PROYECTO FINAL DE INGENIERÍA
Cuadricóptero Open Source

Grandinetti Camesella, Juan Ignacio - LU 1111435
Scoflich, Lautaro - LU 1108371
Ingeniería Electrónica

Tutor:
Dr. Ing. Amet, Leonardo Javier

2025

UNIVERSIDAD ARGENTINA DE LA EMPRESA
FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

Resumen

El presente trabajo de tesis trata sobre el desarrollo de una plataforma abierta, accesible y replicable para el control de vuelo de cuadricópteros. El proyecto integra aspectos teóricos, de simulación, firmware, software y hardware, con el propósito de proporcionar a estudiantes, investigadores e instituciones una plataforma accesible y personalizable para la experimentación en el campo de los vehículos aéreos no tripulados. Se abordan fundamentos de dinámica y control, junto con la identificación de parámetros reales del sistema mediante ensayos prácticos, que permitieron el diseño e implementación de controladores basados en algoritmos PID y técnicas de fusión de mediciones indirectas de estados.

Asimismo, se desarrolló una placa de control propia, un firmware modular para el microcontrolador ESP32 y un software de monitoreo en tiempo real, complementados con un entorno de simulación y pruebas experimentales que validaron el desempeño del sistema. Finalmente, se presenta un análisis económico que expone los costos asociados a la fabricación, desarrollo y puesta en funcionamiento del proyecto, ofreciendo una referencia clara para universidades, empresas o particulares interesados en implementar soluciones similares. En conjunto, la tesis constituye una base sólida para futuros desarrollos en drones de propósito específico, reduciendo los tiempos y costos de investigación y prototipado al proponer una plataforma abierta y documentada que reduce la barrera de entrada y acelera la innovación.

Abstract

This thesis focuses on the development of an open, accessible, and replicable platform for quadcopter flight control. The project integrates theoretical, simulation, firmware, software, and hardware aspects, with the aim of providing students, researchers, and institutions with an affordable and customizable platform for experimentation in the field of unmanned aerial vehicles. It addresses the fundamentals of dynamics and control, along with the identification of real system parameters through practical testing, which enabled the design and implementation of controllers based on PID algorithms and techniques for the fusion of indirect state measurements.

In addition, a custom control board, a modular firmware for the ESP32 microcontroller, and a real-time monitoring software were developed, complemented by a simulation environment and experimental tests that validated the system's performance. Finally, an economic analysis is presented, detailing the costs associated with the project's manufacturing, development, and implementation, providing a clear reference for universities, companies, or individuals interested in adopting similar solutions. Altogether, the thesis establishes a solid foundation for future developments in purpose-specific drones, reducing research and prototyping time and costs by offering an open and well-documented platform that lowers entry barriers and accelerates innovation.

Agradecimientos

Agradecimientos de Grandinetti Camesella, Juan Ignacio:

Agradezco principalmente y enormemente a mis padres por apoyarme en todos los aspectos y darme la oportunidad de ser quien soy hoy en día. Asimismo, agradezco profundamente a Abigail Romero por la comprensión y apoyo incondicional en todo momento, particularmente durante el último tramo de este proyecto. No menos importante, gracias a nuestro tutor Leonardo Amet por brindarnos su conocimiento, experiencia y apoyo, el cual fue sumamente útil en momentos donde no teníamos claro el camino a seguir. Por último, gracias Lautaro por todas las experiencias compartidas juntos y dejarme aprender de vos constantemente, no obstante, agradecerte fundamentalmente por la amistad generada.

Agradecimientos de Scoflich, Lautaro:

Para mis padres, Verónica y Marcelo, por el apoyo, la paciencia y la motivación durante toda la carrera y durante la realización de este trabajo. A mi compañera, Luciana, por su comprensión y sus ánimos durante este trayecto. Para mi amigo y colega Juan Ignacio, por acompañarme con su amistad y profesionalismo en este camino. Finalmente, a nuestro tutor de tesis, Leonardo, por su enorme calidad como docente y como persona. Gracias a todos ellos por acompañarme en este camino, hacerme mejor persona y ayudarme a cumplir mis sueños.

Grandinetti Camesella, Juan Ignacio
Scoflich, Lautaro
6 de noviembre de 2025

Índice de contenido

Índice de figuras

Índice de tablas

Índice de bloques de código

Capítulo 1

Introducción

En este proyecto de tesis de grado se describirá todo el proceso de diseño, desarrollo y pruebas de un controlador de vuelo *"Cuadricóptero Open Source"*. El objetivo de este controlador de código abierto es brindar a individuos e instituciones una herramienta de uso libre y altamente personalizable, incluso desde el bajo nivel de funcionamiento. De esta manera, podrán aprender y experimentar en el mundo de los drones. También para que puedan basarse en este desarrollo para el diseño de drones de propósito específico, centrándose en el valor agregado de su idea y no en lo ya desarrollado en este trabajo. En síntesis, la idea es reducir el tiempo entre la idea y el prototipo, y potenciar el valor agregado de la solución de negocio dando bases sólidas en las que basar nuevos proyectos.

1.1 Cuadricóptero

1.1.1 Conceptos elementales

Un cuadricóptero se puede describir como un vehículo aéreo no tripulado (VANT o dron) con cuatro hélices propulsoras, accionadas por motores, en configuración transversal. La Figura 1.1 muestra la distribución de los motores de un cuadricóptero configurado en "X"[1].

Los dos pares, diagonalmente opuestos, de hélices (1,3) y (2,4) giran en direcciones opuestas. Las hélices 1 y 3 giran en sentido horario, mientras que las hélices 2 y 4 giran en sentido antihorario. Esta configuración de direcciones de giro opuestas permite equilibrar los torques generados, eliminando la necesidad de un rotor de cola.

[1]Esta configuración se caracteriza por definir los ejes (x, y) del cuadricóptero de manera tal, que cada uno quede entre medio de dos motores. En contraste, en la configuración "+", los ejes (x, y) se encuentran alineados con los brazos del cuadricóptero.

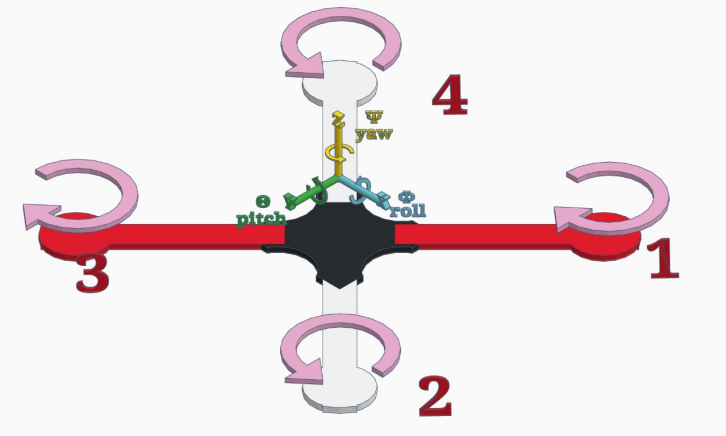


Figura 1.1: Cuadricóptero en configuración X

En la Figura 1.2 se puede observar un diagrama de bloques que resume los componentes principales y sus relaciones entre ellos. Este trabajo se basará principalmente en el desarrollo de un Controlador de Vuelo, cuyas funciones serán explicadas en la sección siguiente.

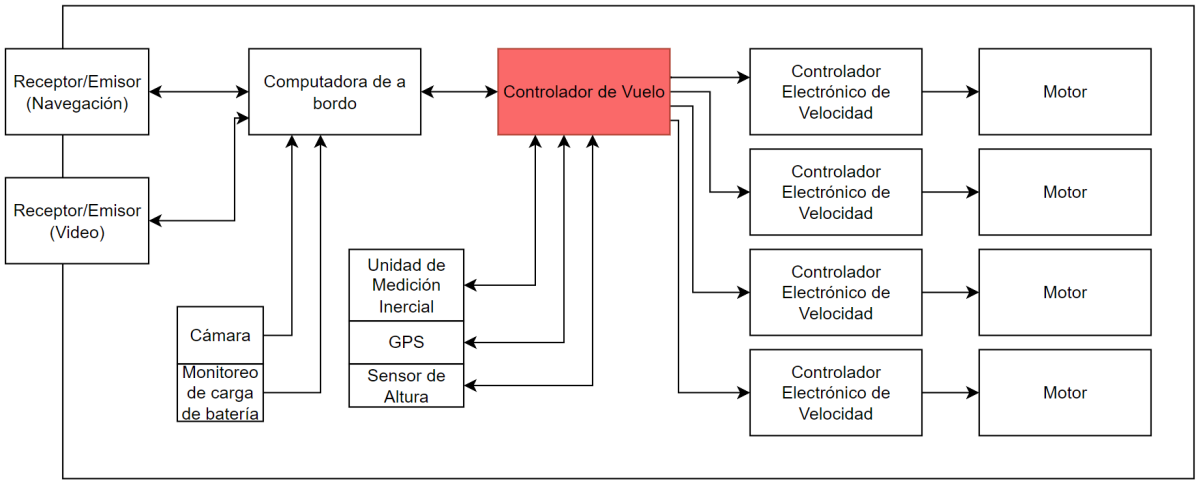


Figura 1.2: Diagrama en bloques del sistema de un cuadricóptero

1.1.2 Estudio de los componentes principales

1.1.2.1 Computadora de a bordo

La computadora de a bordo puede pensarse como el sistema de gestión de misión del dron, que se encarga de dar órdenes a ejecutarse por el Controlador de Vuelo, manejar la comunicación con el piloto o estaciones terrestres, procesar y transmitir el video capturado por la cámara, decidir el perfil de vuelo actual en función de la batería u otras variables, entre otras cosas.

Esta es una computadora de placa única, como una Raspberry Pi o una solución específica del fabricante, que corre un sistema operativo de propósito general como Linux o Robot Operating System (ROS), ejecuta varias tareas de diferentes prioridades, y se encarga de dar las indicaciones necesarias para el cumplimiento de la misión, en vez de ejecutarlas, lo cual es res-

ponsabilidad del controlador de vuelo. Por todo esto, su ciclo de trabajo es de menor frecuencia que el de la computadora de vuelo.

Suele guardar información de los puntos por los que tiene que pasar el dron, ejecutar algoritmos de procesamiento de vídeo e inteligencia artificial para detectar objetivos u obstáculos, y definir el camino óptimo para esquivarlos. Además, en caso de que el dron los posea, es la encargada de procesar datos de sensores más complejos como nubes de puntos de Lidar o múltiples cámaras.

Esta computadora se comunica con el Controlador de Vuelo mediante un protocolo que puede variar según la complejidad, tamaño y requerimientos de inmunidad al ruido del sistema, yendo desde algo tan simple como protocolos serie tales como UART, CAN, MavLink, hasta Ethernet.

Sin embargo, su presencia como componente independiente es opcional, ya que en drones pequeños, recreativos o altamente miniaturizados, pueden implementarse solo algunas funciones necesarias en el controlador de vuelo y prescindir de la computadora de a bordo. En estos casos, el controlador de vuelo puede denominarse también Autopiloto.

1.1.2.2 Controlador de vuelo

El controlador de vuelo, por su parte, auspicia como sistema de control de vuelo de bajo nivel, encargándose de ejecutar las directivas enviadas por la computadora de vuelo en forma de movimientos o velocidades en el espacio específicos. Se encarga de interactuar con los sensores críticos para la estabilización y el vuelo como puede ser la Unidad de Medición Inercial, la brújula o el barómetro.

Es un microcontrolador o computadora pequeña que ejecuta un sistema operativo de propósito específico, o incluso, en el caso de desarrollos pequeños, firmware a bajo nivel sin sistema operativo. Sus tareas son de alta prioridad, se ejecutan en tiempos del orden de milisegundos y la frecuencia de su ciclo de trabajo es mucho mayor que la de la computadora de a bordo.

Ejecuta algoritmos de control tales como PIDs que convierten una señal de control en comandos para los motores, de manera de mantenerse estable y cumplir con las órdenes de la computadora de a bordo.

1.1.2.3 Motores

Es usual utilizar motores de corriente continua (DC) sin escobillas debido a que tienen una mayor relación empuje-peso que aquellos con escobillas. Por otra parte, al ser conmutados electrónicamente, para una velocidad de giro dada presentan un mayor torque. Por otra parte, algunas consideraciones a tener en cuenta al momento de elegir un motor son las siguientes.

Peso del cuadricóptero

Lo primero es conocer el peso del cuadricóptero para calcular aproximadamente cuánto empuje necesita generar la combinación de motores y hélices con tal de garantizar el vuelo.

Relación empuje-peso

Una buena práctica consiste en imponer que el empuje generado sea, como mínimo, el doble del peso del cuadricóptero para garantizar un comportamiento ágil, dinámico y veloz. En contraste, si el empuje proporcionado por los motores es demasiado pequeño, el cuadricóptero no

responderá satisfactoriamente a los comandos.

Tamaño del motor

El tamaño de los motores sin escobillas, en general, está indicado por un número de 4 dígitos. Así, los primeros 2 dígitos indican el diámetro del estátor, mientras que los últimos 2 indican su altura. Luego, mientras más alto sea el estátor, mayor número de revoluciones por minuto tendrá el rotor y, cuanto mayor sea el diámetro del estátor, el par generado será mayor para un valor dado de revoluciones por minuto.

En este caso, se seleccionó el motor DC sin escobillas A2212/13T de 1000 K_v (ver Figura 1.3)

Figura 1.3: Motor DC sin escobillas, A2212/13T 1000 K_v

Constante de velocidad

Es un parámetro que indica el incremento de revoluciones por minuto del motor, sin carga, cuando la tensión de alimentación crece 1 Volt,

$$K_v = \frac{rpm}{V}$$

Una vez incorporadas las hélices, las revoluciones por minuto caen drásticamente debido a la resistencia con el aire. Los motores con K_v más elevados intentan girar las hélices a mayor velocidad, en contraste, los motores de menor K_v generan mayor par motor. Por tanto, para aplicaciones donde se requieran hélices grandes se suelen utilizar motores de bajo K_v porque hace falta más par para hacerlas girar, pero para el caso de hélices pequeñas y ligeras se utilizan motores diseñados para altas velocidades.

Por último, al momento de seleccionar el motor que mejor se adecúe para la aplicación en cuestión se debe realizar un análisis que relacione las rpm que el motor es capaz de generar con la fuerza requerida para producir el efecto deseado en la planta. En concreto, para el cuadricóptero la fuerza de cada motor puede expresarse como (*Sección 2.1.3: Dinámica*)

$$F_m = k_t \omega^2$$

Además, como se acaba de mostrar la velocidad angular de cada motor se relaciona linealmente con la tensión aplicada

$$\omega = k_v v$$

Luego, se puede reescribir la fuerza de cada motor en función de la tensión aplicada según

$$F_m = k_t k_v^2 v^2 = k_m v^2$$

Como mínimo, el cuadricóptero debe ser capaz de igualar la fuerza de atracción gravitatoria para poder sustentarse en el aire, es decir,

$$F_m \geq F_g = mg = 0,9kg \cdot 9,81\frac{m}{s^2} = 8,83N$$

donde se ha utilizado el peso determinado en la *Tabla X: Peso de cada componente*.

Ahora bien, puesto que el cuadricóptero diseñado para el presente proyecto tiene como objetivo final realizar maniobras lentas y controladas, una aproximación razonable podría ser exigir que la fuerza generada por los motores sea el doble de la fuerza gravitatoria esto es, $F_m \geq 2F_g$

Igualando ambas ecuaciones se obtiene

$$F_m = k_t k_v^2 v^2 \geq 2F_g$$

Despejando la constante de velocidad k_v y asumiendo una tensión de alimentación igual a 11V,

$$k_v \geq \sqrt{\frac{2F_g}{k_t v^2}} = \sqrt{\frac{2(8,83N)}{(1,984e^{-7}\frac{N}{rpm^2})(11V)^2}} = 857,69\frac{rpm}{V}$$

Así, se conlcuye que los motores seleccionados con $k_v = 1000\frac{rpm}{V}$ son suficientes para que el cuadricóptero logre realizar las maniobras esperadas.

1.1.2.4 Controlador de velocidad

La velocidad a la cual giran los motores depende de los comandos enviados por el controlador. Ahora bien, la máxima corriente nominal que puede suministrar este último es menor a la requerida por los motores para funcionar correctamente. Por tal motivo, se utilizan controladores electrónicos de velocidad (ESC) como intermediarios. Estos se encargan de recibir la señal del controlador y llevar al motor a la velocidad óptima suministrando la corriente apropiada para ello. En tal caso, se deben de tener en cuenta las siguientes características al momento de seleccionarlos.

Corriente de consumo

Como se mencionó previamente, la corriente requerida por los motores será suministrada por los ESC. Entonces, si el motor demanda mayor corriente que la que puede entregar, este último comenzará a calentarse y, eventualmente, acabará fallando. Hay dos tipos de corriente especificadas: la primera es la corriente nominal y se refiere a la máxima corriente que puede manejar un variador de forma continua sin dañarse. La segunda es la corriente en ráfaga y hace referencia a la máxima corriente que soporta el variador durante un breve período de tiempo.

Programación

Dado que un cuadricóptero utiliza 4 motores simultáneamente, si todos reciben el mismo comando uno esperaría que giren a la misma velocidad. No obstante, si los ESC no son programables, entonces no se puede controlar la relación entre PWM recibida y revoluciones por minuto de salida.

$$\text{PWM} \rightarrow \boxed{\text{ESC}} \rightarrow \text{rpm}$$

Luego, para un comando dado, no todos los motores girarán a la misma velocidad; en consecuencia, los torques no serán iguales y el cuadricóptero presentará inestabilidades.

Para los primeros ESC adquiridos (ver Figura 1.4) algunos de ellos resultaron programables y otros no.

Figura 1.4: ESC no programable 30 A

Por tal motivo, se optó por utilizar los controladores (Hobbywing, s.f.) Hobbywing Skywalker de 40 A (ver Figura 1.5).

Figura 1.5: ESC programable 40 A

Peso y tamaño

Normalmente, el tamaño y peso de un ESC dependen del amperaje que soporta.

Tensión de alimentación

Hay que asegurar que el variador sea compatible con la tensión de alimentación de la batería utilizada o, de lo contrario, se acabará dañando.

1.1.2.5 Sensores

Si bien es el usuario quien define hacia dónde se desplazará el cuadricóptero, ¿Cómo puede el sistema de control determinar si se desplaza en la dirección correcta? Para ello, se emplean sensores que midan las magnitudes físicas de interés y luego, se comparan las mediciones con las referencias definidas por el usuario, tal como sugiere la Figura 1.6. Así, el objetivo será que los valores medidos por los sensores sean lo más parecidos posible a las referencias.

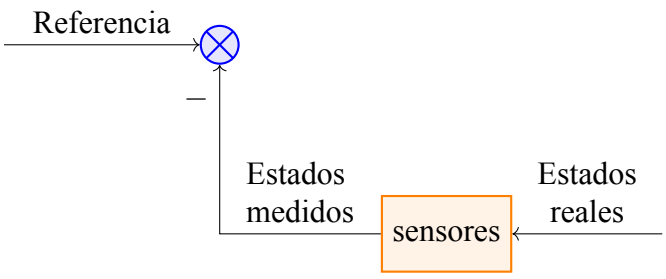


Figura 1.6: Diagrama de sensores

Se puede observar que los estados reales no son los mismos que aquellos medidos por los sensores, lo cual se debe al agregado de ruido[2] en el sistema.

1.1.2.6 Batería

Para que el cuadricóptero funcione, se debe alimentar al controlador, los ESC y motores. Además, puesto que es un vehículo aéreo no es posible alimentarlo a través de un cable. Por ello, en la práctica es común el uso de baterías recargables. Algunas variables relevantes a tener en cuenta son la tensión, capacidad, peso y la velocidad de carga y descarga. Los tipos que se pueden encontrar son los siguientes.

Baterías de níquel-cadmio (Ni-Cd)

Están compuestas de varias células de 1.2V cada una, no obstante, el hecho de cargarlas sin haberlas descargado previamente por completo provoca que no se puedan cargar en toda su capacidad para el resto de su vida útil (efecto memoria). Asimismo, el cadmio es una sustancia altamente contaminante.

Baterías de níquel-metal-hidruro (Ni-MH)

Presentan mayor capacidad de carga, menor efecto memoria y aceptan cargas rápidas. Sin embargo, soportan un menor número de cargas durante su vida útil y tienen una resistencia interna superior, lo cual las limita para alimentar motores de alta potencia.

