante la entrada de señal RC (PWM/PPM/SBUS).

Figura 1.8: Emisora FS-i6S y receptor FS-A8S (Fuente: Amazon)

Módulo Wi-Fi – ESP32-C3

Para establecer comunicaciones inalámbricas de tipo UDP, se ha integrado un módulo ESP32-C3 (Figura 1.9), el cual permite el envío y recepción de datos entre el dron y una estación en tierra (QGroundControl). Este enlace es fundamental para las tareas de monitorización, recepción de comandos o pruebas de navegación sin la necesidad de cables físicos.

Figura 1.9: ESP32-C3-DevKitC-02 (Fuente: Datasheet ESP32)

Estructura del UAV

La estructura del dron ha sido diseñada y montada específicamente para el proyecto (véase Figura 1.10). Sigue una configuración en X y ha sido diseñada como Trabajo de Tin de Máster de Antonio Engelinus. La disposición de los brazos, el soporte de los motores y el alojamiento del Cube Orange+ se ha optimizado para facilitar el acceso a los componentes.

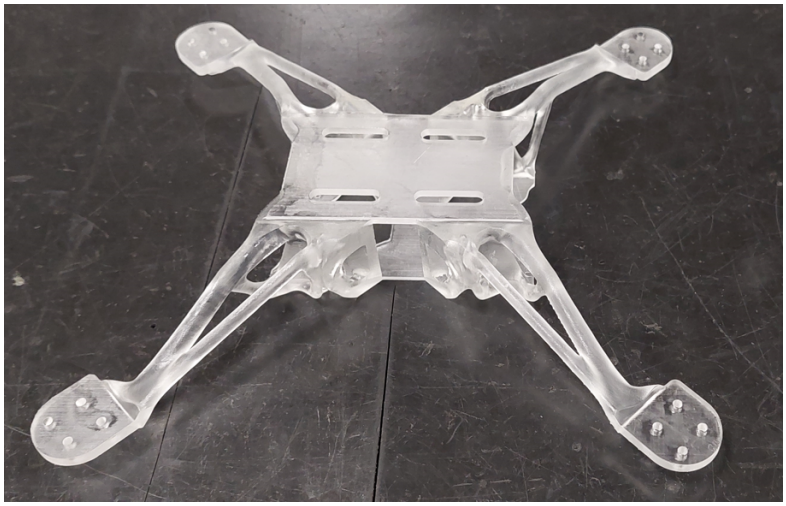

Figura 1.10: Prototipo de la estructura del UAV

1.4.2. Programas y Aplicaciones (Software)

Matlab 2024b/Simulink

Matlab 2024b y Simulink han sido las herramientas a elegir para el desarrollo del sistema de control del cuadricóptero autónomo. Matlab es un entorno de programación de alto nivel que permite el análisis numérico y la simulación, mientras que Simulink proporciona una plataforma gráfica para modelar, simular y analizar sistemas dinámicos. En este proyecto, Matlab/Simulink se ha empleado tanto en la simulación del modelo dinámico como en la implementación de los controladores y la integración con los sensores y actuadores del dron. Por último, se ha empleado el "UAV Toolbox Support Package for PX4 Autopilots", una extensión descargable de Matlab. Se ha utilizado para facilitar la conexión y comunicación entre Simulink y el CubePilot Cube Orange+.

QGroundControl

QGroundControl es una aplicación de control y monitorización de vehículos UAV que se ha utilizado como herramienta para leer las variables de interés del dron durante las pruebas. A través de su interfaz, se puede acceder a información en tiempo real sobre el estado del vehículo, la altitud, la velocidad, la orientación o, en general, cualquier variable que queramos. La comunicación entre el controlador y el ordenador puede establecerse a través de un cable serie o mediante el protocolo MAVLink (véase Figura 1.11).

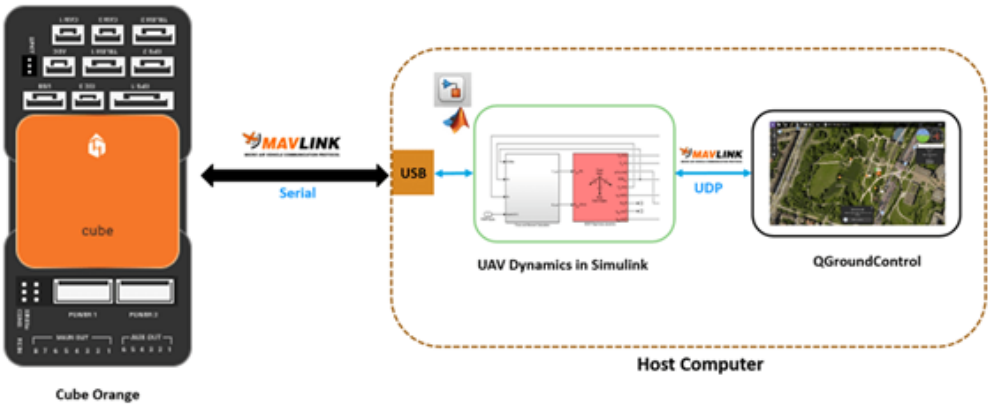

Figura 1.11: Comunicación entre Cube Orange+, Matlab y QGroundControl

En este proyecto, QGroundControl ha sido especialmente útil para monitorear las lecturas de los sensores y ajustar los parámetros del sistema de control durante las fases de prueba del cuadricóptero.

VSCode

VSCode se ha utilizado principalmente para realizar pruebas sencillas de comunicación UDP [4] y para la configuración del sensor de altura TFmini-S. Aunque no es la herramienta principal del proyecto, VSCode ha servido para verificar de una forma rápida la conectividad y el funcionamiento básico de ciertos componentes del sistema de control.

```python
import socket

# Configuracion del socket UDP
UDP_IP = "0.0.0.0"  # Escucha en todas las interfaces
UDP_PORT = 14550    # Puerto usado por DroneBridge/QGroundControl

# Crear socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

print(f"Escuchando MAVLink en UDP puerto {UDP_PORT}...\n")

try:
    while True:
        data, addr = sock.recvfrom(1024)
        print(f"Paquete recibido de {addr}: {len(data)} bytes")
        if data[0] == 0xfe or data[0] == 0xfd:
            print("Paquete MAVLink valido (inicio 0xfe o 0xfd)\n")
        else:
            print("No parece un paquete MAVLink\n")
except KeyboardInterrupt:
    print("Interrumpido por el usuario")
finally:
    sock.close()
```

Código 1.1: Recepción de paquetes MAVLink mediante UDP en Python

2. Estado del Arte

El avance de los sistemas UAV ha sido impulsado en gran medida por los progresos en navegación, control y sensorización. En este capítulo se presenta una revisión de los conceptos fundamentales relacionados con la navegación autónoma de drones, así como una visión general de las principales aplicaciones actuales en distintos sectores.

2.1. Navegación Autónoma

La navegación autónoma es uno de los pilares fundamentales para el funcionamiento independiente de los UAVs. Engloba el conjunto de técnicas que permiten al dron determinar su posición y orientación en el espacio, planificar trayectorias y ejecutar movimientos sin intervención directa de un operador.

Los métodos de navegación se pueden clasificar en dos grandes grupos: los que dependen de sistemas de posicionamiento global (GNSS, como el GPS, véase Figura 2.1) y los que operan sin ellos, como ocurre en entornos interiores.

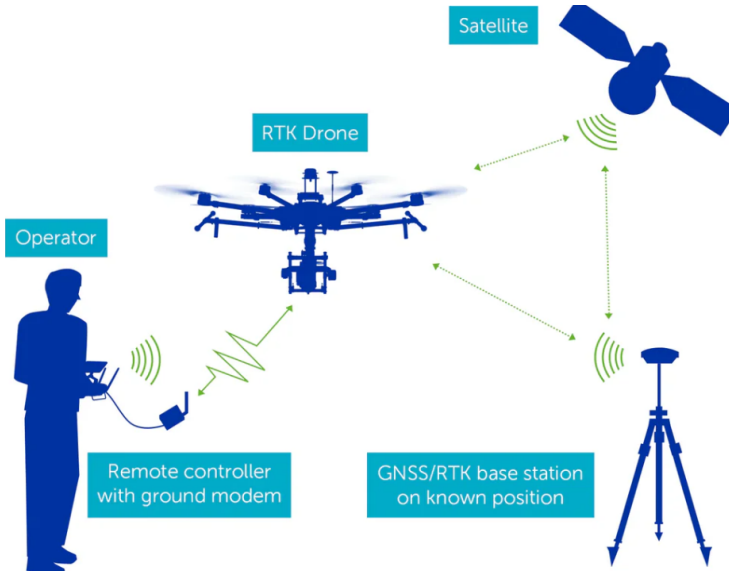

Figura 2.1: Sistema de posicionamiento del dron (Fuente: RCDrone.top)

Este proyecto, está orientado a la navegación en interiores, donde se requiere el uso de sensores inerciales (IMU), estimadores de posición, y algoritmos de control capaces de mantener la estabilidad y seguir trayectorias sin referencias externas.

En trabajos anteriores se han utilizado sensores alternativos al GPS, como el flujo óptico (véase Figura 2.2), para permitir la navegación en interiores mediante estimación visual del movimiento [5]. Concretamente, en entornos interiores lineales como lo puede ser un pasillo, el flujo óptico se ha utilizado con éxito para evitar colisiones y estimar la profundidad del mismo [6].

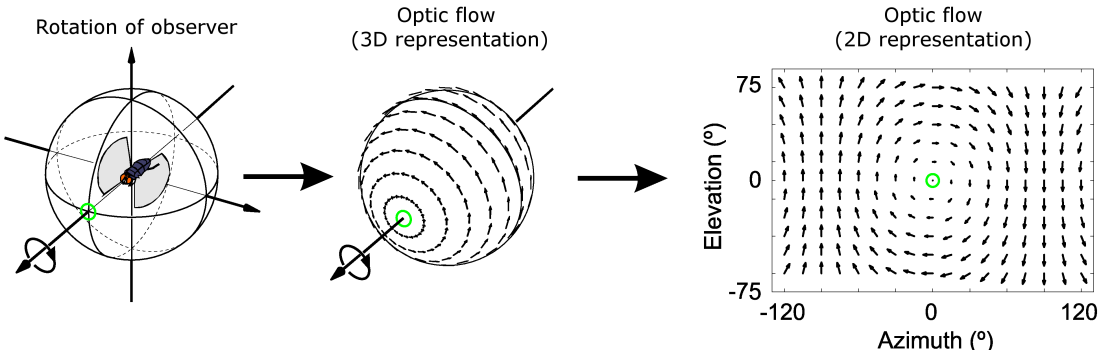

Figura 2.2: Flujo óptico en rotación: modelo 3D y 2D (Fuente: Wikipedia)

Para lograr este objetivo, es necesario un sistema de control robusto que procese en tiempo real la información procedente de los sensores y genere las señales de control adecuadas. El uso de modelos dinámicos precisos y la integración de algoritmos como el LQR o estimadores de estado, como filtros de Kalman, son esenciales para mantener la actitud y controlar la posición del UAV de forma autónoma.

Cabe destacar que el desarrollo de sistemas de navegación autónoma en interiores ha sido abordado desde distintas perspectivas en varios TFGs previos, como el desarrollo de sistemas de navegación autónoma para un UAV por parte de Javier García Aguilar [7] o el desarrollo de control para navegación autónoma de un cuadricóptero en interiores de Jorge Benasar Vázquez [8].

2.2. Aplicaciones

Los UAVs han demostrado su utilidad en una amplia variedad de campos gracias a su versatilidad, bajo coste relativo y facilidad para acceder a zonas complejas o peligrosas. Algunas de las aplicaciones más relevantes incluyen:

- **Inspección y mantenimiento**: Los UAVs se utilizan para inspeccionar visualmente infraestructuras críticas, detectando problemas en líneas eléctricas (véase Figura 2.3), puentes y edificios de forma más segura y eficiente que los métodos tradicionales [9].

Figura 2.3: Aplicación de drones para inspeccionar líneas eléctricas de alta tensión (Fuente: Telefónica Tech)

- **Logística y reparto**: Transporte de paquetes ligeros en zonas urbanas o de difícil acceso (véase Figura 2.4). En logística y gestión de almacenes, los UAVs han destacado para tareas como inventario, inspección y reparto ligero, aunque se han identificado retos críticos en hardware, software y sensorización [10].

Figura 2.4: Dron para el transporte de paquetes ligeros (Fuente: EU Drone Port)

- **Agricultura de precisión**: En agricultura de precisión, los drones permiten monitorizar cultivos (véase Figura 2.5), detectar enfermedades y optimizar el riego mediante imágenes multiespectrales y análisis automatizado [11].

Figura 2.5: Drones en agricultura de precisión para el riego (Fuente: Toll)

- **Vigilancia y seguridad**: En vigilancia y rescate, los UAVs integran sensores térmicos, cámaras y capacidades autónomas para patrullaje y operaciones de búsqueda en zonas inaccesibles [12].

Figura 2.6: Dron multirrotor con cámara térmica y óptica, empleado en aplicaciones de vigilancia y búsqueda

- **Cartografía y topografía**: Los UAVs equipados con fotogrametría y LiDAR permiten generar modelos 3D precisos (ver Figura 2.7), ideales para cartografía y levantamientos topográficos de alta resolución [13].

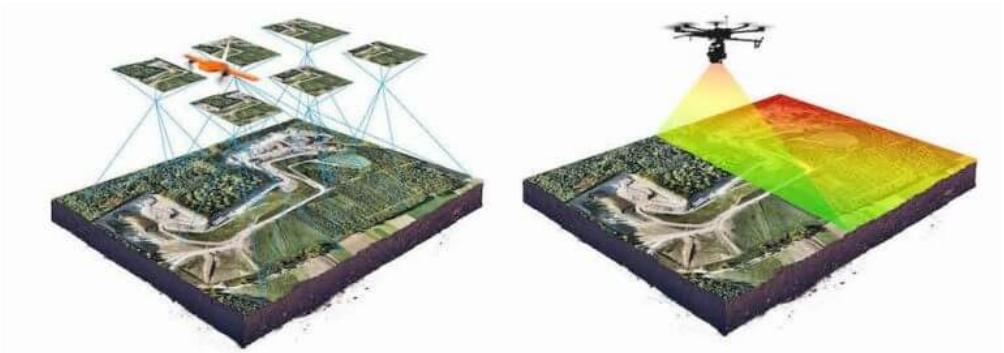

Figura 2.7: Drones en tareas de cartografía y topografía mediante fotogrametría y sensores LiDAR (Fuente Wingtra)

- **Investigación científica**: En investigación científica, los UAVs se utilizan para monitoreo ambiental, muestreo de calidad del aire y seguimiento de fauna en áreas remotas [14].

En los últimos años, también han emergido aplicaciones en entornos cerrados o estructurados, como almacenes automatizados, hospitales o inspección de túneles, donde el uso de UAVs autónomos en interiores abre nuevas posibilidades para automatizar procesos que antes requerían presencia humana directa.

3. Modelo de la Planta

3.1. Modelo Dinámico

En función de la configuración física del dron (número, disposición y orientación de los motores y hélices) se obtienen distintos modelos de la planta. Estas configuraciones afectan directamente a las ecuaciones de movimiento, al sistema de control necesario y a la respuesta del dron ante perturbaciones externas [15]. Las configuraciones más comunes son:

- **Configuración en +**: Disposición en cruz, donde los cuatro motores están alineados con los ejes del dron (adelante, atrás, izquierda y derecha). Es una configuración simétrica y simple de modelar, muy utilizada sobretodo en aplicaciones educativas y de investigación.
- **Configuración en X**: Variante de la configuración en cruz donde los motores están colocados en los vértices de una X, es decir, formando un ángulo de 45° respecto a los ejes del cuerpo. Esta configuración es más común en vuelos reales, ya que proporciona una mejor distribución del empuje y permite una mayor agilidad en maniobras.

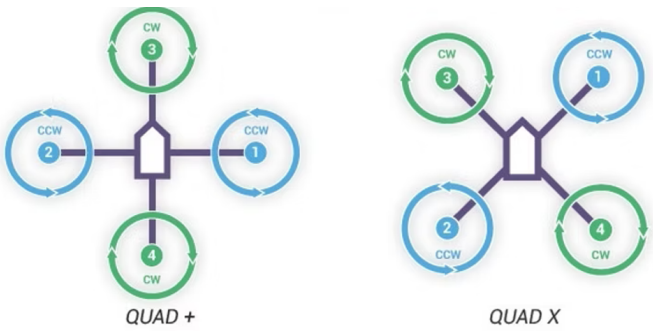

Figura 3.1: Configuración en + y en X de un cuadricóptero (Fuente: Wix)

- **Configuración tricóptero**: Compuesta por tres motores y una cola con un mecanismo de inclinación (servo) en uno de los motores traseros para controlar el eje de guiñada (*yaw*) (véase Figura 3.2). Esta configuración reduce el número de motores necesarios, aunque requiere un control más complejo por la inclusión del servo.

Figura 3.2: Ejemplo de dron con configuración tricóptero (Fuente: Xataca)

- **Configuración helicóptero coaxial**: Utiliza dos rotores principales montados en el mismo eje pero girando en sentidos opuestos, lo que elimina la necesidad de un rotor de cola (véase Figura 3.3). Es una configuración más compacta, habitual en drones de despegue y aterrizaje vertical (VTOL), y requiere un modelo dinámico distinto al de los cuadricópteros debido a su *aerodinámica particular*.

Figura 3.3: Ejemplo de helicóptero coaxial (Fuente: Amazon)

En este proyecto se van a profundizar en las configuraciones más comunes; en '+'y en 'X'. Algunas de las configuraciones clásicas, como la coaxial o tricóptero, están descritas en manuales de mecánica de vuelo II [16].

Cada una de estas configuraciones implica un modelado diferente de la planta y, por tanto, distintos requisitos en cuanto al diseño del controlador y la estrategia de navegación. La planta se compone de cuatro integradores conectados en serie y las salidas de esos integradores representan, en orden, la velocidad angular (*rate*), el ángulo, la velocidad lineal y la posición [17].

Las variables de control son las fuerzas que generan los pares de cabeceo (*pitch*), alabeo (*roll*) y guiñada (*yaw*), así como la fuerza total de empuje.

3.2. Modelo de Actitud

En aeronáutica y robótica, la actitud se refiere a la orientación del vehículo en el espacio, es decir, cómo está rotado respecto a un sistema de referencia (normalmente el terrestre o inercial). Esto incluye los ángulos de roll (alabeo), pitch (cabeceo) y yaw (guiñada).

Para diseñar el control, en primer lugar, se debe obtener el sistema de la planta en tiempo continuo, cuya representación de estado en su versión más general es:

- Ecuaciones de estado:

$$\frac{dx_i}{dt} = f_i(x_1, \ldots, x_n, u_1, u_2, \ldots, u_p, t) = f_i(X, U, t) \quad \forall i = 1, 2, \ldots, n \tag{3.1}$$

- Ecuaciones de salida:

$$y_j = g_j(x_1, \ldots, x_n, u_1, u_2, \ldots, u_p, t) = g_j(X, U, t) \quad \forall j = 1, 2, \ldots, p \tag{3.2}$$

Siendo:

- Vector de Estado:
$$X(t) = [x_1\ x_2\ \ldots\ x_n]^t$$
- Vector de Entrada:
$$U(t) = [u_1\ u_2\ \ldots\ u_m]^t$$
- Vector de Salida:
$$Y(t) = [y_1\ y_2\ \ldots\ y_p]^t$$

En el caso del dron, y definiendo los ángulos de Euler como:

$$\phi \equiv \text{Ángulo de Guiñada (Precesión o } \textit{Yaw}\text{)}$$
$$\theta \equiv \text{Ángulo de Cabeceo (Nutación o } \textit{Pitch}\text{)}$$
$$\psi \equiv \text{Ángulo de Alabeo (Spin o } \textit{Roll}\text{)}$$

Obtenemos la siguiente representación de estado:

- Vector de Estado (en el sistema de referencia del cuerpo):

$$X(t) = [\phi \quad \theta \quad \psi \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T \tag{3.3}$$

- Vector de Entrada:
$$U(t) = [u_1 \quad u_2 \quad u_3 \quad u_4]^T \tag{3.4}$$
- Vector de Salida:
$$Y(t) = [\phi \quad \theta \quad \psi]^T \tag{3.5}$$

Donde:

- u_1: empuje total (para el control de altura),
- u_2: momento de *roll* (alabeo),
- u_3: momento de *pitch* (cabeceo),
- u_4: momento de *yaw* (guiñada).

Además, se define el empuje base como:

$$T_{\text{base}} = m \cdot g \tag{3.6}$$

Donde:

- m es la masa total del dron,
- g es la aceleración de la gravedad (en Madrid, $g \approx 9{,}81\,\text{m/s}^2$).

Este valor representa el empuje necesario para mantener el dron en vuelo estacionario (hover), es decir, para contrarrestar exactamente su peso. Definir T_{base} permite normalizar las entradas de control respecto a este valor de referencia, facilitando el diseño e interpretación del sistema de control.

Por ejemplo, cuando $T_{\text{base}} = m \cdot g$, el empuje total generado iguala al peso del dron y, por tanto, este se mantiene a una altura constante.

3.2.1. Relaciones entre empuje, par de reacción y velocidad angular del motor

En el modelado dinámico de un cuadricóptero, es fundamental establecer las relaciones físicas entre las variables que intervienen en la generación de fuerza y momento por parte de los motores. En particular, para cada motor i-ésimo, se consideran las siguientes magnitudes:

- T_i: Empuje vertical generado por el motor i [N]
- τ_i: Par de reacción sobre el eje z del cuerpo (guiñada) [Nm]

- ω_i: Velocidad angular de giro de la hélice asociada al motor i [rad/s]

Estas magnitudes están relacionadas entre sí mediante los siguientes modelos simplificados:

1. **Relación entre empuje y velocidad angular.** El empuje generado por una hélice es proporcional al cuadrado de su velocidad angular:

$$T_i = k_T \cdot \omega_i^2 \quad i = 1, 2, 3, 4 \tag{3.7}$$

   Donde k_T es una constante que depende de las características de la hélice (geometría, perfil aerodinámico, densidad del aire, etc.).

2. **Relación entre par de reacción y velocidad angular.** El par de reacción τ_i inducido por la hélice sobre el cuerpo del dron también es proporcional al cuadrado de su velocidad angular:

$$\tau_i = k_Q \cdot \omega_i^2 \quad i = 1, 2, 3, 4 \tag{3.8}$$

   Donde k_τ es la constante de par, igualmente dependiente de las propiedades físicas de la hélice. El signo de τ_i depende del sentido de giro del motor: positivo para sentido antihorario y negativo para horario, teniendo en cuenta que el eje z es positivo hacia abajo.

3. **Relación entre par de reacción y empuje.** Combinando las dos relaciones anteriores, se puede expresar el par como función directa del empuje:

$$\tau_i = \frac{k_Q}{k_T} \cdot T_i = K_\tau \cdot T_i \quad i = 1, 2, 3, 4 \tag{3.9}$$

   Esta relación permite modelar el momento de guiñada en función del empuje, sin necesidad de medir directamente la velocidad angular de cada hélice, lo cual resulta útil en controladores simplificados o en simulaciones.

En realidad, las constantes k_T, k_Q son distintas para cada motor. No obstante, si se asume que:

1. Todos los motores y hélices son iguales, garantizando que las características geométricas y aerodinámicas (diámetro, paso, perfil, etc.) sean idénticas
2. Los controladores electrónicos (ESCs) aplican el mismo control y están bien calibrados
3. Las condiciones de montaje (altura, orientación, etc.) son simétricas, es decir, los motores están colocados a la misma distancia del centro de masa, en el mismo plano, sin inclinaciones ni desalineaciones

4. Las variaciones de fabricación son despreciables, es decir, las tolerancias de fabricación no afectan significativamente al modelo dinámico

Entonces:

$$k_{T_1} = k_{T_2} = k_{T_3} = k_{T_4} = k_T \tag{3.10}$$

$$k_{Q_1} = k_{Q_2} = k_{Q_3} = k_{Q_4} = k_Q \tag{3.11}$$

Y por tanto:

$$K_{\tau_1} = K_{\tau_2} = K_{\tau_3} = K_{\tau_4} = K_\tau \tag{3.12}$$

Aunque en la práctica sí puede haber pequeñas diferencias debidas a variaciones de fabricación entre motores o hélices, diferencias debidas al desgaste, desalineaciones mecánicas o estructurales y/o condiciones ambientales (flujo de aire, turbulencias, etc.), estas diferencias son generalmente pequeñas y se compensan con el controlador.

3.3. Variables de Control en Diferentes Configuraciones

En esta sección se analizan las variables de control empleadas en distintas configuraciones geométricas del cuadricóptero. Cada configuración presenta características particulares que afectan la forma en que se relacionan los comandos de entrada con los movimientos del dron.

3.3.1. Cuadricóptero en configuración "+"

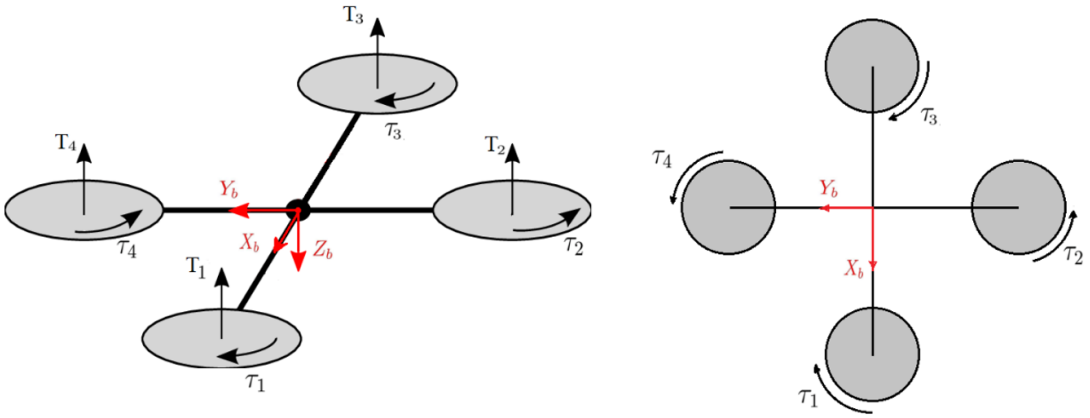

Figura 3.4: Cuadricóptero en configuración "+"

- **Empuje (Thrust):**

$$u_1 = \frac{T_1 + T_2 + T_3 + T_4}{T_{\text{base}}} \tag{3.13}$$

- **Alabeo (Roll):**

$$u_2 = \frac{T_2 - T_4}{T_{\text{base}}} \tag{3.14}$$

- **Cabeceo (Pitch):**

$$u_3 = \frac{T_1 - T_3 \cdot \frac{L_{\text{pitch,r}}}{L_{\text{pitch,f}}}}{T_{\text{base}}} = \{L_{\text{pitch,r}} = L_{\text{pitch,f}}\} = \frac{T_1 - T_3}{T_{\text{base}}} \tag{3.15}$$

- **Guiñada (Yaw):**

$$u_4 = \frac{T_1 - T_2 + T_3 - T_4}{T_{\text{base}}} \tag{3.16}$$

En forma matricial y definiendo los empujes como $T_i^{pu} = \frac{T_i}{T_{\text{base}}}$, se puede expresar el vector de entradas de control como:

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = T_+ \cdot \begin{bmatrix} T_1^{pu} \\ T_2^{pu} \\ T_3^{pu} \\ T_4^{pu} \end{bmatrix} \tag{3.17}$$

Donde:

$$T_+ = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -L_{\text{pitch},r}/L_{\text{pitch},f} & 0 \\ 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

3.3.2. Cuadricóptero en configuración "X"

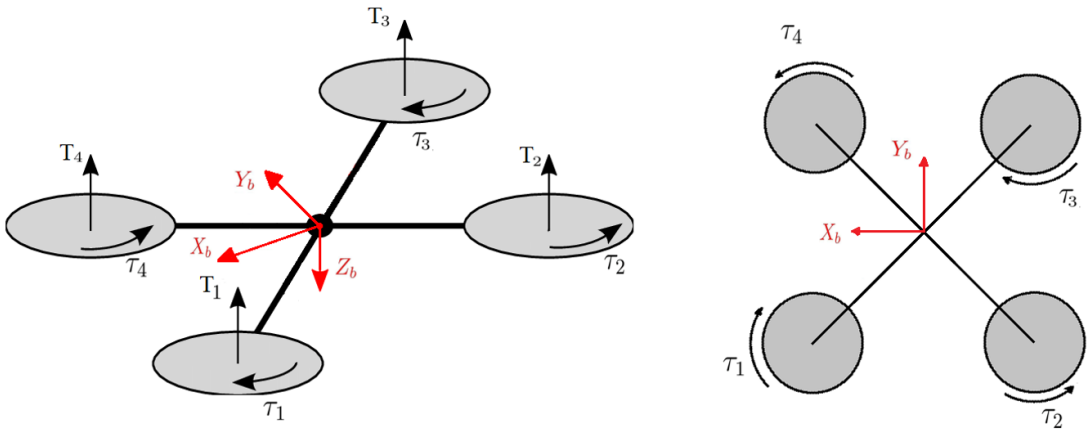

Figura 3.5: Cuadricóptero en configuración "X"

Empuje (Thrust):

$$u_1 = \frac{T_1 + T_2 + T_3 + T_4}{T_{\text{base}}} \tag{3.18}$$

Alabeo (Roll):

$$u_2 = \frac{T_1 + T_2 - T_3 - T_4}{T_{\text{base}}} \tag{3.19}$$

Cabeceo (Pitch):

$$u_3 = \frac{T_1 - T_2 \cdot \frac{L_{\text{pitch},r}}{L_{\text{pitch},f}} - T_3 \cdot \frac{L_{\text{pitch},r}}{L_{\text{pitch},f}} + T_4}{T_{\text{base}}} = \frac{T_1 - T_2 - T_3 + T_4}{T_{\text{base}}} \tag{3.20}$$

Guiñada (Yaw):

$$u_4 = \frac{T_1 - T_2 + T_3 - T_4}{T_{\text{base}}} \tag{3.21}$$

En forma matricial y definiendo los empujes como $T_i^{pu} = \frac{T_i}{T_{\text{base}}}$, se puede expresar el vector de entradas de control como:

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = T_X \cdot \begin{bmatrix} T_1^{pu} \\ T_2^{pu} \\ T_3^{pu} \\ T_4^{pu} \end{bmatrix} \tag{3.22}$$

Donde:

$$T_X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\frac{L_{\text{pitch},r}}{L_{\text{pitch},f}} & -1 & -1 \\ 1 & -\frac{L_{\text{pitch},r}}{L_{\text{pitch},f}} & 1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

3.4. Representación de estado - Modelo basado en Ángulos de Euler

Para el diseño del control, en primer lugar, se debe de obtener el sistema de la planta en tiempo continuo, cuya representación de estado en su versión más general es:

$$\dot{X}(t) = AX(t) + BU(t) \tag{3.23}$$

$$Y(t) = CX(t) + DU(t) \tag{3.24}$$

Siendo:

- Vector de Estado: $X(t)$
- Vector de Entradas: $U(t)$
- Vector de Salidas: $Y(t)$

En nuestro caso:

$$X(t) = \begin{bmatrix} \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^t$$

$$U(t) = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = T \cdot \begin{bmatrix} T_1^{pu} \\ T_2^{pu} \\ T_3^{pu} \\ T_4^{pu} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} T_1^{pu} \\ T_2^{pu} \\ T_3^{pu} \\ T_4^{pu} \end{bmatrix}$$

$$Y(t) = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^t$$

No obstante, aunque el sistema dinámico del dron se puede expresar mediante las ecuaciones de estado convencionales, en este trabajo se ha desarrollado una expresión invertida del sistema, de forma que se calcula las entradas de la planta en función de las variables de estado. Esta inversión es esencial para el control LQR ya que permite incorporar los términos de compensación no lineal y desacoplamiento.

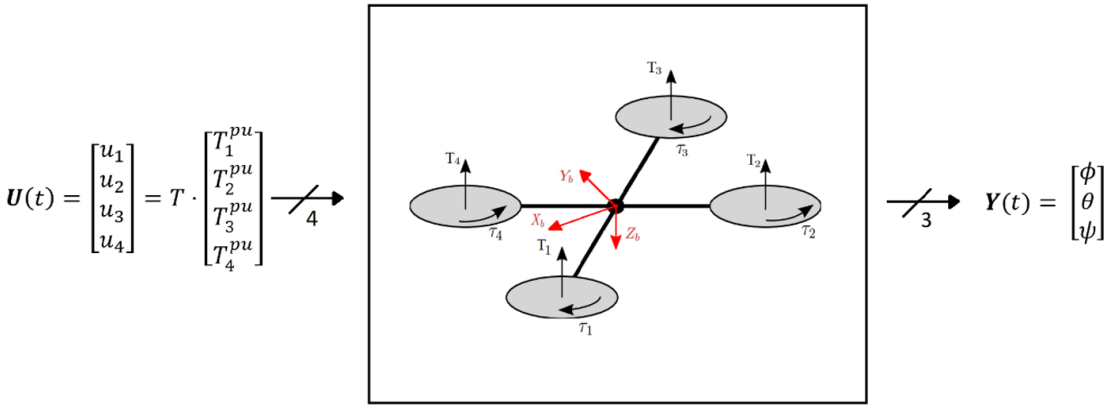

Figura 3.6: Modelo de la planta. Entradas y Salidas

3.4.1. Ecuación del movimiento rotacional del dron

En el caso del dron, y ya particularizando para la configuración en X, para obtener las ecuaciones de estado, se parte de la ecuación sin efectos giróscopos de los motores:

$$I \cdot \dot{\omega}_{xyz} = L \cdot U - \omega_{xyz} \times (I \cdot \omega_{xyz}) \tag{3.25}$$

Donde:

- $\boldsymbol{I}$: Matriz de inercia del dron respecto a su centro de masas, expresada en el sistema de coordenadas del cuerpo. Representa cómo se distribuye la masa del dron en torno a sus ejes rotacionales (roll, pitch, yaw). Además, en configuraciones simétricas, los productos de inercia (elementos fuera de la diagonal) suelen despreciarse, considerando I diagonal.

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

- $\boldsymbol{\omega}_{\text{xyz}}$: Vector de velocidades angulares del dron en el sistema del cuerpo (en los ejes X_b, Y_b, Z_b).

$$\omega_{xyz} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^t$$

- $\dot{\boldsymbol{\omega}}$: Vector de derivadas temporales de las velocidades angulares del dron. Es decir, la aceleración angular en los ejes X_b, Y_b, Z_b.

$$\dot{\omega}_{xyz} = \begin{bmatrix} \dot{\omega}_x & \dot{\omega}_y & \dot{\omega}_z \end{bmatrix}^t$$

- $\boldsymbol{L}$: Matriz de mapeo que transforma los empujes normalizados de cada motor en momentos respecto a los tres ejes del cuerpo (roll, pitch y yaw).

$$L = \begin{bmatrix} 0 & T_{\text{base}} \cdot L_{\text{roll}} & 0 & 0 \\ 0 & 0 & T_{\text{base}} \cdot L_{\text{pitch},f} & 0 \\ 0 & 0 & 0 & T_{\text{base}} \end{bmatrix}$$

Para relacionar las velocidades del cuerpo con los ángulos de Euler, se debe obtener una matriz de transformación de forma que:

$$\omega_{xyz} = M_{\text{Euler}} \cdot \dot{\theta}_{\text{Euler}} \tag{3.26}$$

Dicha matriz se construye a partir de aplicar las tres rotaciones en el siguiente orden:

1. Yaw: rotación de ángulo ψ sobre el eje Z inercial
2. Pitch: rotación de ángulo θ sobre el eje Y intermedio
3. Roll: rotación de ángulo ϕ sobre el eje X del cuerpo

Notación:

$i,\ j,\ k$	ejes del sistema inercial inicial.
$i_1,\ j_1,\ k_1$	sistema tras la primera rotación (yaw, ángulo ϕ).
$i_2,\ j_2,\ k_2$	sistema tras la segunda rotación (pitch, ángulo θ).
$i_b,\ j_b,\ k_b$	ejes finales del cuerpo, tras las tres rotaciones (roll, ángulo ψ).
e_ϕ	dirección del eje de rotación de yaw (Z inercial, k).
e_θ	dirección del eje de rotación de pitch (Y del sistema intermedio, j_1).
e_ψ	dirección del eje de rotación de roll (X del cuerpo, i_b).

Paso 1: Rotación de ángulo ϕ sobre el eje Z inercial

La primera rotación es una rotación de ángulo ϕ (yaw) sobre el eje Z del sistema inercial:

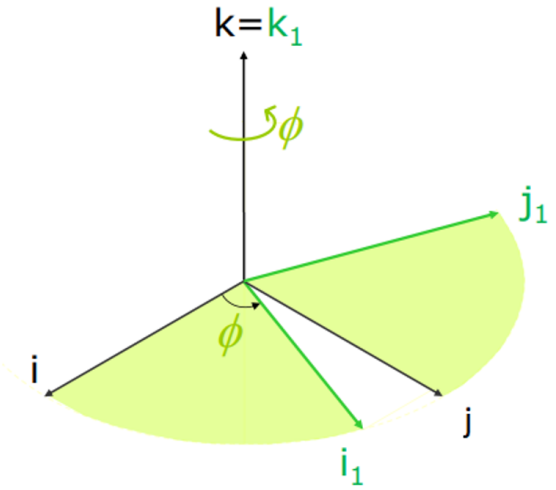

Figura 3.7: Rotación de ángulo ψ sobre el eje Z inercial

$$\begin{bmatrix} \vec{i}_1 \\ \vec{j}_1 \\ \vec{k}_1 \end{bmatrix} = R_z(\phi) \cdot \begin{bmatrix} \vec{i} \\ \vec{j} \\ \vec{k} \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{i} \\ \vec{j} \\ \vec{k} \end{bmatrix} \tag{3.27}$$

Paso 2: Rotación de ángulo θ sobre el eje Y intermedio

La segunda rotación es una rotación de ángulo θ (pitch) sobre el eje Y del sistema intermedio:

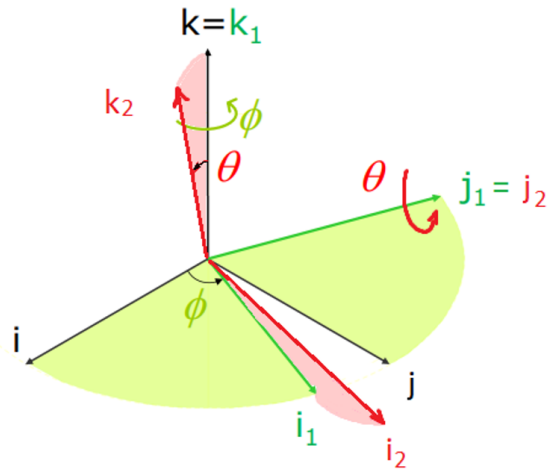

Figura 3.8: Rotación de ángulo θ sobre el eje Y intermedio

$$\begin{bmatrix} \vec{i}_2 \\ \vec{j}_2 \\ \vec{k}_2 \end{bmatrix} = R_y(\theta) \cdot \begin{bmatrix} \vec{i}_1 \\ \vec{j}_1 \\ \vec{k}_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \vec{i}_1 \\ \vec{j}_1 \\ \vec{k}_1 \end{bmatrix} \tag{3.28}$$

Paso 3: Rotación de ángulo ψ sobre el eje X del cuerpo

La tercera rotación es una rotación de ángulo ψ (roll) sobre el eje X del cuerpo:

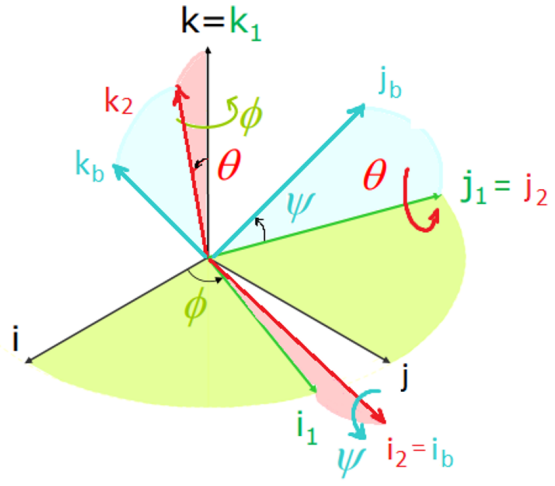

Figura 3.9: Rotación de ángulo ϕ sobre el eje X del cuerpo

$$\begin{bmatrix} \vec{i}_b \\ \vec{j}_b \\ \vec{k}_b \end{bmatrix} = R_x(\psi) \cdot \begin{bmatrix} \vec{i}_2 \\ \vec{j}_2 \\ \vec{k}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \cdot \begin{bmatrix} \vec{i}_2 \\ \vec{j}_2 \\ \vec{k}_2 \end{bmatrix} \tag{3.29}$$

Relación entre vectores de Euler y vectores de la base del cuerpo

El último paso es relacionar los vectores de Euler con los vectores de la base del cuerpo:

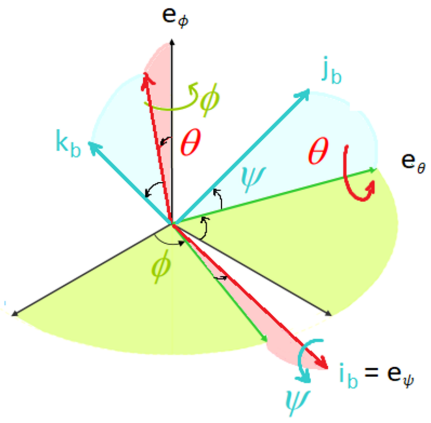

Figura 3.10: Relación entre los vectores de Euler y la base del cuerpo

$$\begin{bmatrix} \vec{e}_\phi \\ \vec{e}_\theta \\ \vec{e}_\psi \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \sin(\psi)\cos(\theta) & \cos(\psi)\cos(\theta) \\ 0 & \cos(\psi) & -\sin(\psi) \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{i}_b \\ \vec{j}_b \\ \vec{k}_b \end{bmatrix} \tag{3.30}$$

Es decir:

$$\vec{e}_\phi^{(b)} = \begin{bmatrix} -\sin(\theta) \\ \sin(\psi)\cdot\cos(\theta) \\ \cos(\psi)\cdot\cos(\theta) \end{bmatrix}, \quad \vec{e}_\theta^{(b)} = \begin{bmatrix} 0 \\ \cos(\psi) \\ -\sin(\psi) \end{bmatrix}, \quad \vec{e}_\psi^{(b)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.31}$$

Finalmente, reordenando y colocando estos vectores en columnas, obtenemos la matriz de Euler:

$$M_{\text{Euler}} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\psi) & \sin(\psi) \cdot \cos(\theta) \\ 0 & -\sin(\psi) & \cos(\psi) \cdot \cos(\theta) \end{bmatrix} \tag{3.32}$$

Otra forma de obtener la matriz es directamente desde la velocidad angular del cuerpo:

$$\omega_{xyz} = \omega_\phi + \omega_\theta + \omega_\psi = \dot{\phi} \cdot \vec{k} + \dot{\theta} \cdot \vec{j}_1 + \dot{\psi} \cdot \vec{i}_b \tag{3.33}$$

Sumando las tres componentes en la base del cuerpo y agrupando:

$$\begin{aligned} \omega_{xyz} = & \left[\dot{\psi} - \dot{\phi} \cdot \sin(\theta)\right] \cdot \vec{i}_b \\ & + \left[\dot{\theta} \cdot \cos(\psi) + \dot{\phi} \cdot \sin(\psi)\cos(\theta)\right] \cdot \vec{j}_b \\ & + \left[\dot{\theta} \cdot \sin(\psi) + \dot{\phi} \cdot \cos(\psi)\cos(\theta)\right] \cdot \vec{k}_b \end{aligned} \tag{3.34}$$

Finalmente, colocando los coeficientes de forma matricial:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\psi) & \sin(\psi)\cos(\theta) \\ 0 & -\sin(\psi) & \cos(\psi)\cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} \tag{3.35}$$

Es decir, se llega a la misma expresión matricial que la obtenida anteriormente en la ecuación (3.32).

Una vez obtenida la relación, podemos derivar la ecuación respecto al tiempo para obtener la relación entre las aceleraciones angulares del cuerpo con la de las aceleraciones angulares de Euler:

$$\dot{\omega}_{xyz} = \frac{d(\omega_{xyz})}{dt} = \frac{d(M_{\text{Euler}} \cdot \dot{\theta}_{\text{Euler}})}{dt} = \dot{M}_{\text{Euler}} \cdot \dot{\theta}_{\text{Euler}} + M_{\text{Euler}} \cdot \ddot{\theta}_{\text{Euler}} \tag{3.36}$$

Despejando se obtiene:

$$\ddot{\theta}_{\text{Euler}} = M_{\text{Euler}}^{-1} \cdot \left(\dot{\omega}_{xyz} - \dot{M}_{\text{Euler}} \cdot \dot{\theta}_{\text{Euler}}\right) \tag{3.37}$$

Donde:

$$M_{\text{Euler}}^{-1} = \begin{bmatrix} 1 & \sin(\psi)\tan(\theta) & \cos(\psi)\tan(\theta) \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \frac{\sin(\psi)}{\cos(\theta)} & \frac{\cos(\psi)}{\cos(\theta)} \end{bmatrix} \tag{3.38}$$

$$\dot{M}_{\text{Euler}} = \begin{bmatrix} 0 & 0 & -\cos(\theta)\cdot\dot{\theta} \\ 0 & -\sin(\psi)\cdot\dot{\psi} & \cos(\psi)\cos(\theta)\cdot\dot{\psi} - \sin(\psi)\sin(\theta)\cdot\dot{\theta} \\ 0 & -\cos(\psi)\cdot\dot{\psi} & -\sin(\psi)\cos(\theta)\cdot\dot{\psi} - \cos(\psi)\sin(\theta)\cdot\dot{\theta} \end{bmatrix} \tag{3.39}$$

Nota:
La inversa de la matriz de Euler (M_{Euler}^{-1}) es válida siempre que $\cos(\theta) \neq 0$*, es decir, siempre que* $\theta \neq \pm\pi/2$ *para evitar la singularidad de Gimbal Lock [18]. En esa situación, dos de los tres ejes de rotación se vuelven coplanarios, lo que provoca la pérdida de un grado de libertad. En este caso, el sistema ya no puede distinguir entre rotaciones de yaw y roll, ya que ambas afectan al mismo eje. No obstante, nunca nos encontramos con dicho problema, porque el ángulo de cabeceo nunca alcanza los* $\pm 90°$*.*

Este resultado se guarda en la matriz de variables de control virtuales u_p:

$$u_p = M_{\text{Euler}}^{-1} \cdot \left(\dot{\omega}_{xyz} - \dot{M}_{\text{Euler}} \cdot \dot{\theta}_{\text{Euler}} \right) \tag{3.40}$$

Sustituyendo, despejando y utilizando variable simbólica se obtiene la ecuación matricial que representa el modelo del dron:

$$u = C_1 \cdot u_p + C_2 \cdot \omega_{xyz} \times (I \cdot \omega_{xyz}) + C_3 \cdot \omega_{xyz} \tag{3.41}$$

Donde:

- $C_1 \in \mathbb{R}^{3\times 4}$

Mapea los empujes de los motores a la aceleración angular deseada (control efectivo).

$$C_1(t) = \begin{bmatrix} 0 & \dfrac{L_{\text{roll}} \cdot T_{\text{base}}}{I_{xx}} & \dfrac{L_{\text{pitch},f} \cdot T_{\text{base}} \cdot \sin(\psi) \cdot \sin(\theta)}{I_{yy}\cos(\theta)} & \dfrac{T_{\text{base}} \cdot \cos(\psi) \cdot \sin(\theta)}{I_{zz}\cos(\theta)} \\ 0 & 0 & \dfrac{L_{\text{pitch},f} \cdot T_{\text{base}} \cdot \cos(\psi)}{I_{yy}} & -\dfrac{T_{\text{base}} \cdot \sin(\psi)}{I_{zz}} \\ 0 & 0 & \dfrac{L_{\text{pitch},f} \cdot T_{\text{base}} \cdot \sin(\psi)}{I_{yy}\cos(\theta)} & \dfrac{T_{\text{base}} \cdot \cos(\psi)}{I_{zz}\cos(\theta)} \end{bmatrix}$$

- $C_2 \in \mathbb{R}^{3\times3}$

Compensa el término no lineal de acoplamiento rotacional: $(\omega_{xyz} \times (I \cdot \omega_{xyz}))$.

$$C_2(t) = \begin{bmatrix} -\dfrac{1}{I_{xx}} & -\dfrac{\sin(\psi)\sin(\theta)}{I_{yy}\cos(\theta)} & -\dfrac{\cos(\psi)\sin(\theta)}{I_{zz}\cos(\theta)} \\ 0 & -\dfrac{1}{I_{yy}} \cdot \dfrac{\cos(\psi)}{\sin(\psi)} & -\dfrac{1}{I_{zz}} \cdot \dfrac{\sin(\psi)}{\cos(\psi)} \\ 0 & -\dfrac{1}{I_{yy}} \cdot \dfrac{1}{\cos(\theta)} & -\dfrac{1}{I_{zz}} \cdot \dfrac{1}{\cos(\theta)} \end{bmatrix}$$

- $C_3 \in \mathbb{R}^{3\times3}$

Corrige el efecto del cambio de base entre los ángulos de Euler y la base rotacional $(\dot{M}_{\text{Euler}})$.

$$C_3(t) = \begin{bmatrix} 0 & \dfrac{\sin(\theta)\cdot\dot{\psi}}{\cos(\theta)} & \dfrac{\dot{\theta}}{\cos(\theta)} \\ 0 & 0 & -\cos(\theta)\cdot\dot{\psi} \\ 0 & \dfrac{\dot{\psi}}{\cos(\theta)} & \dfrac{\sin(\theta)\cdot\dot{\theta}}{\cos(\theta)} \end{bmatrix}$$

Resumen

El sistema se modela como un doble integrador idealizado para el diseño del control, donde la aceleración angular de los ángulos de Euler es la entrada del controlador:

$$u_p = \ddot{\theta}_{\text{Euler}} \tag{3.42}$$

Y para obtener los momentos reales que deben aplicarse al dron, se usa la ecuación de control no lineal basada en linealización por realimentación y desacoplo, dada por la ecuación (3.41).Esta ecuación compensa las dinámicas no lineales, acoplamientos giroscópicos y transformaciones cinemáticas del sistema. Las matrices C_1, C_2, C_3 dependen de los ángulos de Euler y se actualizan en tiempo real.

Además, existen parametrizaciones alternativas como los cuaterniones o los parámetros de Euler sin singularidades que han sido exploradas en otros trabajos [19].

3.5. Modelo linealizado, invariante en el tiempo y desacoplado

En el modelo linealizado invariante en el tiempo, las ecuaciones de la representación de estado son las siguientes:

$$\dot{X}(t) = A \cdot X(t) + B \cdot u_p(t) \tag{3.43}$$

$$Y(t) = C \cdot X(t) + D \cdot u_p(t) \tag{3.44}$$

Siendo:

- **Vector de Estado:**

$$X(t) = \begin{bmatrix} \theta_{\text{Euler}} & \dot{\theta}_{\text{Euler}} \end{bmatrix}^t = \begin{bmatrix} \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^t$$

- **Vector de Entrada:**

$$u_p(t) = \ddot{\theta}_{\text{Euler}} = \begin{bmatrix} \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \end{bmatrix}^t$$

- **Vector de Salida:**

$$Y(t) = [\phi \quad \theta \quad \psi]^t$$

En el modelo linealizado y desacoplado, el sistema de actitud se representa como un conjunto de tres dobles integradores **independientes**. El vector de estado incluye los ángulos de Euler y sus derivadas:

Como el sistema está desacoplado por diseño (gracias al control no lineal con C_1, C_2, C_3), para cada eje se define un sistema SISO (una entrada, una salida).

Modelo para Yaw

Ecuaciones de la representación de estado para Yaw:

$$\dot{x}_\psi(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_\psi(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_\psi(t) \tag{3.45}$$

$$y_\psi(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_\psi(t) \tag{3.46}$$

Donde:

- **Vector de Estado:**
$$x_\psi(t) = \begin{bmatrix} \psi & \dot{\psi} \end{bmatrix}^t$$
- **Entrada:**
$$u_\psi(t) = \ddot{\psi}$$
- **Salida:**
$$y_\psi(t) = \psi$$

Modelo para Pitch

Ecuaciones de la representación de estado para Pitch:

$$\dot{x}_\theta(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_\theta(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_\theta(t) \tag{3.47}$$

$$y_\theta(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_\theta(t) \tag{3.48}$$

Donde:

- **Vector de Estado:**
$$x_\theta(t) = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^t$$
- **Entrada:**
$$u_\theta(t) = \ddot{\theta}$$
- **Salida:**
$$y_\theta(t) = \theta$$

Modelo para Roll

Ecuaciones de la representación de estado para Roll:

$$\dot{x}_\phi(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_\phi(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_\phi(t) \tag{3.49}$$

$$y_\phi(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_\phi(t) \tag{3.50}$$

Donde:

- **Vector de Estado:**

$$x_\phi(t) = \begin{bmatrix} \phi & \dot{\phi} \end{bmatrix}^t$$

- **Entrada:**

$$u_\phi(t) = \ddot{\phi}$$

- **Salida:**

$$y_\phi(t) = \phi$$

Matrices del sistema

Podemos concluir que, para cada eje (yaw, pitch y roll), las matrices de la representación de estado son:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0 \tag{3.51}$$

3.6. Discretización exacta de una representación de estado lineal

Una vez obtenido el modelo del sistema en espacio de estados en tiempo continuo, definido por las matrices A, B, C y D, es necesario transformarlo a su forma discreta para su implementación digital.

Para ello, se emplea una discretización exacta basada en un mantenedor de orden cero (Zero-Order Hold, ZOH), uno de los métodos estándar para implementar modelos continuos en sistemas digitales, como se describe en obras clásicas de control [20]. Esta discretización asume que la entrada $u_p(t)$ se mantiene constante durante cada período de muestreo T_s.