Baterías de litio-ión (Li-Ion)

Su capacidad es aproximadamente el doble que las baterías anteriormente mencionadas y, la tensión de cada célula es de 3.7V. Asimismo, dado que el litio es el metal más ligero, resultan más livianas. Por otra parte, no se ven afectadas por el “efecto memoria”.

Baterías de polímero de litio (Li-Po)

Se fabrican en una mayor variedad de formas y tamaños que las baterías de litio-ion ya que utilizan un polímero. La principal característica de este tipo de baterías es su elevada densidad energética, por unidad de peso, la cual les permite proveer gran potencia en un espacio reducido. Esto resulta esencial ya que los cuadricópteros deben ser ligeros para poder volar eficientemente. Por otra parte, son de carga rápida debido a que presentan una pequeña resistencia interna, lo cual implica que pueden manejar grandes corrientes de carga sin necesidad de sobrecalentamiento o daños en la batería. A pesar de lo mencionado previamente, las baterías de Li-Po son sensibles a la temperatura y a la carga en exceso, pues esto puede causar que se sobrecaliente. En base a las características mencionadas, se eligió este tipo de batería para alimentar al cuadricóptero (ver Figura 1.7)

[2]Para esta aplicación, el ruido es principalmente mecánico ocasionado por las vibraciones de la estructura del cuadricóptero al girar los motores, y electromagnético que afecta las mediciones de la unidad inercial o IMU.

Figura 1.7: Batería Li-Po 2200 mAh

Finalmente, para seleccionar la batería a utilizar se debe tener en cuenta la fuerza máxima que podrían llegar a requerir generar los motores, lo cual está ligado a su velocidad angular según se detalló en la *Sección 1.1.2.3: Motores*. Además, también hay que tener en cuenta la corriente de consumo de los motores, puesto que esta limita la vida útil del cuadricóptero.

Para responder la primer inquietud, se despeja la tensión de alimentación, en vez de la constante de velocidad k_v quedando así,

$$v \geq \sqrt{\frac{2F_g}{k_t k_v^2}} = \sqrt{\frac{2(8.83N)}{(1.894e^{-7}\frac{N}{rpm^2})(1000\frac{rpm}{V})^2}} = 9.43V$$

Nótese que haber calculado previamente la constante de velocidad, k_v, con $v = 11V$ fue una elección razonable.

Luego, para responder la segunda cuestión comenzamos por determinar la cantidad de rpm necesarias para lograr una fuerza en los motores igual a $F_m = 2F_g$

$$\omega \geq \sqrt{\frac{2F_g}{k_t}} = \sqrt{\frac{2(8,83N)}{1,984e^{-7}\frac{N}{rpm}}} = 9434,63\ rpm$$

A partir de la *Figura 3.30: Curva característica I-W del motor utilizado a partir de los gráficos obtenidos*, se puede observar que la corriente de consumo no supera los 11A, asi que se utilizará este valor como peor escenario.

Suponiendo que la capacidad de la batería es de $C = 2200\ mAh = 2,2\ Ah$, entonces el máximo tiempo de vuelo estimado será de $t_{max} = \frac{C}{Consumo_{max}} = \frac{2,2\ Ah}{11A} = 0,2\ h\frac{60min}{h} = 12min.$

Si bien el tiempo de vuelo no es tan prolongado, el cuadricóptero tiene como finalidad realizar maniobras específicas para analizar y comprobar el rendimiento del sistema de control y del firmware. Por tanto y con la finalidad de no encarecer el componente de manera innecesaria, se justifica su elección.

1.1.2.7 Transmisor de Navegación

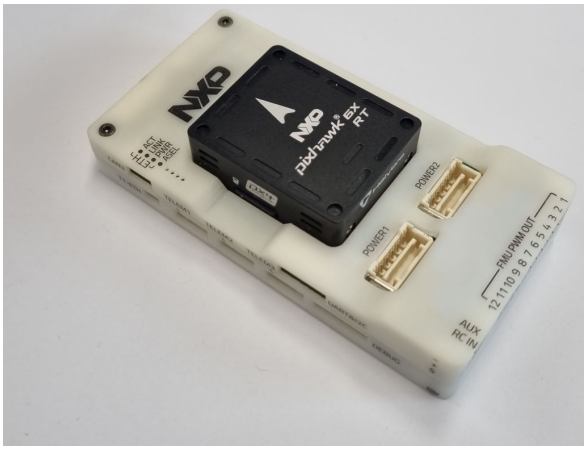
Es un sistema de antena y modulador/demodulador de radio, que se encarga de transmitir desde el operador hacia el dron los comandos de vuelo como velocidad, altura, orientación, junto a algunas variables adicionales como perfil de uso de motores, modo de vuelo y demás. La frecuencia más utilizada es de 2.4GHz, y 72MHz en caso de transmisores más antiguos.

En el caso de sistemas comerciales más avanzados, o propios del fabricante, suelen también permitir una comunicación bidireccional para el intercambio de información de telemetría de datos de vuelo, como datos históricos de variables de vuelo o de misión.

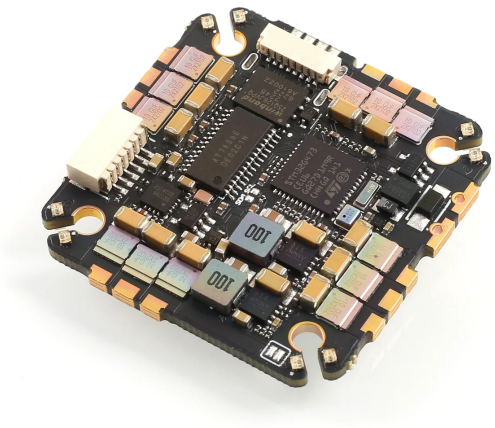
1.1.3 Antecedentes de controladores de vuelo

A lo largo de los años, han sido desarrolladas diversas plataformas de código abierto y comercial que permiten el funcionamiento autónomo de los drones. Los desarrollos de la comunidad son, por lo general y como se mostrará a continuación, firmwares de control pensados para ser compatibles con distintas placas de control de vuelo o computadoras de a bordo. La filosofía general de la comunidad es ofrecer, dentro de ciertas restricciones, la capacidad de utilizar en una misma placa distintos firmwares de control, siempre que la arquitectura del microprocesador sea compatible con el código. De estos programas de control utilizados para los controladores de vuelo, los más populares son:

- **PX4 Autopilot**: Sistema de control de vuelo de código abierto desarrollado por la Fundación Dronecode. Ofrece estabilización avanzada y planificación de misiones.(PX4, 2025)
- **Betaflight**: Inicialmente diseñado para carreras de drones, se caracteriza por su alto rendimiento en control de orientación y respuesta en tiempo real. (Betaflight, 2025)
- **INAV**: Orientado a aeronaves de ala fija y multicópteros, con énfasis en capacidades de navegación autónoma. (Betaflight, 2025)
- **ArduPilot**: Plataforma de código abierto que soporta múltiples configuraciones de aeronaves. Proporciona capacidades de vuelo autónomo, navegación por *waypoints* y seguimiento de misiones.(Ardupilot, 2025b) Anteriormente ofrecían una placa de controlador de vuelo, pero esta fue discontinuada y el proyecto centra sus esfuerzos en el desarrollo de firmware que corra en otras plataformas.(Ardupilot, 2025a)



(a) Placa de control de vuelo N□XP MR-VMU-RT1176,□ que soporta el firmware de PX4



(b) Placa de control de vuelo Äirbot Fenix G4 35A AIO,□ que soporta el firmware de Betaflight

Figura 1.8: Ejemplos de placas de control de vuelo

Analizando las prestaciones de tanto las plataformas y las placas que estas soportan, se puede encontrar que comparten características comunes fundamentales:

- Software de código abierto basado en arquitecturas modulares
- Algoritmos de estabilización mediante filtros complementarios o Kalman
- Comunicación mediante protocolos UART, I2C, MAVLink o MSP
- Soporte para sensores inerciales (IMU) y sistemas de posicionamiento global (GPS)
- Capacidades de telemetría y registro de datos de vuelo
- Interfaz de configuración mediante aplicaciones *ground control*
- Controladores electrónicos de velocidad integrados en la placa

En cuanto a su implementación hardware, estas plataformas están predominantemente basadas en microcontroladores de la familia ARM Cortex, particularmente procesadores STM32. El software es desarrollado principalmente en lenguaje C o C++ y opera sobre sistemas en tiempo real o *bare-metal*. La mayoría utilizan el entorno de desarrollo Arduino o PlatformIO, y se distribuyen mediante licencias de código abierto (GPL/LGPL).

La evolución de estos controladores ha sido impulsada por comunidades de desarrolladores y empresas que colaboran en el perfeccionamiento continuo de los algoritmos de navegación, estabilización y autonomía.

1.2 Marco legal en Argentina

1.2.1 Instituciones reglamentarias en Argentina

En Argentina, en el año 2007, se establece mediante el Decreto 239/2007(Poder Ejecutivo Nacional, 2007) a la Administración Nacional de Aviación Civil (ANAC) como entidad oficial y Autoridad Aeronáutica Nacional, con facultad para regular, fiscalizar y administrar la aviación civil en el país de acuerdo al Código Aeronáutico dictaminado en la Ley 17.285(Poder Ejecutivo Nacional, 1967) y demás tratados internacionales.

Estos tratados internacionales son principalmente convenios con la Organización de Aviación Civil Internacional (OACI).

Al día de publicación de este trabajo, se encuentra vigente la Resolución ANAC N° 550/2025(Administración Nacional de Aviación Civil (A.N.A.C), 2025) que incluye las Regulaciones Argentinas de Aviación Civil (RAAC) 100, 101 y 102 para la clasificación y regulación de aeronaves tripuladas a distancia.

1.2.2 Clasificación de drones

La regulación actual de la ANAC clasifica los drones en 3 categorías claras:

1.2.2.1 Abierta

Esta categoría aplica a operaciones que no requieran autorización previa, como aquellas realizadas dentro del alcance visual del operador o en zonas rurales, a una altura máxima de 122 metros. Aplica a drones de menos de 25 kilogramos de peso, salvo en zonas rurales donde el operador es responsable de evitar interferir con la operación de otras aeronaves.

Con la normativa actual no es necesaria licencia, certificado de explotación de trabajo ni certificación médica aeronáutica.

1.2.2.2 Específica

La categoría específica aplica a operaciones que requieran autorización por parte de la autoridad competente, los prestadores de servicio de navegación aérea y los operadores de aeródromos involucrados, como operaciones más allá de la visibilidad directa, vuelos en zonas urbanas y operaciones nocturnas. Aplica a drones entre 25 kilogramos y 150 kilogramos, o cualquier dron que pese más de 250 gramos pero opere fuera de las condiciones propias de la categoría abierta.

El vehículo aéreo no tripulado debe poseer un peso máximo al despegue comprendido entre 25 kg y 150 kg. Asimismo, tiene que estar inscrito en el Registro Nacional de Aeronaves de la ANAC, debiendo contar con el Certificado de Matrícula y el Certificado de Aeronavegabilidad vigente, cuando este último sea aplicable según la normativa.

Por su parte, el piloto debe poseer una licencia aeronáutica de piloto a distancia emitida por la ANAC, y tener una certificación médica aeronáutica clase 3. Esta última se obtiene y renueva en conformidad a la parte 67 de la RAAC.

La institución explotadora (puede ser el piloto o una empresa que contrate al piloto) debe tener un Certificado de Explotador de Trabajo Aéreo (CETA) mediante la presentación y aprobación por parte de la ANAC sobre el Manual de Operaciones del Explotador (MOE) y un Sistema de Gestión de Mitigación de Riesgo.

1.2.2.3 Certificada

Esta categoría engloba operaciones que requieran autorización directa de la Autoridad Aeronáutica, como operaciones de movilidad aérea avanzada o urbana y aplica a drones de 150 kilogramos o más.

Los requisitos de la misma son iguales a aquellos aplicables a aeronaves tripuladas.

1.2.3 Conclusión del marco legal

La ANAC establece ciertas limitaciones para el piloto y para el dron, por lo que la computadora a bordo del dron debe ser capaz de operar dentro del marco legal. Es por eso que el controlador de vuelo Cuadricóptero Open Source debe ser capaz de detectar y alertar sobre ciertas situaciones:

1. Altura máxima de 120 metros en espacio aéreo no controlado [3], 43 metros a menos de 5 kilómetros de aeropuertos.

2. No poder sobrevolar zonas prohibidas como aeropuertos, áreas militares y demás, incluidas zonas especiales designadas como “Zonas de No Vuelo” para drones.

3. Registro de datos de vuelo para auditorías.

[3] Salvo que se pida una autorización especial de la ANAC

1.3 Objetivos y Alcance

El objetivo general de este trabajo es desarrollar una plataforma abierta de control de vuelo para cuadricópteros que sea replicable en entornos educativos y de investigación y desarrollo. Esta plataforma busca reducir el tiempo de desarrollo de nuevas ideas o proyectos tecnológicos al ofrecer una base técnica funcional y accesible sobre la cual pueden incorporarse módulos o algoritmos adicionales según las necesidades del usuario. De esta manera, se contribuye a disminuir los costos de ingeniería asociados al desarrollo inicial de sistemas de control, al tiempo que se promueve la investigación, la experimentación y la innovación en el ámbito de los sistemas embebidos y el control de vehículos aéreos no tripulados.

El objetivo de este trabajo no es demostrar un cuadricóptero volando en modo autónomo completo, sino desarrollar y validar una plataforma abierta, modular y replicable de control de vuelo, incluyendo: modelado dinámico con parámetros identificados experimentalmente, firmware de control en tiempo real sobre ESP32 con FreeRTOS, hardware de control propio con puntos de test, y herramientas de monitoreo y ajuste en línea, que sea replicable en entornos educativos y de investigación y desarrollo.

El vuelo libre estable se propone como trabajo futuro sobre una base ya calibrada y documentada.

1.3.1 Objetivos específicos

Los objetivos específicos de este trabajo incluyen modelar la dinámica del cuadricóptero y determinar sus parámetros físicos reales mediante técnicas de identificación experimental. Se busca diseñar e implementar controladores en cascada para la estabilización de la actitud, utilizando un microcontrolador ESP32 que ejecuta FreeRTOS. Además, se desarrollará una placa de control propia que incorpore puntos de medición, junto con una herramienta de monitoreo y ajuste en tiempo real. Finalmente, se validará experimentalmente el lazo interno de control de actitud y se compararán los resultados obtenidos de la realidad con los obtenidos en simulación.

1.3.2 Alcance

El alcance del presente trabajo comprende el desarrollo del modelado y la simulación del sistema, la identificación de los parámetros físicos relevantes, la implementación del firmware de control en tiempo real, y el diseño del hardware dedicado para dicho control. Asimismo, se incluyen las herramientas de visualización y monitoreo necesarias para la puesta a punto, junto con la realización de pruebas en una bancada restringida que permita validar el funcionamiento del sistema en condiciones controladas.

1.3.3 Futuras líneas

A partir de la plataforma desarrollada en este trabajo se abre la posibilidad de profundizar en las siguientes líneas de trabajo futuro: la integración de elementos relacionados con el vuelo libre estabilizado en todos los ejes, la navegación autónoma completa mediante puntos de guía, la integración de visión artificial, múltiples sensores, inteligencia artificial y la combinación de estos para la detección y prevención de colisiones. Además, se podría integrar este controlador en un sistema que cuenta con una computadora de vuelo, y utilizar ESC internos, montados en la misma placa.

Capítulo 2

Marco teórico

2.1 Física del cuadricóptero

En esta sección, se expondrán los conceptos de físicos y matemáticos asociados al movimiento del cuadricóptero.

2.1.1 Sistemas de referencia

Figura 2.1: Sistema de referencia solidario a un objeto

El sistema de referencia utilizado para medir la posición relativa del cuadricóptero se denomina “sistema de referencia solidario a un objeto” (ver Figura 2.1) y se denota como $\{B\}$. Se utiliza para definir cómo un cuerpo está orientado. Los vectores unitarios B_x, B_y, B_z están fijos al cuerpo del cuadricóptero en el sentido en que rotan o giran según este lo haga. Esto significa que se puede rastrear hacia dónde apunta o hacia dónde está orientado. Por otra parte, la posición del cuadricóptero se mide con respecto a un sistema de referencia inercial $\{E\}$ (ver Figura 2.2), es decir, no acelera ni tampoco rota.

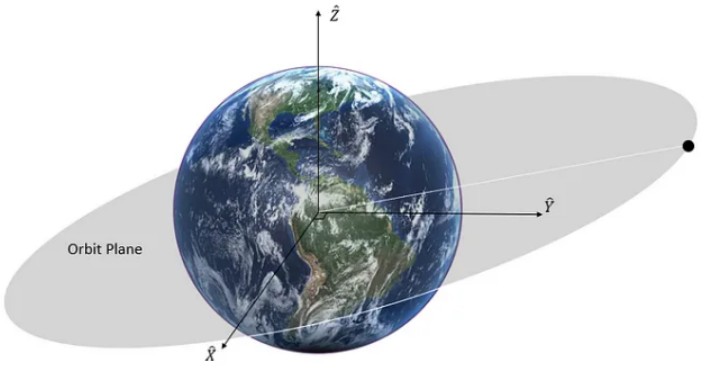


Figura 2.2: Sistema de referencia inercial

2.1.2 Mecánica

Mediante la variación de la velocidad del motor, se puede cambiar la fuerza de sustentación y crear movimiento como se describe en la Figura 2.3.

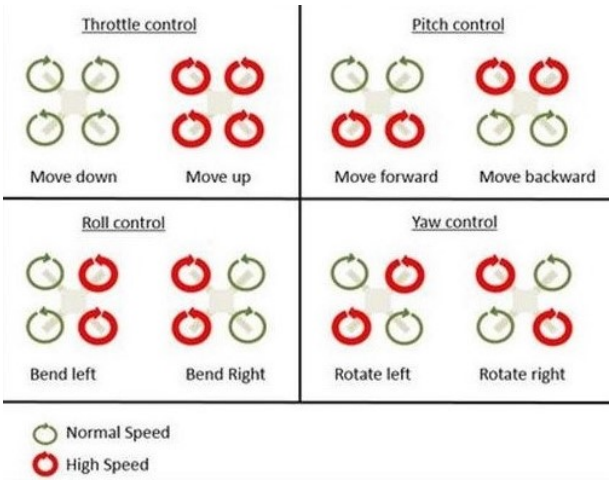


Figura 2.3: Cambio de velocidad en hélices para generar movimiento

Esta configuración permite cuatro movimientos básicos, los cuales habilitan al cuadricóptero a alcanzar una posición exacta. A continuación, se describen estos movimientos básicos.

Empuje

Este movimiento es proporcionado por la variación de velocidad de todas las hélices en la misma cantidad generando así una fuerza que hace subir, bajar, o mantener suspendido en el aire al cuadricóptero respecto al sistema inercial.

Roll

Este movimiento es proporcionado por el incremento en la velocidad de las hélices $(2, 3)$ y la disminución en la velocidad de las hélices $(1, 4)$ o viceversa. El resultado es un movimiento neto hacia la izquierda o bien, hacia la derecha.

Pitch

En este caso, el incremento (o disminución) en la velocidad de las hélices $(1, 2)$ y la disminución (o incremento) en la velocidad de las hélices $(3, 4)$ produce un movimiento neto hacia delante (o hacia atrás).

Yaw

Por último, el incremento (o disminución) de las hélices $(2, 3)$ y la disminución (o incremento) de las hélices $(1, 4)$ genera una rotación en sentido horario (o antihorario).

2.1.3 Dinámica

Como se puede observar en la Figura 1.1, cada motor provee una fuerza en la dirección $z > 0$, que es proporcional a ω^2, siendo $\omega \geq 0$ la velocidad de giro de cada motor, medida en $\frac{rad}{s}$. Luego, la fuerza, en Newton N, que estos generan puede expresarse como $F_i = k_t \omega_i^2$ (Quan, 2017, Sec. 6.1.3.2), donde k_t es la constante de empuje de los motores[1], medida en $N\left(\frac{s}{rad}\right)^2$. Por tanto, la fuerza total del sistema solidario al cuadricóptero $\{B\}$ sobre el eje z, se puede calcular como la sumatoria de las fuerzas de cada motor. Es decir,

$$F_z^{\{B\}} = \sum_{i=1}^{n=4} F_i = k_t(w_1^2 + w_2^2 + w_3^2 + w_4^2) \tag{2.1}$$

No obstante, las coordenadas de referencia que se envían al sistema de control se encuentran descritas en el sistema inercial $\{E\}$ y la manera de convertir una coordenada de un sistema cualquiera a otro es a través de la matriz de rotación R. Entonces, una manera de obtener la fuerza generada por el cuadricóptero, en el sistema inercial $\{E\}$, es integrando la matriz de rotación a la segunda ley de Newton.

$$F^{\{E\}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{1}{m} R^{\{B\} \longrightarrow \{E\}} \begin{bmatrix} 0 \\ 0 \\ F_z^{\{B\}} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

donde m representa la masa total del cuadricóptero, y se mide en kg. De esta manera, se pueden obtener las aceleraciones lineales $(\ddot{x}, \ddot{y}, \ddot{z})$[2].