El sistema discreto resultante tiene la forma:

$$X[k+1] = A_d \cdot X[k] + B_d \cdot u_p[k] \tag{3.52}$$

$$Y[k] = C_d \cdot X[k] + D_d \cdot u_p[k] \tag{3.53}$$

Donde:

$$A_d = e^{AT_s} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \tag{3.54}$$

$$B_d = \int_0^{T_s} e^{A\tau} B \, d\tau = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix} \tag{3.55}$$

$$C_d = C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{3.56}$$

$$D_d = D = \begin{bmatrix} 0 \end{bmatrix} \tag{3.57}$$

3.7. Código de Matlab

Este proceso puede realizarse directamente en `MATLAB` utilizando `c2d` [21]:

```
% Definicion del Periodo de Muestreo (seg)
Ts = 0.01;

% Matrices del sistema linealizado
matA = [0 1; 0 0];
matB = [0; 1];
matC = [1 0];
matD = 0;

% Funcion c2d
EULER_ANG_SS_MODEL = ss(matA, matB, matC, matD);
sys_d = c2d(EULER_ANG_SS_MODEL, Ts, 'zoh');

% Extraer Ad, Bd, Cd y Dd
matAd = sys_d.A;
matBd = sys_d.B;
matCd = sys_d.C;
matDd = sys_d.D;
```

Código 3.1: Discretización del modelo linealizado

4. Control LQR (Linear Quadratic Regulator)

Con el objetivo de obtener un control preciso y estable del dron en configuración en "X", se ha implementado un controlador óptimo del tipo LQR (Linear Quadratic Regulator). Este tipo de control es utilizado en sistemas dinámicos lineales debido a su capacidad para balancear de manera eficiente el rendimiento del sistema y el esfuerzo de control. Aunque este trabajo se centra en el control LQR, existen técnicas más avanzadas como el control predictivo por modelo (MPC), [22].

El control por realimentación de estado es la estrategia de control más apropiada, al menos en teoría, para optimizar la respuesta de un sistema multivariable (véase Figura 4.1). La característica principal de dicho control es que utiliza la información de todas las medidas disponibles para generar los mandos de todos los actuadores presentes en el sistema.

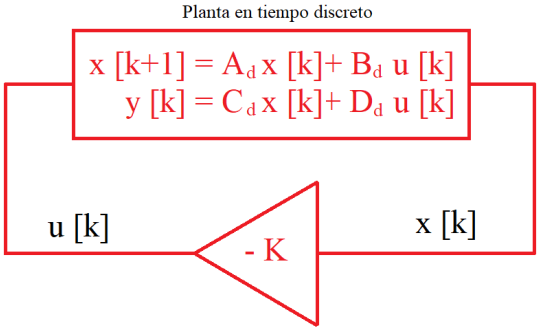

Figura 4.1: Esquema de control en tiempo discreto con realimentación de estado

El control LQR consiste en un regulador por realimentación de estado para que la respuesta libre del sistema en lazo cerrado, ante un estado inicial distinto al del punto de operación, evolucione hacia dicho punto de operación de forma óptima según una determinada función de coste, dando lugar a lo que se conoce como regulador lineal cuadrático (LQR).

A pesar de que esta memoria no tiene como objetivo desarrollar en profundidad la teoría del control, resulta necesario introducir brevemente el significado de las variables involucradas y el origen de las matrices empleadas. Esta contextualización permitirá comprender mejor el fundamento del controlador por realimentación de estados utilizado, en concreto, el regulador lineal cuadrático (LQR).

4.1. Formulación general

Supongamos que tenemos el modelo discreto de la planta, linealizado en el punto de operación:

$$\Delta X[k+1] = A_d \cdot \Delta X[k] + B_d \cdot \Delta U[k] \tag{4.1}$$

Nota:
Todas las variables son incrementales respecto al punto de operación. Para simplificar la notación, en adelante se omitirán los símbolos Δ, aunque debe entenderse que tanto $X[k]$ como $U[k]$ representan variaciones respecto a dicho punto.

Supongamos además que se parte de un estado inicial $X[0]$ diferente al del punto de operación deseado. Entonces, la función objetivo cuadrática $J(N)$ empleada en el diseño del regulador LQR, definida en un horizonte infinito, es la siguiente:

$$J = \sum_{k=0}^{\infty} X^t[k] \cdot Q \cdot X[k] + \sum_{k=0}^{\infty} U^t[k] \cdot R \cdot U[k] \tag{4.2}$$

Donde Q y R son matrices de peso definidas positivas correspondientes al error de regulación y al esfuerzo de actuación:

- **Q:** penaliza los errores del estado.
- **R:** penaliza el esfuerzo de control.

Cuanto mayor sea un valor en Q, mayor será la prioridad de minimizar el error asociado a ese estado. Por otro lado, mayores valores en R penalizarán más los movimientos agresivos en los actuadores.

Las matrices de peso permiten relativizar la importancia en la función de coste entre ambos términos y, dentro de cada término, las prioridades entre las distintas variables de estado y entre las distintas variables de control.

El diseño óptimo se obtiene resolviendo una ecuación de Riccati mediante programación dinámica, desarrollada por Bellman en la década de 1960.

4.2. LQR aplicado al cuadricóptero

El uso de un controlador LQR optimizado para el seguimiento de trayectorias ha sido ya validado en plataformas de cuadricópteros, donde se ha demostrado la efi-

cacia del control en actitud y posición [23]. En una comparación entre PID y LQR, este último demostró una robustez superior frente a perturbaciones [24].

Recordemos que, en el caso del cuadricóptero, el sistema discreto para cada eje de actitud (roll, pitch, yaw) queda definido por las siguientes ecuaciones de estado:

$$x_{\theta_i}[k+1] = A_d \cdot x_{\theta_i}[k] + B_d \cdot u_{p,\theta_i}[k] \quad i \in \{\text{roll}, \text{pitch}, \text{yaw}\} \tag{4.3}$$

$$y_{\theta_i}[k] = C_d \cdot x_{\theta_i}[k] + D_d \cdot u_{p,\theta_i}[k] \quad i \in \{\text{roll}, \text{pitch}, \text{yaw}\} \tag{4.4}$$

Donde: θ_i representa uno de los tres ángulos de Euler, y:

- **Vector de Estado:**
$$x_{\theta_i} = \begin{bmatrix} \theta_i & \dot{\theta}_i \end{bmatrix}^t$$
- **Vector de Entrada:**
$$u_{p,\theta_i} = \begin{bmatrix} \ddot{\theta}_i \end{bmatrix}$$
- **Vector de Salida:**
$$y_{\theta_i} = \begin{bmatrix} \theta_i \end{bmatrix}$$

Y las matrices discretas del sistema son las obtenidas previamente en la Sección 3.6.

$$A_d = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}, \quad B_d = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix}, \quad C_d = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D_d = \begin{bmatrix} 0 \end{bmatrix} \tag{4.5}$$

Es decir, lo único que nos quedaría es establecer cuáles son las matrices Q y R.

4.2.1. Modelo extendido con retardos e integrador – Matrices Q y R

Para el diseño del controlador LQR aplicado al sistema de actitud del cuadricóptero (roll, pitch y yaw), se ha considerado que el modelo incluye los efectos de los retardos introducidos por la dinámica de los actuadores (ESC + motor) [25].

Este pequeño cambio afecta al vector de estado para cada eje:

$$x_{\theta_i} = \begin{bmatrix} \theta_i[k] & \dot{\theta}_i[k] & d_1 & \dots & d_{n_{\text{delays}}} & e_{\text{int}}[k] \end{bmatrix}^t \tag{4.6}$$

Donde:

- $d_1, \dots, d_{n_{\text{delays}}}$ modelan los retardos en la actuación.
- $e_{\text{int}}[k]$ representa la acumulación discreta del error de seguimiento.

$$e_{\text{int}}[k] = \sum_{i=0}^{k} e[i] \cdot T_s \tag{4.7}$$

En este trabajo se ha modelado el retardo total $r_o = 20\,\text{ms}$, un período de muestreo $T_s = 10\,\text{ms}$, y por tanto, el número total de retardos se ha calculado como:

$$n_{\text{delays}} = \text{round}\left(\frac{r_o + \dfrac{T_s}{2}}{T_s}\right) = \text{round}(2{,}5) = 3 \tag{4.8}$$

Modelado de la integración del error

Además del retardo, el modelo extendido también incluye una acción integral mediante un estado adicional e_{int}, que acumula el error de seguimiento con respecto a la referencia deseada. Esta inclusión permite eliminar el error en régimen permanente y mejora el seguimiento de referencias constantes.

En un sistema continuo, la acción integral se define como:

$$e_{\text{int}}(t) = \int_0^t (r(t) - y(t))\ dt \tag{4.9}$$

Donde $r(t)$ es la referencia deseada y $y(t)$ la salida del sistema.

Sin embargo, en un sistema discreto como el que se utiliza en este trabajo, la integral se aproxima mediante una suma acumulativa ponderada por el período de muestreo T_s:

$$e_{\text{int}}[k] = \sum_{i=0}^{k} (r[i] - y[i]) \cdot T_s \tag{4.10}$$

Dado que en la mayoría de los controladores considerados en esta memoria la referencia se mantiene constante y se asume igual a cero durante la fase de estabilización, esta acumulación se implementa simplemente como:

$$e_{\text{int}}[k+1] = e_{\text{int}}[k] + T_s \cdot (-\theta[k]) \tag{4.11}$$

Esta dinámica se refleja directamente en la matriz A_d, donde el estado e_{int} se realimenta con el valor actual de $\theta[k]$. Esta estructura permite que el controlador LQR tenga en cuenta no solo el error instantáneo, sino también su acumulación en el tiempo.

Definición de las matrices Q y R

Las matrices de peso del regulador LQR se han definido específicamente para cada controlador de actitud. Estas matrices son cuadradas y de tamaño igual al número de estados del modelo extendido (es decir, $2 + n_{\text{delays}} + 1$):

- **Roll y Pitch:**

$$Q_{\text{roll/pitch}} = \begin{bmatrix} 0 & & & & & 0 \\ & 0 & & & & 0 \\ & & \ddots & & & \vdots \\ & & & 0 & & 0 \\ & & & & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n_{\text{delays}}+3)\times(n_{\text{delays}}+3)} \tag{4.12}$$

$$R_{\text{roll/pitch}} = 5 \cdot 10^{-7} \tag{4.13}$$

- **Yaw:**

$$Q_{\text{yaw}} = \begin{bmatrix} 0 & & & & & 0 \\ & 0 & & & & 0 \\ & & \ddots & & & \vdots \\ & & & 0 & & 0 \\ & & & & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n_{\text{delays}}+3)\times(n_{\text{delays}}+3)} \tag{4.14}$$

$$R_{\text{yaw}} = 1 \cdot 10^{-3} \tag{4.15}$$

La matriz Q da mayor peso a la integración del error, buscando eliminar el error en régimen permanente. Y la matriz R se ha mantenido en valores bajos para no

restringir excesivamente las acciones de control, permitiendo maniobras dinámicas sin saturación de los actuadores.

Al extender el vector de estado para incluir los retardos en la actuación y un integrador del error, las matrices de la representación en espacio de estados también se modifican.

En el caso particular de $n_{\text{delays}} = 3$, el vector de estado queda definido como:

$$x[k] = \begin{bmatrix} \theta[k] & \dot{\theta}[k] & d_1[k] & d_2[k] & d_3[k] & e_{\text{int}}[k] \end{bmatrix}^t$$

Las matrices discretas resultantes del sistema extendido son:

- Matriz de estados $A_d \in \mathbb{R}^{6\times 6}$:

$$A_d = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & T_s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ T_s & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matriz de entrada $B_d \in \mathbb{R}^{6\times 1}$:

$$B_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- Matriz de salida $C_d \in \mathbb{R}^{1\times 6}$:

$$C_d = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(salida = ángulo)

- Matriz de transmisión directa $D_d \in \mathbb{R}^{1\times 1}$:

$$D_d = 0$$

Justificación de la estructura del modelo extendido con retardos

En el modelo extendido utilizado para el diseño del controlador LQR, los retardos introducidos por los actuadores (ESC + motor) se modelan explícitamente mediante una cadena de variables de estado que simulan un retardo puro de varios pasos.

La entrada $u[k]$ no actúa directamente sobre el sistema, sino que se introduce en el primer nodo de esta cadena (estado d_1), y posteriormente se propaga hacia adelante en los siguientes pasos de tiempo hasta llegar al sistema físico a través del último nodo $d_{n_{\text{delays}}}$.

Este comportamiento se puede visualizar en la siguiente tabla:

Tiempo	**Entrada** $u[k]$	**Retardo 1** (d_1)	**Retardo 2** (d_2)	**Retardo 3** (d_3)
$k = 0$	1.0	1.0	0.0	0.0
$k = 1$	2.0	2.0	1.0	0.0
$k = 2$	3.0	3.0	2.0	1.0
$k = 3$	4.0	4.0	3.0	2.0

Tabla 4.1: Propagación de la entrada a través de la cadena de retardos.

El valor de entrada $u[k]$ afecta directamente al primer retardo d_1, y se transmite sucesivamente hasta llegar a d_3, que actúa sobre el sistema real. En este esquema, puede observarse cómo el valor introducido en $u[k]$ tarda tres ciclos completos en llegar al estado d_3, que es el que afecta directamente a la evolución de la velocidad angular $\dot{\theta}$. Esta estructura se implementa en la matriz A_d desplazando los valores de los retardos en una cadena tipo "shift register", mientras que la matriz B_d únicamente tiene un valor distinto de cero en la posición correspondiente al primer retardo (d_1).

Este enfoque permite modelar de forma explícita y precisa el efecto del retardo total en el sistema, lo que es especialmente útil para el diseño robusto del controlador.

4.3. Código de Matlab

```
% Parametros
Ts = 0.01;
n_delays = 3;
n_states = 2 + n_delays + 1;  % theta, dot_theta, d1..dn, e_int

% Construccion de matrices Ad y Bd extendidas
Ad = eye(n_states);

% theta[k+1] = theta[k] + Ts * dot_theta[k]
Ad(1,2) = Ts;

% dot_theta[k+1] = dot_theta[k] + Ts * u[k - delay]
Ad(2,2) = 1;
Ad(2,2 + n_delays) = Ts;  % input delayed

% Desplazamiento de retardos (shift register)
Ad(3:n_delays+2,3:n_delays+2) = diag(ones(n_delays-1,1), -1);

% Integrador del error
Ad(end,end) = 1;          % mantiene su valor anterior
Ad(end,1) = Ts;           % suma del error: ref - theta = -theta

% Matriz Bd: entrada afecta al primer delay
Bd = zeros(n_states,1);
Bd(n_delays + 2) = 1;

% Matrices Q y R
Q = zeros(n_states);
Q(end,end) = 1;          % solo penaliza el error integrado
R = 5e-7;

% Diseno del controlador LQR discreto
[K,~,~] = dlqr(Ad, Bd, Q, R);
```

Código 4.1: Diseño del controlador LQR discreto. Modelo extendido Roll/Pitch

En este caso, la ganancia óptima obtenida mediante el comando `dlqr` [26] para el modelo extendido del eje *roll/pitch* es:

$$K = \begin{bmatrix} 2{,}572 \cdot 10^{2} & 2{,}488 \cdot 10^{1} & 1{,}792 \cdot 10^{-1} & 2{,}011 \cdot 10^{-1} & 2{,}243 \cdot 10^{-1} & 1{,}264 \cdot 10^{3} \end{bmatrix}$$

5. Simulink

Como ya se ha mencionado anteriormente, se ha utilizado MATLAB 2024b y su entorno Simulink para el desarrollo del sistema de control del proyecto.

Se ha empleado el fichero `UAV_CONTROL_SYSTEM`, inspirado en la estructura modular que utilizábamos en el fichero `CAR_CONTROL_SYSTEM` en las asignaturas de Regulación Automática y Control Digital.

El modelo Simulink está organizado en distintas capas o niveles jerárquicos, que agrupan los distintos subsistemas según su funcionalidad. Estas capas principales son: **Control**, **Hardware**, **Simulation** y **Monitoring**.

Esta organización jerárquica ha facilitado el desarrollo, la depuración y la validación del sistema.

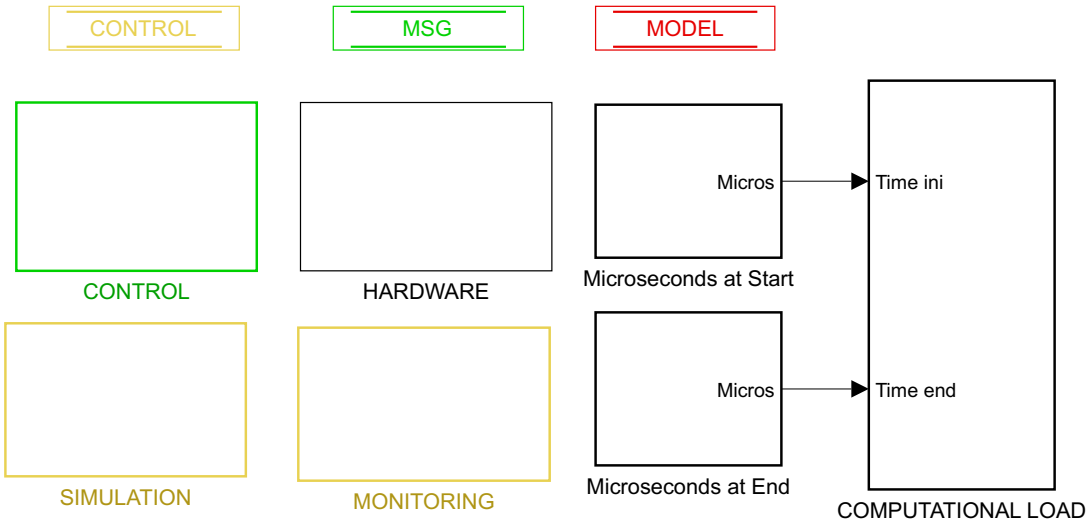

Figura 5.1: Diagrama de control del UAV.

5.1. Control

El subsistema *Control* agrupa todos los bloques relacionados con la generación de referencias, el control de actitud, el control de navegación y la planificación de misiones. Este bloque es el núcleo lógico del sistema, donde se toma la decisión de cómo debe comportarse el dron en función de su estado y de los comandos recibidos.

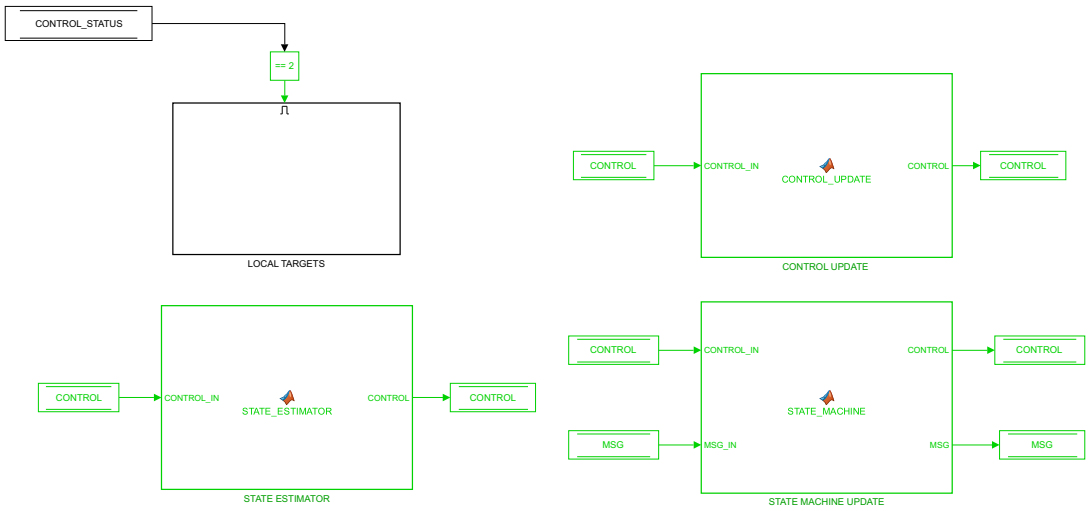

Figura 5.2: Subsistema de control

A continuación se describen los bloques más relevantes de este subsistema.

5.1.1. Local Targets

El bloque *Local Targets* se encarga de generar las referencias internas que seguirán los controladores del sistema, en función del estado actual de la misión y de las fuentes externas de comandos.

Internamente, el subsistema se divide en tres módulos principales:

a) **Attitude Targets:** genera referencias de actitud (roll, pitch y yaw) cuando el sistema se encuentra en un estado compatible (por ejemplo, armado o en vuelo) y la fuente de referencia de actitud está habilitada.

b) **Navigation Targets:** genera referencias de navegación en el plano horizontal (posición X e Y), activándose cuando el estado del sistema y la fuente de referencia de navegación indican que el control de navegación está habilitado.

c) **Mission Targets:** activa la generación de referencias de navegación en misión (waypoints) cuando el sistema entra en modo misión y la fuente de referencia correspondiente está configurada en modo autónomo.

Estos tres bloques se activan de forma condicional mediante operaciones lógicas (AND y OR) que verifican múltiples criterios. Esto permite asegurar que las referencias se generan únicamente en los modos adecuados de funcionamiento, garantizando seguridad y robustez frente a configuraciones erróneas o estados no deseados.

5.1.2. State Estimator

El subsistema *"State Estimator"* se encarga de estimar el estado completo del UAV a partir de las mediciones de sensores y del estado interno del sistema. Su función es proporcionar (en tiempo real) una estimación precisa de las variables necesarias para el control: actitud, velocidad angular, posición, velocidad y aceleración.

Los filtros complementarios, especialmente en su versión no lineal, son una buena alternativa para estimar la orientación en plataformas inerciales [27]. Este bloque se activa únicamente cuando el sistema ha superado la fase de inicialización. Su estructura se divide en tres grandes bloques funcionales:

a) **Estimación de actitud:** se realiza mediante un filtro complementario o un observador extendido de Kalman (EKF), dependiendo del modo de operación.

   i) El filtro complementario combina medidas de acelerómetros y giróscopos para estimar los ángulos de Euler (*roll, pitch, yaw*).

   ii) El EKF proporciona una estimación más precisa y robusta, especialmente útil cuando existen ruidos o perturbaciones en los sensores inerciales.

b) **Estimación de posición y navegación:** la navegación puede realizarse en dos modos:

   i) *Interior:* mediante filtrado de las posiciones y velocidades del sistema de captura de movimiento (MCS).

   ii) *Exterior:* utilizando las mediciones directas del GPS y el altímetro barométrico.

c) **Filtrado y retrasos:** se emplean filtros y retardos sobre variables como la aceleración, velocidad angular y ángulos de Euler para suavizar las señales y alimentar correctamente el EKF. Además, se calcula la matriz de rotación entre el cuerpo del vehículo y el marco de referencia inercial, necesaria para transformar correctamente magnitudes vectoriales.

En resumen, este subsistema actúa como la "percepción" del vehículo, convirtiendo datos brutos de sensores en información estructurada y útil para los controladores. Es un bloque fundamental para garantizar la estabilidad y precisión del sistema desde el punto de vista del control completo.

5.1.3. Control Update

El subsistema *"Control Update"* es el núcleo del sistema de control. Es el responsable de calcular las señales de control que serán enviadas a los actuadores del dron. Esta etapa integra algoritmos avanzados de estimación y control, incluyendo observadores de estado tipo EKF (Filtro de Kalman Extendido), técnicas de control por realimentación de estados y mecanismos de compensación de retardos computacionales.

A diferencia de otros bloques más simples, *Control Update* se apoya en varios scripts de MATLAB externos (ficheros `.m`). En concreto, emplea tres funciones externas: `EKF_NAV.m`, `EKF_ATT.m` y `SFC.m`. El bloque se organiza en torno a dos ramas principales:

- Estimadores de estado (actitud y navegación)
- Controlador con realimentación de estados (*State Feedback Control*)

El algoritmo principal evalúa las condiciones del sistema y aplica la lógica necesaria para seleccionar entre distintas configuraciones de observadores y controladores, en función del modo de operación y de las referencias activas.

Filtro de Kalman Extendido

El Filtro de Kalman Extendido (Kalman Filter, EKF) es un algoritmo de estimación recursiva que permite inferir el estado interno de un sistema dinámico a partir de medidas ruidosas y modelos matemáticos del sistema. En el contexto del control de vehículos autónomos, como un UAV, su aplicación es fundamental para obtener estimaciones precisas de variables como la posición, velocidad y orientación, las cuales no siempre pueden medirse directamente con sensores.

Los EKF se aplican habitualmente para estimación no lineal en robótica móvil [28], incluyendo UAVs y el proceso de estimación se basa en dos etapas principales: Predicción y Corrección.

- **Paso de predicción:**
  Se actualiza la estimación del estado y su incertidumbre en base al modelo dinámico del sistema y a las entradas conocidas.

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_{k-1} \tag{5.1}$$
$$P_{k|k-1} = AP_{k-1|k-1}A^t + Q \tag{5.2}$$

- **Paso de corrección:**
  Se ajusta la estimación previa utilizando las mediciones disponibles, ponderando en función de la incertidumbre tanto del modelo como de los sensores.

$$K_k = P_{k|k-1}C^t(CP_{k|k-1}C^t + R)^{-1} \tag{5.3}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) \tag{5.4}$$
$$P_{k|k} = (I - K_kC)P_{k|k-1} \tag{5.5}$$

Donde:

- $\hat{x}$: estimación del estado.
- P: matriz de covarianza del error de estimación.
- Q, R: matrices de covarianza del ruido del modelo y de los sensores, respectivamente.
- K_k: ganancia de Kalman.
- y_k: medida en el instante k.

El Filtro de Kalman minimiza la varianza del error de estimación bajo el supuesto de que las dinámicas del sistema son lineales y que tanto los ruidos del modelo como de las mediciones son de tipo gaussiano. Por ello, su uso es especialmente adecuado en entornos donde se requiere estimación precisa en tiempo real, como en la navegación y control de vehículos aéreos no tripulados.

Estimación del Estado

La estimación del estado se realiza mediante dos filtros EKF independientes, cada uno especializado en una función concreta:

a) `EKF_ATT.m` (Estimador de actitud): se encarga de calcular los ángulos de Euler (*roll, pitch, yaw*) y sus derivadas a partir de los datos de acelerómetros, giróscopos y magnetómetros. Utiliza una formulación no lineal, con actualización de la matriz de covarianza y rotaciones para mantener la coherencia de la estimación.

b) `EKF_NAV.m` (Estimador de navegación): estima la posición y la velocidad del vehículo en el marco de referencia inercial. Para ello, integra múltiples sensores (GPS, barómetro, sonar, sensor óptico, LIDAR, sistema de captura de movimiento, etc.) mediante un modelo dinámico linealizado.

Ambos estimadores realizan tres tareas fundamentales: predicción del estado, cálculo de jacobianos y actualización con las medidas disponibles. Su activación depende del modo de operación configurado en el sistema y del tipo de referencias habilitadas en cada momento.

Existen propuestas que combinan estimación de estado mediante EKF con controladores LQR basados en planificadores de trayectoria, aplicadas con éxito en drones [29].

Controlador SFC

La función `SFC.m` implementa un controlador por realimentación de estados con varias funcionalidades avanzadas:

a) **Compensación de retardos computacionales:** el controlador incorpora el histórico de las entradas previas para anticipar el efecto de los retardos introducidos por el procesamiento interno. Esto resulta esencial cuando se trabaja con frecuencias de muestreo elevadas.

b) **Control integral configurable:** permite activar acción integral sobre el error de seguimiento, con varias opciones de discretización (Euler hacia adelante, hacia atrás, trapezoidal), e incluye un sistema anti-windup para evitar la saturación del integrador.

c) **Saturación de variables manipuladas:** las señales de salida se limitan entre valores máximos y mínimos definidos, de forma que no se superen los límites físicos de los actuadores del dron.

d) **Control incremental o regulador en punto de operación:** según se active o no el control integral, el controlador puede funcionar de forma incremental (acumulando errores) o como regulador respecto a un punto de operación definido (linealización local).

En cada ciclo de control se realizan los siguientes pasos:

1. Lectura del estado y referencias actuales y previas.
2. Actualización del estado integral (si aplica).
3. Cálculo de la nueva acción de control.
4. Saturación de la salida y actualización del estado interno del controlador.