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}(F_1 + F_2 + F_3 + F_4)(\sin(\phi)\sin(\varphi) + \cos(\phi)\cos(\varphi)\sin(\vartheta)) \\ \frac{1}{m}(F_1 + F_2 + F_3 + F_4)(\cos(\phi)\sin(\varphi)\sin(\vartheta) - \cos(\varphi)\sin(\phi)) \\ \frac{1}{m}(F_1 + F_2 + F_3 + F_4)(\cos(\phi)\cos(\vartheta)) - g \end{bmatrix} \tag{2.2}$$

Por otra parte, las rotaciones alrededor de cada eje se pueden calcular a partir de los torques que genera cada motor, recordando que $Torque = Fuerza \cdot Distancia \cdot sin(\alpha)$.

[1] Se utiliza el mismo valor que aquel encontrado en (Quan, 2017, Sec. 6.3.4.3) (al parámetro que aquí se denominó k_t, allí se lo denomina c_T), dado que el experimento fue llevado a cabo con los mismos motores, hélices y ESC. Por otra parte, en caso de ser necesario, allí se encuentra detallado su proceso de obtención.

[2] El desarrollo para la obtención de una expresión para la matriz de rotación se encuentra en la sección 2.1.4.

$$\begin{aligned}\tau_x &= (F_1 + F_4 - F_2 - F_3)\left(\frac{\sqrt{2}}{2}\right)l = k_t\left(\frac{\sqrt{2}}{2}\right)l(\omega_1^2 + \omega_4^2 - \omega_2^2 - \omega_3^2)\\ \tau_y &= (F_1 + F_2 - F_3 - F_4)\left(\frac{\sqrt{2}}{2}\right)l = k_t\left(\frac{\sqrt{2}}{2}\right)l(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)\\ \tau_z &= (F_1 + F_3 - F_2 - F_4)l = k_t l(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2)\end{aligned} \tag{2.3}$$

donde $(\tau_x = \tau_{roll}, \tau_y = \tau_{pitch}, \tau_z = \tau_{yaw})$ y l es la longitud, en metros, desde el centro del cuadricóptero hacia cualquiera de los 4 motores[3]. Además, nótese que el sin no aparece en la ecuación porque la fuerza que produce cada motor es siempre perpendicular a los brazos del cuadricóptero y, por tanto, $\sin(\frac{\pi}{2}) = 1$.

Para obtener las aceleraciones angulares $(\ddot{\phi}, \ddot{\vartheta}, \ddot{\varphi})$ se hace uso de la ley de Euler del movimiento de cuerpos rígidos, que establece lo siguiente.

$$\begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \begin{bmatrix}j_{xx} & 0 & 0\\ 0 & j_{yy} & 0\\ 0 & 0 & j_{zz}\end{bmatrix}\begin{bmatrix}\ddot{\phi}\\ \ddot{\vartheta}\\ \ddot{\varphi}\end{bmatrix} + \begin{bmatrix}\dot{\phi}\\ \dot{\vartheta}\\ \dot{\varphi}\end{bmatrix} \times \begin{bmatrix}j_x & 0 & 0\\ 0 & j_y & 0\\ 0 & 0 & j_z\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\vartheta}\\ \dot{\varphi}\end{bmatrix}$$

Debido a que el sistema $\{B\}$ tiene su origen en el centro del cuadricóptero y su estructura es simétrica respecto a este, la matriz de inercia resulta ser una matriz diagonal cuyos elementos son, j_{xx}, j_{yy}, j_{zz}, medidos en m^2kg. Luego de realizar el despeje, las aceleraciones son las siguientes[4].

$$\begin{bmatrix}\ddot{\phi}\\ \ddot{\vartheta}\\ \ddot{\varphi}\end{bmatrix} = \begin{bmatrix}\frac{k_t(\frac{\sqrt{2}}{2})l(\omega_1^2+\omega_4^2-\omega_2^2-\omega_3^2)}{j_{xx}}\\ \frac{k_t(\frac{\sqrt{2}}{2})l(\omega_1^2+\omega_2^2-\omega_3^2-\omega_4^2)}{j_{yy}}\\ \frac{k_t l(\omega_1^2+\omega_3^2-\omega_2^2-\omega_4^2)}{j_{zz}}\end{bmatrix} + \begin{bmatrix}\dot{\vartheta}\dot{\varphi}\frac{(j_{yy}-j_{zz})}{j_{xx}}\\ \dot{\phi}\dot{\varphi}\frac{(j_{zz}-j_{xx})}{j_{yy}}\\ \dot{\vartheta}\dot{\phi}\frac{(j_{xx}-j_{yy})}{j_{zz}}\end{bmatrix} \tag{2.4}$$

Se observa que se puede controlar el movimiento del cuadricóptero a partir de las velocidades de cada motor w_i. Luego, el objetivo es que los controladores encuentren sus valores apropiados para que el cuadricóptero alcance las referencias enviadas por el usuario.

2.1.4 Matrices de transformación

Sea el sistema de referencia inercial $\{E\}$ y otro sistema $\{B\}$ con traslaciones relativas respecto a $\{E\}$ luego, la matriz de rotación $R^{\{B\}\longrightarrow\{E\}}$ permite trasladar puntos del sistema $\{B\}$ al sistema $\{E\}$. En esencia, para realizar tal conversión hay que proyectar un punto ubicado en $\{B\}$, en cada eje de $\{E\}$. Esto es

$$R^{\{B\}\longrightarrow\{E\}} = \begin{bmatrix}\hat{x_B}\cdot\hat{x_E} & \hat{y_B}\cdot\hat{x_E} & \hat{z_B}\cdot\hat{x_E}\\ \hat{x_B}\cdot\hat{y_E} & \hat{y_B}\cdot\hat{y_E} & \hat{z_B}\cdot\hat{y_E}\\ \hat{x_B}\cdot\hat{z_E} & \hat{y_B}\cdot\hat{z_E} & \hat{z_B}\cdot\hat{z_E}\end{bmatrix} \text{ tal que,}$$

$$\begin{bmatrix}x_E\\ y_E\\ z_E\end{bmatrix} = R^{\{B\}\longrightarrow\{E\}}\begin{bmatrix}x_B\\ y_B\\ z_B\end{bmatrix}$$

[3] El motivo del $\frac{\sqrt{2}}{2}$ se detalla en el apéndice A.
[4] El despeje matemático se encuentra detallado en el apéndice E.

La manera de obtenerla es a partir de la multiplicación de las matrices de rotación R_x, R_y, R_z sobre cada eje x, y, z, respectivamente. Con base en la Figura 2.4, se calcula R_x como

$$R_x = \begin{bmatrix} \cos(0) & \cos(90) & \cos(90) \\ \sin(90) & \cos(\phi) & -\sin(\phi) \\ \cos(90) & \sin(\phi) & \cos(\phi) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

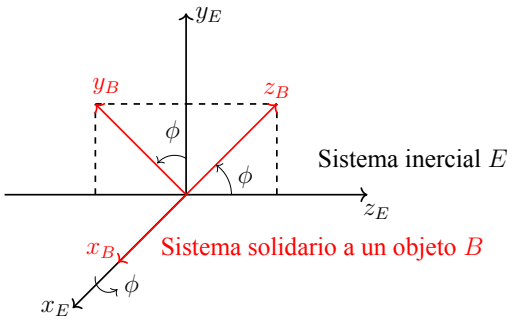


Figura 2.4: Rotación alrededor del eje x $(roll)$

Luego, según la Figura 2.5

$$R_y = \begin{bmatrix} \cos(\vartheta) & \cos(90) & \sin(\vartheta) \\ \cos(90) & \cos(0) & \cos(90) \\ -\sin(\vartheta) & \cos(90) & \cos(\vartheta) \end{bmatrix} = \begin{bmatrix} \cos(\vartheta) & 0 & \sin(\vartheta) \\ 0 & 1 & -\sin(\vartheta) \\ -\sin(\vartheta) & 0 & \cos(\vartheta) \end{bmatrix}$$

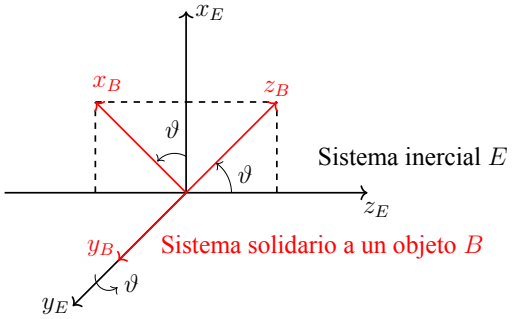


Figura 2.5: Rotación alrededor del eje y $(pitch)$

Finalment, se puede calcular R_z en base a la figura Figura 2.6

$$R_z = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & \cos(90) \\ \sin(\varphi) & \cos(\varphi) & \cos(90) \\ \cos(90) & \cos(90) & \cos(0) \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

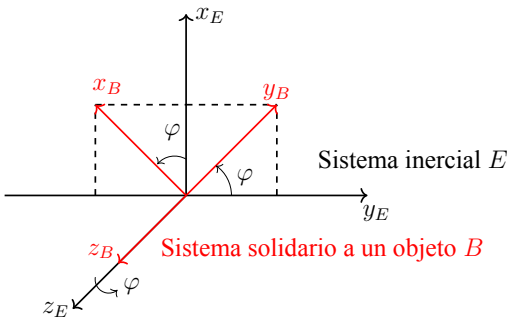


Figura 2.6: Rotación alrededor del eje z (yaw)

Continuando, para obtener la matriz de rotación se utilizan los Ángulos de Euler (Quan, 2017, Sec. 5.2.1.1), que consisten en rotar el sistema $\{B\}$ según la secuencia ordenada, $z \rightarrow y \rightarrow x$ o bien, $\varphi \rightarrow \vartheta \rightarrow \phi$. Al rotar sobre el eje z se obtiene, $P^E = R_z(\varphi)P^1$. Luego, rotando sobre el eje y se tiene que, $P^1 = R_y(\vartheta)P^2$. Por último, rotar sobre el eje x da como resultado, $P^2 = R_x(\phi)P^3$, donde

- P^1 representa un punto en el espacio luego de rotar sobre el eje z
- P^2 representa un punto en el espacio luego de rotar sobre el eje y
- P^3 representa un punto en el espacio luego de rotar sobre el eje x

Así, sustituyendo cada resultado en la ecuación $P^E = R_z(\varphi)P^1$ se obtiene,

$$P^E = R_z(\varphi)R_y(\vartheta)R_x(\phi)P^3 = R^{\{B\}\longrightarrow\{E\}}P^3$$

$$R^{\{B\}\longrightarrow\{E\}} = \begin{bmatrix} \cos(\varphi)\cos(\vartheta) & \cos(\varphi)\sin(\phi)\sin(\vartheta) - \cos(\phi)\cos(\varphi) & \sin(\phi)\sin(\varphi) + \cos(\phi)\cos(\varphi)\sin(\vartheta) \\ \cos(\vartheta)\sin(\varphi) & \cos(\varphi)\cos(\phi) + \sin(\phi)\sin(\vartheta)\sin(\varphi) & \cos(\phi)\sin(\varphi)\sin(\vartheta) - \cos(\phi)\sin(\phi) \\ -\sin(\vartheta) & \cos(\vartheta)\sin(\phi) & \cos(\phi)\cos(\vartheta) \end{bmatrix}$$

2.1.5 Fuerza aerodinámica y perfil alar

Se entiende por *perfil alar* al corte transversal del ala de una aeronave. Con ayuda de la Figura 2.7 se observa que, a medida que el ala avanza, las partículas de aire que se encuentran por debajo de la misma, (círculos rojos) se van a comprimir y así aumentar la presión de esa zona.

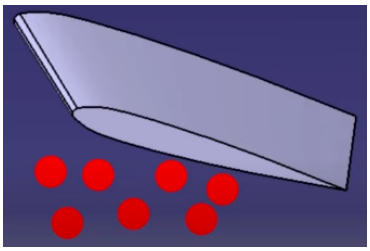


Figura 2.7: Partículas de aire debajo del perfil alar

En cambio, en la Figura 2.8 se observa que el área que ocupan las partículas de aire sobre el ala aumentará a medida que la misma avanza, lo cual hará disminuir la presión en esa zona. Entonces, como hay menos presión, las partículas de aire pueden expandirse, separándose unas de otras. Consecuentemente, su velocidad aumenta y provoca que el aire termine siendo empujado hacia abajo. Por último, a partir de la tercera ley de Newton el aire que es empujado hacia abajo por la curvatura del ala provocará una fuerza igual y opuesta. Ésta última es la causa de que las aeronaves vuelen.

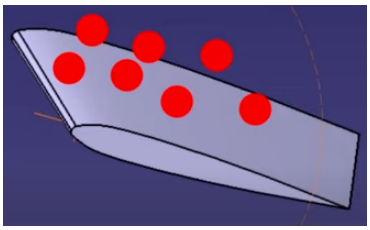


Figura 2.8: Partículas de aire sobre el perfil alar

2.2 Principio de funcionamiento de los sensores

2.2.1 Posición

2.2.1.1 Ultrasónico

Estos tipos de dispositivos utilizan el sonido para determinar la distancia entre el sensor y el objeto más próximo en su camino. En esencia, su principio de funcionamiento se basa en emitir una onda sonora a través de su módulo transmisor T_x, a una frecuencia superior a la audible, y esperar a que esta rebote (se refleje) en un objeto y regrese, para asi ser captada por el módulo receptor R_x (Figura 2.9).

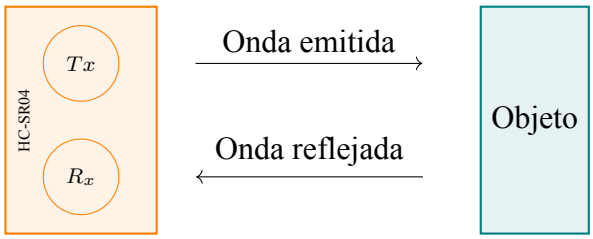


Figura 2.9: Principio de funcionamiento del sensor HC-SR04

Además, el sensor mide el tiempo que tarda la onda en ir y volver, de manera que conociendo la velocidad a la cual viaja es posible determinar la distancia entre el sensor y el objeto en cuestión. Luego, la distancia se puede obtener a partir de la ecuación

$$d = v_a \frac{t_s}{2}$$

siendo t_s el tiempo medido por el sensor, en segundos, y $v_a = 343.2 \frac{m}{s}$ la velocidad del sonido en el aire[5], en condiciones atmosféricas. De igual modo, nótese que el 2 dividiendo al tiempo t_s

[5] Si el medio de transmisión es distinto del aire, se debe usar el valor que corresponda para la velocidad.

se debe a que la onda viajó hasta el objeto y regresó al sensor, por ello, solo se debe utilizar la mitad del tiempo medido.

En concreto, el HC-SR04 posee 4 pines de los cuales

- VCC (5 V según Cytron Technologies, 2013, Cap. 4) y GND son los pines de alimentación.
- TRIG es el pin de control.
- ECHO es el pin que contiene la duración del recorrido de la onda.

La Figura 2.10 muestra cómo utilizar el sensor ultrasónico en conjunto con un microcontrolador para obtener la distancia relativa hacia un objeto dado. A continuación, se detallan los pasos a seguir.

1. El microcontrolador envía un pulso de mínimo 10 μs, al pin $TRIG$ del sensor.
2. Al recibir el pulso, el parlante (T_x) incorporado al sensor emite una ráfaga de 8 ciclos a 40 kHz.
3. Al mismo instante, el pin $ECHO$ cambia su estado de $BAJO$ a $ALTO$, y se mantiene con ese valor.
4. La onda emitida se refleja al impactar contra un objeto y retorna al micrófono (R_x) del sensor.
5. Al detectar el regreso de la onda, el pin $ECHO$ vuelve a cambiar su estado de $ALTO$ a $BAJO$.
6. El ancho del pulso emitido por el pin $ECHO$ representa la duración de la onda[6], y es el tiempo que se debe utilizar para obtener la respectiva distancia.

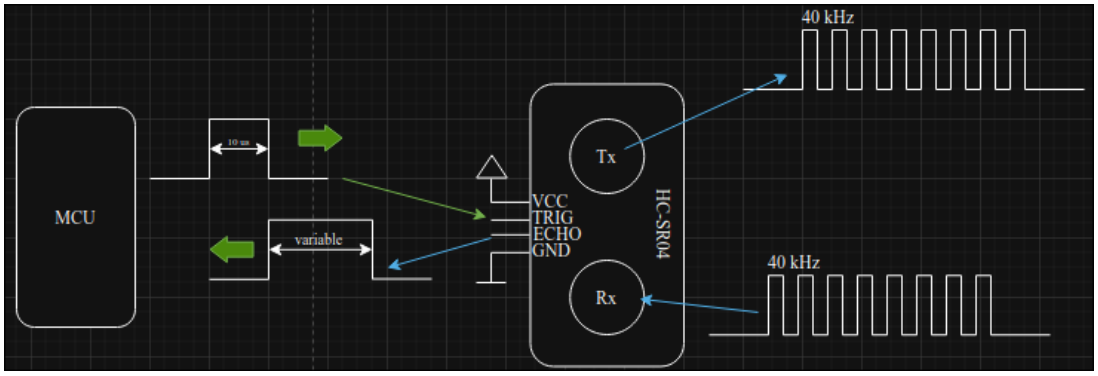


Figura 2.10: Funcionamiento del sensor HC-SR04 junto con un microcontrolador

[6]Según (Cytron Technologies, 2013, Cap. 5), el rango normal de operación es de 150 μs hasta 25 ms, tal que el sensor medirá 38 ms si no detecta ningún obstáculo.

2.2.1.2 Barométrico

Una alternativa para medir la altura en distancias donde el sensor ultrasónico ya no sea preciso o ni siquiera pueda obtener el sonido de retorno, es medir la presión atmosférica y temperatura ambiente. La presión atmosférica es consecuencia del peso de una columna de aire que descansa sobre una superficie debajo de él, por lo que mientras más alto se encuentre dicha superficie menor será la presión atmosférica percibida. Este fenómeno se puede emplear para calcular la altura a la que se encuentra un objeto, pero también hay que realizar correcciones debido a variaciones que pueda tener la presión en función de la humedad y la temperatura.

En este caso, se usará el sensor de presión barométrica BMP390, fabricado por BOSCH, que mide presión atmosférica y temperatura ambiente. Se eligió esto por encima de otras familias como BMP180/BMP280/BMP380 puesto que, en comparación, presenta una mejor precisión de ±3Pa equivalente a ±0.25 m (Bosch Sensortec, 2021b, Key features, Table 1).

Por otra parte, para medir presión utiliza un elemento piezoresistivo y un mezclador de señales ASIC (Application-Specific Integrated Circuits). Este último lleva a cabo conversiones A/D (analógicas a digitales) y provee tales resultados junto con valores de compensación, propios del sensor, a través de una interfaz digital. Estos son necesarios para obtener valores de presión y temperatura precisos y se deben aplicar junto con su respectiva formula (Bosch Sensortec, 2021b, Sec. 8.4, Sec. 8.5 y Sec. 8.6) sobre los datos crudos. Por otra parte, son almacenados en su memoria no volatil (NVM) (Bosch Sensortec, 2021b, Sec. 3.11.1, Table 24) durante su fabricación.

Siguiendo, el comportamiento del BMP390 se encuentra basado en una máquina de estados, tal como se puede apreciar en la Figura 2.11.

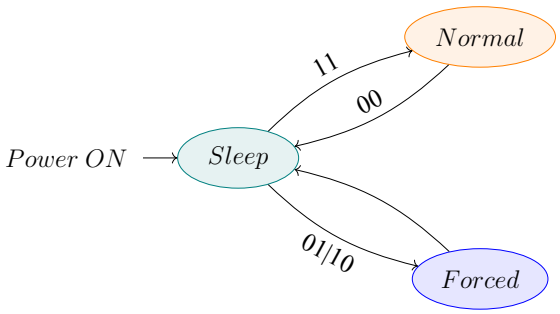


Figura 2.11: Diagrama de la máquina de estados del sensor BMP390

- Sleep mode: Una vez energizado, el sensor se encuentra en estado de reposo sin tomar mediciones.

- Normal mode: Se activa una única vez, de forma tal que el sensor se encarga de tomar nuevas mediciones automáticamente.

- Forced mode: El sensor toma una única medición y vuelve al modo “Sleep mode”. Luego, para volver a medir se debe de activar el modo “Forced mode” nuevamente.

Es posible modificar la precisión del sensor tanto para medir presión (Bosch Sensortec, 2021b, Sec. 3.4.1, Table 6) como para medir temperatura (Bosch Sensortec, 2021b, Sec. 3.4.1,

Table 7) ya que cuenta con 6 modos configurables. Sin embargo, es importante remarcar que aumentar la precisión de las mediciones equivale a un mayor costo de procesamiento, lo cual tiene un impacto directo en la frecuencia de muestreo disponible a nivel hardware.

En (Bosch Sensortec, 2021b, Sec. 3.9.2) se encuentra una ecuación que describe el tiempo total que tarda el sensor en obtener mediciones de presión y temperatura.

$$T_{conv} = 234\mu s + press\_en \cdot (392\mu s + 2^{osr\_p} \cdot 2020\mu s) + temp\_en \cdot (163\mu s + 2^{osr\_t} \cdot 2020\mu s)$$

donde osr_p y osr_t hacen referencia a la precisión de presión y temperatura, respectivamente. Luego la máxima frecuencia de muestreo (típica) para el "Forced mode" es

$$f_{max\_forced} = \frac{1}{T_{conv}[\mu s] \cdot 10^6}$$

mientras que para el "Normal mode" queda definida de la siguiente forma

$$f_{max\_normal} = \frac{200Hz}{2^{odr\_sel}}$$

siendo odr_sel el modo, de 4 bits, para configurar el registro ODR del sensor (Bosch Sensortec, 2021b, Sec. 4.3.19).

Por último pero no menos importante, el sensor cuenta con un filtro pasa bajos interno de tipo IIR, que sirve para mitigar cambios abruptos en el entorno y mejorar la precisión de las mediciones. En (Bosch Sensortec, 2021a, Sec. 3.4.3) se encuentra detallado el algoritmo que utiliza el sensor para calcular los próximos valores filtrados. En (Bosch Sensortec, 2021b, Sec. 3.5, Table 10) se puede observar una tabla con aplicaciones pre-definidas que contienen las configuraciones recomendadas por el fabricante para cada caso, lo cual es una buena opción como punto de partida.

2.2.1.3 GNSS

Los sensores anteriormente explicados sirven para dar una medición precisa de la altura cuando esta toma valores relativamente pequeños, pero a medida que aumenta, el ultrasonido deja de ser útil y por ello solo se puede utilizar la presión atmosférica. Esto es un problema debido a la poca precisión de la misma, y la deriva en la medición que presenta en grandes períodos de tiempo. Por otra parte, la posición también consta de dos componentes principales que son la latitud y la longitud, las cuales no hay manera de estimar a partir de estos sensores. Para ello, no se pueden usar únicamente sensores analógicos tradicionales, sino que habrá que emplear un Sistema Global de Navegación por Satélite, también conocido como GNSS.

El sensor que se utilizará es el NEO-7M desarrollado por u-blox (u-blox, 2018). Este es capaz de utilizar satélites de diversos sistemas GNSS como

- GPS, SBAS, QZSS
- GLONASS

El principal sistema utilizado por el receptor es el GPS, basado en una red de satélites y estaciones de seguimiento y control de Estados Unidos, que proveen la ubicación exacta y hora a cualquier sistema que cuente con un receptor GPS. Funciona mediante el concepto de trilateración, midiendo la distancia de cada satélite al receptor GPS y haciendo que se cumplan las

condiciones. Esta distancia se mide de la siguiente manera: El satélite envía una onda de radio con información muy exacta del tiempo en el que se generó, y el receptor resta el tiempo de envío al tiempo de recepción y multiplica eso por la velocidad de la luz.

Como el reloj de un receptor suele ser poco preciso, habría mucho error si se utilizase el tiempo medido por el mismo sin antes corregirlo, por lo que, para solucionar esto, se emplea un cuarto satélite que ayuda a determinar la desviación de tiempo. También existe el GPS asistido, que se beneficia de internet y datos móviles, el cual se asiste con la información de posición del enrutador más cercano.

Por otra parte, el sistema SBAS (Sistemas de Aumento Basado en Satélites) es una tecnología para GPS que permite añadir información para corregir las soluciones de navegación o mediciones. Fundamentalmente, en determinadas ubicaciones se encuentran estaciones de monitoreo o RIMS (Ranging Integrity Monitoring Stations) que proveen retrasos en la propagación de señales, desviaciones de órbita, errores en los relojes de los satélites, entre otras. Entonces, esta información es enviada a satélites SBAS y estos la reportan al receptor.

Con respecto a la comunicación entre el microcontrolador y el receptor GNSS, este último cuenta con una interfaz de comunicación flexible, ya que es multiprotocolo y multipuerto. En esencia, multiprotocolo hace referencia a la capacidad de transmitir simultáneamente tanto mensajes NMEA como mensajes UBX. Luego, multipuerto significa que cada protocolo puede ser transmitido a través de más de un puerto (I2C, UART, USB, SPI) a la vez.

El protocolo NMEA (National Marine Electronics Association) es el estándar de comunicación universal para los dispositivos marinos. Esencialmente, la estructura de una trama NMEA es la siguiente (u-blox, 2018, Sec. 18).

Figura 2.12: Formato de un mensaje NMEA

- $: Es el caracter que define el comienzo de un mensaje NMEA
- Tipo de receptor: Hace referencia al sistema GNSS utilizado por el receptor. Para el caso de un receptor GPS, se colocan las iniciales GP.
- Mensaje NMEA: Se refiere al tipo de información recibida. Por ejemplo, el mensaje GGA significa que el campo “Datos” contiene información sobre la calidad de la señal.
- Datos: Es la información útil.
- Checksum: Representa la integridad del mensaje y se calcula como la OR exclusiva entre todos los caracteres entre “$” y “*”.
- CR, LF: Representa el fin del mensaje.

Por otra parte, el protocolo UBX es propietario del fabricante u-blox y presenta la siguiente estructura (u-blox, 2018, Sec. 27).

Figura 2.13: Formato de un paquete UBX

- SYNC CHAR 1/2 (2 bytes): Todos los mensajes UBX comienzan con el encabezado 0xB5 0x62.
- CLASS (1 byte): Todos los mensajes UBX están agrupados según el tipo de información que contienen, o bien, la clase. Por ejemplo, la clase NAV contiene resultados de navegación como la posición, velocidad, tiempo, entre otras. En cambio, la clase CFG contiene información de la configuración del receptor.
- ID (1 byte): Dentro de una clase en concreto, existen diversos mensajes que contienen distinta información. Por ejemplo, el mensaje UBX-NAV-PVT contiene información de la posición, velocidad y tiempo, mientras que el mensaje UBX-NAV-DOP contiene información de la precisión de las mediciones.
- LENGTH (2 bytes): Es la longitud del payload en formato little endian (LSB-MSB).
- PAYLOAD (Variable): Es la información útil y su longitud queda definida por el contenido del campo LENGTH.
- CK_A/B (2 bytes): Representa la integridad del paquete y utiliza el algoritmo de 8-bit Fletcher para calcular ambos bytes de checksum (u-blox, 2018, Sec. 29).

Si bien el protocolo NMEA es el estándar universal para la comunicación, en este desarrollo se optó por utilizar el protocolo propietario de UBX por varios motivos.

1. La implementación de jerarquías a través de clases e IDs mejora la organización y distribución de la información.
2. Es más versatil puesto que permite configurar una mayor cantidad de modos del sensor, en comparación con el protocolo NMEA.
3. La transmisión de mensajes no es aleatoria, sino más bien los mensajes son de tipo *polling*[7] o periódicos en caso de configurarlos para tal modo de trabajo.

2.2.2 Orientación

En esta sección se detallarán todos los sensores empleados por el cuadricóptero para determinar su orientación actual. Estos sensores se construyen en chips integrados basados en Microelectromecánica (MEM), una tecnología que emplea fenómenos mecánicos a muy pequeña escala (micrómetros) para variar una propiedad eléctrica y generar una señal a medir.

[7]Por defecto, el receptor no envía ningún comando UBX a ningún puerto, es decir, para obtener información es necesario realizar una consulta al dispositivo. No obstante, es posible configurar ciertos mensajes para que el receptor los transmita periódicamente.

2.2.2.1 Giroscopio

Un giroscopio es un dispositivo capaz de medir la velocidad angular a la que rota un objeto en sus tres ejes. Para lograr esto mediante dispositivos MEM se hace uso del efecto Coriolis, que plantea que desde el sistema de referencia de un objeto moviéndose a una velocidad constante, si ese objeto gira, se percibirá una fuerza de Coriolis tal que genere un desplazamiento perpendicular del mismo. A partir de este desplazamiento, se puede generar un cambio en la capacidad entre dos masas a partir de la velocidad de giro en un eje (similar a como se verá en el acelerómetro).

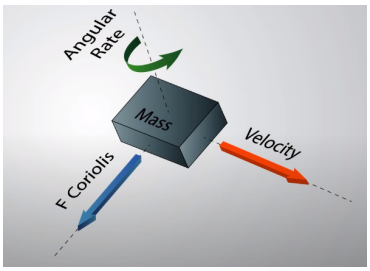


Figura 2.14: Objeto con velocidad que al girar percibe la fuerza Coriolis

Para obtener la velocidad de giro en los tres ejes, se pueden emplear tres dispositivos como estos colocados en distintas orientaciones.

2.2.2.2 Acelerómetro

Se entiende por acelerómetro a un dispositivo que es capaz de detectar aceleraciones lineales en los tres ejes de un objeto.

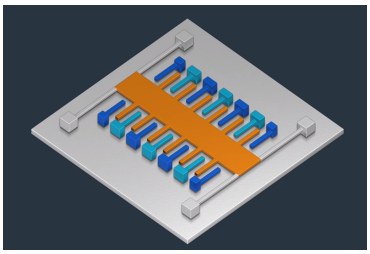


Figura 2.15: Vista de un acelerómetro

Las partes principales del acelerómetro son la masa de prueba (Figura 2.18), una masa de peso conocido con forma de H fijada al sustrato (Figura 2.17) de tal manera que puede moverse horizontalmente, y los electrodos (Figura 2.18) que están fijos al sustrato y no pueden moverse.

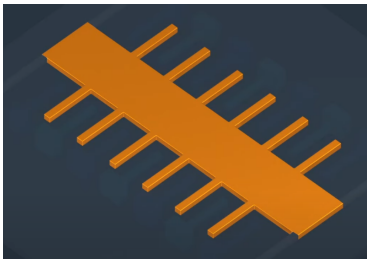


Figura 2.16: Masa de prueba

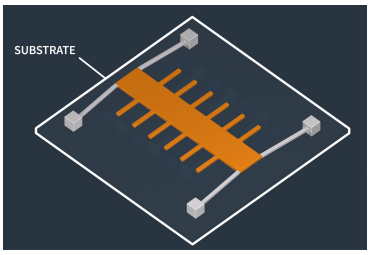


Figura 2.17: Masa de prueba moviendose, fijada al sustrato

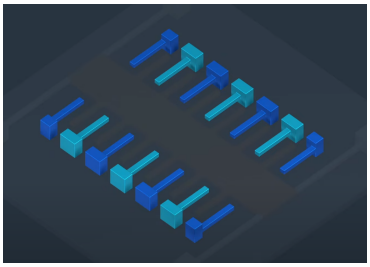


Figura 2.18: Electrodos

Al haber una aceleración en el eje horizontal, la masa de prueba se desplazará en la misma dirección que la misma, y variará la capacidad entre la masa y un grupo de electrodos. De esta manera se logra generar una señal eléctrica a partir de una aceleración. Si se quieren medir aceleraciones lineales en 3 ejes, simplemente colocamos 3 dispositivos como estos.

2.2.2.3 Magnetómetro

El magnetómetro es un dispositivo capaz de medir la intensidad y dirección del campo magnético percibido mediante el efecto Hall, y se usa especialmente para medir el campo magnético de la Tierra y así conocer la orientación respecto al norte.

El efecto Hall indica que si circula una corriente por un conducto y se aplica un campo magnético, los electrones se verán deflectados de su circulación normal y tenderán a ir a un lado del conductor, generando así una región con carga negativa y otra con carga positiva.

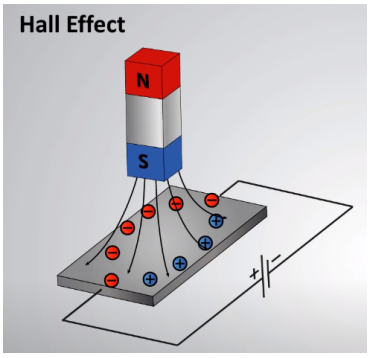


Figura 2.19: Efecto Hall y distribución de electrones

Si ahora se mide la tensión entre ambos lados de la placa conociendo la corriente que circula, se puede tener una medida del campo magnético percibido.

2.2.3 Otros

2.2.3.1 Carga de batería

Para medir la carga restante de la batería, se empleará un sistema que mida la tensión entregada por la misma y luego, a partir de un cálculo de ajuste a la curva de Tensión/Capacidad, se podrá determinar el estado de carga de esta.

A continuación, se muestra un ejemplo de una gráfica Tensión/Carga para distintos tipos de baterías.

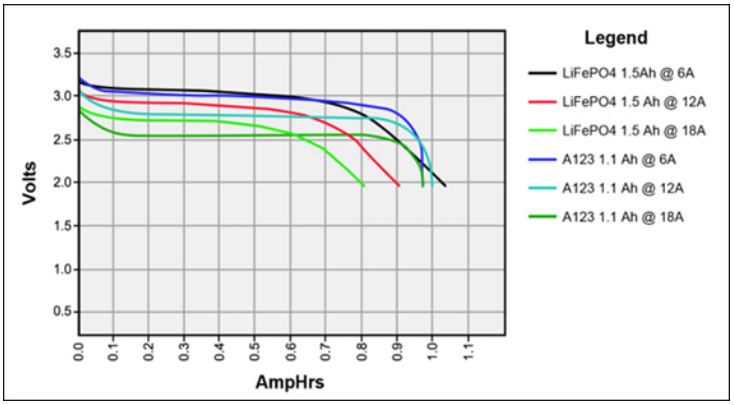


Figura 2.20: Gráfica de tensión/carga para distintos tipos de baterías

2.3 Teoría de control

2.3.1 Definiciones

Se entiende por Control Automático a un campo interdisciplinario de la ingeniería y la matemática aplicada, encargado del estudio y manipulación del comportamiento de un sistema dinámico. Durante este estudio, se emplean algunas definiciones.

2.3.1.1 Variables y señales

Variable controlada o de salida

Cantidad o condición que se mide y se controla, es decir, la salida o estado del sistema que es de interés estudiar. El valor de esta variable es el motivo de aplicar el sistema de control en primer lugar, pero no es modificable de manera directa, por lo que se recurre a modificar otras variables que sí pueden ser alteradas, y de las cuales depende el valor de la variable controlada. A estas variables que pueden ser modificadas se las denomina variables manipuladas.

Variable manipulada

Cantidad o condición que el controlador puede modificar, con el fin de aprovechar su influencia sobre la variable controlada a modo de alterar su comportamiento.

Señal de entrada

Es la señal que utiliza un elemento externo al sistema (puede ser un ser humano, o un sistema de control externo) para indicar al sistema de control la magnitud de acción de control que debe aplicarse al proceso.

Perturbación

Es una señal fuera de nuestro control que afecta a las variables de nuestro sistema. Su efecto es alterar la señal proveniente de los actuadores.

Ruido

Es inherente al sistema y afecta directamente la medición de los sensores. En consecuencia, la percepción de la realidad que tiene un sistema se ve degradada.

Señal de error

Es una variable exclusiva de los sistemas realimentados, y se define como la diferencia entre el valor medido de una variable presente en el sistema de control (salida o perturbación), y el valor de entrada.

2.3.1.2 Sistemas

Sistema

Se entiende como sistema a la combinación de componentes que actúan juntos y realizan un objetivo determinado, en este trabajo se hará uso del estudio, modelado y control de sistemas físicos, pero es interesante hacer notar que estos pueden ser abstractos, como aquellos que se trabajan en áreas como economía o sociología.

Planta

Se conoce como planta a la parte del proceso que, a partir de la variable manipulada, produce la variable de salida o variable controlada. En el caso puntual del cuadricóptero, la planta es la estructura misma que produce los estados de salida a partir de las velocidades angulares (variable manipulada) generadas por cada motor (actuadores).

A su vez, la planta puede ser dividida en subsistemas que se encargan de modelar una parte del proceso, con motivo de entenderlo a distintos niveles de abstracción y en distintos dominios de conocimiento, separando la parte eléctrica de la química, de la física, etc. Volviendo al cuadricóptero, se puede dividir la planta en tres subsistemas.

1. Fuerza en función de velocidad angular.
2. Torque en función de fuerza.
3. Aceleración en función de torque[8].

Controlador

Utiliza el error como parámetro para determinar qué tan cerca se encuentra la variable controlada del valor deseado. Además, también define cuál tiene que ser el valor de la *variable manipulada* para que esto suceda.

Actuador

En función del valor de la *variable manipulada*, genera una fuerza que actúa directamente sobre la planta y produce cambios en la variable controlada.

Por último, la Figura 2.21 integra todos estos conceptos en un mismo diagrama.

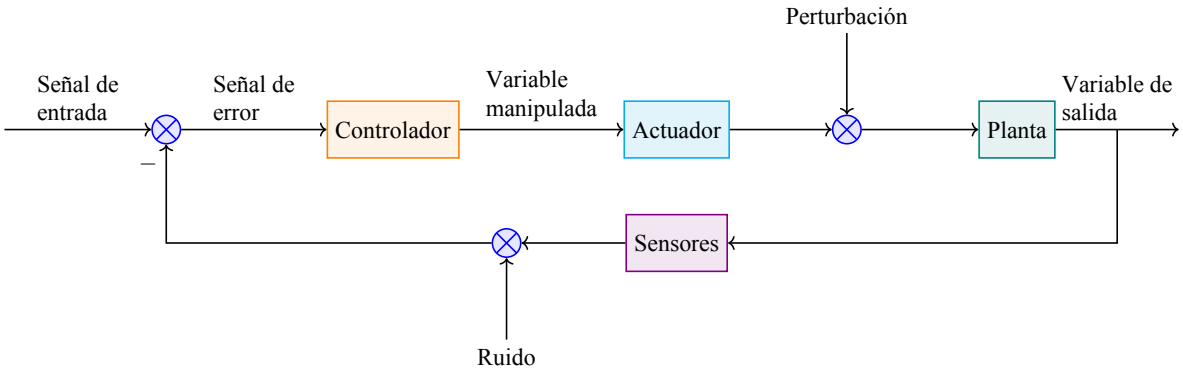


Figura 2.21: Diagrama de un sistema de control genérico

2.3.2 Tipos de control

2.3.2.1 Control pasivo

Aquellas estrategias de control de una variable que no requiera una entrega de energía a la planta durante la evolución del sistema dinámico. Por ejemplo, maquinados que mejoren el perfil aerodinámico de un vehículo, disipadores de calor de aluminio, etc. Siempre que puedan ser utilizados es conveniente hacerlo ya que reducen la complejidad del sistema.

[8]Realmente, este bloque devuelve las pocisiones y velocidades, tanto lineales como angulares. No obstante, en un ambiente de simulación como lo es el software Simulink de Matlab, esto consiste simplemente en agregar bloques integradores $\frac{1}{s}$ a la salida de cada aceleración.

2.3.2.2 Control activo

Son aquellas estrategias que requieren la entrega de energía durante la evolución del sistema dinámico a través de un elemento controlador. Estos sistemas activos pueden clasificarse a su vez en *lazo abierto* y *lazo cerrado*.

Lazo abierto

Son aquellos sistemas que trabajan sin información sobre el estado de la variable de salida. Esto hace al sistema simple de implementar, pero propenso a funcionar mal ante perturbaciones y errores o simplificaciones empleadas en el modelo físico en el cual se basó la construcción del controlador.

La Figura 2.22 muestra el diagrama de un sistema de control a lazo abierto. La idea general es que, el controlador en este caso es una persona encargada de manipular los actuadores para que la respuesta de la planta sea lo más parecida posible a los valores deseados.