Función Control Update

El script del bloque *Control Update* evalúa:

- Si se debe activar el control (`ENABLE_CONTROL`).
- Si se ha inicializado correctamente el sistema (`INITIALIZATION_DONE`).
- El tipo de referencias activas (actitud o navegación).
- El modo de observador configurado.

Según estas condiciones, el sistema actualiza los estados estimados y selecciona la estructura de control adecuada, bien para el bucle de actitud (cuando la referencia es en términos de ángulo y velocidad angular) o para el bucle de navegación (cuando la referencia es de posición y velocidad en tierra).

5.1.4. Máquina de Estados

El bloque *"State Machine Update"* es el cerebro lógico del sistema de control. Gestiona las distintas fases operativas del dron. Está basado en una máquina de estados finita codificada en MATLAB, cuya variable principal `CURRENT_STATUS_SYS` identifica en qué estado se encuentra el vehículo de forma que se tomen decisiones coherentes con la situación.

Estados definidos

La máquina de estados contempla nueve estados numerados del 0 al 8, cada uno con una función específica dentro del flujo de control:

0) **BOOTING**
   Estado inicial tras el arranque. El sistema está cargando los módulos esenciales y preparando el entorno de control.

1) **SENSOR CALIBRATION**
   Se realizan las calibraciones necesarias de sensores inerciales y externos. Hasta que esta fase no finaliza, el sistema no puede activarse. Sirve para calcularse los *offset* de los sensores para luego restarlos y que no intervengan.

2) **READY**
   Indica que el sistema ha sido correctamente inicializado y calibrado. Espera una orden de armado para continuar.

3) **DISARMED MOTORS**
   Los motores están desactivados pero el sistema ya está operativo. Es una fase intermedia segura antes del armado.

4) **ARMING MOTORS**
   Transición hacia el armado. Durante este estado se validan las condiciones de seguridad y se inicializan variables clave del controlador.

5) **ARMED MOTORS**
   Los motores están armados y listos para volar. El sistema puede pasar a fases activas de control.

6 **RC FLIGHT**
   El dron está siendo controlado manualmente mediante mando remoto. Se desactivan referencias automáticas.

7) **ALTITUDE CONTROL**
   Se activa el control de altura, típicamente usando barómetro, ópticos o sonar. El dron puede mantenerse estable en el eje vertical, pero sin navegar horizontalmente.

8) **NAVIGATION CONTROL**
   Estado más avanzado, con control total de posición y orientación. Se ejecutan referencias de trayectoria completas.

Variables fundamentales

Las transiciones entre estados están determinadas por un conjunto de variables booleanas y condiciones temporales clave:

Variable	Tipo	Descripción
t	Temporal	Tiempo actual desde el arranque del sistema.
`init_done`	Booleana	Indica si la inicialización y calibración de sensores se ha completado.
`arming_req`	Booleana	Solicitud de armado de los motores.
`disarm_req`	Booleana	Solicitud de desarmado del sistema.
`safety_sw`	Booleana	Estado del interruptor de seguridad física.
`stop_sw`	Booleana	Parada de emergencia: puede activarse en cualquier momento.
`rc_mode`	Booleana	Activa el modo de vuelo manual mediante radiocontrol.
`nav_mode`	Booleana	Activa el modo de navegación autónoma.
`alt_mode`	Booleana	Activa el modo de control automático de altitud.
$t_{\min}$	Temporal	Tiempo mínimo para completar el proceso de armado de motores.

Tabla 5.1: Variables de la Máquina de Estados.

Diagrama de Estados

Las transiciones entre estos estados se producen en función de una combinación de condiciones booleanas, temporizadores internos, banderas de inicialización y la presencia o ausencia de referencias válidas.

La mejor forma de explicar la lógica es mediante el siguiente diagrama:

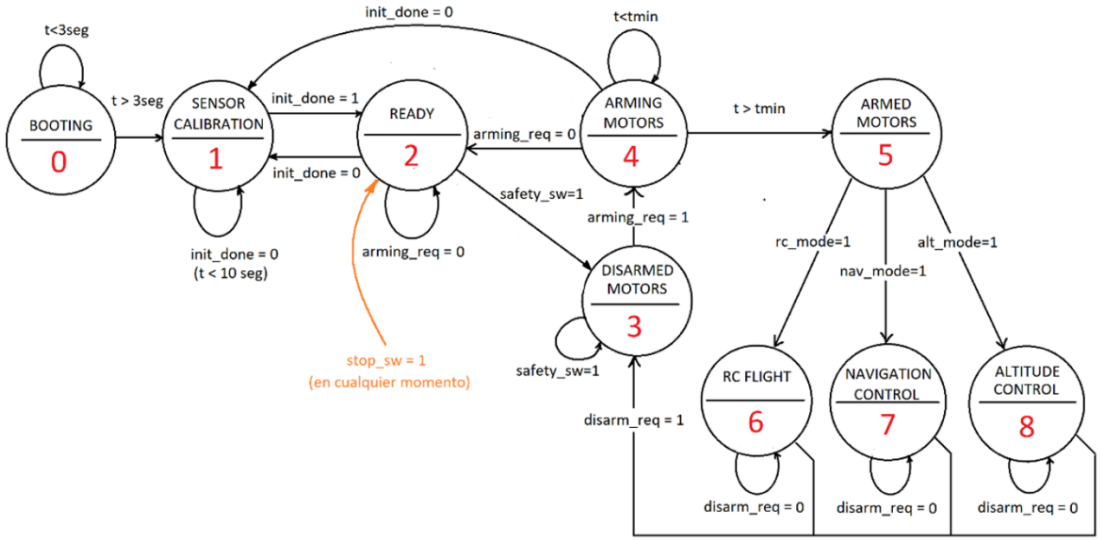

Figura 5.3: Diagrama de estados

Lógica de funcionamiento del sistema

El sistema se inicia en el estado **BOOTING (0)**, donde permanece durante los primeros tres segundos ($t < 3$ s) para garantizar que el hardware esté completamente encendido. Una vez superado este umbral, transiciona a **SENSOR CALIBRATION (1)**, donde se lleva a cabo la calibración de sensores inerciales y otros elementos críticos. Este estado permanece activo hasta que se cumple la condición `init_done = 1` (se establece a 1 automáticamente tras 10 segundos de espera).

Al finalizar la calibración, el sistema pasa al estado **READY (2)**, en el que se encuentra preparado para el armado. Desde aquí, se puede o bien: iniciar el proceso de activación de motores **ARMING MOTORS (4)** si `arming_req = 1`, o bien, permanecer desarmado en **DISARMED MOTORS (3)** si la solicitud de armado no se realiza. Para mayor seguridad, el armado también requiere que se active `safety_sw = 1` y que transcurra un tiempo mínimo ($t > t_{\min}$), tras lo cual se accede al estado **ARMED MOTORS (5)**.

Una vez armados los motores, el sistema puede operar en tres modos de control diferenciados:

- **RC FLIGHT (6):** activado mediante `rc_mode = 1`, habilita el control manual del dron mediante emisora.

- **NAVIGATION CONTROL (7):** activado con `nav_mode = 1`, habilita el control autónomo mediante planificación de trayectoria.
- **ALTITUDE CONTROL (8):** activado con `alt_mode = 1`, habilita el mantenimiento automático de altitud.

Desde cualquier modo de vuelo o estado armado, el sistema puede ser desarmado mediante `disarm_req = 1`, retornando de forma segura al estado **DISARMED MOTORS (3)**. Adicionalmente, una condición de emergencia `stop_sw = 1` puede forzar la transición inmediata a **READY (2)**, garantizando una parada controlada.

Esta estructura lógica permite que no se activen motores si no se cumplen las condiciones críticas de seguridad y permite una rápida transición entre los diferentes modos de operación.

5.2. Hardware

El hardware se ha dividido en tres grandes partes: sensores, actuadores y comunicaciones.

5.2.1. Sensores

Este subsistema agrupa todos los sensores del controlador, encargados de proporcionar la información del entorno y del estado del dron. Entre ellos se encuentran sensores inerciales, de seguridad y de medida de distancia.

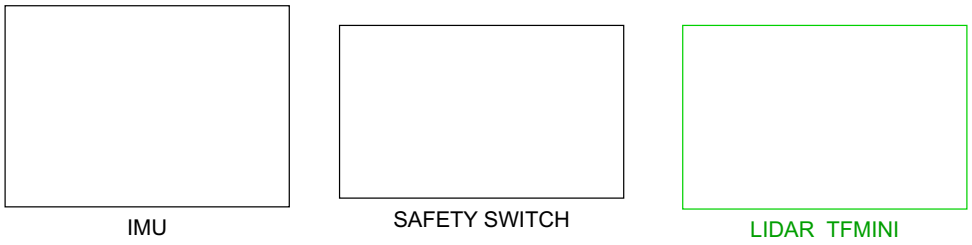

Figura 5.4: Subsistema de sensores

A continuación se describen los bloques más relevantes de este subsistema.

IMU

El subsistema *"IMU"* adquiere los datos brutos de la Unidad de Medición Inercial (IMU, por sus siglas en inglés). Estos datos se filtran para calibrar tanto el acelerómetro como el giroscopio, eliminando los posibles *offsets* que las medidas en bruto introducen inevitablemente.

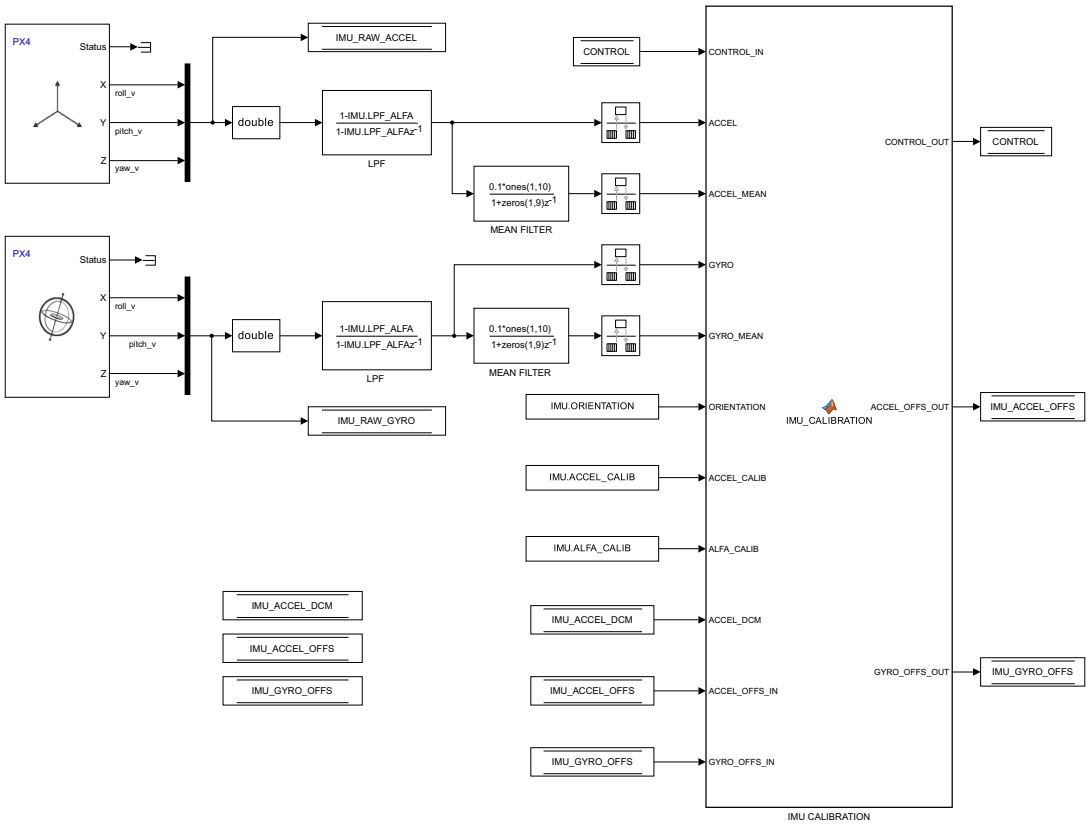

Figura 5.5: Diagrama de la IMU

El sistema parte de dos bloques de entrada PX4, uno para el acelerómetro y otro para el giroscopio, desde los cuales se extraen las lecturas crudas de los tres ejes: X, Y y Z.

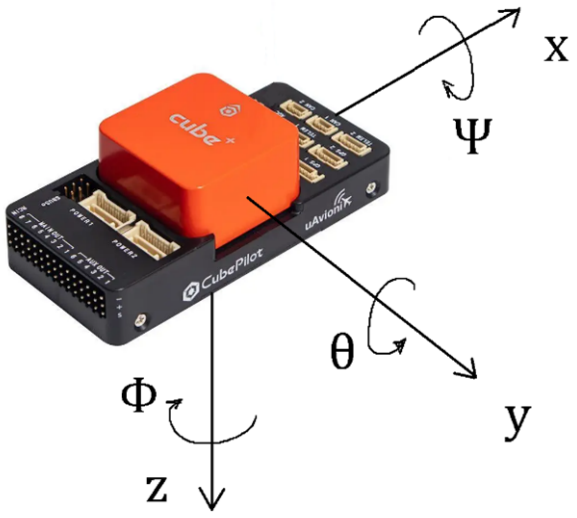

Figura 5.6: Ejes de referencia en el Cube Orange+

Estas señales se procesan en primer lugar mediante un filtro paso bajo digital (LPF) de primer orden, diseñado para atenuar el ruido de alta frecuencia sin introducir mucha latencia [30]. La función de transferencia de ambos filtros es:

$$H(z) = \frac{1-\alpha}{1-\alpha \cdot z^{-1}} \quad \text{con } 0 < \alpha < 1 \tag{5.6}$$

- Si $\alpha \to 0$, el filtro atenúa casi todo (respuesta lenta).
- Si $\alpha \to 1$, el filtro deja pasar casi todo (menos atenuación).

Se eligió un valor de $\alpha = 0{,}9$ para los filtros.

Posteriormente, se aplica un filtro de media móvil (*Mean Filter*) que suaviza aún más la señal, promediando las últimas diez muestras.

Todas estas señales (las filtradas por el LPF y las filtradas dos veces: por LPF y luego por el *Mean Filter*) se envían al bloque denominado `IMU_CALIBRATION`, que cumple varias funciones clave:

a) **Compensación por orientación física**
En función del parámetro `ORIENTATION`, el bloque reordena los ejes del acelerómetro y giroscopio para adaptarse a la orientación real de la IMU dentro de la estructura del dron. Están previstas las dos configuraciones más típicas: en "+" y en "X".

b) **Estimación de *offsets* en tiempo real**
Si el sistema se encuentra en modo de calibración (`CURRENT_STATUS_SYS = 1`) y se ha activado la opción de calibrar el acelerómetro, se calcula el vector de compensación necesario para que la aceleración medida en reposo coincida con la aceleración gravitatoria. Para el giroscopio, se estima un *offset* que compense el error del sensor cuando el dron está en reposo.

c) **Aplicación de compensaciones**
Una vez finalizada la fase de calibración, los datos brutos se corrigen restando los *offsets* calculados y aplicando la matriz de orientación de la IMU (`ACCEL_DCM`), obteniendo así las medidas inerciales ya compensadas.

Finalmente, las señales corregidas se almacenan en el bus `CONTROL_OUT`, que se utiliza como entrada para el control del dron.

Este proceso garantiza que las medidas inerciales empleadas en los algoritmos de control y navegación sean precisas, estables y coherentes, incluso ante variaciones en la orientación física de la IMU o desviaciones debidas al envejecimiento de los sensores. La navegación visual-inercial es especialmente útil en entornos donde se requiere alta precisión sin condiciones iniciales conocidas [31].

Interruptor de Seguridad (Safety Switch)

El subsistema *"Safety Switch"* se encarga de leer el estado de seguridad del sistema de control de vuelo a través del *topic* `VehicleStatus` de PX4. Posteriormente, la información leída se guardará dentro de una variable booleana en el bus de control general del sistema.

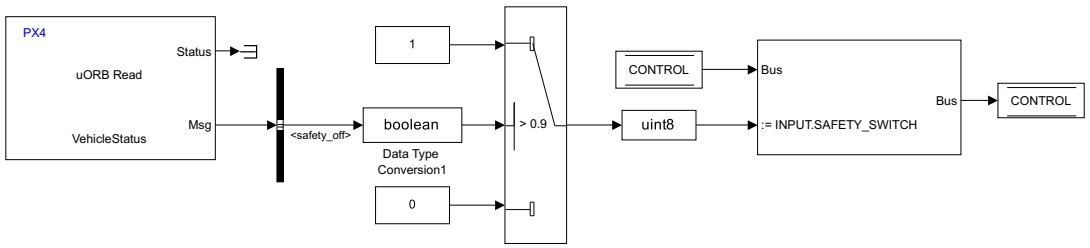

Figura 5.7: Diagrama de Safety Switch

Para ello, se utiliza un bloque `uORB Read` [32], que se suscribe a uno de los *topic* que ofrece la propia librería: `VehicleStatus`, específicamente a la variable `safety_off`.

Esta variable indica si el sistema de seguridad está desactivado (es decir, si el dron está armado y listo para operar).

El valor leído se compara mediante una estructura condicional: si el valor es mayor que 0.9, se interpreta como `true` (seguridad desactivada), y se asigna el valor 1. En caso contrario, se asigna el valor 0. Esta salida se convierte a tipo `uint8` para asegurar la compatibilidad con el resto del sistema. Finalmente, la señal resultante se inyecta en el bus de control mediante un bloque `Bus Assignment`, sobrescribiendo el campo `INPUT.SAFETY_SWITCH`.

La inclusión de un safety switch es fundamental para prevenir el armado accidental de los motores, tal y como destaca la documentación oficial de sistemas autopilotados [33]. De esta forma, el sistema reacciona de forma coherente al estado del interruptor de seguridad: si no se arma de forma explícita (pulsando un botón), el dron no se armará.

Lidar TF Mini

El subsistema *"Lidar TFMini"* se encarga de guardar las medidas de distancia del sensor Lidar TFmini-S, conectado al controlador de vuelo (Cube Orange+) a través del bus I2C.

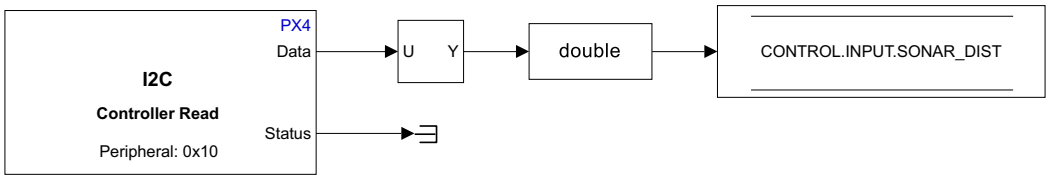

Figura 5.8: Diagrama del sensor Lidar TF Mini

Para ello, se utiliza el bloque `I2C Controller Read` de PX4 [34], configurado para leer desde el bus I2C 2, con dirección de periférico `0x10`. El bloque devuelve un vector de 9 valores en formato `single` (4 bytes por valor), representando los datos transmitidos por el sensor.

De entre todos los valores adquiridos, se selecciona la componente de interés mediante un bloque `Selector` (medida de la distancia en mm). Finalmente, la medida se almacena en la variable `CONTROL.INPUT.SONAR_DIST`.

Este subsistema permite que el controlador reciba medidas de distancia al suelo de forma continua, ya que, a pesar de que el Cube Orange+ es capaz de estimar la altura integrando las aceleraciones medidas por la IMU, la incorporación de un sensor de distancia externo permite obtener una medida más directa, estable y fiable, especialmente en maniobras cercanas al suelo o en entornos con ruido dinámico.

5.2.2. Actuadores

En este proyecto, los únicos actuadores que se pueden controlar directamente son los motores, encargados de generar la propulsión necesaria para el vuelo del dron.

BLDC Motors

El subsistema *"BLDC Motors"* se encarga de generar las señales de control PWM necesarias para accionar los cuatro motores *brushless* (BLDC) del dron. Las señales se mandan directamente al controlador PX4 a través del bloque `PX4 PWM Output`.

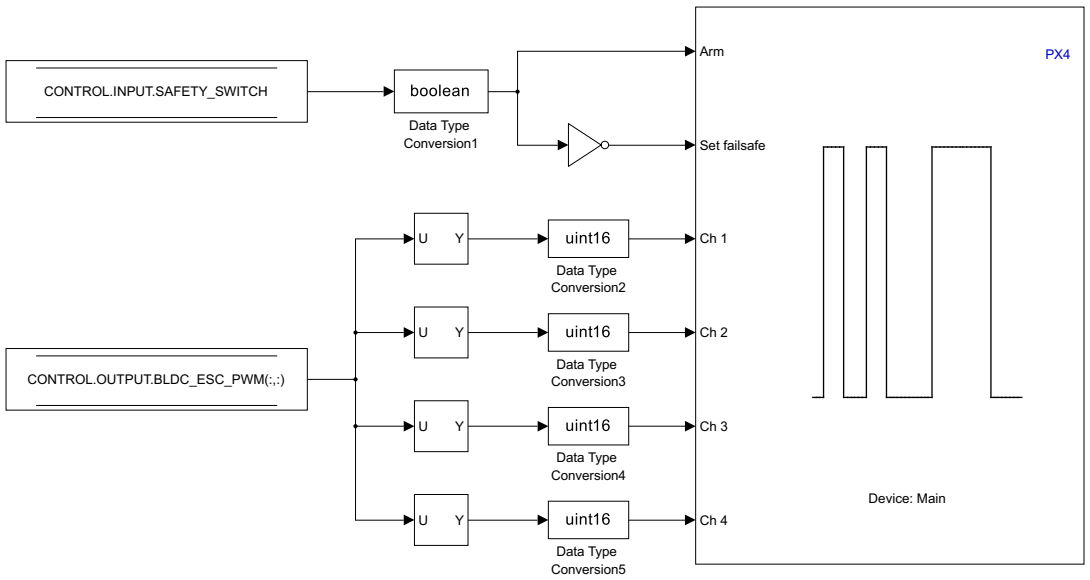

Figura 5.9: Diagrama de los motores

El bloque recibe dos tipos de entradas:

a) **Señal de seguridad (Safety Switch)**
Se lee desde el bus compartido el valor `CONTROL.INPUT.SAFETY_SWITCH`, el cual indica si el sistema está armado. La señal se convierte a booleana (el bloque `PX4 PWM Output` está configurado internamente para que sea booleana), y se utiliza en dos entradas del mismo:

   i) `Arm:` permite la activación de los motores.

   ii) `Set Failsafe:` se activa automáticamente cuando la señal de armado es falsa, a través de un bloque `NOT`.

   Esta lógica garantiza que, mientras el sistema no esté armado, los motores permanezcan desactivados y en estado seguro.

b) **Señales PWM para cada motor**
Se accede también al bus `CONTROL.OUTPUT.BLDC_ESC_PWM`, que contiene un vector de 8 elementos con valores comprendidos entre 1000 y 2000, correspondientes al ancho de pulso en microsegundos para cada ESC (*Electronic Speed Controller*). Mediante bloques `Selector`, se extraen las señales correspondientes a los primeros cuatro canales del vector (uno por cada motor). Cada valor es convertido al tipo `uint16`, como requiere el bloque `PX4 PWM Output`.

5.2.3. Comunicaciones

El sistema de comunicaciones permite la interacción entre el operador y el dron. A través de la emisora se envían comandos que son procesados por el controlador a bordo, facilitando el pilotaje manual o semiautomático.

Emisora

El subsistema *"Emisora"* se encarga de leer, calibrar y transformar las señales recibidas por radiocontrol (RC) a través del *topic RcChannels* de PX4. A partir de dichas señales, genera referencias manuales para los algoritmos de control de actitud, velocidad y navegación.

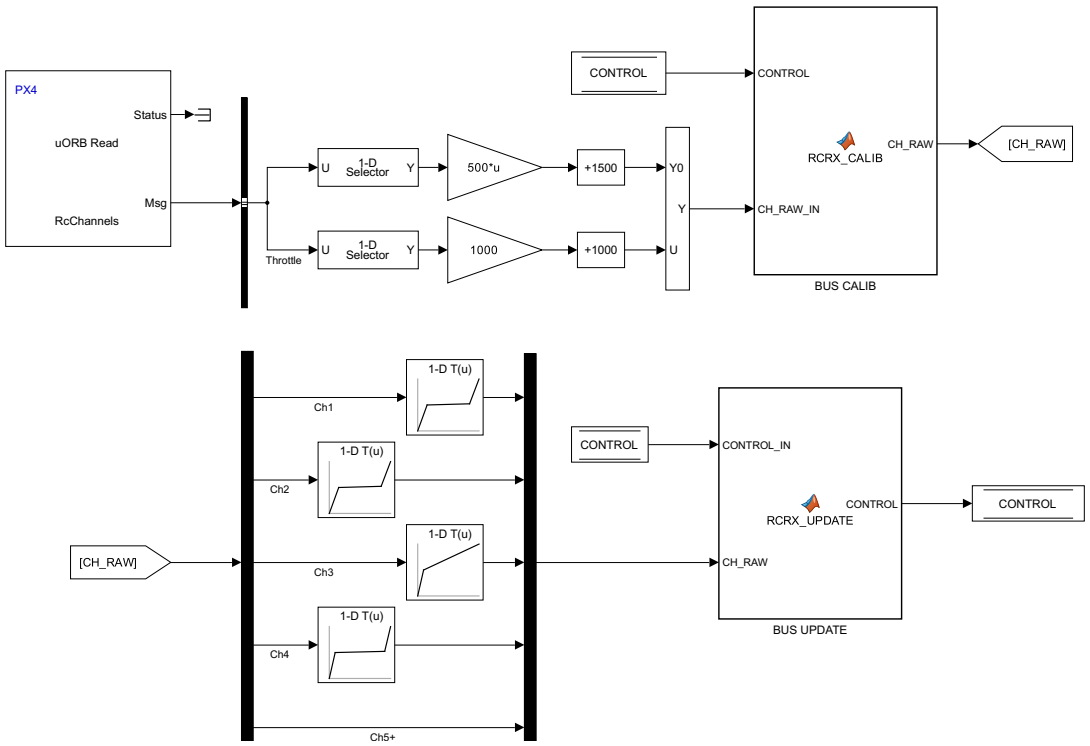

Figura 5.10: Diagrama de la emisora

El proceso se puede dividir en cuatro etapas:

1. **Lectura y tratamiento de emisora RC:**
   Mediante un bloque `uORB Read`, se accede al *topic* `RcChannels`, que contiene las señales de los *sticks* del mando, con valores normalizados en los rangos:

   - $[-1, 1]$ para los canales de *Roll*, *Pitch* y *Yaw*.
   - $[0, 1]$ para el canal de *Throttle*.

   Estas señales son posteriormente escaladas al rango típico de los canales RC físicos:

   - **Roll, Pitch y Yaw:**

   $$500 \cdot u + 1500 \quad \Rightarrow \quad [1000, 2000]$$

   - **Throttle:**

   $$1000 \cdot u + 1000 \quad \Rightarrow \quad [1000, 2000]$$

Los canales transformados se agrupan en el vector de entrada (`CH_RAW_IN`) y se envían al bloque `RCRX_CALIB`. Cabe mencionar que la emisora emite 14 canales, pero en este proyecto se han utilizado solo seis:

Canal	**Función asignada**	**Descripción**
Ch1	Roll	Movimiento lateral (alabeo)
Ch2	Pitch	Movimiento longitudinal (cabeceo)
Ch3	Throttle	Potencia general de los motores
Ch4	Yaw	Rotación sobre el eje vertical (guiñada)
Ch5	Control Mode	Conmutador para elegir el modo (Altitude/Navigation/RC)
Ch6	Stop Switch	Parada de emergencia
Ch7+	—	No utilizados en este proyecto

Tabla 5.2: Asignación de canales de la emisora en el sistema de control.