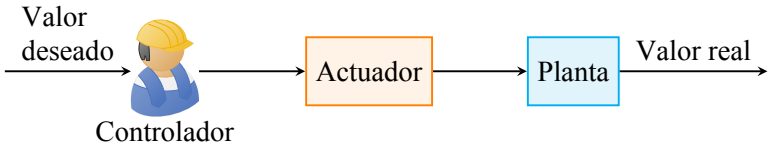


Figura 2.22: Sistema de control a lazo abierto

Lazo cerrado o basados en sensores

Son aquellos sistemas que funcionan mediante la medición del estado de alguna variable. Su implementación es la que más estudio conlleva, tanto por el comportamiento intrínseco de un sistema realimentado (estabilidad, velocidad, precisión, etc) como del estudio de los sensores, ya que si bien se puede asumir que los mismos van a medir las variables de forma idónea, esto es solo una suposición. A pesar de esta complejidad, son los más empleados debido a sus buenos resultados ante imperfecciones del modelo, perturbaciones y porque dan los mejores resultados en cuanto a velocidad y estabilidad.

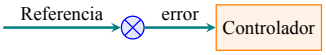
Estos sistemas tienen la particularidad de poseer una señal de error y una variable medida. Dependiendo de cuál sea la variable medida, se pueden clasificar estos sistemas en

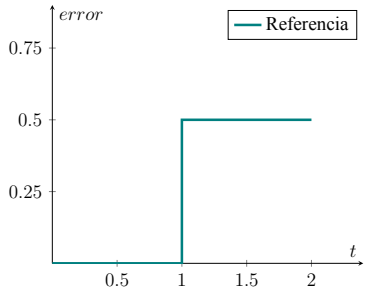
- Prealimentados (Feedforward).
- Retroalimentados (Feedback).

Como se vio en la sección 2.3.1.2, cualquier señal que produzca error provocará una reacción en el controlador. A continuación se estudia cuales son las señales[9] que componen al error[10].

1. Cuando la referencia cambia, el sistema no tiene tiempo suficiente para reaccionar de manera que se traduce instantáneamente en error.

[9]Las curvas graficadas de las señales *referencia*, *perturbación* y *ruido* son ejemplos para explicar el concepto.
[10]En las figuras Figura 2.23a, Figura 2.24a y Figura 2.25a, a modo de simplificación, se diagraman únicamente los bloques de interés para cada caso.



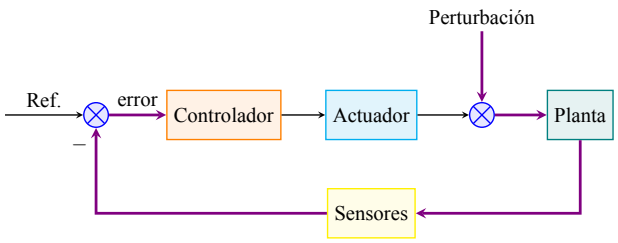


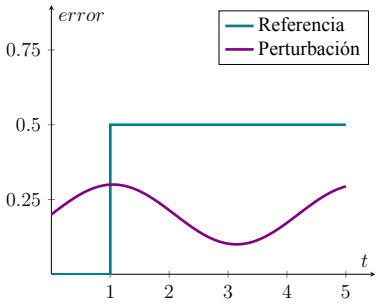
(a) Diagrama en bloques del error en función de la referencia

(b) Gráfico del error en función de la referencia

Figura 2.23: Error en función de la referencia

2. Cuando una perturbación afecta al sistema, lo hace directamente en la salida de este. Luego, esto es medido por los sensores y traducido en error.



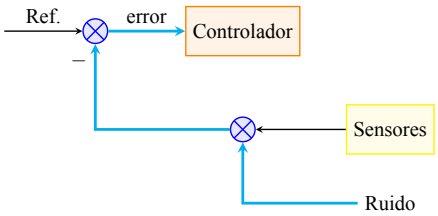


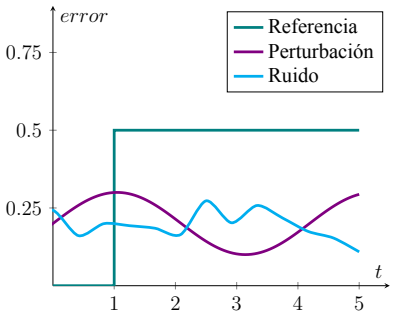
(a) Diagrama en bloques del error en función de la perturbación

(b) Gráfico del error en función de la perturbación

Figura 2.24: Error en función de la perturbación

3. Por último, la última señal capaz de producir error es el ruido en los sensores, afectando así las mediciones de los estados del sistema.





(a) Diagrama en bloques del error en función del ruido

(b) Gráfico del error en función del ruido

Figura 2.25: Error en función del ruido

Es de interés que el controlador reaccione para corregir la referencia y las perturbaciones.

Sin embargo, dado que lo único que hace el ruido es afectar negativamente la percepción de la realidad del sistema, es importante que lo ignore.

En primera instancia, el problema radica en que las tres señales, *referencia*, *perturbación* y *ruido*, ingresan al controlador al mismo tiempo, como se observa en la Figura 2.26. Consecuentemente, este ve la suma de todas ellas de manera que es incapaz de tratarlas por separado.

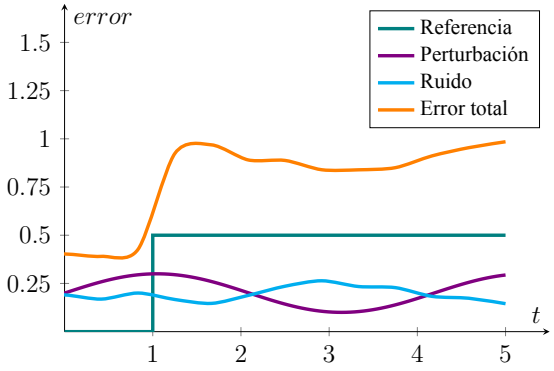
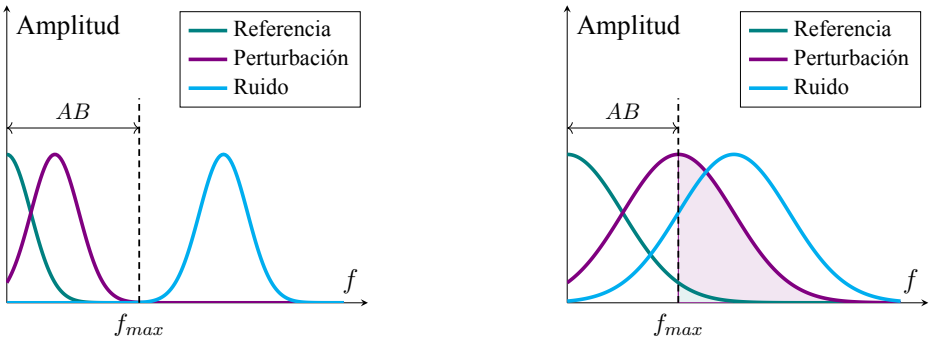


Figura 2.26: Error total

En esta situación, una posible solución es analizar la frecuencia de las tres señales que producen error y tratar de aislarlas, para así trabajar con cada una individualmente. Luego, en la Figura 2.27a, se grafica un primer caso en donde la frecuencia de la señal de ruido es considerablemente mayor con respecto a las otras dos señales. En este caso, se puede eliminar el ruido simplemente ajustando la frecuencia máxima del controlador, f_{max}[11].



(a) Análisis en frecuencia - Ruido de alta frecuencia

(b) Análisis en frecuencia - Ruido de baja frecuencia

Figura 2.27: Análisis en frecuencia de las señales que producen error

Ahora bien, en la Figura 2.27b está graficada la situación en la cual la frecuencia del ruido es semejante a las otras dos señales. Si bien se puede quitar ruido del sistema ajustando f_{max}, también se vería afectada la capacidad del controlador de responder a las señales de referencia y perturbación[12], lo cual no es deseable.

[11] El controlador sólo responderá frente a señales cuya frecuencia se encuentre dentro su ancho de banda, AB.

[12] Nótese las áreas coloreadas debajo de las curvas de las señales de referencia y perturbación, luego esos son los valores frente a los cuales el controlador no sería capaz de reaccionar.

Esta es una situación en la cual resulta útil implementar un controlador prealimentado. La idea de este es mitigar el error producido por la referencia y la perturbación antes de que lleguen al controlador.Así, el controlador solo vería el ruido, tal que se podría modificar f_{max} sin preocuparse por la referencia y perturbación, como denota la Figura 2.28.

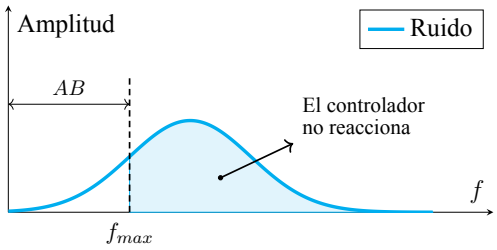


Figura 2.28: Análisis en frecuencia - Filtrar ruido únicamente

Por último, en la Figura 2.29 se visualiza el diagrama en bloques que implementa controladores de tipo *feedforward* para aislar la referencia y perturbación, y un controlador de tipo *feedback* para el ruido.

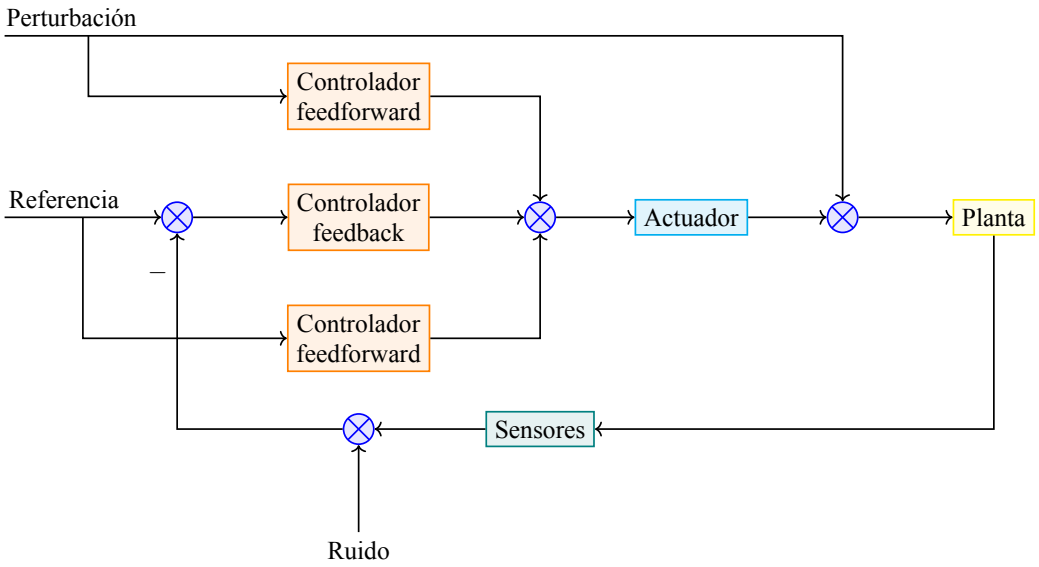


Figura 2.29: Diagrama de un sistema de control con controladores feedforward y feedback

2.3.3 Conceptos

2.3.3.1 Estabilidad

Si se implementa, por ejemplo, un control únicamente para la altura. Luego, diversas perturbaciones como podría ser el viento, pueden modificar el ángulo de roll y/o pitch provocando que el cuadricóptero se incline hacia un lado. Entonces, dado que sí existe un control para la altura, la fuerza de empuje (thrust) va a aumentar para que no pierda altitud, pero esto también dará como resultado una fuerza horizontal hacia la dirección y sentido de la inclinación.

El resultado es que, si bien la altura se mantiene en el valor deseado, el cuadricóptero se desplazaría indeseadamente hacia los lados. Por tanto, se tiene que aplicar control tanto para el ángulo *Roll* como a *Pitch*.

2.3.3.2 Velocidad

Es de interés que el tiempo de respuesta del cuadricóptero sea el menor posible y, para ello, el sistema de control tendrá que aplicar las correcciones de cada variable de estado lo más rápido posible. Sin embargo, el hecho de enfocarnos únicamente en la velocidad de respuesta provoca que los movimientos del cuadricóptero sean demasiado bruscos. En esencia, la estrategia será modificar, en base a pruebas, los parámetros del sistema de control para obtener así una respuesta relativamente rápida, asegurando movimientos suaves y controlados.

2.3.3.3 Oscilación

La consecuencia que conlleva un tiempo de respuesta muy pequeño es la oscilación en la curva de respuesta del cuadricóptero. Es decir, a medida que se ajustan los parámetros para que este responda rápidamente, las oscilaciones serán cada vez mayores y, aunque sigan siendo amortiguadas, es factible que se alejen considerablemente del valor deseado pudiendo provocar, en conjunto con la oscilación de otras variables, maniobras indeseadas. Por ello, una buena práctica es limitar dentro de un rango los valores pico que pueden tener las respuestas de cada estado.

2.3.4 Controlador PID

El algoritmo utilizado en este trabajo será el controlador proporcional, integral derivativo (PID), que funciona de la siguiente manera.

1. Se toma la señal de error como $e = x_{ref} - x_{medida}$.
2. Se calcula la derivada del error (s en el dominio de Laplace).
3. Se calcula la integral del error ($\frac{1}{s}$ en el dominio de Laplace).
4. Se calcula el producto del error por una ganancia, k_p.
5. Se calcula el producto de la integral del error por una ganancia, k_i.
6. Se calcula el producto de la derivada del error por una ganancia, k_d.
7. Se obtiene la señal de control como la suma de los 3 términos, $u = u_p + u_i + u_d$.

Gráficamente, se puede describir el algoritmo como:

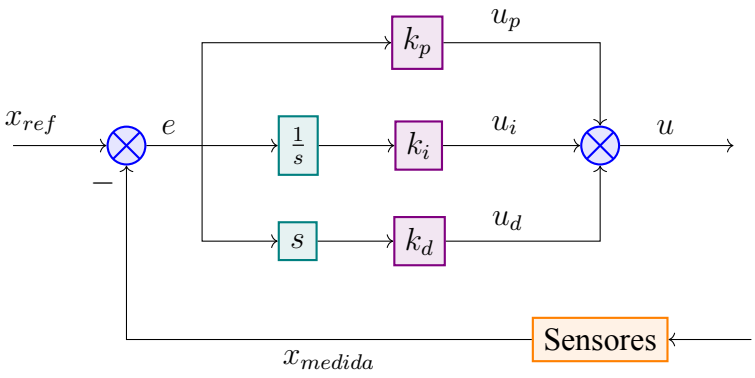


Figura 2.30: Diagrama de control PID

También se puede reducir el diagrama a un solo bloque PID, como se puede ver en la Figura 2.31.

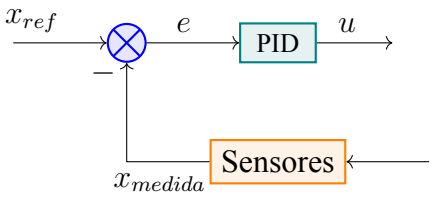


Figura 2.31: Bloque de control PID simplificado

Este es un algoritmo basado en error, ya que genera una señal de control únicamente a partir del error presente en el sistema, y no su modelado matemático. Esto lo hace sencillo de implementar en software, necesitando únicamente calcular productos, derivadas e integrales numéricas. Sin embargo, también agrega un paso extra al momento de diseñarlo conocido como sintonización o tuneado, el cual es el proceso de determinar los valores óptimos de cada ganancia, k_p, k_i, k_d, a utilizar. Para ello, existen métodos analíticos que consisten en analizar cómo reaccionaría la planta frente a diversos estímulos, así como también métodos empíricos que pueden dar una guía, pero principalmente se emplea la prueba y error a partir de un modelado matemático de la planta.

2.3.5 Filtro complementario

En primera instancia, no se dispone de sensores que midan directamente la posición angular, es decir, el estado de la planta. No obstante, tanto el giróscopo como el acelerómetro son capaces de medir una magnitud física que se relaciona con la posición angular. Entonces, de esta manera es posible medir el estado de la planta de forma indirecta.

El giróscopo mide velocidad angular, tal que para obtener la posición angular hay que recurrir a la integral. Así, el algoritmo que se utilizó es un método de integración numérico conocido como “Método de Euler hacia delante” que consiste en multiplicar la medición del giróscopo por el tiempo de muestreo y sumarle la última estimación del ángulo en cuestión.

$$\mathrm{y}[n] = \mathrm{y}[n-1] + gyro \cdot \Delta t$$

Donde y es la posición angular medida indirectamente.

Por otra parte, el acelerómetro mide las aceleraciones lineales en x, y, z tal que la rotación angular sobre cada eje puede determinarse haciendo uso de relaciones trigonométricas.

$$\theta\ (pitch) = \arctan\left(\frac{-A_{CC_X}}{\sqrt{A_{CC_Y}^2 + A_{CC_Z}^2}}\right)$$

$$\phi\ (roll) = \arctan\left(\frac{A_{CC_Y}}{\sqrt{A_{CC_X}^2 + A_{CC_Z}^2}}\right)$$

Entonces, una vez calculadas ambas mediciones indirectas de la posición angular, se requiere de algún algoritmo para combinarlas en función de qué tanto se desee que cada una aporte al

valor final obtenido. Un algoritmo popular, e implementado en el firmware que se desarrolló, es el filtro complementario. Su ventaja radica en el bajo costo de procesamiento y los buenos resultados que ofrece.

Este proceso se ve gráficamente en la Figura 2.32, donde z^{-1} representa un retraso de una muestra $[n-1]$ y $k \in [0,1]$ representa el aporte de cada sensor sobre el valor final del ángulo obtenido.

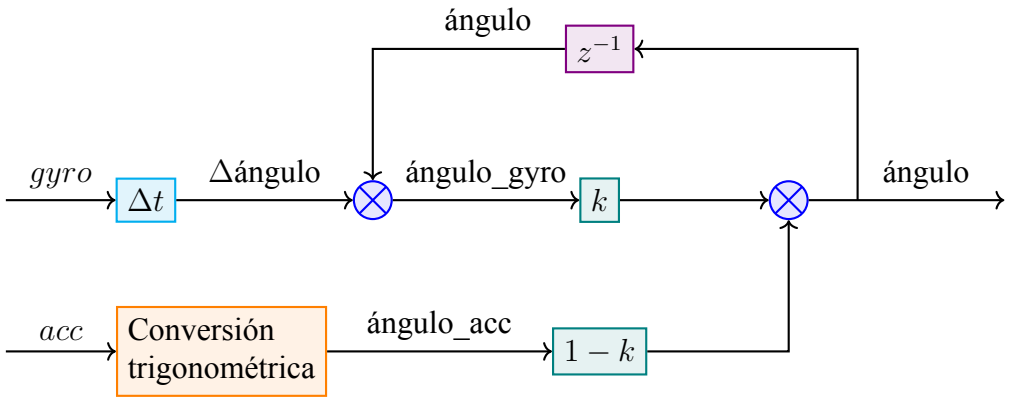


Figura 2.32: Diagrama en bloques de un filtro complementario para estimar posición angular

El acelerómetro presenta ruido de manera que sus mediciones en el corto plazo no son tan precisas. Asimismo, el giróscopo también presenta ruido, tal que al integrar sus mediciones se agrega una desviación creciente con respecto al ángulo real que se acumula a largo plazo. Por tanto, al combinar ambos sensores se logran mitigar las desviaciones introducidas por el acelerómetro así como también eliminar la desviación introducida por el hecho de integrar el giróscopo.

Por último, mencionar que *yaw* puede ser estimado integrando la velocidad angular medida por el giróscopo.

2.4 Filtros digitales

El objetivo de esta sección es dar una visión general de qué son los filtros digitales, para qué se utilizan y cuáles son sus principales características.

En primera instancia, la problemática radica en que los sensores introducen ruido en el sistema, que entra directamente en los controladores y empeora su respuesta ya que provoca una visión errónea de los estados. Por ello, el objetivo será desarrollar un algoritmo que lo mitigue y logre un resultado similar al de la Figura 2.33.

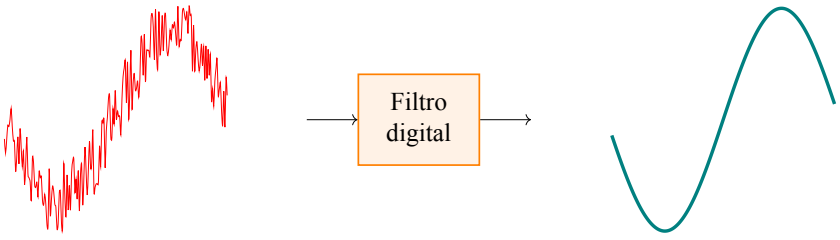


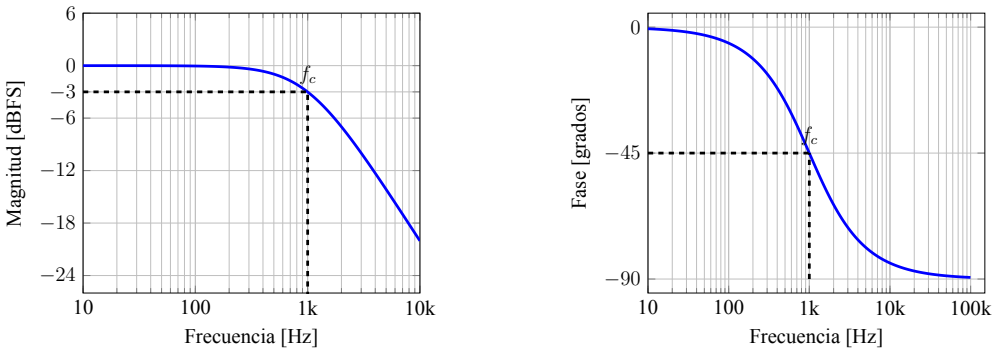
Figura 2.33: Función de un filtro digital

Puesto que el filtro es digital, significa que la señal de entrada tendrá que ser muestreada cada cierto tiempo $t_s = \frac{1}{f_s}$. Además, a partir del teorema de Nyquist (Alan V. Oppenheim, s.f., Sec. 7.1, p. 515) se sabe que para poder reconstruir la señal de entrada en su totalidad se debe imponer que $f_s \geq 2f$, siendo f la frecuencia de la señal de entrada.

Antes de continuar, se mencionarán algunas definiciones en el ámbito de las señales discretas para facilitar la comprensión de las sub-secciones siguientes. En el dominio temporal, la señal sinusoidal, sin desfasaje, se define como $f(t) = \sin(\omega t), t \in \mathbb{R}$ con $[w] = \frac{rad}{s}$. Sin embargo, los dispositivos solo son capaces de asignarle valores a las funciones cada un cierto tiempo. Por tanto, en el dominio discreto, la señal sinusoidal se define como $x[n] = \sin(\omega n), n \in \mathbb{N}$ con $[w] = \frac{rad}{muestra}$. Entonces, si en el dominio temporal un ciclo completo ocurre cuando $\omega T = 2\pi$, en el dominio discreto ocurrirá cuando $\omega N = 2\pi \Leftrightarrow \omega = \frac{2\pi}{N}$, siendo N la cantidad de muestras tomadas en un ciclo completo de la señal. Asimismo, N relaciona la frecuencia física y la frecuencia de muestreo de la siguiente manera, $N = \frac{f_s}{f}$ tal que es posible reescribir la señal discreta como $x[n] = \sin(2\pi \frac{f}{f_s} \cdot n)$, donde $\omega = 2\pi \frac{f}{f_s}$.

El último concepto presentado es la frecuencia de Nyquist, que ocurre cuando $f = f_N = \frac{f_s}{2}$ tal que $\omega_N = 2\pi \frac{f_s}{2f_s} = \pi$.

Ahora bien, lo relevante a la hora de analizar los filtros digitales es su respuesta a medida que varía la frecuencia. Por ello, un método para analizar la respuesta en frecuencia de un sistema es a través de los diagramas de Bode. Estos permiten visualizar el efecto que un sistema produce sobre la magnitud (Figura 2.34a) y fase (Figura 2.34b) de la señal de entrada.



(a) Diagrama de Bode - Magnitud

(b) Diagrama de Bode - Fase

Figura 2.34: Diagramas de Bode para un filtro pasa bajos de primer orden

La Figura 2.34 representa los diagramas de Bode para un filtro pasa bajos de primer orden cuya función de transferencia es $H(s) = \frac{1}{\tau s + 1}, s = j\omega = j2\pi f$. La frecuencia de corte del filtro, f_c, se define como aquel valor que provoca una atenuación en la señal de entrada de $-3dB$. Luego, el rango $f < f_c$ se denomina "banda de paso", mientras que el rango $f > f_c$ se denomina "banda de supresión". La idea básica es que todas aquellas señales cuya frecuencia se encuentre dentro de la banda de paso ingresarán al sistema sin verse mayormente afectadas por el filtro. En contraste, aquellas señales cuya frecuencia se encuentre contenida en la banda de supresión se verán gradualmente atenuadas.

A continuación se presentan algunas topologías básicas de filtros digitales y sus principales características.

2.4.1 Filtro FIR

En la Figura 2.35 se puede apreciar el diagrama en bloques de un filtro FIR genérico de primer orden.

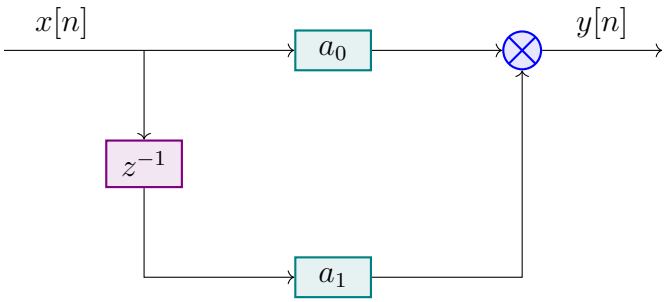


Figura 2.35: Diagrama en bloques de un filtro FIR de primer orden

A esta topología se la describe como “Feed Forward” puesto que el camino de la entrada hacia la salida es directo, Por otra parte su ecuación en diferencias se expresa como

$$y[n] = a_0 x[n] + a_1 x[n-1]$$

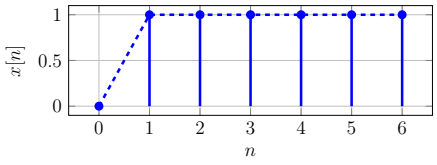
Continuando, se desarrollarán dos métodos para analizar la respuesta en frecuencia del filtro, el primero es uno experimental y el segundo es analítico.

2.4.1.1 Método experimental

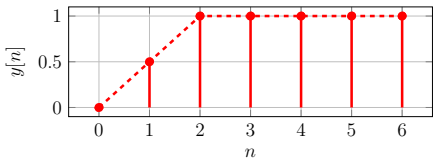
Este método consiste en inyectar distintas señales de prueba, y observar la señal de salida. Luego, se hará uso de los datos obtenidos para trazar el diagrama de Bode característico del filtro. Debajo se encuentra el análisis para cada caso, donde se definió que $a_0 = a_1 = 0.5$.

x[n]	y[n]
0	0
1	0.5
1	1
1	1
1	1
1	1
1	1

Tabla I: Respuesta del filtro FIR frente a una señal DC



(a) Señal de entrada $x[n]$



(b) Señal de salida $y[n]$

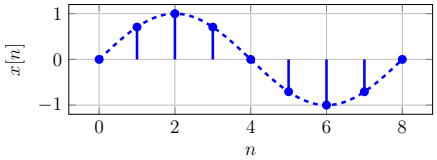
Figura 2.36: Respuesta del filtro FIR frente a una señal DC

La primera señal inyectada será una señal DC con frecuencia nula (Figura 2.36). Nótese que la respuesta presenta un retardo de una muestra en alcanzar el valor final 1, esto se debe precisamente a que únicamente hay un delay, z^{-1}, presente en el sistema.

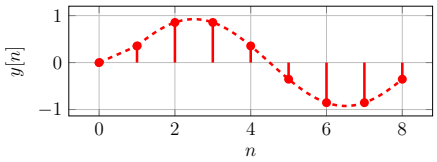
A continuación, en la Figura 2.37 se puede apreciar el efecto de inyectar la señal 1/4 de Nyquist. Primero, se remarca el hecho de que la señal tarda 8 muestras en volver a su valor inicial tal como lo expresan la columna $x[n]$ de la Tabla II y la Figura 2.36a. Además, recordando que $\omega = \frac{2\pi}{N} = \frac{2\pi}{8} = \frac{\pi}{4} = \frac{\omega_N}{4}$, de ahí que se denomine "1/4" de Nyquist. Luego, a partir de la Figura 2.36b se puede observar que la amplitud se redujo un $100\,\% - 100\,\%\frac{0.924}{1} = 7.6\,\%$. Por otra parte, la señal de entrada $x[n]$ se puede interpolar según la función $f(x) = \sin(\frac{\pi}{4}x)$ y la de salida $y[n]$ con $g(x) = \sin(\frac{\pi}{4}x) - \frac{\pi}{8}$ tal que el desfasaje entre ellas es de $g(x = 0) - f(x = 0) = \frac{\pi}{8} - 0 = \frac{\pi}{8} = 22.5°$.

x[n]	y[n]
0	0
0.707	0.354
1	0.854
0.707	0.854
0	0.354
-0.707	-0.354
-1	-0.854
-0.707	-0.854
0	-0.354

Tabla II: Respuesta del filtro FIR frente a la señal de 1/4 Nyquist



(a) Señal de entrada $x[n]$, $f = \frac{f_s}{8}$



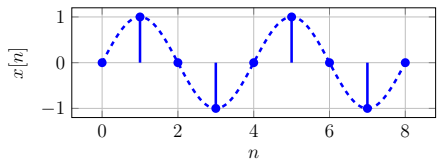
(b) Señal de salida $y[n]$ con $|y_{max}| \approx 0.924$

Figura 2.37: Respuesta del filtro FIR frente a la señal de 1/4 Nyquist

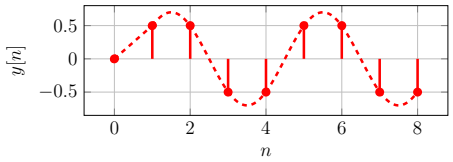
En la Figura 2.38 se repite el proceso para la señal 1/2 de Nyquist, de 4 muestras por ciclo, tal que la amplitud se reduce un $100\,\% - 100\,\%\frac{0.7}{1} = 30\,\%$. Por otra parte, la entrada $x[n]$ se puede interpolar a través de la función $f(x) = \sin(\frac{\pi}{2}x)$ y la salida con $g(x) = \sin(\frac{\pi}{2}x) - \frac{\pi}{4}$. Entonces, el desfasaje entre estas es de $g(x = 0) - f(x = 0) = \frac{\pi}{4} - 0 = \frac{\pi}{4} = 45°$.

$x[n]$	y[n]
0	0
1	0.5
0	0.5
-1	-0.5
0	-0.5
1	0.5
0	0.5
-1	-0.5
0	-0.5

Tabla III: Respuesta del filtro FIR frente a la señal de 1/2 Nyquist



(a) Señal de entrada $x[n]$, $f = \frac{f_s}{4}$



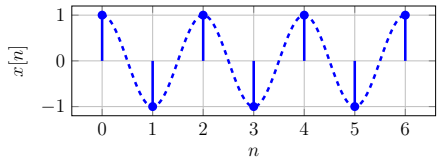
(b) Señal de salida $y[n]$ con $|y_{max}| \approx 0.7$

Figura 2.38: Respuesta del filtro FIR frente a la señal de 1/2 Nyquist

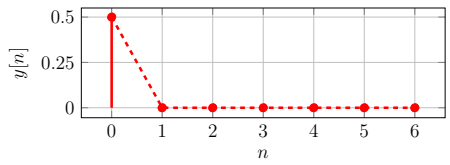
Inyectando la señal de Nyquist, de 2 muestra por ciclo, (Figura 2.39) al sistema se observa un cambio dramático en la salida, tal que esta cae abruptamente a 0 a partir de la segunda muestra en adelante, dando lugar a la gráfica de la Figura 2.38b.

x[n]	y[n]
1	0.5
-1	0
1	0
-1	0
1	0
-1	0
1	0

Tabla IV: Respuesta del filtro FIR frente a la señal de Nyquist



(a) Señal de entrada $x[n]$, $f = \frac{f_s}{2}$

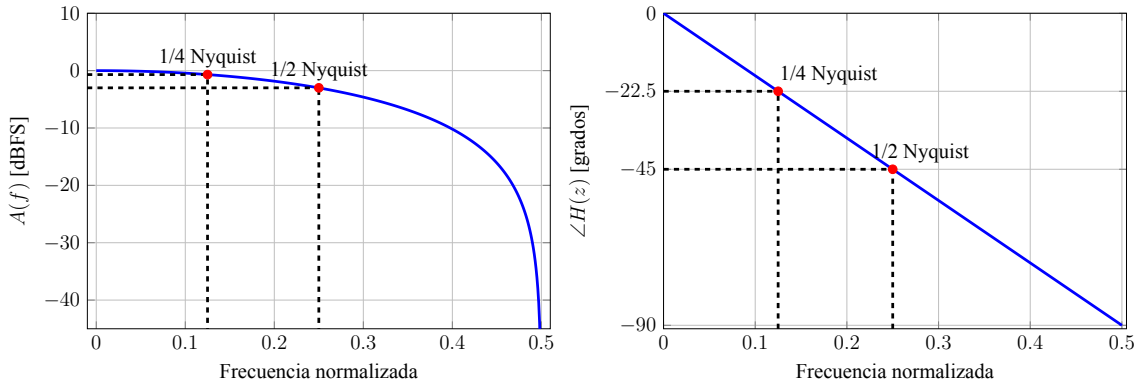


(b) Señal de salida $y[n]$

Figura 2.39: Respuesta del filtro FIR frente a la señal de Nyquist

El orden en el cual fueron apareciendo las distintas señales de prueba no es aleatorio. Nótese que a medida que se fueron graficando, estas cada vez iban aumentando más su frecuencia. El resultado obtenido es una disminución gradual en la amplitud de la señal de salida y un desfasaje también gradual, ambos hasta alcanzar una amplitud nula para la señal de Nyquist y un desfasaje total de -90°. Entonces, con los datos obtenidos es posible trazar el diagrama de Bode característico del filtro FIR de primer orden[13], donde se ha utilizado una frecuencia de muestreo de 1kHz.

[13] Se han realizado ambas gráficas con el eje x normalizado con respecto a la frecuencia de muestreo f_s puesto que la respuesta es relativa a este valor de frecuencia.



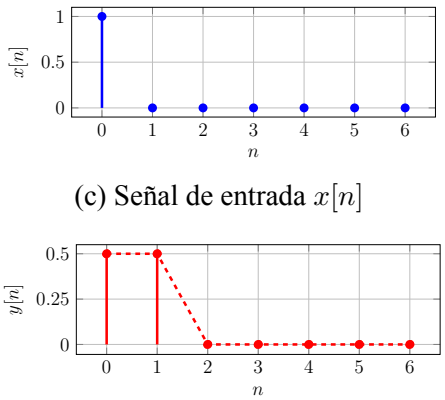
(a) Magnitud $A(f)$ (b) Fase $\angle H(z)$

Figura 2.40: Diagramas de Bode para un filtro FIR de primer orden

Se obtuvo así la respuesta en frecuencia del filtro FIR; no obstante, no se ha comentado nada acerca de su nombre. Para ello, se analiza su respuesta frente a un impulso unitario como el de la Figura 2.40c.

x[n]	y[n]
1	0.5
0	0.5
0	0
0	0
0	0
0	0
0	0

Tabla V: Respuesta del filtro FIR frente a un impulso



(c) Señal de entrada $x[n]$

(d) Señal de salida $y[n]$

Figura 2.41: Respuesta del filtro FIR frente a un impulso

Como demuestra la Figura 2.41, la respuesta frente a un impulso presenta dos valores no nulos, es decir, los dos primeros valores de 0.5 (Figura 2.40d). Por tanto, puesto que la cantidad de elementos no nulos es exacta (2), se denomina al filtro como “Respuesta frente a un Impulso Finita” o FIR.

2.4.1.2 Método analítico

Este método consiste en obtener la ecuación en diferencias del filtro, $y[n]$, y hallar una ecuación para su función de transferencia, $\frac{Y(z)}{X(z)}$.

Analizando la Figura 2.35 es sencillo llegar a la conclusión de que $y[n] = a_0x[n]+a_1x[n-1]$. Luego, aplicando la transformada Z (Alan V. Oppenheim, s.f., Cap. 10) se obtiene

$$Y(z) = a_0X(z) + a_1X(z)z^{-1}$$

despejando para obtener la función de transferencia se obtiene que,

$$H(z) = \frac{Y(z)}{X(z)} = a_0 + a_1 z^{-1}$$

Ahora bien, puesto que z es una variable compleja tal que $z = e^{j\omega},\ \omega \in \mathbb{R}$ y, recordando la fórmula de Euler $e^{-j\omega} = \cos(\omega) - j\sin(\omega)$, se reemplaza y obtiene

$$H(z) = a_0 + a_1 e^{-j\omega} = a_0 + a_1(\cos(\omega) - j\sin(\omega))$$

Separando la parte real y la imaginaria se obtiene

$$H(z) = \underbrace{a_0 + a_1\cos(\omega)}_{\text{Parte real}} - j\ \underbrace{a_1\sin(\omega)}_{\text{Parte imaginaria}}$$

Continuando, para obtener la función de transferencia que describa la variación de la amplitud con la frecuencia se debe determinar $|H(z)| = \sqrt{a^2 + b^2}$.

$$|H(z)| = \sqrt{\Big(a_0 + a_1\cos(\omega)\Big)^2 + \Big(a_1\sin(\omega)\Big)^2}$$

Desarrollando ambos términos,

$$|H(z)| = \sqrt{(a_0^2 + 2a_0a_1\cos(\omega) + a_1^2\cos^2(\omega) + a_1^2\sin^2(\omega)}$$

Utilizando la identidad trigonométrica $1 = \cos^2(\omega) + \sin^2(\omega)$ y recordando que los coeficientes del filtro se definieron como $a_0 = a_1 = 0.5$

$$|H(z)| = \sqrt{\frac{1 + \cos(\omega)}{2}}$$

Se utiliza otra propiedad trigonométrica que establece que, $\dfrac{1 + \cos(2w)}{2} = \cos^2(w)$

$$|H(z)| = \sqrt{\cos^2(\frac{\omega}{2})} = \cos(\frac{\omega}{2})$$

Por último, recordando del comienzo de la sección que la frecuencia digital se puede reescribir como $\omega = 2\pi\dfrac{f}{f_s}$,

$$|H(z)| = A(f) = \cos\left(\frac{2\pi f}{2f_s}\right) = \cos\left(\pi\frac{f}{f_s}\right)$$

$$\boxed{A(f) = \cos\left(\frac{2\pi f}{f_s}\right)} \tag{2.5}$$

Luego, para determinar la fase se tiene que hallar una ecuación para $\angle H(z) = \arctan\left(\dfrac{b}{a}\right)$

$$\boxed{\angle H(z) = \arctan\left(\frac{-\sin(\frac{2\pi f}{f_s})}{1 + \cos(\frac{2\pi f}{f_s})}\right)} \tag{2.6}$$

El resultado de graficar ambas ecuaciones son los gráficos de la Figura 2.40.

2.4.2 Filtro IIR

En la Figura 2.42 se puede observar el diagrama genérico de un filtro IIR de primer orden.

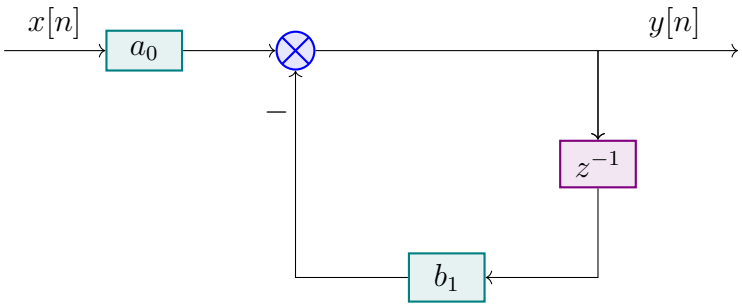


Figura 2.42: Diagrama en bloques de un filtro IIR de primer orden

A este tipo de filtros se los denomina “Feed Back” puesto que la salida se realimenta en el sistema. Adicionalmente, la ecuación en diferencias queda determinada como

$$y[n] = a_0 x[n] - b_1 y[n-1]$$

Para este filtro, únicamente se realizará un análisis analítico puesto que la metodología de la forma experimental ya quedó clara en la sección previa. Luego, se continuará hallando una expresión para las funciones de transferencia tanto para la amplitud como para la fase.