2. **Calibración personalizada por canal:**
   Este bloque aplica una calibración por tramos configurable para cada canal. Sirve para compensar las asimetrías y *offsets* de hardware. Cada canal se calibra usando dos rectas, dependiendo de si su valor está por debajo o por encima del punto central definido (`trim`). Esto permite una respuesta no lineal adaptada al mando específico utilizado.

   Para cada canal n, se define una calibración por tramos mediante la siguiente ecuación:

$$\mathrm{CH}_{\mathrm{RAW}}(n) = \begin{cases} a_1 \cdot x + b_1, & \text{si } x < \text{trim} \\ a_2 \cdot x + b_2, & \text{si } x \geq \text{trim} \end{cases} \tag{5.7}$$

   Donde:

   - x: valor de entrada del canal (escalado)
   - `trim`: valor central del canal
   - a_1, b_1 y a_2, b_2: coeficientes de calibración de cada tramo

Para obtener estos coeficientes, se resuelve un sistema de ecuaciones lineales a partir de tres puntos de referencia por canal:

- Valor mínimo del canal:

$$CH_{\text{mín}} = 1000$$

- Valor central:

$$CH_{\text{trim}} = 1500$$

- Valor máximo del canal:

$$CH_{\text{máx}} = 2000$$

Se plantea el siguiente sistema matricial para obtener los cuatro coeficientes para cada canal n:

$$\begin{bmatrix} CH_{\text{mín}} & 1 & 0 & 0 \\ CH_{\text{trim}} & 1 & 0 & 0 \\ 0 & 0 & CH_{\text{máx}} & 1 \\ 0 & 0 & CH_{\text{trim}} & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1000 \\ 1500 \\ 2000 \\ 1500 \end{bmatrix} \tag{5.8}$$

Donde la matriz de términos independientes representa los valores deseados para cada punto de calibración. Este sistema se resuelve por cada canal individualmente, y sus soluciones se almacenan en la matriz `CALIB_PARAM`, con una fila por canal:

$$\texttt{CALIB\_PARAM}(n, :) = [a_1 \quad b_1 \quad a_2 \quad b_2] \tag{5.9}$$

3. **Conversión a magnitudes físicas:**
   Una vez calibradas las señales mediante `RCRX_CALIB`, se aplican transformaciones adicionales utilizando bloques `1-D Lookup Table`. Estos bloques permiten aplicar curvas personalizadas para suavizar o comprimir distintas zonas del recorrido del *stick*:

   - *Roll, Pitch y Yaw* (Ch1, Ch2 y Ch4): curvas que suavizan la respuesta alrededor del valor central (1500), reduciendo la sensibilidad cerca del *trim* y ampliándola en los extremos.
   - *Throttle* (Ch3): se da mayor resolución en la zona baja (1000–1100), permitiendo un control más preciso en el despegue o vuelo estacionario.

Estas curvas están definidas mediante pares de puntos (*breakpoints* y *table data*) que representan la relación entre el valor de entrada y el valor transformado.

4. **Generación de referencias para el control:**
   La generación de referencias desde la emisora se realiza en el bloque `BUS_UPDATE`, que convierte las señales brutas recibidas en comandos útiles para el sistema de control. Este bloque toma como entrada el vector de canales calibrados `CH_RAW` y la estructura `CONTROL_IN`, y devuelve una nueva estructura `CONTROL` con las referencias actualizadas en función del modo de operación del dron.

   La Figura 5.11 muestra la asignación de cada canal utilizado para este proyecto en la emisora.

Figura 5.11: Distribución de canales de la emisora

Explicación detallada del bloque `BUS_UPDATE`

En primer lugar, las señales de la emisora se transforman al rango normalizado $[-1, 1]$, tomando 1500 como el valor central. Esta transformación se almacena en el vector `CH_PU` (*per-unit*):

$$CH_{PU} = \frac{CH_{RAW} - 1500}{500} \tag{5.10}$$

Dado que el canal correspondiente al *Throttle* debe operar entre 0 y 1, se reescala por separado mediante:

$$CH_{PU}^{\text{Throttle}} = \frac{CH_{PU}(3) + 1}{2} \tag{5.11}$$

Posteriormente, y según el estado actual del dron (`CURRENT_STATUS_SYS`), el sistema selecciona cómo interpretar estas señales y en qué variables internas deben aplicarse:

a) **Modo *ALTITUDE CONTROL***
En este modo se activa el control automático de altura, mientras que los demás ejes (actitud y guiñada) siguen siendo controlados manualmente mediante la emisora. Las referencias se generan como:

- *Roll y Pitch*: referencias angulares (en radianes) en torno a 0, con límite de $\pm 15°$.
- *Yaw*: velocidad angular (rad/s), hasta $\pm 360°/s$.
- Altura: velocidad de consigna en el eje vertical (m/s), con escala limitada.

b) **Modo *NAVIGATION CONTROL***
En este modo, el control es completamente autónomo y las señales de la emisora representan velocidades horizontales deseadas y velocidad de ascenso/descenso:

- X e Y: velocidades horizontales (m/s), hasta ± 20 m/s.
- *Yaw* rate: velocidad de guiñada (rad/s).
- Altura: velocidad vertical deseada (m/s), hasta $\pm 2{,}5$ m/s.

c) **Modo *RC FLIGHT (Manual)***
En el modo manual, las señales de la emisora representan:

- Ángulo de *Roll* y *Pitch*
- Velocidad angular de *Yaw*
- Potencia de motor *Throttle*

Todas las referencias generadas se almacenan en el campo `CONTROL.TARGET`, donde se diferencian claramente según el tipo: referencias angulares para *roll* y *pitch* (rad), referencia de velocidad angular en *yaw* (rad/s), referencia de velocidad en el plano horizontal (m/s), velocidad vertical deseada (en vez de posición) y potencia de motor (0 a 1 o 2).

Finalmente, se guarda el vector completo `RC_CH_PU` para poder monitorizarlo en cualquier momento.

5.3. Simulation

El subsistema denominado *SIMULATION* es el núcleo del entorno de simulación del dron. Se encarga de reproducir la evolución temporal del estado del sistema en función de las entradas de control y las condiciones dinámicas del entorno. El subsistema representa el modelo matemático discreto del dron a partir de una formulación en espacio de estados.

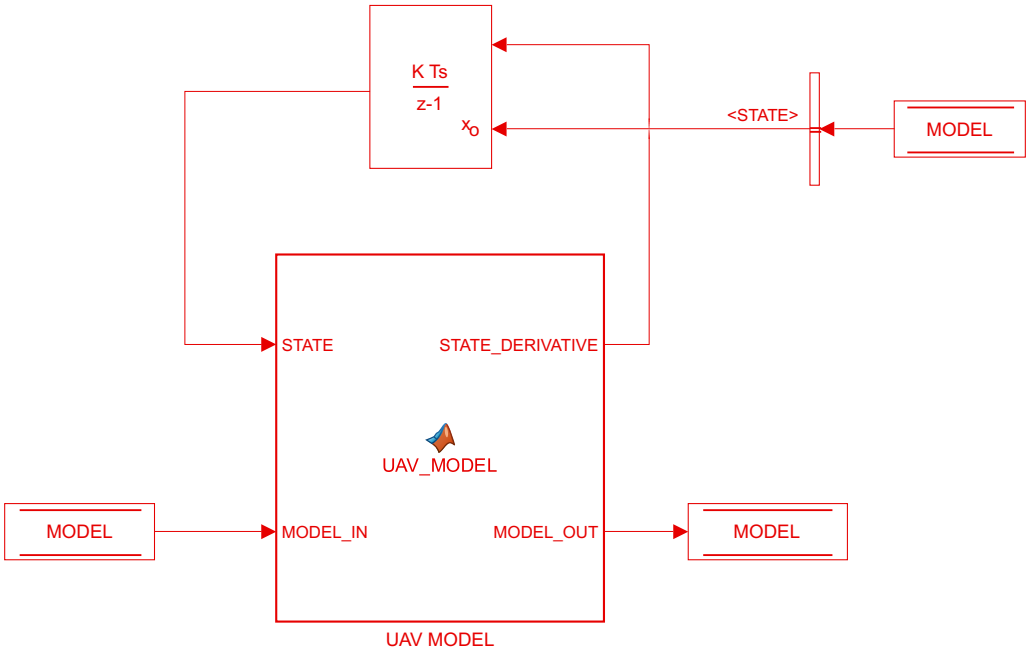

Figura 5.12: Diagrama de simulación

Está compuesto por tres bloques principales:

- **Bloque UAV_MODEL:** módulo principal que define la dinámica completa del vehículo no tripulado (UAV).
- **Bloque Discrete-Time Integrator:** integrador temporal discreto que actualiza el estado del sistema [35].
- **Bloques Data Store Read y Data Store Write:** encargados de leer y actualizar el estado del sistema almacenado en memoria compartida a través de variables globales definidas en `CONFIG_MODEL.m` [36].

De esta forma, se simula el comportamiento físico del dron en función de las entradas, devolviendo el estado actualizado en cada iteración del ciclo de simulación. Dicho ciclo sigue una secuencia definida:

1. **Lectura del estado actual (STATE):** mediante un bloque `Data Store Read`, se accede al vector de estado almacenado en memoria compartida (variable `"State"`).
2. **Cálculo de la derivada del estado:** el bloque `UAV_MODEL` evalúa la dinámica del sistema, generando el vector `STATE_DERIVATIVE` en función del estado actual y de las entradas contenidas en `MODEL_IN`.
3. **Integración del estado:** se aplica un integrador discreto del tipo:

$$\frac{K \cdot T_s}{z - 1}$$

   para así obtener el nuevo estado. Este modelo considera que la derivada del estado se mantiene constante durante el periodo de muestreo T_s, es decir, se está aplicando un retenedor de orden cero.
4. **Escritura del nuevo estado:** el estado actualizado se almacena mediante un bloque `Data Store Write` en la variable `"STATE"`, cerrando así el bucle de simulación.

5.3.1. Bloque UAV_MODEL

El bloque `UAV_MODEL` es el núcleo dinámico del sistema simulado. Modela el comportamiento físico completo del dron a partir del estado actual y de las entradas de control, permitiendo predecir la evolución del sistema.

La función `UAV_MODEL.m` recibe el vector de estado `STATE` y la estructura de parámetros y entradas `MODEL_IN`, y devuelve:

- `STATE_DERIVATIVE`: la derivada temporal del estado del dron.
- `MODEL_OUT`: copia extendida de `MODEL_IN` que incluye información completa de la simulación (salidas, variables intermedias, estados auxiliares, etc.).

a) Estructura del estado

El vector de estado `STATE` incluye las principales variables físicas del UAV:

- Velocidad angular en el cuerpo: `BODY_RATE`
- Velocidad y posición en el sistema terrestre: `EARTH_VEL`, `EARTH_POS`

- Orientación: representada mediante los ángulos de Euler
- Velocidad de giro de los motores: `MOT_SPEED`
- Voltaje de batería: `BAT_VOLT`

Es decir:

$$X = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \\ x \\ y \\ z \\ \phi \\ \theta \\ \psi \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ V_{bat} \end{bmatrix} \tag{5.12}$$

Cabe destacar que, al tratarse de una simulación, se puede incluir un vector de estado completo que contempla todas las variables físicas relevantes del sistema. Esto permite representar de forma precisa el comportamiento dinámico del dron y analizar su evolución en distintas condiciones de operación.

En un sistema de control real, sin embargo, no se utilizan todos estos estados. En ese caso, el modelo se simplifica para reducir la carga computacional y adaptarse a las limitaciones del hardware. Los estados utilizados en ese contexto se describen en la Sección 3. Modelo de la Planta.

b) Parámetros de configuración

Los parámetros utilizados en la simulación se cargan desde `MODEL_IN.PARAM` y son definidos previamente en el archivo `CONFIG_MODEL.m`. Entre los más relevantes se encuentran:

- Masa del UAV, matriz de inercia y geometría del chasis
- Densidad del aire y gravedad

- Datos eléctricos del motor (resistencia, constante de par, constante de velocidad, inercia)
- Curva PWM–*thrust* experimental
- Parámetros de batería: capacidad, tensión máxima y modo de simulación (variable o constante)

c) Modelado de motores y propulsión

El modelo simula los motores *brushless* (BLDC) alimentados por señales PWM. A partir de la señal de entrada `ESC_PWM`, se calcula:

- Tensión aplicada al motor (`MOT_VOLT`)
- Corriente de motor (`MOT_CURRENT`)
- Par generado (`MOT_TORQUE`)
- Fuerza de empuje (interpolada a partir de curvas PWM–*thrust*)
- Par de arrastre

A partir de los ángulos de Euler, se calcula la matriz de rotación del sistema cuerpo al sistema terrestre (DCM, *Direction Cosine Matrix*), utilizando la convención de rotaciones ZYX y la derivada de estos ángulos.

El modelo soporta múltiples configuraciones físicas, pero en este trabajo se ha centrado principalmente en el cuadricóptero en configuración "X", donde los torques generados por los motores se calculan como combinaciones lineales del empuje de cada hélice respecto a los brazos de palanca. También se incorpora el efecto del momento giroscópico debido a los motores.

Es decir, el sistema considera tres fuerzas:

1. **Fuerza gravitatoria**, proyectada en el sistema cuerpo:

$$\left|\vec{F}_g\right| = m \cdot g \tag{5.13}$$

2. **Fuerza de resistencia (drag)**:

$$\left|\vec{F}_{\text{drag}}\right| = \frac{1}{2} \cdot \rho \cdot A \cdot C_d \cdot v^2 \tag{5.14}$$

3. **Empuje de hélices**, que teóricamente es proporcional al cuadrado de la velocidad angular del motor:

$$\left|\vec{F}_i\right| = k_F \cdot \omega_i^2 \quad i = 1, 2, 3, 4 \tag{5.15}$$

Antes de aplicar la ecuación de Newton, se representa en la Figura 5.13 el diagrama de cuerpo libre, donde se muestran las principales fuerzas que actúan sobre el dron en el sistema cuerpo.

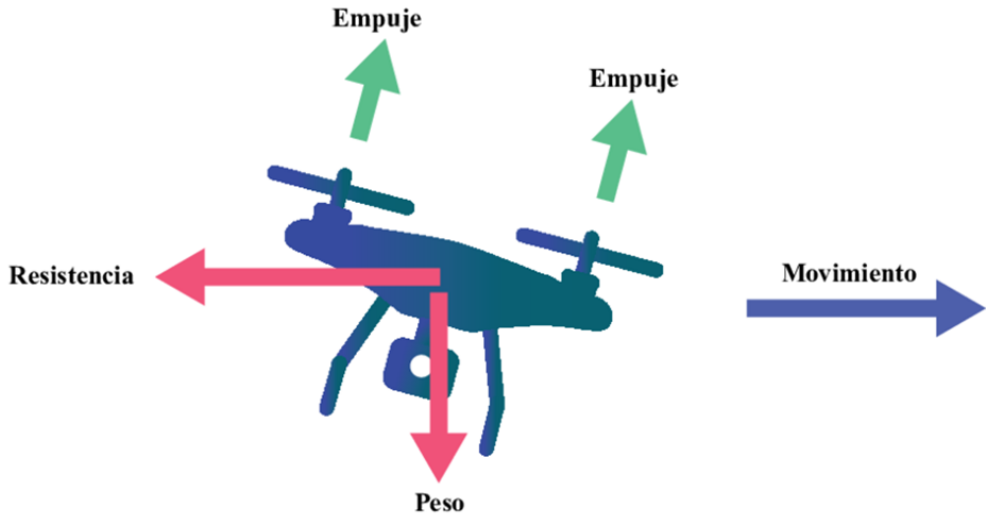

Figura 5.13: Diagrama de Cuerpo Libre (DCL) del dron en el sistema cuerpo

Estas fuerzas se suman y se utiliza la segunda ley de Newton en el sistema de referencia no inercial que rota para calcular la aceleración del cuerpo:

$$\sum \vec{F} - \vec{\omega} \times (m \cdot \vec{v}) = m \cdot \frac{d\vec{v}}{dt} \tag{5.16}$$

Donde la aceleración resultante se transforma luego al sistema terrestre para actualizar las variables de posición y velocidad. El término $-\vec{\omega} \times (m \cdot \vec{v})$ representa la fuerza de Coriolis que "corrige" el análisis dinámico por estar en un sistema que rota [37].

Aproximación experimental del empuje

Cabe destacar que se ha optado por una aproximación más precisa basada en datos experimentales. Durante los ensayos se elaboró una tabla que relaciona directamente el valor de la señal PWM (entre 1000 y 2000 μs) con el empuje generado por cada motor (medido en gramos).

PWM (µs)	Empuje medio (g)
1000	0.00
1100	11.38
1200	31.89
1300	64.03
1400	106.23
1500	148.50
1600	186.16
1700	224.17
1800	260.65
1900	311.57
2000	364.75

Tabla 5.3: Relación entre PWM y el empuje medio generado por los motores.

En el modelo, esta relación se implementa mediante interpolación lineal en intervalos de 100 µs, lo que estima el empuje con mayor fidelidad sin necesidad de calcular la velocidad angular de las hélices. Gráficamente:

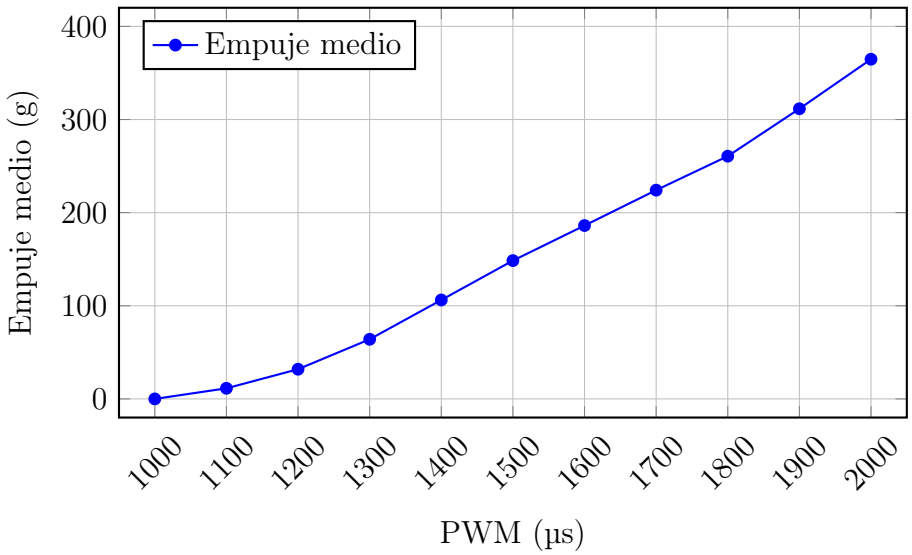

Figura 5.14: Curva experimental de empuje medio en función de la señal PWM.

Nota:
Aunque teóricamente el empuje es proporcional al cuadrado de la velocidad angular del motor, la relación experimental entre la señal PWM y el empuje generado no sigue exactamente una parábola, aunque sí presenta una tendencia similar.

d) Modelo de batería

Cuando se activa la simulación de batería variable, el voltaje se reduce según el consumo total de corriente y la capacidad restante. El modelo implementado se basa en una descarga proporcional de la batería según la corriente consumida por los motores, como se presenta en modelos simplificados de dinámica energética para UAVs [38].

Tasa de descarga de Capacidad:

$$\dot{Q}(t) = -\frac{1000}{3600} \cdot \sum |I_{\text{motor, i}}| \tag{5.17}$$

Donde:

- $Q(t)$: capacidad restante de la batería [mAh].
- $I_{\text{motor, i}}$: intensidad consumida por el motor i [A].
- $\frac{1000}{3600}$: factor que convierte de A en mAh/s.

Derivada del Voltaje:

$$\frac{dV_{\text{bat}}(t)}{dt} = \frac{\dot{Q}}{Q_{\text{máx}}} \cdot V_{\text{bat}}^{\text{máx}} \tag{5.18}$$

Donde:

- $V_{\text{bat}}(t)$: voltaje actual de la batería [V].
- $Q_{\text{máx}}$: capacidad nominal máxima [mAh].
- $V_{\text{bat}}^{\text{máx}}$: voltaje nominal máximo [V].

e) Salidas del modelo

Todos los valores calculados en la simulación se almacenan en `MODEL_OUT.OUTPUT`, incluyendo: ángulos de Euler y sus derivadas, velocidades lineales y angulares, matrices de rotación, fuerzas y torques aplicados, tensiones, corriente y capacidad de batería y, en general, el estado físico completo del UAV.

5.3.2. Integrador Discreto

El bloque *Discrete-Time Integrator* actualiza el estado completo del sistema UAV simulando su evolución temporal paso a paso. Actúa como un integrador numérico implementado en tiempo discreto, asumiendo que las derivadas del estado se mantienen constantes durante cada periodo de muestreo.

Para poder simular el comportamiento en un entorno digital (como MATLAB/*Simulink*), es necesario transformar el sistema continuo en uno discreto. Para ello, se ha utilizado un periodo de muestreo T_s de 10 ms, y se ha aplicado el método de integración *Forward Euler*, cuya fórmula general es:

$$X_{k+1} = X_k + T_s \cdot \dot{X}_k \tag{5.19}$$

Donde:

- X_k: Estado del sistema en el instante kT_s
- $\dot{X}_k$: Derivada del estado del sistema en el instante kT_s

Este método asume que la derivada del estado se mantiene constante durante el intervalo de muestreo. Es utilizado por su simplicidad y bajo coste computacional, lo que lo hace especialmente adecuado para simulaciones en tiempo real o con alta frecuencia de muestreo.

Si se compara con otros métodos de integración, como "Backward Euler" y "Trapezoidal", estos presentan mejores propiedades numéricas (mayor estabilidad o precisión), pero requieren resolver ecuaciones implícitas en cada paso (lo cual aumenta mucho su coste computacional). La Tabla 5.4 resume sus diferencias.

Método	**Fórmula**	**Características**
Forward Euler	$X_{k+1} = X_k + T_s \cdot \dot{X}_k$	Simple, rápido, pero menos estable
Backward Euler	$X_{k+1} = X_k + T_s \cdot \dot{X}_{k+1}$	Implícito, más estable, requiere resolver una ecuación
Trapezoidal	$X_{k+1} = X_k + \frac{1}{2}T_s \cdot (\dot{X}_k + \dot{X}_{k+1})$	Mejor precisión, también implícito

Tabla 5.4: Métodos de Integración.

5.4. Monitoring

El subsistema *Monitoring* se encarga de seleccionar y almacenar las variables internas más relevantes del sistema de control, con el objetivo de permitir su visualización y análisis posterior desde la estación de tierra.

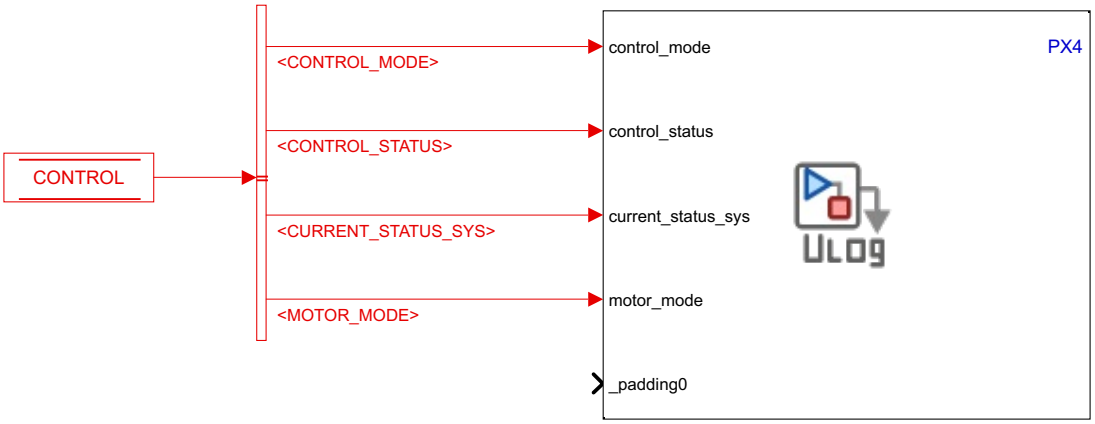

Figura 5.15: Ejemplo de diagrama de monitorización

El diagrama se compone de los siguientes bloques principales:

1. **Data Store Read:** accede al campo `CONTROL`, que contiene toda la estructura interna con las variables generadas por los controladores, sensores y lógica del sistema.
2. **Bus Selector:** se utiliza para extraer únicamente las señales de interés dentro de la estructura `CONTROL`, tales como referencias, estimaciones, errores, estados de control y entradas de actuador, entre otras.
3. **PX4 uLOG:** todas las señales seleccionadas se redirigen a este bloque, que genera los mensajes de log en tiempo real. Estos datos quedan registrados en el archivo `ulog` (véase la documentación del proyecto PX4 [39]), y pueden visualizarse posteriormente en QGroundControl.

Este sistema de monitorización es especialmente útil para depuración, análisis de rendimiento y verificación en vuelo, ya que permite revisar el comportamiento interno del dron sin necesidad de intervenir el código o los algoritmos de control.

Bibliografía

[1] MathWorks, "Uav toolbox support package for px4 autopilots," 2025. [En línea]. Disponible en: https://es.mathworks.com/help/uav/px4-spkg.html [Último acceso: 25 abr. 2025].

[2] CubePilot, "The cube module overview," 2025. [En línea] Disponible en: https://docs.cubepilot.org/user-guides/autopilot/the-cube-module-overview [Último acceso: 20 jun. 2025].

[3] PX4 Autopilot, "Cubepilot cube orange flight controller," 2025. [En línea]. [Último acceso: 11 jun. 2025].

[4] W. Lee, "Enabling reliable uav control by utilizing multiple protocols and paths for transmitting duplicated control packets," *Sensors*, vol. 21, no. 9, p. 3295, 2021. [En línea]. Disponible en: https://doi.org/10.3390/s21093295 [Último acceso: 25 abr. 2025].

[5] N. González García, "Control de un cuadricóptero para navegación en interiores usando un sensor de flujo óptico." Trabajo Fin de Grado, Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas, 2016. [En línea] Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/14540 [Último acceso: 2 may. 2025].

[6] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "Mav navigation through indoor corridors using optical flow," in *ICRA*, 2010. [En línea]. Disponible en: https://rpg.ifi.uzh.ch/docs/ICRA10_zingg.pdf [Último acceso: 22 jun. 2025].

[7] J. García Aguilar, "Desarrollo de sistemas de navegación autónoma para un uav." Trabajo Fin de Grado, Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas, 2018. [En línea] Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/22629 [Último acceso: 10 may. 2025].

[8] J. J. B. Vázquez, "Control de navegación autónoma de un cuadricóptero en interiores." Trabajo Fin de Grado, Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas, 2018. [En línea] Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/32609 [Último acceso: 18 jun. 2025].

[9] NASA review, "A review of uav technology in power line inspection," 2020. [En línea]. Disponible en: https://ntrs.nasa.gov/api/citations/20205002834/downloads/A%20Review%20of%20Unmanned%20Aerial%20Vehicle%20Technology%20in%20Power%20Line%20Inspection%20Abstract%20V1.6.pdf [Último acceso: 26 may. 2025].