Llevando la ecuación en diferencias al dominio Z se encuentra que,

$$Y(z) = a_0 X(z) - b_1 Y(z) z^{-1}$$

tal que la función de transferencia queda expresada como

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0}{1 + b_1 z^{-1}} = \frac{a_0}{1 + b_1(\cos(\omega) - j\,\sin(\omega))} = \frac{a_0}{\Big(1 + b_1\cos(\omega)\Big) - j\Big(b_1\sin(\omega)\Big)}$$

En estos casos, una forma de eliminar la parte imaginaria del denominador y obtener un número complejo de la forma $z = a + jb$, es multiplicando tanto el numerador como el denominador por el conjugado de este último. A saber,

$$H(z) = \frac{a_0}{\Big(1 + b_1\cos(\omega)\Big) - j\Big(b_1\sin(\omega)\Big)} \cdot \frac{\Big(1 + b_1\cos(\omega)\Big) + j\Big(b_1\sin(\omega)\Big)}{\Big(1 + b_1\cos(\omega)\Big) + j\Big(b_1\sin(\omega)\Big)}$$

Desarrollándose, se obtiene,

$$H(z) = \underbrace{\frac{a_0(1 + b_1\cos(\omega))}{1 + 2b_1\cos(\omega) + b_1^2}}_{\text{Parte real}} + j\underbrace{\frac{a_0 b_1\sin(\omega)}{1 + 2b_1\cos(\omega) + b_1^2}}_{\text{Parte imaginaria}}$$

Trabajando la parte real e imaginaria por separado, se tiene que

$$a^2 = \frac{a_0^2(1 + 2b_1\cos(\omega) + b_1^2\cos^2(\omega))}{\Big(1 + 2b_1\cos(\omega) + b_1^2\Big)^2}$$

$$b^2 = \frac{a_0^2 b_1^2 \sin^2(\omega)}{\left(1 + 2b_1 \cos(\omega) + b_1^2\right)^2}$$

Finalmente, agrupando ambas expresiones para determinar $|H(z)| = \sqrt{a^2 + b^2}$, y realizando algunas simplificaciones se obtiene la expresión para la función de transferencia que expresa cómo varía la amplitud en función de la frecuencia, $A(f)$.

$$\boxed{A(f) = \sqrt{\frac{a_0^2}{1 + 2b_1 \cos\left(\frac{2\pi f}{f_s}\right) + b_1^2}}} \tag{2.7}$$

donde se ha reemplazado ω por $\frac{2\pi f}{f_s}$. Para el filtro FIR, la función de transferencia hallada para la amplitud (ecuación 2.5) es estable para cualquier valor de frecuencia y parámetros a_0, a_1. Sin embargo, para el filtro IIR esto no es así, ya que las raíces del denominador de la ecuación 2.7 son puntos singulares o inestables del sistema.

$$1 + 2b_1 \cos\left(\frac{2\pi f}{f_s}\right) + b_1^2 = 0$$

Analizando las situaciones clave donde el polinomio podría llegar a ser nulo, surgen los siguientes casos.

Caso 1)

Si $\cos(\omega) = 1$,

$1 + 2b_1 + b_1^2 = 0$

$(1 + b_1)^2 = 0$

$\boxed{b_1 = -1}$

Caso 2)

Si $\cos(\omega) = -1$,

$1 - 2b_1 + b_1^2 = 0$

$(1 - b_1)^2 = 0$

$\boxed{b_1 = 1}$

Caso 3)

Si $\cos(\omega) = 0$,

$1 + b_1^2 = 0$

$\nexists\, b \in \mathbb{R} \,/\, \{1 + b^2 = 0\}$

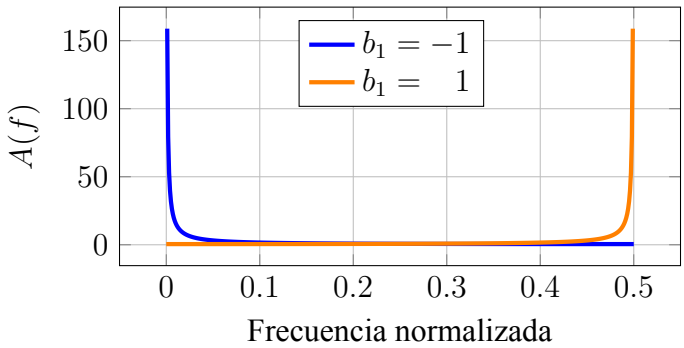


Figura 2.43: Singularidades de la ecuación 2.7

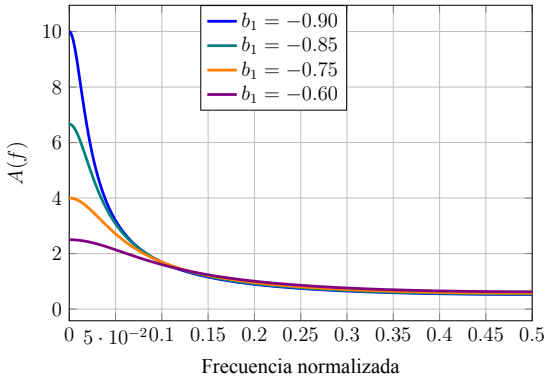
Luego, para hallar la función de transferencia que describe el comportamiento de la fase en función de la frecuencia, se hace uso de las expresiones halladas para la parte real e imaginaria de $H(z)$.

$$\angle H(z) = \arctan\left(\frac{b}{a}\right) = \arctan\left(\frac{b_1 \sin(\omega)}{1 + b_1 \cos(\omega)}\right)$$

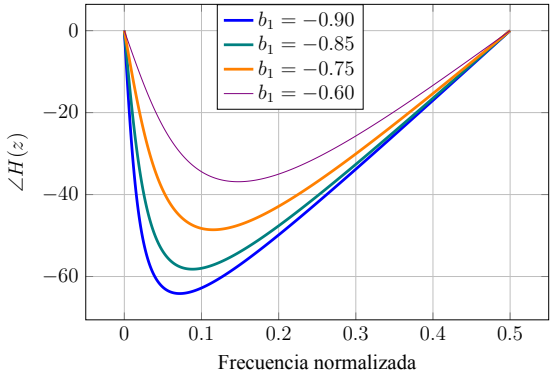
y reescribiendo ω se obtiene

$$\boxed{\angle H(z) = \arctan\left(\frac{b_1 \sin\left(\frac{2\pi f}{f_s}\right)}{1 + b_1 \cos\left(\frac{2\pi f}{f_s}\right)}\right)} \tag{2.8}$$

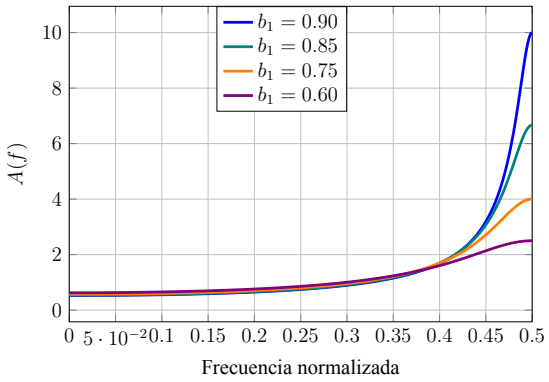
En la Figura 2.44 se encuentran graficadas la ecuación 2.7 y 2.8. Es importante remarcar que los valores de $b_1 < 0$ provocan que el filtro se comporte como un filtro pasa bajos, mientras que para $b_1 > 0$ el filtro se comporta como un filtro pasa altos.



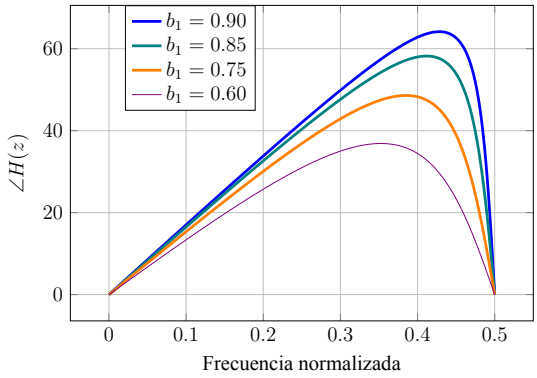
(a) Magnitud $A(f)$ con $a_0 = 1, b_1 < 0$



(b) Fase $\angle H(z)$ con $b_1 < 0$



(c) Magnitud $A(f)$ con $a_0 = 1, b_1 > 0$



(d) Fase $\angle H(z)$ con $b_1 > 0$

Figura 2.44: Diagramas de Bode para un filtro IIR de primer orden

Por último, al igual que para el filtro FIR, el nombre del filtro “IIR” está ligado a su respuesta frente a un impulso.

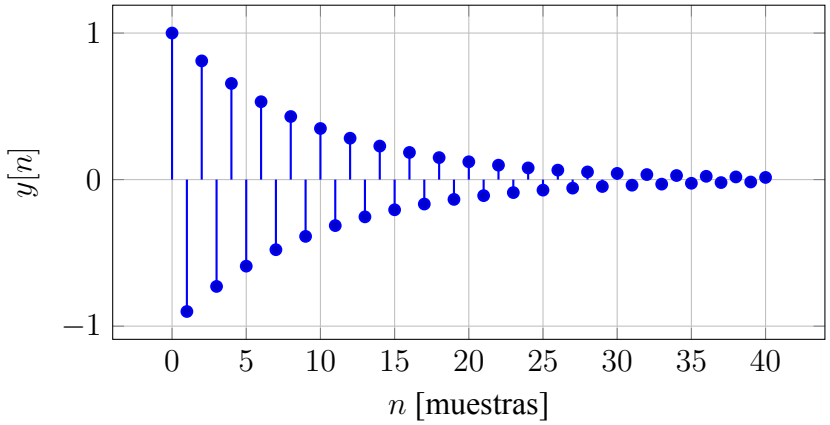


Figura 2.45: Respuesta al impulso con $a_0 = 1, b_1 = -0.9$

Luego, esta se encuentra graficada en la Figura 2.45 tal que se puede observar su comportamiento oscilatorio amortiguado. Sin embargo, a pesar de que la respuesta tiende a un valor estable, en este caso 0, nunca lo alcanza realmente. Es por ello que el filtro recibe el nombre de "Respuesta frente a un Impulso Infinita" o bien, "IIR".

Ahora que ya se han presentado ambas topologías es posible definir algunos aspectos relevantes de estos.

- Los filtros FIR introducen ceros en el sistema.
- Los filtros IIR introducen polos en el sistema.
- La respuesta de un filtro FIR es siempre estable.
- La respuesta de un filtro IIR es estable en cualquier punto exceptuando en sus polos.

Dada una función de transferencia genérica,

$$H(s) = \frac{Y(s)}{X(s)} = \frac{(s - z_1)(s - z_2)(s - z_n)}{(s - p_1)(s - p_2)(s - p_n)}$$

las raíces del numerador representan los ceros del sistema, mientras que las raíces del denominador representan los polos del sistema.

2.5 Comunicaciones

Para lograr que los distintos componentes del proyecto sean capaces de comunicarse, se usarán los protocolos detallados en esta sección.

2.5.1 Bluetooth

Tanto para enviar comandos como para controlar al cuadricóptero, se utiliza el protocolo Bluetooth Classic. Luego, este es un protocolo de comunicación inalámbrica de corta distancia (aproximadamente 10 metros) orientado a transmitir datos de manera continua. Además, su frecuencia de trabajo es de $2.4GHz$ dividiendo el espectro en 79 canales de $1MHz$.

2.5.2 I2C

I2C es el protocolo a utilizar en este proyecto para la comunicación entre el microcontrolador y los sensores. En concreto, es un protocolo de comunicación serial, síncrono, multidispositivo y semidúplex que permite la coexistencia de múltiples maestros y esclavos en el mismo bus. Adicionalmente, utiliza dos líneas bidireccionales en la configuración open drain, estas son la línea de datos serial (SDA) y la línea de reloj o clock serial (SCL)[14] activadas mediante resistencias de pull-up[15] (ver Figura 2.46). La ESP32 cuenta con 2 controladores I2C responsables de manejar la comunicación en el bus.

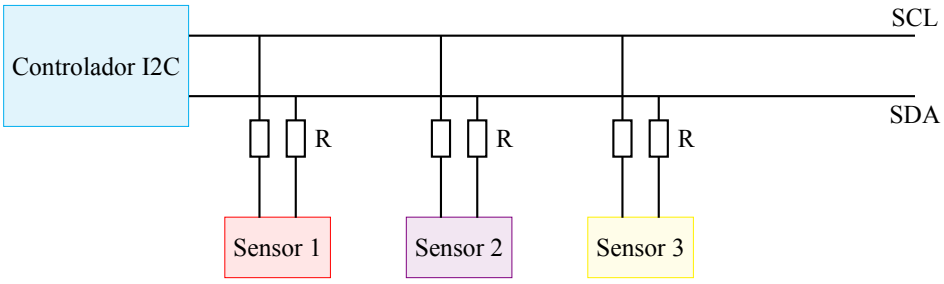


Figura 2.46: Bus maestro I2C con dispositivos conectados

2.5.2.1 Condiciones START y STOP

La comunicación es inicializada por el dispositivo maestro enviando la condicion de START, y finalizada por este al enviar la condición STOP (Valdez & Becker, 2015, Sec. 2.1.1, p. 4). Ambas condiciones se encuentran definidas en la Figura 2.47.

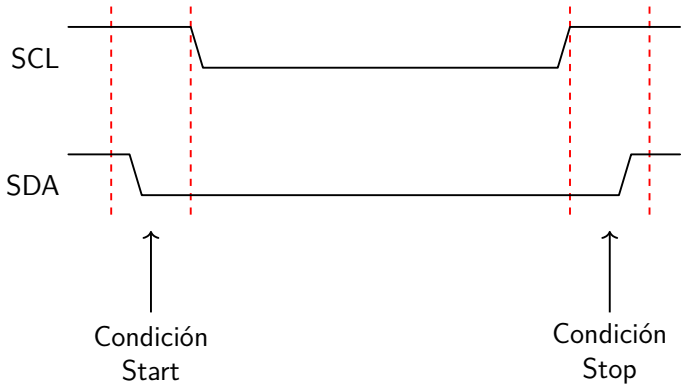


Figura 2.47: Condiciones START y STOP

y detalladas a continuación.

- START: Se define como la transición de la señal SDA del estado ALTO al estado BAJO, mientras la señal SCL se encuentra en el estado ALTO.
- STOP: Se define como la transición de la señal SDA del estado BAJO al estado ALTO, mientras la señal SCL se encuentra en el estado ALTO.

[14] La frecuencia del clock SCL en modo maestro no debe ser mayor a $400kHz$, según Espressif Systems, 2023.
[15] La ESP32 ya cuenta con resistores internos de pull-up, no obstante, en caso de utilizar resistencias adicionales se recomienda usar entre $2k\Omega$ a $5k\Omega$ (Espressif Systems, 2023).

2.5.2.2 Formato y lectura de un byte

Por otra parte, cada byte transferido a través de la señal SDA comienza por el MSB (*Most Significant Bit*) y finaliza por el LSB (*Least Significant Bit*). Por otra parte, cada bit transmitido es leído en un ciclo del reloj SCL (ver Figura 2.48).

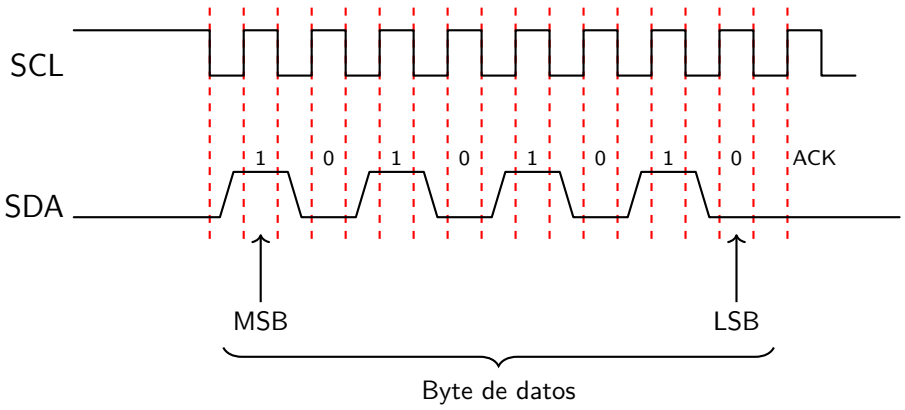


Figura 2.48: Transferencia de datos a través de I2C

2.5.2.3 Acknowledge (ACK) y Not Acknowledge (NACK)

Cada byte de datos, incluyendo el byte de dirección, es seguido por un bit denominado ACK (esto se puede visualizar en la Figura 2.48) enviado por el sensor. Este bit permite que el sensor le comunique al maestro que efectivamente recibió el byte.

Cuando la señal SDA permanece en el estado ALTO durante un evento ACK, este es interpretado como un NACK. Las posibles condiciones que llevan a la generación de un NACK son las siguientes.

1. El sensor es incapaz de recibir o transmitir datos debido a que se encuentra realizando cálculos en tiempo real y, por tanto, no puede comenzar una comunicación con el maestro.
2. Durante una transferencia de datos, el sensor recibe datos o comandos que es incapaz de interpretar.
3. Durante una transferencia de datos, el sensor no puede recibir más bytes.
4. El maestro finalizó la lectura de datos y se lo indica al sensor a través de un NACK.

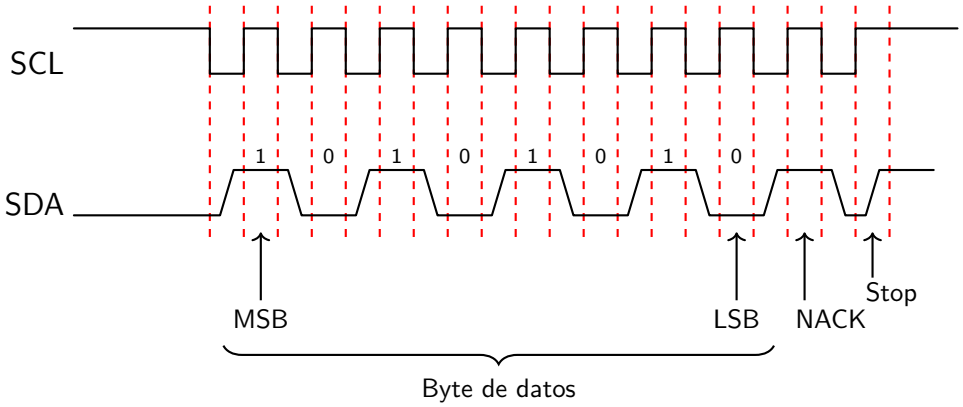


Figura 2.49: Diagrama NACK

2.5.2.4 Escritura

El maestro comienza enviando una condición de $START$ en el bus, junto con la dirección del sensor[16], con el cual desea comunicarse, y el último bit (el bit $R/\overline{W}$) seteado en 0. Este proceso se puede visualizar en la Figura 2.50.

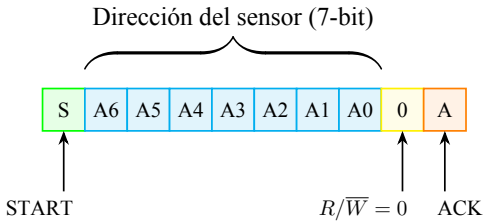


Figura 2.50: I2C Write - Dirección del sensor

Luego de que el sensor envía un ACK, el maestro continua enviando la dirección del registro al cual pretende escribir y el sensor responde de vuelta con otro ACK para confirmar que ya se encuentra preparado para recibir el byte de datos (ver Figura 2.51).

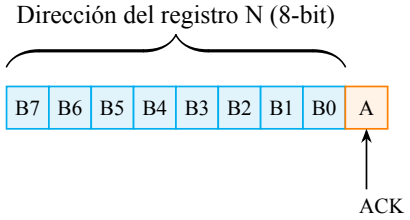


Figura 2.51: I2C Write - Dirección del registro

Entonces, este último envía los datos y espera a que el sensor envíe otro ACK de confirmación. Por último, el maestro envía una condición de $STOP$ indicando así, que finalizó la escritura de datos (ver Figura 2.52).

[16]La dirección del sensor puede ser de 10 o 7 bits. Referirse al datasheet del sensor utilizado.

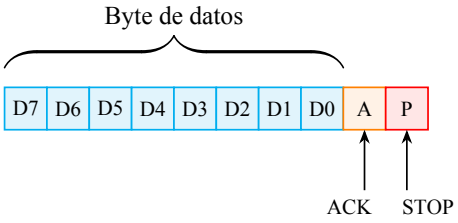


Figura 2.52: I2C Write - Byte de datos

El proceso completo se visualiza en la Figura 2.53. Nótese que el sensor únicamente es responsable de enviar los ACK al maestro.