[10] C. Malang, P. Charoenkwan, and R. Wudhikarn, "Implementation and critical factors of unmanned aerial vehicle (uav) in warehouse management: A systematic literature review," *Drones*, vol. 7, no. 2, p. 80, 2023. [En línea]. Disponible en: https://doi.org/10.3390/drones7020080 [Último acceso: 4 may. 2025].

[11] A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, "Drones in agriculture: A review and bibliometric analysis," *Agricultural Systems*, 2022. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0168169922003349 [Último acceso: 1 jun. 2025].

[12] C. Osorio and J. Martinez-Carranza, "Unmanned aerial systems in search and rescue," *Drones and Rescue Journal*, 2025. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S2212420925000238 [Último acceso: 3 jun. 2025].

[13] L. Kovanič, B. Topitzer, P. Peťovský, and et al, "Review of photogrammetric and lidar applications of uav," *Applied Sciences*, vol. 13, no. 11, 2023. [En línea]. Disponible en: https://www.mdpi.com/2076-3417/13/11/6732 [Último acceso: 3 jun. 2025].

[14] S. Manfreda, M. Francis, P. E. Miller, and R. Lucas, "On the use of unmanned aerial systems for environmental monitoring," *Environmental Monitoring Journal*, 2024. [En línea]. Disponible en: https://www.researchgate.net/publication/323755402_On_the_Use_of_Unmanned_Aerial_Systems_for_Environmental_Monitoring [Último acceso: 4 jun. 2025].

[15] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012.

[16] M. Ángel, M. Pérez, and C. Puentes, "Mecánica de vuelo ii," 2025. Apuntes docentes, Universitat Politècnica de Catalunya, Escuela Superior de Ingenierías Industrial, Aeroespacial y Audiovisual de Terrassa.

[17] A. Patankar and B. Smith, "Modeling and nonlinear control of a quadcopter for stabilization," *Journal of Aerospace Engineering*, 2022. [En línea]. Dispo-

nible en: https://onlinelibrary.wiley.com/doi/10.1155/2022/2449901 [Último acceso: 10 jun. 2025].

[18] MathWorks, "Understanding gimbal lock," 2025. [En línea]. Disponible en: https://es.mathworks.com/help/aerotbx/ug/dcm2angle.html?searchHighlight=gimbal+lock&s_tid=srchtitle_support_results_2_gimbal+lock [Último acceso: 2 jun. 2025].

[19] Alexander T. Miller and Anil V. Rao, "Nonsingular euler parameterizations for motion of a point mass in atmospheric flight." Preprint, 2021. [En línea]. Disponible en: https://arxiv.org/abs/2104.08972 [Último acceso: 31 may. 2025].

[20] G. F. Franklin and J. David Powell, *Feedback Control of Dynamic Systems*. Pearson, 7th ed., 2015. [En línea]. Disponible en: https://www.daslhub.org/unlv/courses/00coursesUsb/labviewCourseDevelopment/labview-0X-LeadLag/Franklin%20PE%206th%20-%20textbook%20but%20perhaps%20too%20much%20-%20no%20simple%20Lag%20control%20example.pdf [Último acceso: 3 jun. 2025].

[21] MathWorks, "c2d — convert model from continuous to discrete time." Documentación en línea, Control System Toolbox, 2025. [En línea]. Disponible en: https://www.mathworks.com/help/control/ref/dynamicsystem.c2d.html [Último acceso: 17 may. 2025].

[22] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[23] L. Martin, C. Cardeira, and P. Oliveira, "Linear quadratic regulator for trajectory tracking of a quadrotor," *Control Engineering Practice*, 2019. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896319311450 [Último acceso: 25 abr. 2025].

[24] P. Saraf, M. Gupta, and A. Manga Parimi, "A comparative study between a classical and optimal controller for a quadrotor," 2020. [En línea]. Disponible en: https://arxiv.org/abs/2009.13175 [Último acceso: 19 abr. 2025].

[25] K. Ogata, *Discrete-Time Control Systems*. Prentice Hall, 2nd ed., 2010.

[26] MathWorks, "dlqr – discrete-time linear-quadratic regulator design." Documentación en línea, Control System Toolbox, 2025. [En línea]. Disponible en: https://www.mathworks.com/help/control/ref/dlqr.html [Último acceso: 29 may. 2025].

[27] R. Mahony, T. Hamel, and J. Pflimlin, "Nonlinear complementary filters on the special orthogonal group," *IEEE Transactions on Automatic Control*, vol. 53, no. 5, pp. 1203–1218, 2008. [En línea]. Disponible en: https://doi.org/10.1109/TAC.2008.923738 [Último acceso: 18 may. 2025].

[28] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[29] S. Rauniyar, S. Bhalla, D. Choi, and D. Kim, "Ekf-slam for quadcopter using differential flatness-based lqr control," *Electronics*, vol. 12, no. 5, p. 1113, 2023. [En línea]. Disponible en: https://doi.org/10.3390/electronics12051113 [Último acceso: 28 may. 2025].

[30] J. O. Smith, *Introduction to Digital Filters with Audio Applications*. W3K Publishing, 2003. [En línea]. Disponible en: https://ccrma.stanford.edu/~jos/filters/ [Último acceso: 25 may. 2025].

[31] A. Martinelli, "Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 61–76, 2012. [En línea]. Disponible en: https://doi.org/10.1109/TRO.2011.2160468 [Último acceso: 27 may. 2025].

[32] MathWorks, "Px4 uorb read (simulink, uav toolbox support package for px4 autopilots)." Documentación en línea, MATLAB & Simulink Help, 2025. [En línea]. Disponible en: https://www.mathworks.com/help/uav/px4/ref/px4uorbread.html [Último acceso: 1 jun. 2025].

[33] ArduPilot Dev Team, "Safety switch — copter documentation." Documentación en línea, ArduPilot, 2024. [En línea]. Disponible en: https://ardupilot.org/copter/docs/common-safety-switch-pixhawk.html [Último acceso: 1 jun. 2025].

[34] MathWorks, "I2c controller read (simulink, uav toolbox support package for px4 autopilots)." Documentación en línea, MATLAB & Simulink Help, 2025. [En línea]. Disponible en: https://www.mathworks.com/help/uav/px4/ref/i2ccontrollerread.html [Último acceso: 5 jun. 2025].

[35] MathWorks, "Discrete-time integrator (simulink)." Documentación en línea, MATLAB and Simulink Help, 2025. [En línea]. Disponible en: https://www.mathworks.com/help/simulink/slref/discretetimeintegrator.html [Último acceso: 5 jun. 2025].

[36] MathWorks, "Datastoreread and datastorewrite (simulink)." Documentación en línea, MATLAB and Simulink Help, 2025. [En línea]. Disponible en: https://es.mathworks.com/help/simulink/ug/data-store-basics.html [Último acceso: 5 jun. 2025].

[37] Physics LibreTexts, "Rotating reference frames," 2020. [En línea]. Disponible en: https://phys.libretexts.org/Bookshelves/University_Physics/Mechanics_and_Relativity_(Idema) [Último acceso: 7 jun. 2025].

[38] Y. Chen, D. Baek, A. Bocca, and et al, "A case for a battery-aware model of drone energy consumption," *IEEE Transactions on Transportation Electrification*, vol. 6, no. 4, pp. 1472–1481, 2020. [En línea]. Disponible en: https://www.researchgate.net/profile/Yukai-Chen/publication/330477358_A_Case_for_a_Battery-Aware_Model_of_Drone_Energy_Consumption/links/5c424ed192851c22a37fa647/A-Case-for-a-Battery-Aware-Model-of-Drone-Energy-Consumption.pdf [Último acceso: 12 jun. 2025].

[39] PX4 Autopilot, "Ulog file format," 2025. [En línea]. Disponible en: https://docs.px4.io/main/en/dev_log/ulog_file_format.html [Último acceso: 22 jun. 2025].

A. Puesta en marcha del sistema y entorno de desarrollo

Este anexo documenta de forma detallada todos los procedimientos necesarios para replicar la instalación del entorno de desarrollo, la configuración del sistema de control, y la conexión con el hardware utilizado en el proyecto. El objetivo es facilitar que futuros estudiantes puedan continuar este trabajo sin partir desde cero.

A.1. Requisitos previos

- Sistema operativo: **Windows 10 o Windows 11 / Ubuntu 22.04**
- Cuenta activa en MathWorks para acceder a MATLAB
- Acceso físico al hardware (Cube Orange+, emisora, sensores...)

A.2. Instalación de MATLAB 2024b

1. Descargar MATLAB desde `https://www.mathworks.com/downloads/`
2. Instalar con licencia académica de la Universidad.
3. Durante la instalación, incluir Simulink, UAV Toolbox y UAV Toolbox Support Package for PX4 Autopilots.

De tener previamente Matlab 2024b instalado, solo se debe descargar el UAV Toolbox Support Package for PX4 Autopilots desde la pestaña "Get Add-Ons".

A.3. Instalación del soporte para PX4

1. Abrir MATLAB y clickar en: Add-Ons ->Manage Add-Ons.

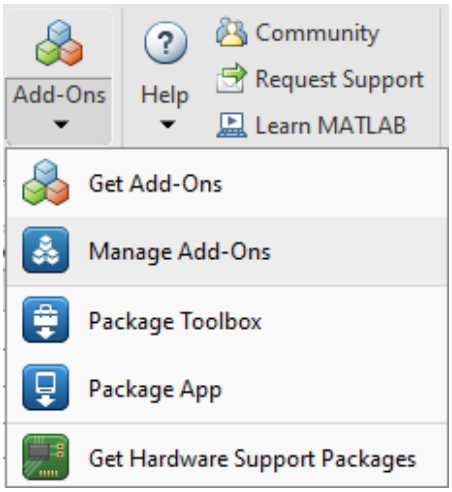

Figura A.1: Manage Add Ons

2. Abrir el setup (símbolo del engranaje) del UAV Toolbox Support Package for PX4 Autopilots

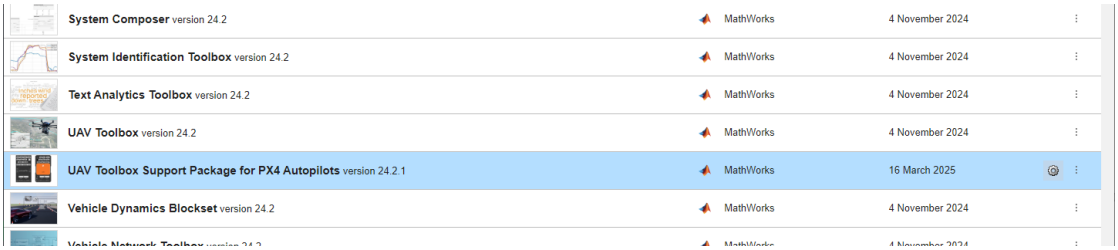

Figura A.2: Setup UAV Toolbox Support Package for PX4 Autopilots

3. Seguir el asistente para instalar todo.

Paso 1: Instalar el subsistema Linux WSL2
En primer lugar, si su sistema operativo es Windows, aparecerá esta pestaña:

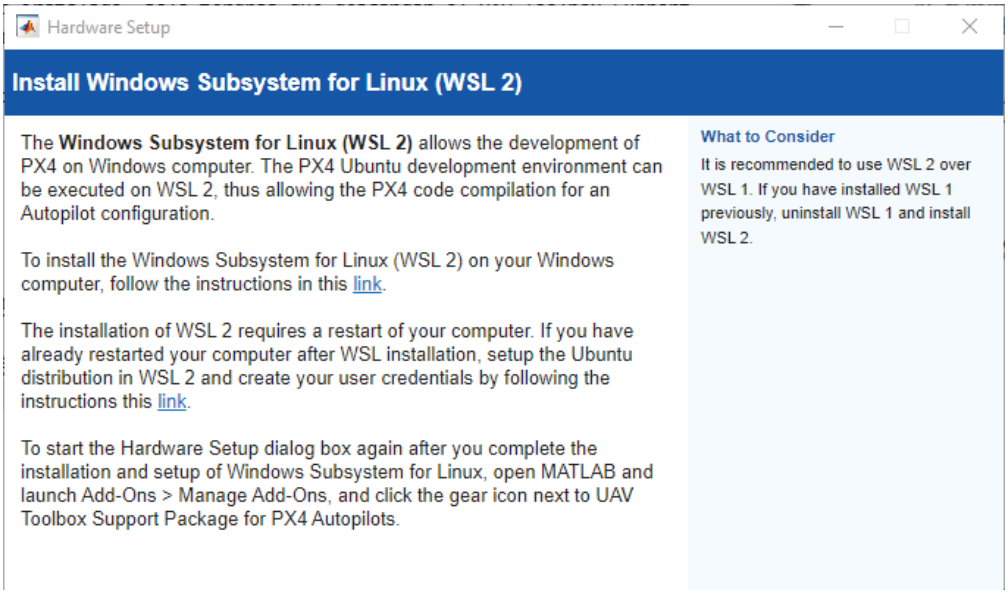

Figura A.3: Instalación de Linux WSL 2

Es necesario seguir el enlace indicado en la ventana, que redirige a la documentación oficial para instalar el Subsistema de Windows para Linux (WSL 2). Si su equipo ya tenía WSL 1 instalado, debe desinstalarlo antes de proceder.

PRIMER LINK

El primer link sirve para Instalar el subsistema Linux WSL2. Siga todos los pasos y una vez instalado, es necesario reiniciar el ordenador.

Para instalar correctamente la distribución de Ubuntu 22.04, es necesario abrir una ventana de terminal (PowerShell o CMD) con permisos de administrador y ejecutar el siguiente comando:

```
wsl --install -d Ubuntu-22.04
```

Una vez completada la instalación de Ubuntu 22.04, es necesario reiniciar el ordenador para que se apliquen correctamente los cambios del Subsistema de Windows para Linux (WSL 2).

La primera vez que se inicie Ubuntu 22.04 (ya sea automáticamente tras reiniciar o manualmente desde el menú de inicio), aparecerá una ventana como la mostrada en la Figura A.4, indicando que se está completando la instalación. Este proceso puede tardar unos minutos.

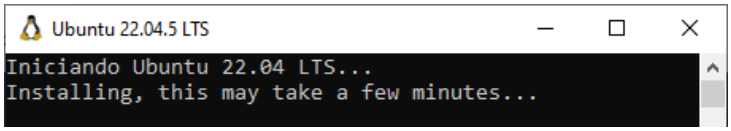

Figura A.4: Espera unos minutoss

Tras reiniciar, se debe abrir de nuevo una terminal (CMD o PowerShell) y establecer Ubuntu 22.04 como la distribución por defecto mediante el siguiente comando:

```
wsl --set-default Ubuntu-22.04
```

Este paso asegura que MATLAB utilice la versión de Ubuntu recién instalada como entorno principal para compilar el firmware PX4. Si no se realiza, pueden producirse errores durante la ejecución del asistente de configuración.

SEGUNDO LINK

Una vez instalado el Subsistema de Windows para Linux (WSL 2) y reiniciado el equipo, se debe instalar la distribución Ubuntu desde la Microsoft Store y crear un usuario, los pasos a seguir están en el segundo link. Esta distribución será utilizada por MATLAB para compilar el firmware PX4.

Durante la primera ejecución de Ubuntu, el sistema solicitará la creación de un nombre de usuario y una contraseña. Este usuario se utilizará como cuenta principal dentro del entorno WSL.

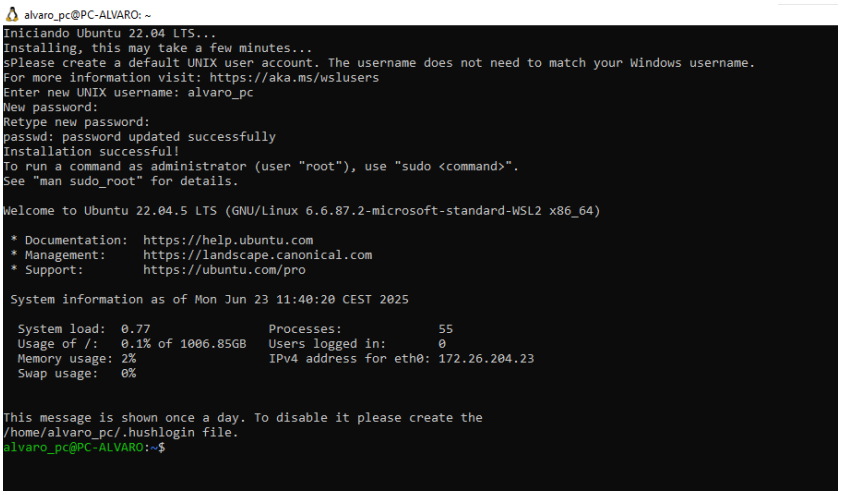

Figura A.5: Primera inicialización de Ubuntu 22.04 en WSL.

Una vez terminado, podrá escribir en la barra de tareas de búsqueda “Ubuntu” y le aparecerá la aplicación.

Paso 2: instalación de Python 3.8.2

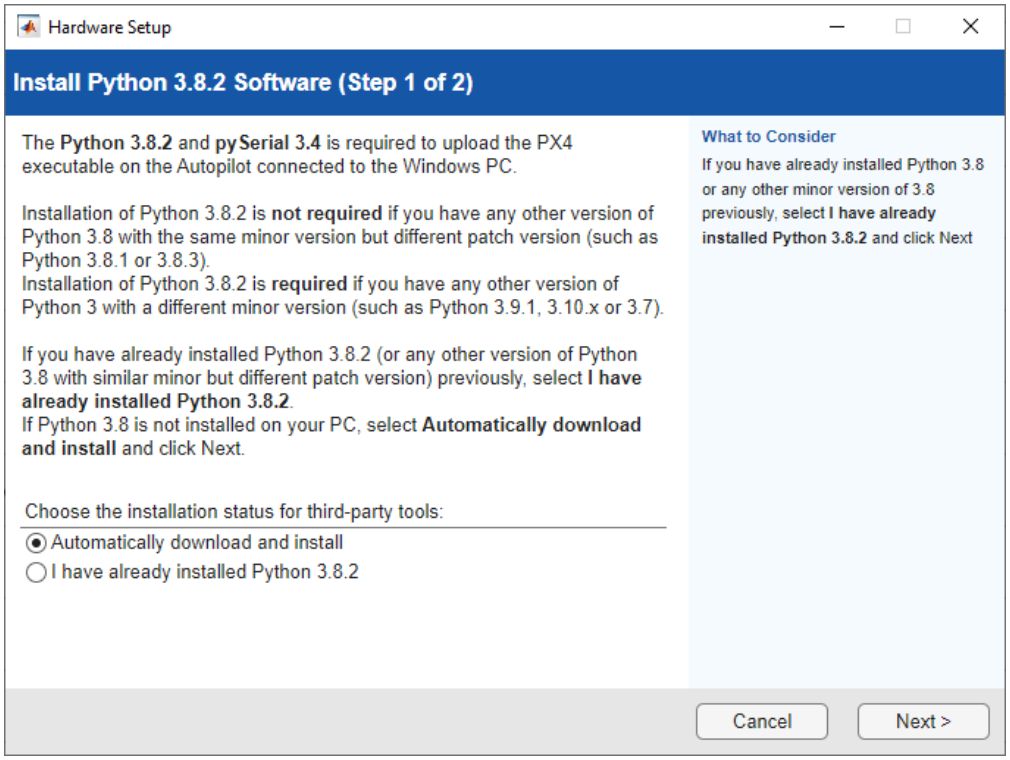

Figura A.6: Instalación de Python 3.8.2 y pySerial 3.4

Para posibilitar la carga del firmware PX4 en el autopiloto desde Windows, es necesario disponer de una versión compatible de Python 3.8.2 y del paquete `pySerial 3.4`.

- Si el sistema ya tiene instalada alguna versión de Python 3.8 con el mismo número de versión menor (por ejemplo, 3.8.1 o 3.8.3), no es necesario instalar Python 3.8.2 exactamente.
- Si el sistema tiene una versión distinta de Python (por ejemplo, 3.7, 3.9, 3.10...), sí será necesario instalar exactamente Python 3.8.2.

Se puede elegir una de las siguientes opciones:

- **Automatically download and install**: el asistente descargará automáticamente Python 3.8.2 y pySerial 3.4.
- **I have already installed Python 3.8.2**: si ya está instalado previamente.

Paso 3: Confirmación de instalación de Python 3.8.2 y pySerial 3.4

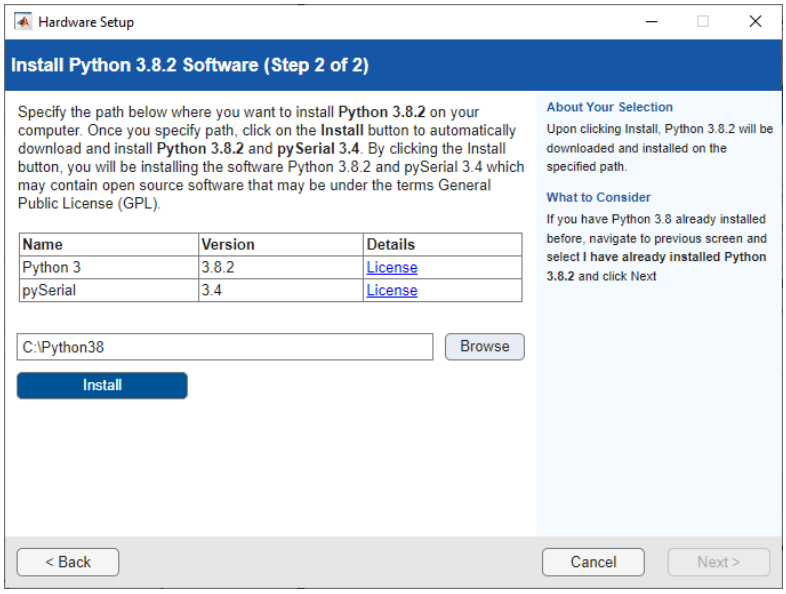

Figura A.7: Selección de ruta para instalar Python 3.8.2 y pySerial 3.4

En esta ventana se especifica la ruta donde se instalarán Python 3.8.2 y pySerial 3.4. Por defecto, la ruta sugerida es `C:\Python38`.

Para continuar, basta con hacer clic en el botón **Install**. El asistente descargará automáticamente las herramientas necesarias y las instalará en el sistema.

Se recomienda no modificar la ruta por defecto salvo que exista un motivo específico (por ejemplo, conflictos con otra instalación de Python).

Una vez completada la instalación, el asistente mostrará un mensaje de confirmación indicando que tanto **Python 3.8.2** como **pySerial 3.4** se han instalado correctamente, como se muestra en la Figura A.8.

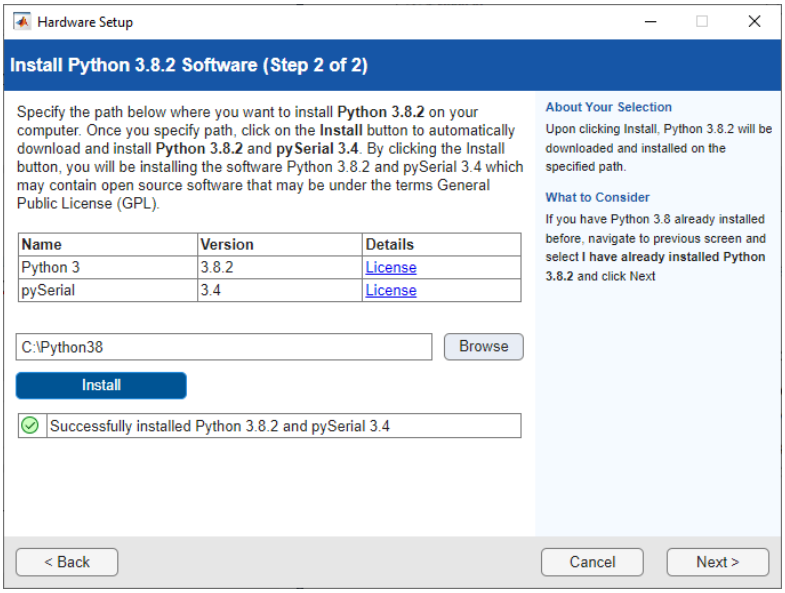

Figura A.8: Confirmación de instalación exitosa de Python 3.8.2 y pySerial 3.4

Para comprobar que la instalación se ha realizado correctamente, se recomienda abrir una ventana de terminal (CMD) y ejecutar los siguientes comandos:

```
python --version
```

Este comando debería devolver:

```
Python 3.8.2
```

Asimismo, es conveniente comprobar que el paquete `pySerial` está instalado correctamente con el siguiente comando:

```
pip list
```

En la lista de paquetes instalados, deberá aparecer una línea similar a:

```
pyserial 3.4
```

Figura A.9: Verificación de instalación correcta de Python 3.8 y de pyserial 3.4

Si al ejecutar

```
python -version
```

aparece un mensaje de error como el mostrado en la Figura A.10, es necesario añadir manualmente la ruta `C:\Python38` a las variables de entorno del sistema.

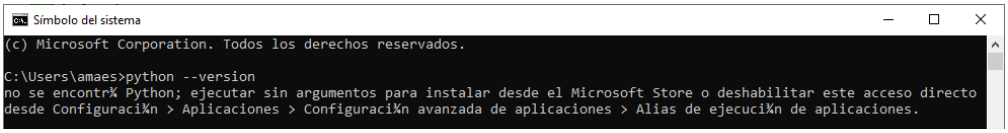

Figura A.10: Ejemplo de error al ejecutar `python -version`

Si alguno de estos comandos no devuelve el resultado esperado, asegúrese de que la ruta de instalación (por ejemplo, `C:\Python38`) se ha añadido correctamente a las variables de entorno del sistema. Para ello, abra el menú de inicio, escriba *"variables de entorno"*, y edite la variable **Path** del entorno de usuario o del sistema. Añada la carpeta donde se haya instalado Python, por ejemplo:

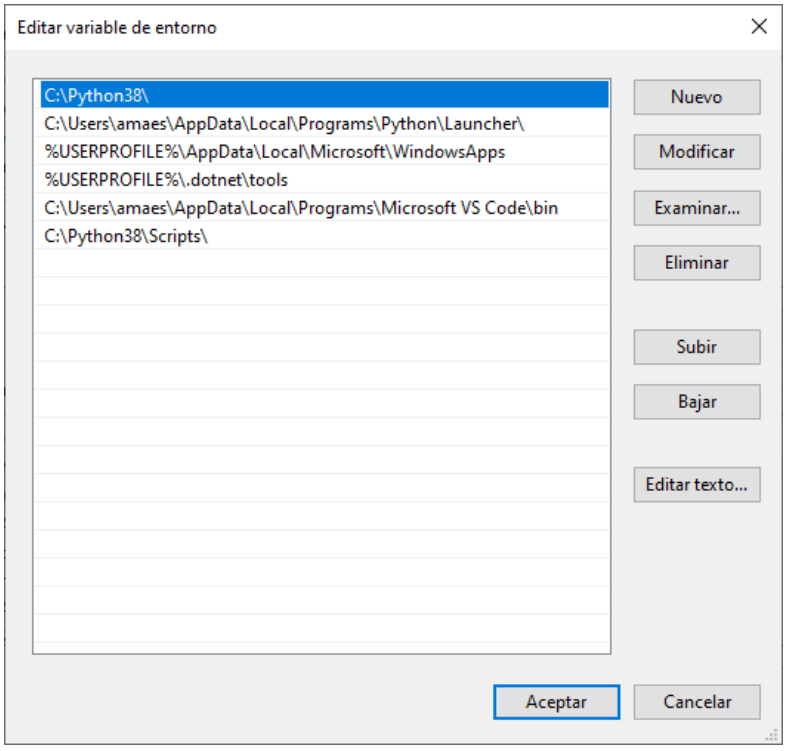

Figura A.11: Adición de la carpeta de instalación de Python al `Path` del sistema.

Paso 4: Descarga del código fuente de PX4

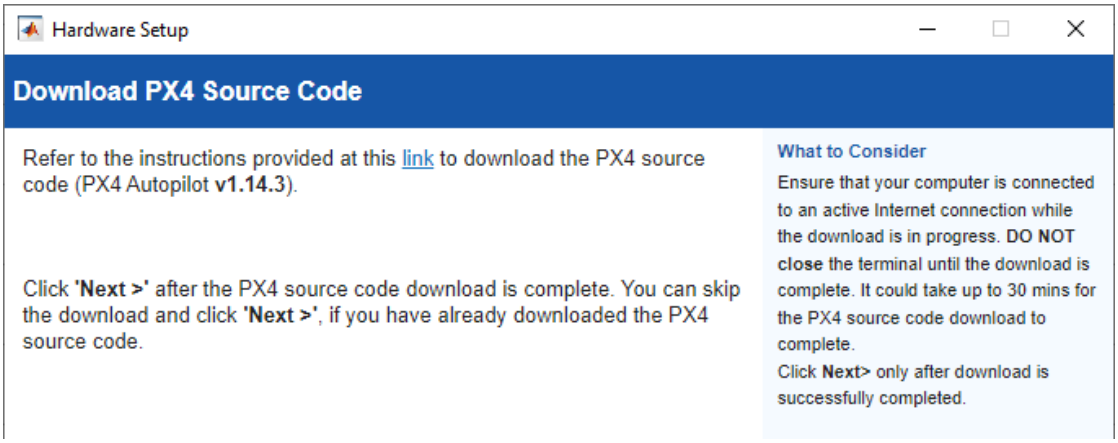

Figura A.12: Descarga del Código fuente de PX4

Para continuar con la instalación, es necesario descargar el código fuente del firmware PX4 (versión **v1.14.3**). Este proceso debe realizarse en el entorno WSL2 para evitar errores de permisos y problemas de rendimiento.

Los pasos a seguir son:

1. Abrir el terminal de Ubuntu (WSL2).
2. Navegar al directorio home con:

   ```
   cd ~
   ```

3. Clonar el repositorio de PX4 desde GitHub:

   ```
   git clone https://github.com/PX4/PX4-Autopilot.git --recursive
   ```

4. Acceder al directorio descargado:

   ```
   cd PX4-Autopilot
   ```

5. Cambiar a la versión estable v1.14.3:

   ```
   git checkout v1.14.3 -f
   ```

6. Inicializar todos los submódulos del repositorio:

   ```
   git submodule update --init --recursive
   ```

Este proceso puede tardar varios minutos. Una vez completado, se puede continuar con los siguientes pasos del asistente de configuración.

Paso 5: Validación del código fuente de PX4

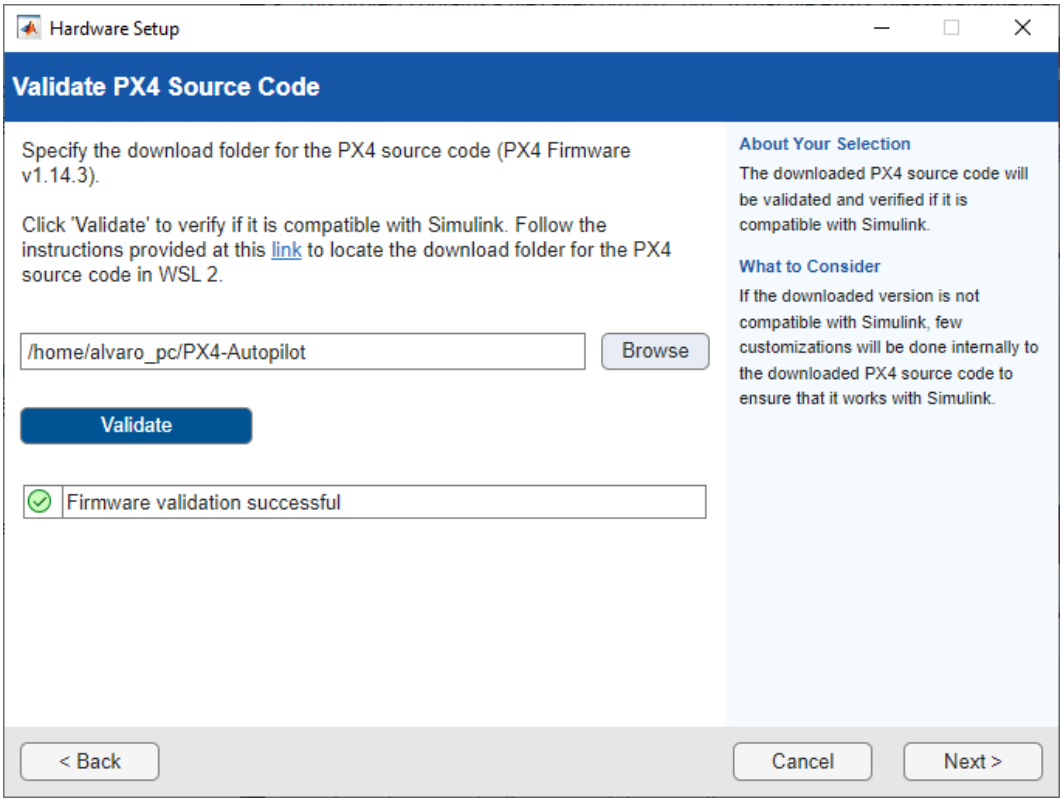

Figura A.13: Validación del código fuente de PX4 desde el asistente de instalación.

Una vez descargado el repositorio PX4 en el entorno WSL2, es necesario validar que la versión es compatible con Simulink. Para ello:

1. En la ventana del asistente, introducir la ruta completa al repositorio, por ejemplo:

   `/home/alvaro_pc/PX4-Autopilot`

2. Pulsar el botón **Validate**.

Si la validación es correcta, aparecerá el mensaje *Firmware validation successful*. Esto confirma que el repositorio contiene una versión adecuada del firmware (v1.14.3) y que puede ser utilizado con el entorno de desarrollo de Simulink.

Paso 6: instalación del entorno de desarrollo (toolchain) de PX4 en WSL2

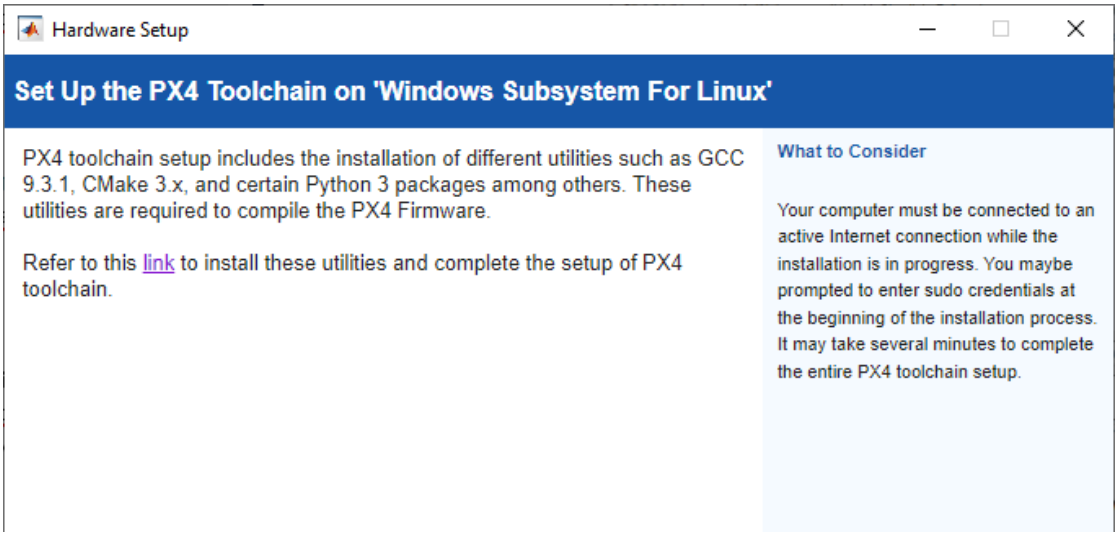

Figura A.14: Instalación del entorno de desarrollo (PX4 Toolchain)

Una vez validado el código fuente del firmware, el asistente solicita instalar el entorno de desarrollo necesario para compilar PX4. Este entorno incluye herramientas como `GCC 9.3.1`, `CMake 3.x`, `Ninja`, `Git` y varios paquetes de Python.

Para ello, es necesario seguir las instrucciones indicadas en el enlace proporcionado, que puede resumirse en los siguientes pasos:

1. Abrir una terminal WSL2.
2. Navegar a la carpeta donde se descargó el firmware:

   ```
   cd ~/PX4-Autopilot
   ```

3. Ir a la carpeta del script de instalación:

   ```
   cd Tools/setup
   ```

4. Ejecutar el script de instalación del toolchain:

   ```
   bash ./ubuntu.sh
   ```

5. Cuando el proceso haya terminado, reiniciar el sistema

Paso 6: desactivar controladores por defecto de PX4

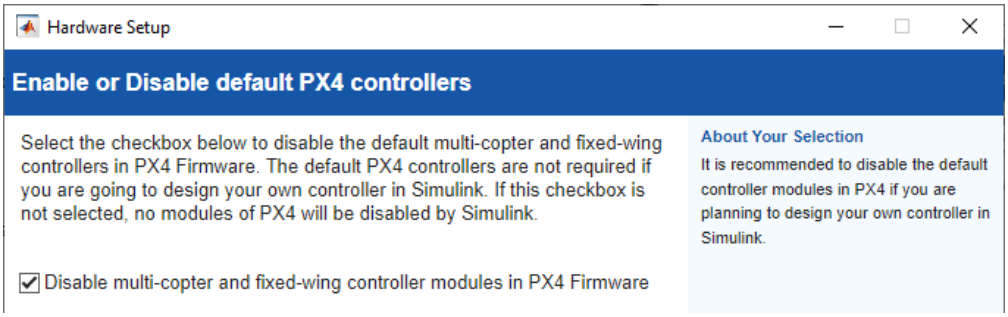

Figura A.15: Desactivación de controladores por defecto de PX4

Marca la casilla para desactivar los controladores por defecto de PX4. Es recomendable si vas a usar tu propio controlador en Simulink.

Paso 7: seleccionar el autopiloto Cube Orange+

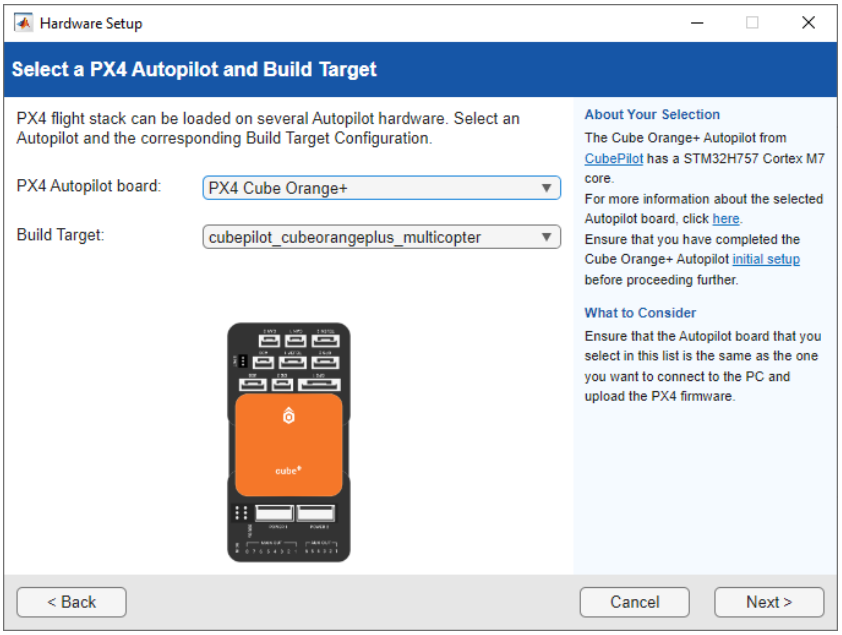

Figura A.16: Selección del autopiloto Cube Orange+ y el Build Target

Seleccionar:

- **PX4 Autopilot board:** `PX4 Cube Orange+`
- **Build Target:** `cubepilot_cubeorangeplus_multicopter`

Paso 8: seleccionar script de inicio del sistema

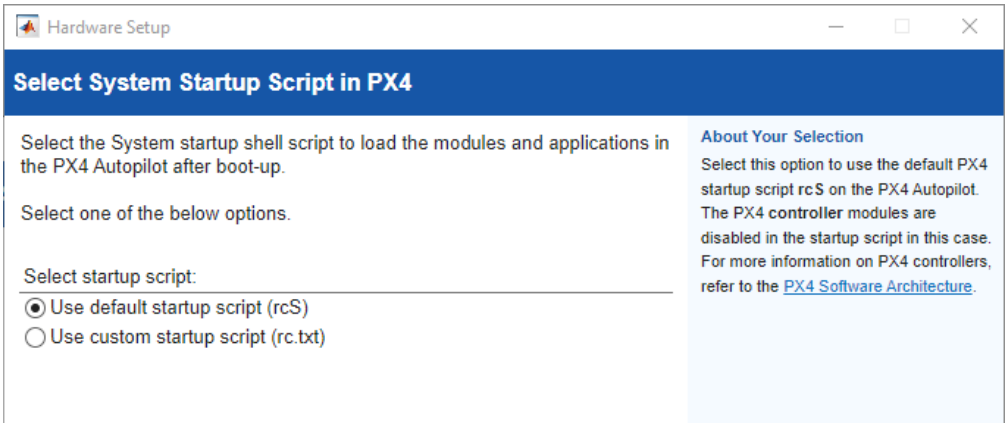

Figura A.17: Selección del script de arranque por defecto en PX4

Marcar la opción:

- `Use default startup script (rcS)`

Paso 9: instalación de QGroundControl

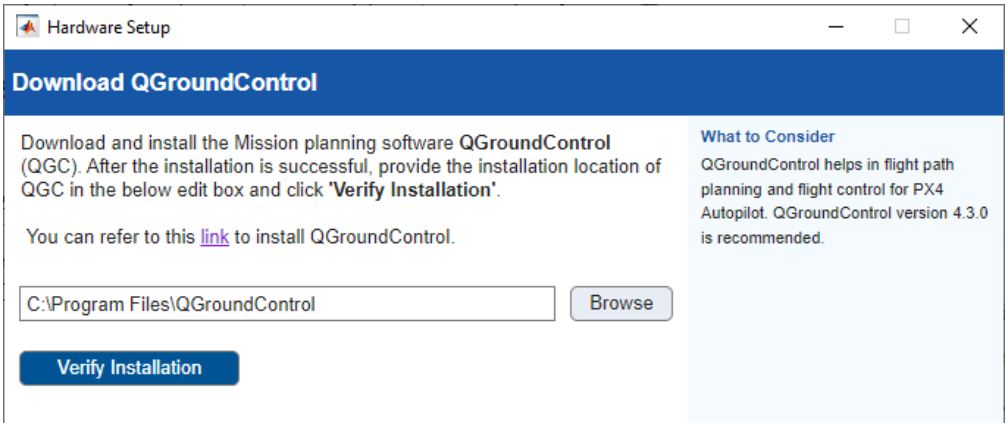

Figura A.18: Instalación de QGroundControl

QGroundControl (QGC) es el software de planificación de misiones y control de vuelo compatible con PX4. Para instalarlo:

1. Acceder al repositorio oficial (click en link)

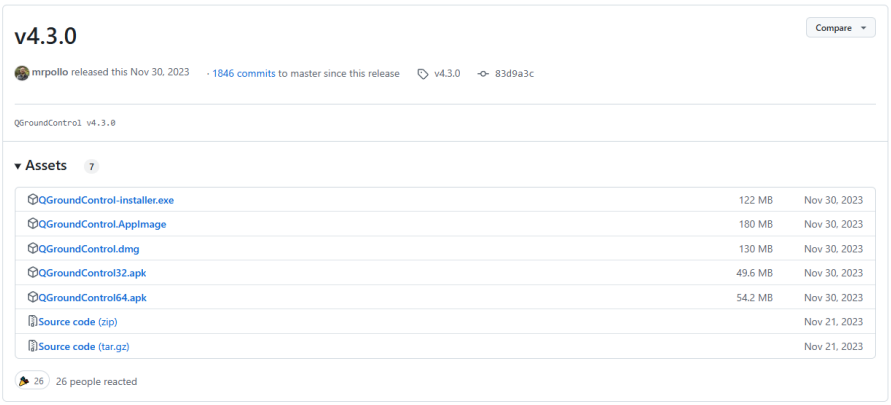

Figura A.19: Repositorio oficial de QGroundControl (v4.3.0) en GitHub

2. Descargar el archivo `QGroundControl-installer.exe`.
3. Ejecutar el instalador y seguir los pasos.

   Durante el proceso de instalación de QGroundControl, el sistema solicita instalar ciertos *drivers* de dispositivos USB, como los proporcionados por *Arduino LLC*. Estos controladores son necesarios para establecer la comunicación entre el ordenador y el autopiloto (por ejemplo, el Cube Orange+), ya que permiten reconocer correctamente el dispositivo cuando se conecta por puerto USB.

Tras la instalación, se debe verificar que el instalador se ha ubicado correctamente. Por defecto, QGroundControl se instala en:

`C:\Program Files\QGroundControl`

Una vez seleccionado el directorio de instalación (Browse), hacer clic en **Verify Installation**. Si la instalación es válida, se permitirá continuar con la configuración.

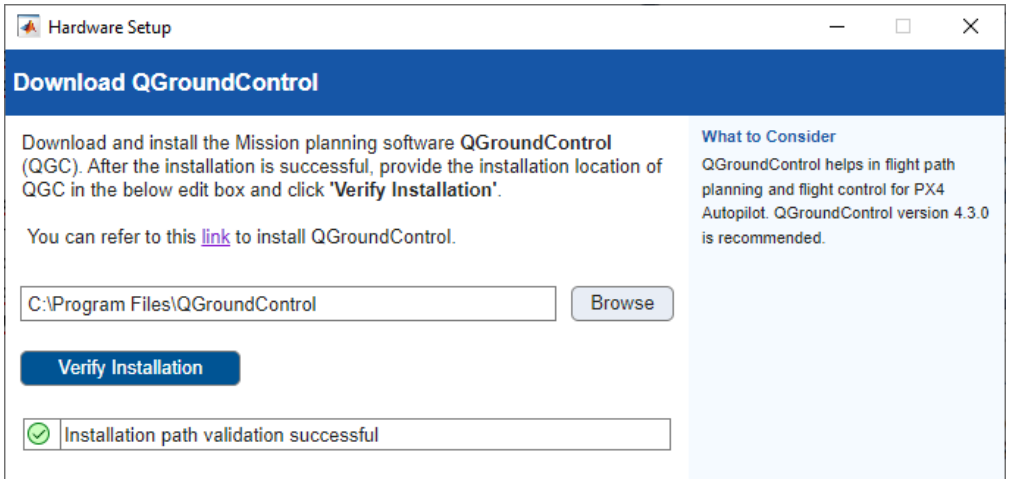

Figura A.20: Verificación de QGroundControl

Paso 10: Seleccionar el *Airframe* en QGroundControl

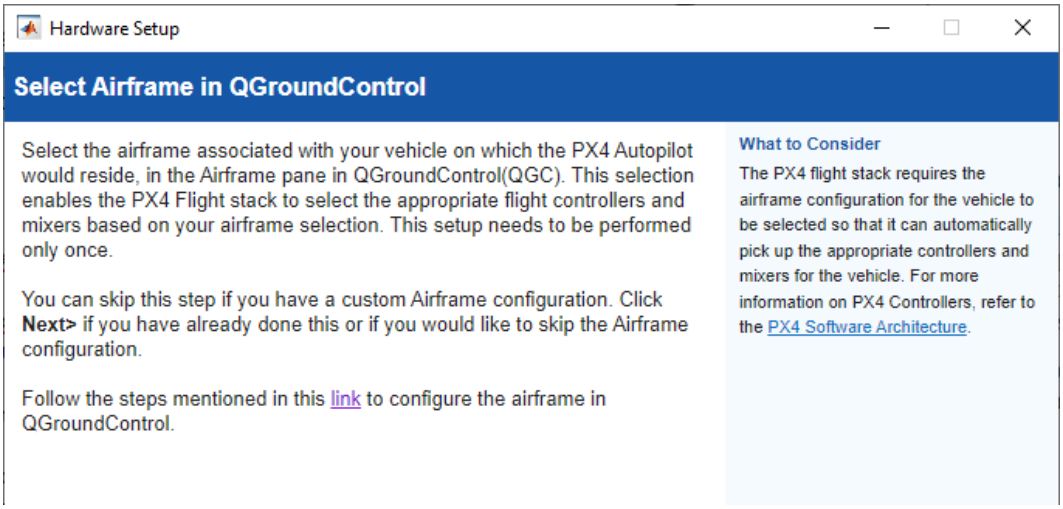

Figura A.21: Verificación de QGroundControl

Para completar la configuración del sistema PX4, es necesario especificar la geometría del vehículo que se está utilizando. Esta configuración permite que PX4 cargue los controladores y mezcladores adecuados para el tipo de dron. En este caso, se seleccionó un cuadricóptero en configuración X. Para ello, se siguieron los siguientes pasos en QGroundControl:

1. Iniciar **QGroundControl** y conectar el vehículo.
2. Desde QGroundControl, navegar a `Vehicle Setup >Airframe` (en la barra lateral) para abrir el panel de configuración de *Airframe*.
3. Dentro del grupo **Quadrotor X**, seleccionar la opción **Generic Quadrotor X geometry**, que corresponde a un cuadricóptero en configuración X.
4. Hacer clic en el botón **Apply and Restart**, situado en la parte superior derecha de la pantalla.
5. En el aviso emergente, confirmar pulsando **Apply** para guardar la configuración y reiniciar el vehículo.

En este proyecto se seleccionó la opción **Quadcopter en configuración "X"**, por ser la que mejor se adapta a la disposición física del dron.

Paso 11: Compilación del firmware PX4

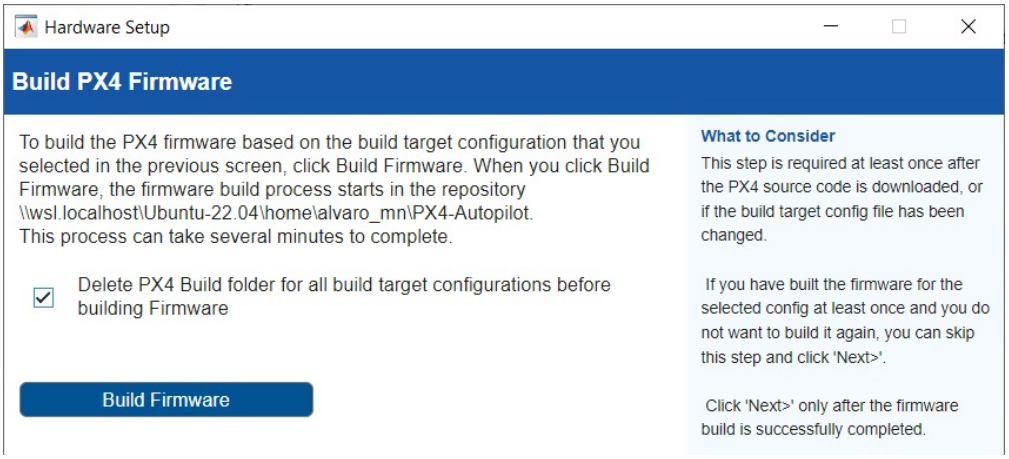

Figura A.22: Compilación exitosa del firmware PX4.

Una vez seleccionada la configuración del hardware y el airframe correspondiente, se procede a compilar el firmware de PX4 desde el asistente de configuración de hardware de Simulink.

Para ello, se marca la opción `Delete PX4 Build folder for all build target configurations before building Firmware` y se pulsa el botón **Build Firmware**. Este paso puede tardar varios minutos, ya que se compilan todos los módulos necesarios.

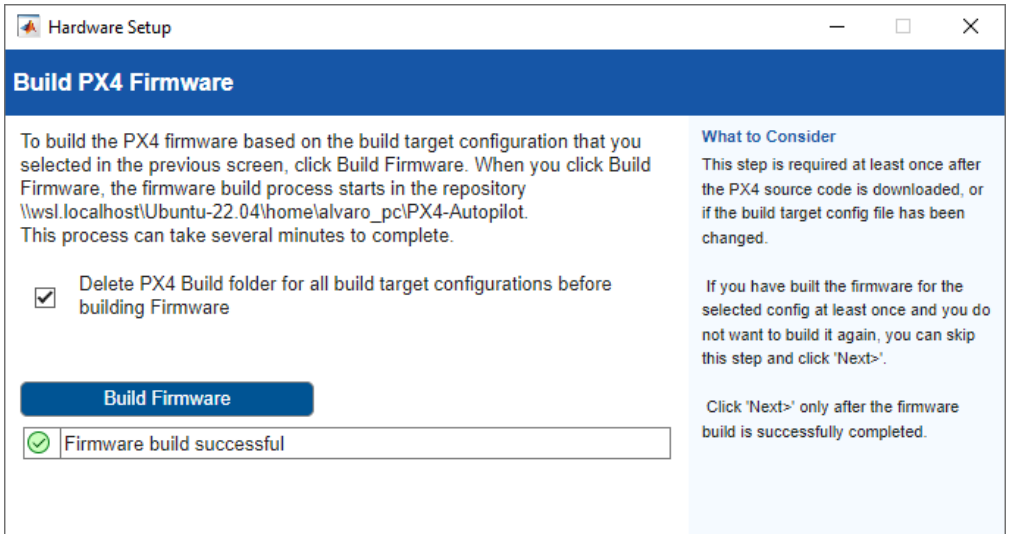

Figura A.23: Compilación exitosa del firmware PX4.

Una vez finalizado, debe aparecer el mensaje `Firmware build successful`. En ese caso, se puede continuar al siguiente paso.

Paso 12: Test de conexión

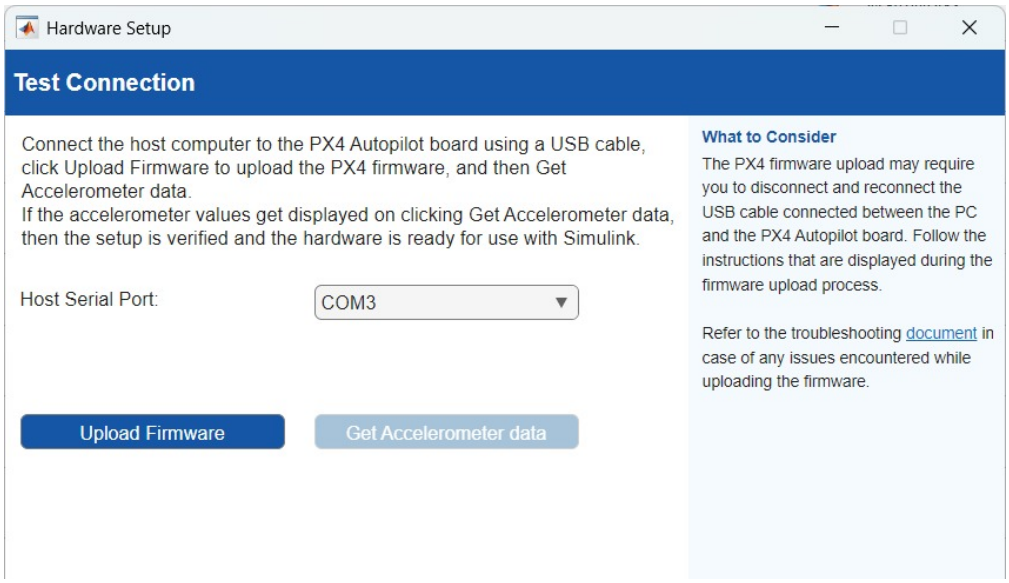

Figura A.24: Test de Conexión

Seleccione el puerto al que está conectado el Cube Orange+ (se puede confirmar en administrador de dispositivos) y pulse en "Upload Firmware". Le aparecerá la siguiente ventana:

Figura A.25: Desconectar y Conectar el cable USB

Desconecte el cable USB, pulse OK y vuelva a conectar. Tras breves minutos, si la conexión ha sido exitosa, le aparecerá el mensaje: "Firmware uploaded successfully".

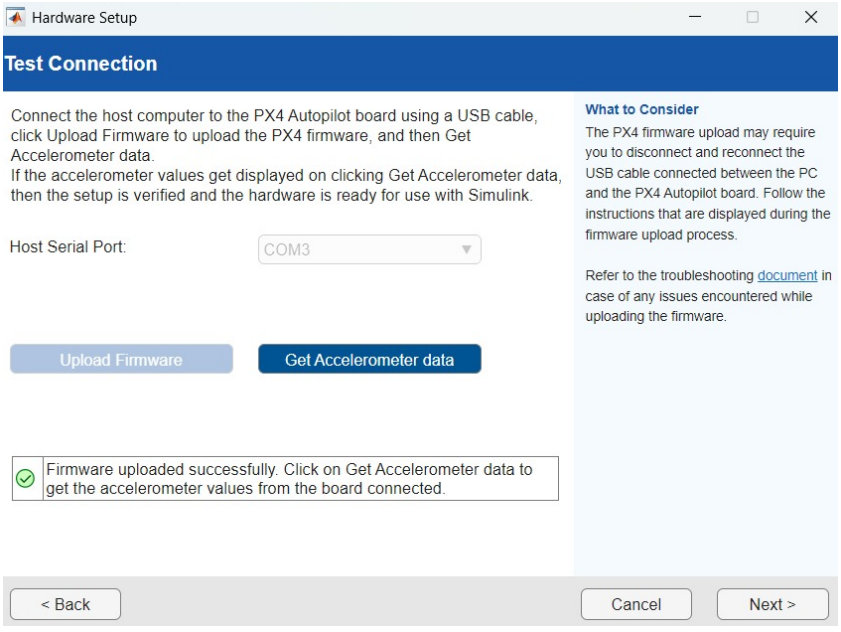

Figura A.26: validación Test de Conexión

Paso 13: Hardware Setup Complete

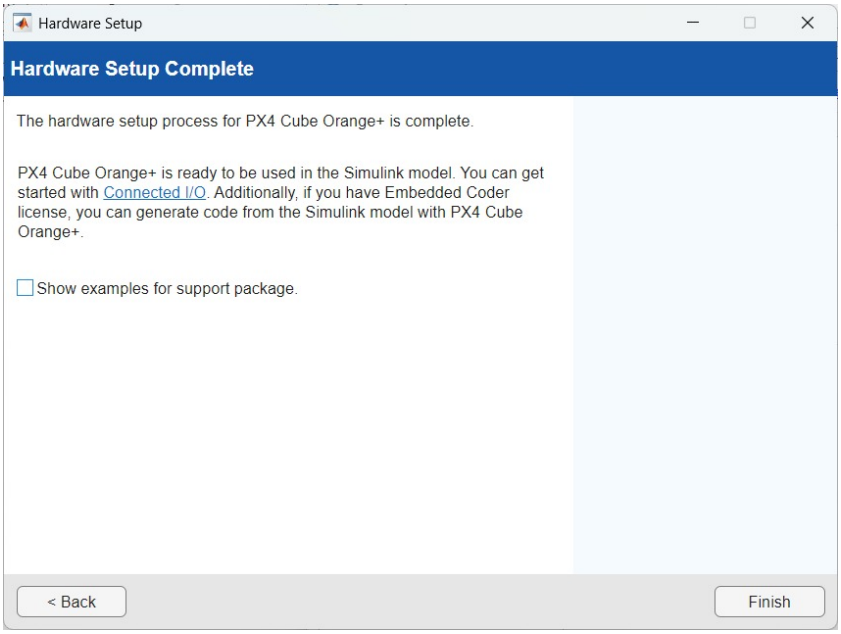

Figura A.27: Hardware setup completado

Haga click en Finish. Enhorabuena, has terminado de configurar el hardware.

A.4. Configuración en Simulink

Una vez compilado correctamente el firmware, es necesario configurar el modelo `UAV_CONTROL_SYSTEM` en Simulink para establecer correctamente los parámetros de hardware y generación de código. A continuación se enumeran los pasos seguidos:

1. Abrir el modelo `UAV_CONTROL_SYSTEM` en Simulink.
2. Acceder a `Hardware Settings` mediante el menú: `Hardware >Hardware Settings`. En la sección `Hardware Implementation`, se seleccionan los siguientes parámetros:
   - **Hardware Board:** `PX4 Cube Orange+`
   - En `Target hardware resources`, dentro del grupo `Build options`, marcar la opción:

- `Automatically determine serial port for firmware upload`

- En el grupo `MAVLink`, habilitar:
  - `Enable MAVLink on /dev/ttyACM0`

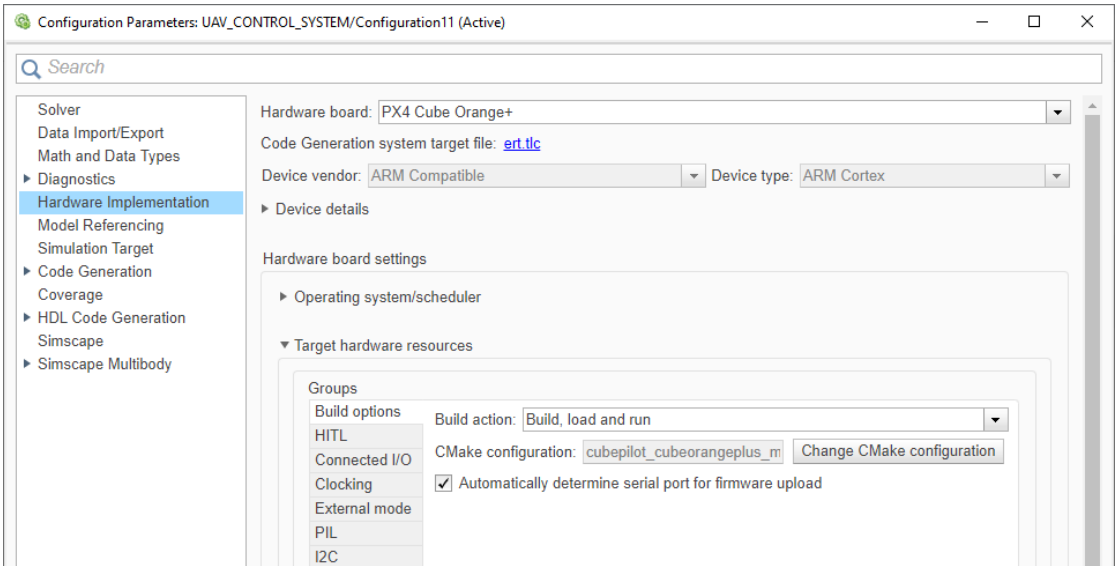

Figura A.28: Opciones de compilación seleccionadas

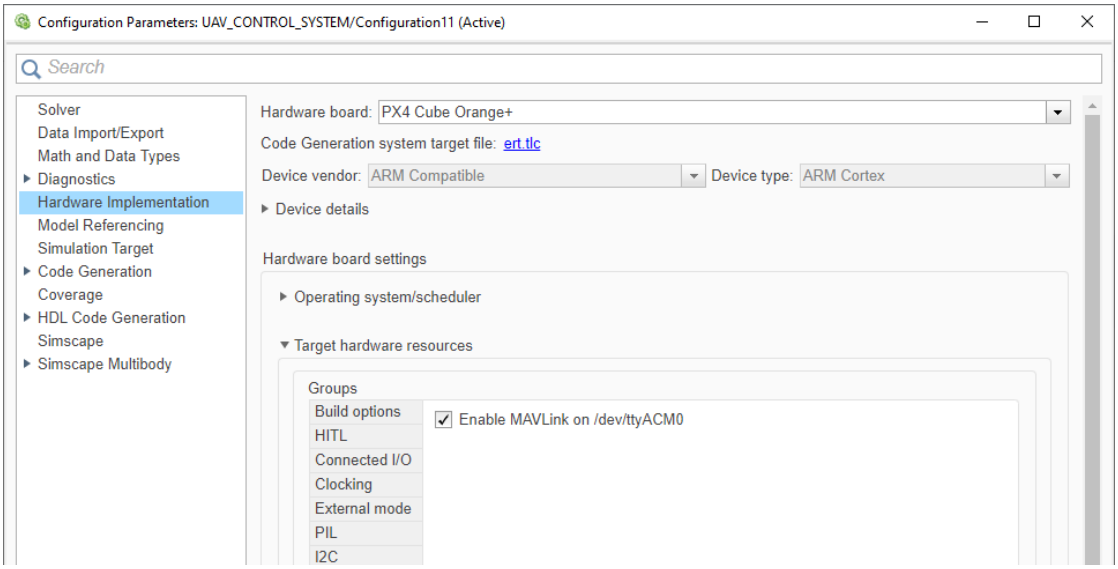

Figura A.29: Habilitación de MAVLink en `/dev/ttyACM0`

3. Finalmente, en la pestaña `Code Generation >Custom Code`, se deben realizar las siguientes modificaciones:

   - En `Code Information` añadir la cabecera:

```
#include <poll.h>
```

Código A.1: Incluir cabecera para compilación

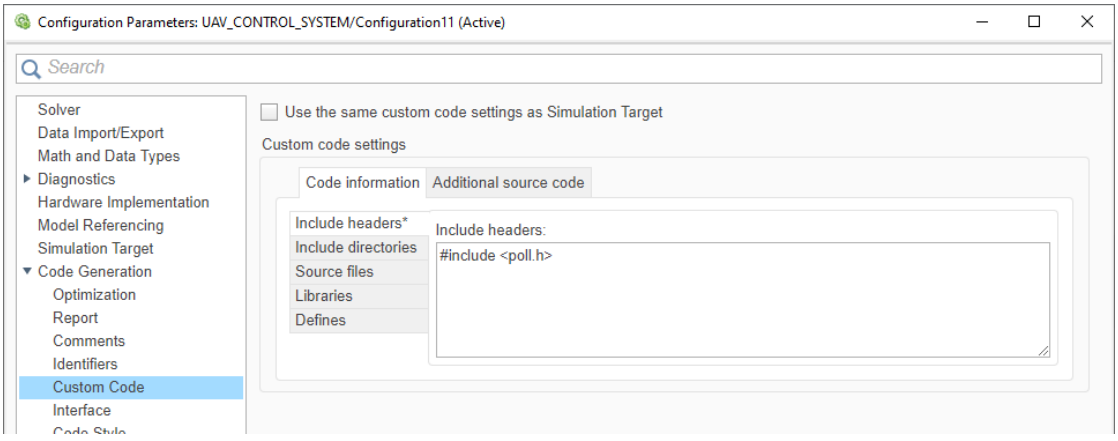

Figura A.30: Incluir cabecera `poll.h`

- En `>Additional source code` incluir la definición del tipo `pollfd`:

```
typedef struct pollfd pollfd_t;
```

Código A.2: Inicialización del tipo pollfd

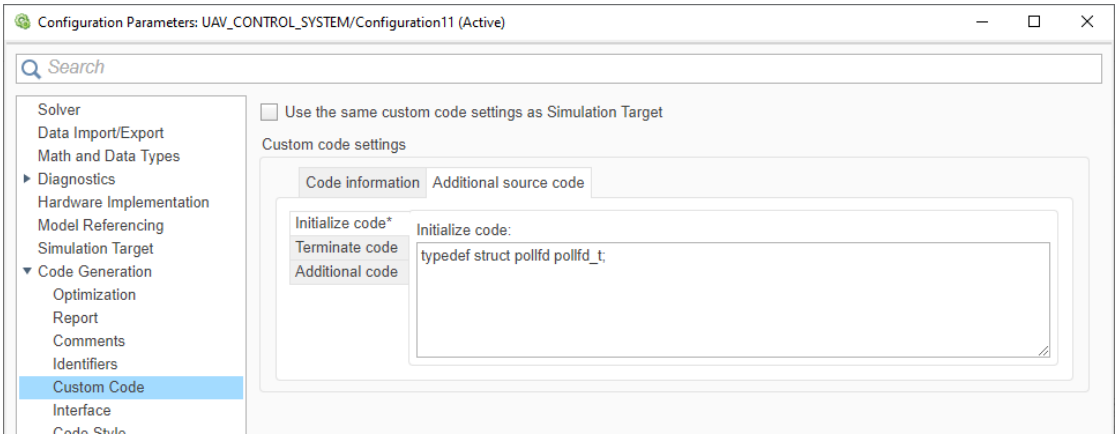

Figura A.31: Incluir la definición del tipo `pollfd`

4. Una vez configurado, para subir un nuevo firmware, hay que seleccionar dentro de Hardware: Run on board (External mode) y clickar en "Build Deploy and Start".

   Asegúrese previamente de haber ejecutado el script `CONFIG_UAV.m`)

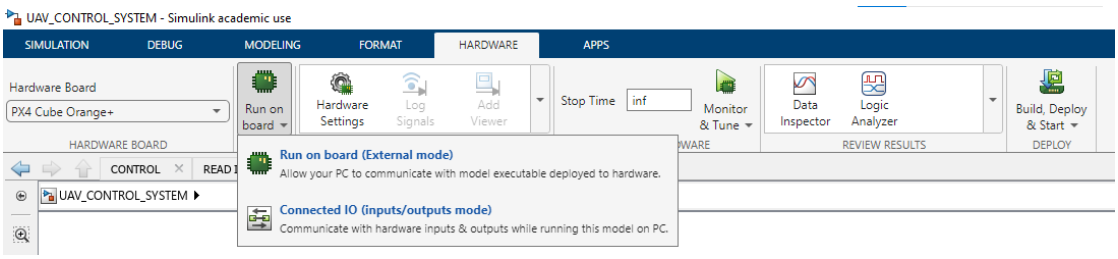

Figura A.32: External mode y Build Deploy & Start

Cuando haya terminado de construirse el firmware le aparecerá de nuevo la pestaña de conectar y desconectar. Tal y como indica dicha ventana, debe desconectar el cable, clickar en “OK” y reconectar el cable.

A.5. Configuración en QGroundControl

Para monitorizar las variables de interés seleccionadas en el bloque PX4 uLOG, necesitamos comunicar el Cube Orange+ con la estación de tierra (QGroundControl). Para ello, debe dirigirse a:

`QGroundControl >Application Settings >Comm Links`

Allí debe añadir (Icono Add) una nueva configuración:

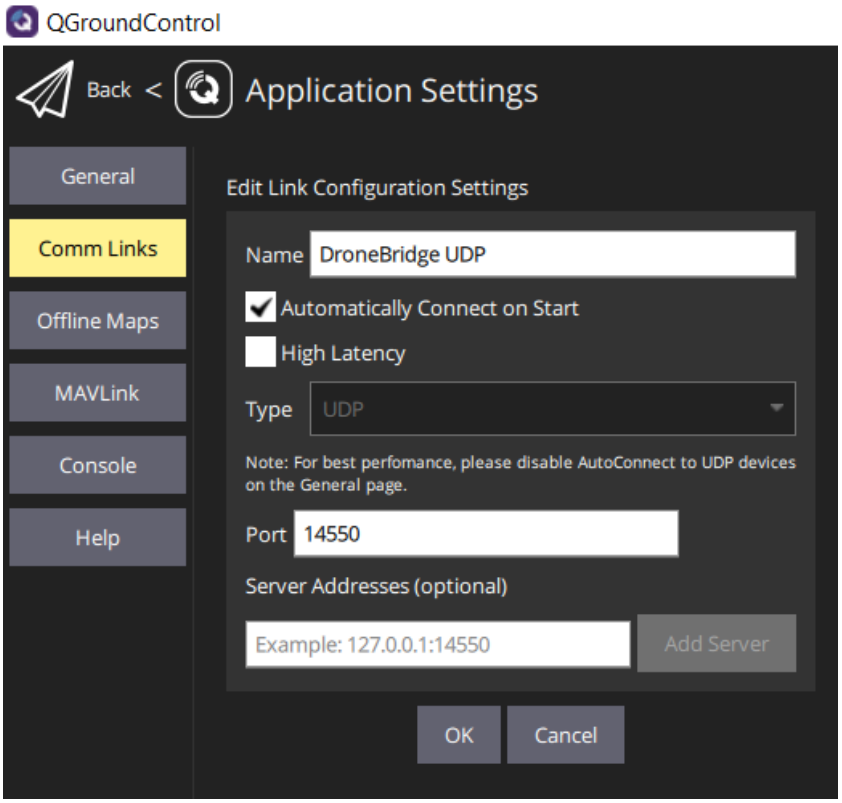

Figura A.33: Configuración conexión UDP en QGroundControl

Cuando haya terminado, pulse en OK y en Connect. Conecte la batería y verá que se encenderá un led rojo en el ESP32 y se le habilitará un punto de acceso a internet llamado: DroneBridge for ESP32. Seleccionalo, y ya estará todo configurado.

De esta forma, en: `QGroundControl >Analize tools >MavLink console` podrá monitorizar las variables de interés.

B. Construcción de Mensajes MAVLink

En este anexo se detalla el proceso a realizar para crear mensajes MAVLink, poder leerlos en tiempo real, y posteriormente realizar un análisis rápido de las medidas realizadas durante el vuelo.

B.1. Creación de un mensaje

Para crear el mensaje, debe seguirse la siguiente estructura en la pestaña de Command Window de Matlab:

`createPX4uORBMessage(messageName, uORBField)`

Es importante que el nombre del mensaje empiece en mayúsculas, ya que dará error en caso contrario. Las variables pueden ser de cualquier tipo, sean enteros, flotantes, vectores... Véase Código B.1.

```
>> createPX4uORBMessage('Intentov', 'uint8 control_mode', '
   uint8 control_status', 'uint8 current_status_sys', 'uint8
   motor_mode', 'float64[4] ch_pu', 'float64[3] euler_ang_ref
   ','double[4] emisora_in','double[4] emisora_out', 'uint16
   [4] pwm', 'double sensor_dist','boolean safety_sw', '
   float64 aux1', 'float64 aux2','float64 aux3')
```

Código B.1: Creación de un mensaje uORB

Al ejecutar el comando, saldrá el siguiente aviso (Código B.2):

```
The custom uORB message "Intentov.msg" is successfully
   included in the PX4 firmware.
To use the message, set up PX4 firmware using the Hardware
   Setup screens and build the PX4 firmware for the selected
   CMake.
To log the message in ulog format, update the logger_topics.
   txt in SD Card.
```

Código B.2: Confirmación en Command Window de MATLAB

Una vez realizados los pasos de instalación del firmware (véase Anexo para más información), el fichero .msg debe aparecer en el directorio /PX4-Autopilots/msg y aparecerá en el listado de mensajes del archivo CMakeLists.txt en ese mismo directorio. De esta forma Simulink reconoce el mensaje.

En Simulink, dentro de la librería de UAV Toolbox Support Package for PX4 Autopilots, se encuentra el bloque utilizado para almacenar las variables dentro del mensaje. El bloque se llama uLog, y dentro del bloque se debe incluir el Topic que se acaba de crear. El nombre debe ser idéntico al que se puso, respetando las mayúsculas y minúsculas (Figura B.1).

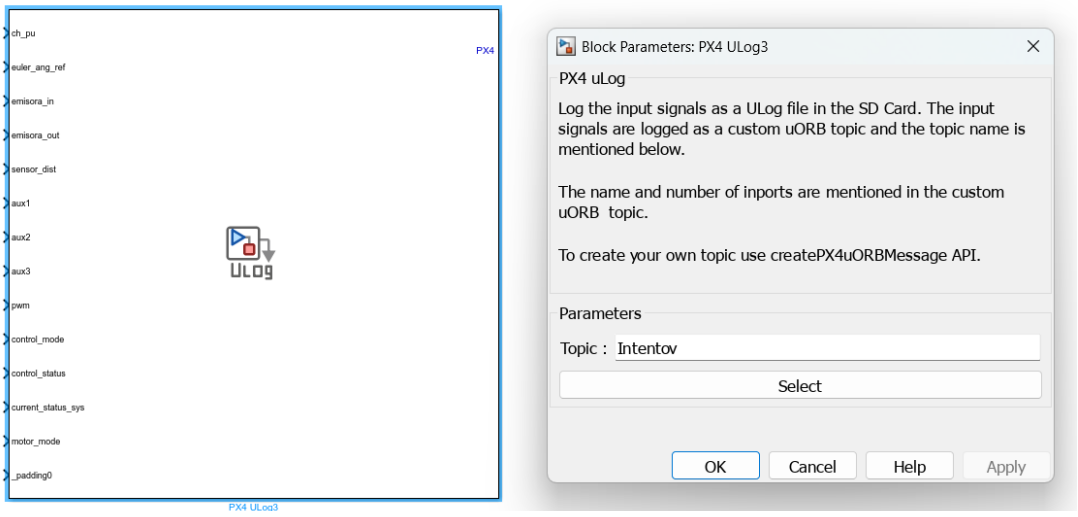

Figura B.1: Configuración del bloque uLog

Una vez conectadas las variables en su puerto correspondiente, y descargado el diagrama en el Cube Orange+, se puede proceder a la lectura de los mensajes.

B.2. Lectura de los mensajes

Para la lectura de los mensajes debemos abrir la consola de MAVLink en QGroundControl.

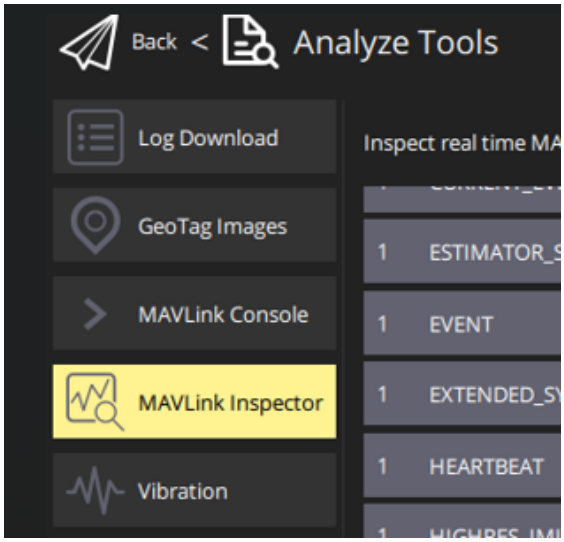

Figura B.2: MAVLink en QGroundControl

En MAVLink Inspector se pueden leer los mensajes predeterminados, como el HEARTBEAT, ATTITUDE y la lectura de los canales de la emisora. En MAVLink Console se pueden ver los mensajes personalizados, ejecutando un sencillo comando *listener messageName*. Importante que el nombre esté escrito en minúsculas, de lo contrario no lo reconocerá. Cada vez que se ejecute el comando, los valores se actualizarán según se haya programado.

```
nsh> listener intentov

TOPIC: intentov
 intentov
    timestamp: 59949425 (0.002207 seconds ago)
    ch_pu: [0.000000, 0.000000, 0.000000, 0.000000]
    euler_ang_ref: [0.000000, 0.000000, 0.000000]
    emisora_in: [1500.000000, 1500.000000, 1000.000000, 1500.000000]
    emisora_out: [1500.000000, 1500.000000, 1000.000000, 1500.000000]
    sensor_dist: 0.000000
    aux1: 0.000000
    aux2: 0.000000
    aux3: 0.000000
    pwm: [1000, 1000, 1000, 1000]
    control_mode: 0
    control_status: 1
    current_status_sys: 2
    motor_mode: 0
```

Figura B.3: Ejemplo de un mensaje MAVLink personalizado

Es aconsejable ejecutar en primer lugar *logger status*. Este comando indicará si la información se está guardando en la tarjeta SD (y dónde) o no.

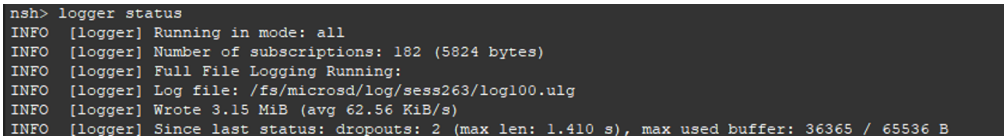

Figura B.4: Comando logger status

Puede ser que en vez de devolver información como la de la Figura B.4, devuelva el aviso *not logging*. Este fallo es habitual, y es probable que sea debido a que la tarjeta SD esté llena. Puede comprobarse fácilmente ejecutando el comando *logger start*. Si la tarjeta está llena, saldrá una advertencia en la parte superior de la pantalla indicándolo. Para solucionarlo, basta con extraerla del Cube Orange+, borrar el contenido de la carpeta /log e insertarla de nuevo. El problema debería estar solucionado.

Es recomendable ejecutar *logger stop* si no se van a leer mensajes MAVLink pero sí se va a mantener el dron conectado a QGroundControl. Si luego se desea continuar almacenando información, solo es necesario ejecutar *logger start*.

Si solo se están realizando pruebas de los componentes o la respuesta de los motores, es posible que resulte más cómodo configurar el parámetro SD_LOG_MODE a 2, para que guarde información desde el momento que se conecta hasta que se apaga. Si, por el contrario, se van a realizar pruebas de vuelo, entonces sería más indicado configurarlo a 0 (el predeterminado), ya que solo guardará la información que haya tenido lugar desde que se armaron los motores hasta que se desarmaron.

C. Alineación con los Objetivos de Desarrollo Sostenible

Este proyecto se encuentra alineado con varios de los Objetivos de Desarrollo Sostenible (ODS) propuestos por la Organización de las Naciones Unidas (ONU), especialmente en lo relativo a la eficiencia energética, la innovación tecnológica y la formación académica en ingeniería. A continuación, se detallan los principales objetivos con los que guarda relación:

ODS 4: Educación de calidad

Este proyecto se enmarca dentro de un Trabajo de Fin de Grado en Ingeniería Industrial, aplicando de forma práctica los conocimientos adquiridos a lo largo de los estudios. Además, contribuye a fortalecer competencias en áreas clave como el control de sistemas, la programación y el modelado matemático. Al tratarse de un trabajo colaborativo, también fomenta habilidades como la coordinación y el trabajo en equipo.

ODS 7: Energía asequible y no contaminante

Se ha priorizado el uso de componentes electrónicos con bajo consumo energético. La selección de estos elementos ha considerado factores como el peso, la eficiencia, la fiabilidad, el lugar de procedencia y el consumo energético, con el objetivo de lograr una optimización energética global del sistema.

ODS 12: Producción y consumo responsables

El proyecto promueve el uso responsable de recursos mediante la elección de componentes eficientes y la reutilización de materiales disponibles. Esta práctica contribuye a minimizar el impacto ambiental, tanto en términos de consumo energético como en la durabilidad y ciclo de vida de los materiales empleados.