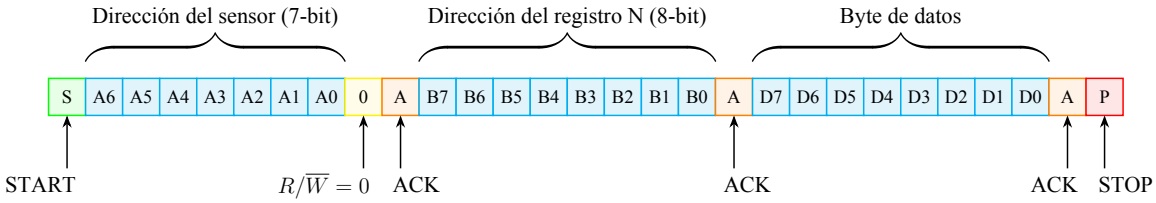


Figura 2.53: I2C Write - Proceso completo

2.5.2.5 Lectura

La primera parte del proceso de lectura es idéntica al de escritura, tal como muestra la Figura 2.54.

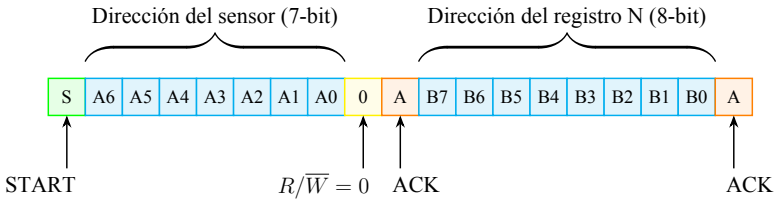


Figura 2.54: I2C Read - Primera parte del proceso

Una vez que el sensor envía un ACK avisando que ya esta preparado para recibir más datos, el maestro vuelve a enviar una condición de $START$ seguido de la dirección del sensor (Figura 2.55) y el bit $R/\overline{W}$ seteado en 1, indicando una operación de lectura. Por último, el sensor envía un ACK.

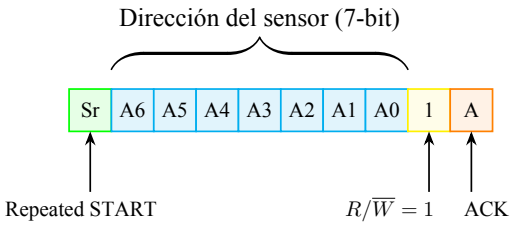


Figura 2.55: I2C Read - Condición $START$ repetida

Finalmente, el sensor toma el control de la línea SDA y envía el byte de datos hasta que el maestro lo recibe, toma de vuelta el control de la línea SDA y envía un $NACK$ seguido de una condición de $STOP$, finalizando así la operación de lectura (ver Figura 2.56).

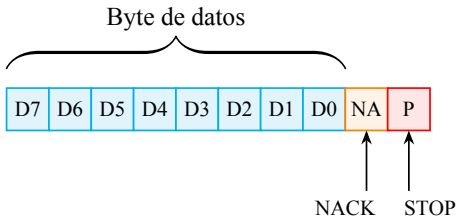


Figura 2.56: I2C Read - Byte de datos

El proceso completo de lectura se puede visualizar en la Figura 2.57.

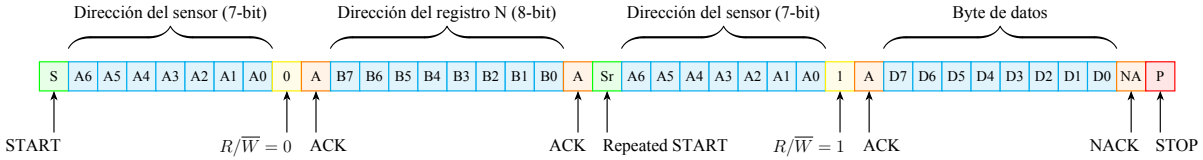


Figura 2.57: I2C Read - Proceso completo

2.5.3 SPI

SPI (Interfaz Periférica Serial) es el protocolo utilizado para la comunicación entre el microcontrolador y la tarjeta microSD, cuya función es la de recopilar y guardar grandes cantidades de información.

En la Figura 2.58 se observa un diagrama genérico del conexionado entre el controlador y un periférico, que establecen una comunicación a través de SPI.

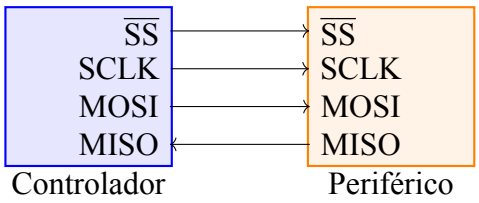


Figura 2.58: SPI: Diagrama de conexión genérico entre un controlador y un periférico

- SS: Permite seleccionar de entre varios periféricos, con cuál de ellos establecer una comunicación.
- SCLK: Es un tren de pulsos que sirve para sincronizar las transmisiones de datos entre el controlador y un periférico. Además, este es compartido entre todos los periféricos en uso.
- MOSI: Se utiliza para enviar datos desde el controlador a un periférico en particular.

- MISO: Se utiliza para enviar datos desde algún periférico hacia el controlador. Cuando los periféricos no se encuentren seleccionado por el controlador a través del pin SS, el pin MISO presentará una alta impedancia.

2.5.3.1 Escritura

En la Figura 2.59 se diagramó el esquema temporal de una operación de escritura.

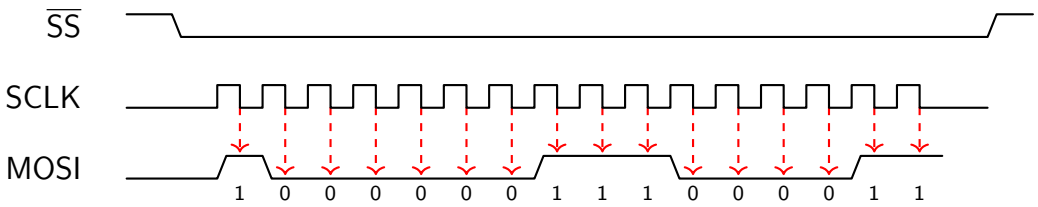


Figura 2.59: Diagrama de tiempos para una operación de escritura en SPI

En este caso, el controlador envía la palabra de ejemplo $81C3_h = 1000\ 0001\ 1100\ 0011_b$ (podría ser para activar o setear alguna configuración) al periférico (línea MOSI), donde cada bit se determina en cada flanco descendente del reloj (línea SCLK).

2.5.3.2 Lectura

Suponiendo que la escritura fue para activar o setear alguna configuración, en la Figura 2.60 se ha diagramado la respuesta $E475_h = 1110\ 0100\ 0111\ 0101_b$, a modo de ejemplo, del periférico.

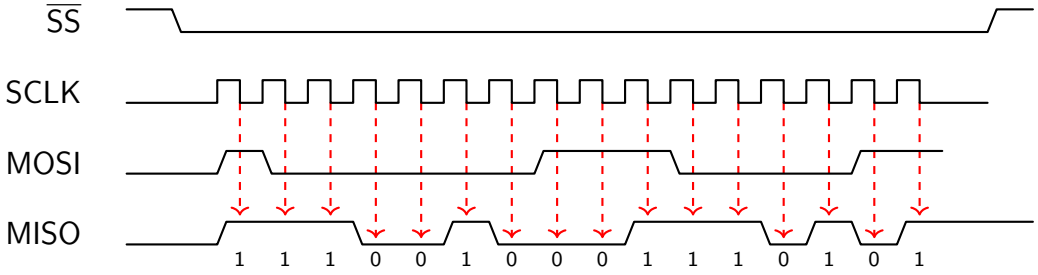


Figura 2.60: Diagrama de tiempos para una operación de lectura en SPI

2.5.3.3 Topologías

La primer topología presentada consiste en que todos los periféricos compartan las líneas de SCLK, MOSI y MISO, pero cada uno con su conexión individual para el pin SS. Esto permite que el controlador decida con cuál periférico intercambiar información. Además, recordando que cuando el pin SS de un periférico no se encuentra seleccionado (es decir esta en el valor lógico 1), luego el pin MISO presenta una alta impedancia que impide la transmición de información hacia el controlador.

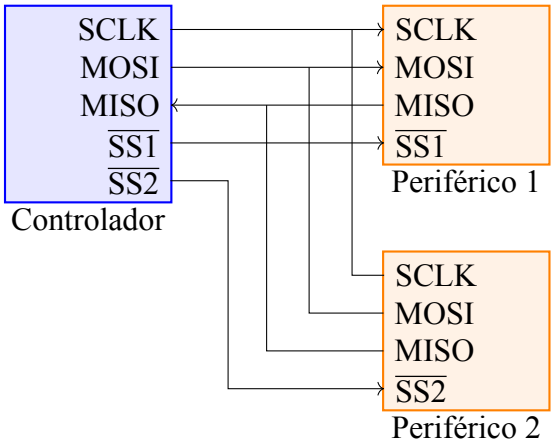


Figura 2.61: SPI: Comunicación entre varios periféricos a través de múltiples pines SS

En la Figura 2.62 se ha diagramado una configuración de comunicación SPI en cadena entre múltiples periféricos. La idea fundamental es que un único pin de SS del controlador se encargue de gestionar a todos los periféricos. Entonces, el controlador envía la información, a través su pin MOSI, hacia el primer periférico y luego el pin MISO de este se conecta al MOSI del periférico siguiente. De esta forma, la información se va transmitiendo de un periférico a otro hasta llegar al último, cuyo pin MISO se conecta directamente al pin MISO del controlador.

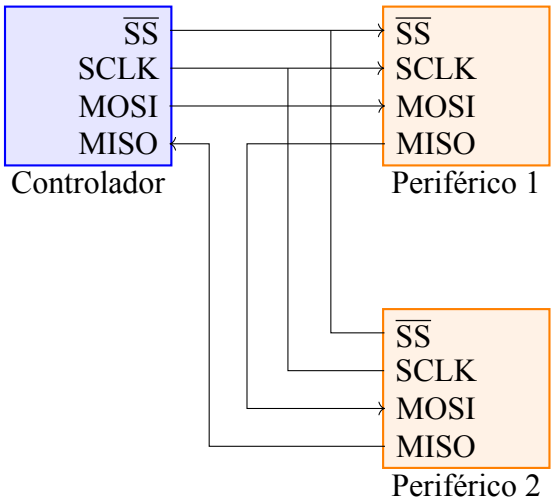


Figura 2.62: SPI: Comunicación en cadena entre varios periféricos

2.5.4 UART

El protocolo UART (Receptor/Transmisor Universal Asíncrono) es una convención que permite transmitir datos entre dos dispositivos de manera serie a través de dos líneas, Rx y Tx, así como también admite configurar la velocidad de transmisión. Por otra parte, puesto que es asíncrono, significa que no hay ningún reloj presente que sincronice el envío de datos, no obstante, esto se resuelve configurando la misma tasa de transmisión $\frac{bit}{s}$ para ambos controladores UART.

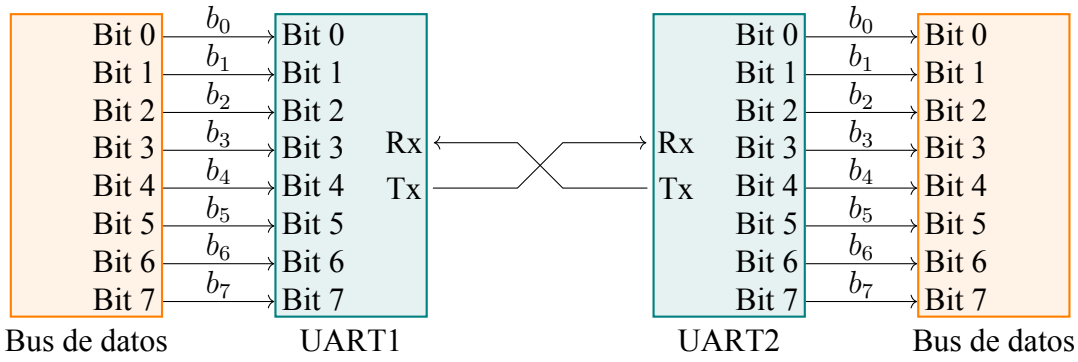


Figura 2.63: Diagrama genérico de un sistema de comunicación UART

En la Figura 2.63, se ha diagramado un esquema genérico de comunicación a modo de ejemplo, donde el controlador UART1 transmite información hacia el controlador UART2. Luego, nótese que el bus de datos Tx le envía la información al controlador UART1 en formato paralelo, quien posteriormente se encarga de enviar los datos bit por bit al controlador UART2. Por último, el controlador UART2 se encarga de convertir nuevamente los datos recibidos a formato paralelo para enviarselos al bus de datos Rx.

Continuando, el formato de un paquete UART es el siguiente

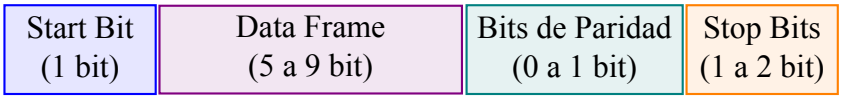


Figura 2.64: Formato de un paquete UART

- **Start Bit**: La línea de transmisión se encuentra normalmente en un nivel lógico alto (1) cuando no hay un intercambio de datos. Entonces, para comenzar una transmisión el controlador que transmite los datos (en el ejemplo de la Figura 2.63 es el UART1), lleva la línea hacia un nivel lógico bajo (0). Luego, cuando el controlador receptor (UART2) detecte el cambio de nivel alto a bajo, comenzará a leer los bits del data frame a una frecuencia que dependerá de la tasa de transmisión configurada.

- **Data Frame**: Son los bits útiles del paquete y representan la información pura transmitida desde el controlador transmisor.

- **Bits de Paridad**: Reflejan si la cantidad de 1 presentes[17] en el numero total transmitido es un valor par (0) o impar (1). Luego, suponiendo que sea par y el valor de este campo es un 1, entonces significa que hubo un error en la transmisión y los datos recibidos son inválidos.

- **Stop Bits**: Para finalizar la transmisión de datos, el controlador que transmite lleva la línea de transmisión del nivel bajo a alto, durante 1 o 2 bits de duración.

[17]El valor de este campo va a depender de la configuración del controlador UART. Es decir, si se lo configuró para detectar paridad ya sea par o impar, entonces este tendrá que definir el valor del campo para que la cantidad de 1 presentes en el número final transmitido se corresponda con lo configurado.

Por último, se realizará un ejemplo para clarificar el proceso completo.

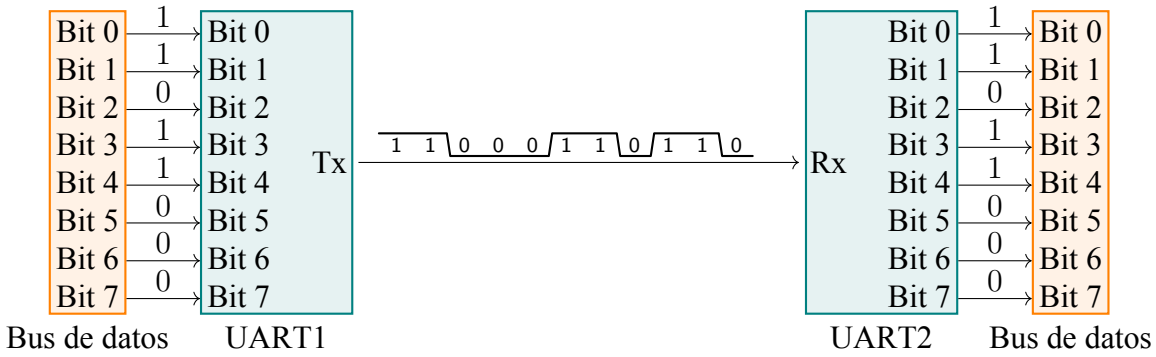


Figura 2.65: Diagrama de transmisión por UART

1. Para este ejemplo, el data frame a transmitir será el número $D8_h = 1101\ 1000_b$. Como se observa en la Figura 2.65, este es transmitido desde el bus de datos Tx hacia el controlador transmisor (UART1) en formato paralelo.

2. Entonces, el controlador transmisor recibe el número $1101\ 1000_b$ y comienza agregándole el bit de start al comienzo del paquete $\underline{0}\ 1101\ 1000_b$ y el bit de stop al final $\underline{0}\ 1101\ 1000\ p\ \underline{1}_b$. Por último, suponiendo que el controlador se configuró para detectar paridad de tipo par y, dado que la cantidad de 1 presentes en el paquete sin contar el bit "p" es 5 (impar), entonces para que la cantidad de 1 sea par p debe valer 1 quedando, $0\ 1101\ 1000\ \underline{1}\ 1_b$.

3. Una vez que el paquete ya se encuentra completo, el controlador transmisor comienza a transmitir bit por bit, desde el LSB hasta el MSB, hacia el pin Rx del controlador receptor.

4. Al detectar el cambio en la línea de transmisión de 1 a 0[18], el controlador receptor identifica la condición de "start" y comienza a leer los bits presentes en la línea a una frecuencia que dependerá de la tasa de transmisión previamente configurada.

5. Al identificar la condición de "stop", el controlador receptor separa los bits de start, stop y paridad quedándose únicamente con los bits útiles o data frame. Luego, procede a sumar la cantidad de 1 presentes en la totalidad del paquete recibido y verificar si el bit de paridad se encuentra acorde a la suma realizada. En caso negativo, el controlador identificará que los datos recibidos no son válidos.

6. Finalmente, el controlador receptor le envía los datos en formato paralelo al bus de datos Rx.

2.6 Sistemas operativos

Un OS (Operating System) es un programa que ofrece servicios a otros programas, ambos corriendo en una computadora o microcontrolador. En concreto, un OS que permite la ejecución

[18]Como se mencionó previamente, la línea de transmisión suele mantenerse a un nivel lógico alto (1). De manera que, cuando el receptor recibe el primer bit es el de start y vale 0, por tanto esa transición le permite identificar la condición de start.

de múltiples programas simultáneamente se lo caracteriza como multi-tarea. Fundamentalmente, cada núcleo del procesador puede ejecutar hasta 1 hilo o tarea a la vez, y es el "organizador" (scheduler en inglés) quien define qué programa se ejecuta en qué momento.

Entonces, el organizador de un RTOS (Real Time Operating System) está diseñado para que su accionar sea predecible, es decir, el objetivo es que el usuario conozca el orden de prioridad de cada tarea y, por tanto, en qué orden se ejecutarán.

Las principales tareas empleadas en el cuadricóptero son las siguientes,

1. Medir y actualizar los estados del sistema.

2. Ejecutar el ciclo de control para actualizar las velocidades angulares de cada motor.

Ahora bien, medir los estados cada cierto tiempo t_k significa que entre t_k y t_{k+1}, con $\Delta t = t_{k+1} - t_k$, no se tiene información sobre nuestro sistema. Entonces, si Δt es demasiado grande los controladores no serán lo suficientemente rápidos como para poder corregir el error y, por lo tanto, mientras más pequeño sea este mejor se desenvolverá el sistema en cuestión. Pero, ¿qué tiene que ver esto con un sistema operativo? Para eso hay que pensar en un flujo secuencial en el cual primero se ejecuta el código asociado a (1) y luego el código asociado a (2). En tal caso, el tiempo total de ejecución sería $\Delta t = t_1 + t_2$. No obstante, como se acaba de mencionar, mientras menor sea el tiempo entre muestras, Δt, mejor será la respuesta del sistema. Esto lleva a plantearse si es posible ejecutar ambas tareas en paralelo y, la respuesta frente a esta problemática es un sistema operativo.

La ESP32 cuenta con un driver que funciona como un sistema operativo en tiempo real, denominado FreeRTOS que permite, entre otras cosas, ejecutar tareas en paralelo. Para logarlo, FreeRTOS admite que el usuario asigne prioridades de ejecución a cada tarea, de manera que el organizador sepa cuál es la tarea que tiene que ser ejecutada luego.

Capítulo 3

Modelado y Simulación

En la sección 2.3.1.2 se ha explicado cómo se compone un sistema de control genérico y detallado la función de cada uno de sus componentes. Ahora bien, el objetivo de este capítulo es adaptarlo a la aplicación de este proyecto y explicar el funcionamiento de cada bloque.

Se comenzará mostrando y explicando el modelo de los componentes físicos del cuadricóptero, seguido del modelo de todos aquellos algoritmos desarrollados a nivel firmware, tales como los controladores, mapeos de variables, entre otros. Luego, se mostrará el resultado obtenido en las simulaciones realizadas por software[1]. Por último, se dejará una sección dedicada a detallar el proceso de obtención de todos los parámetros físicos de los modelos desarrollados.

3.1 Modelos

A partir de la Figura 2.21 se pueden identificar que los bloques físicos de un sistema de control son los actuadores, la planta y los sensores.

3.1.1 Modelo de la planta

Como se observa en la Figura 3.1, la planta es un sistema cuyas entradas son las velocidades angulares de cada motor y su salida son las variables de estado[2], es decir, las pociciones y velocidades del cuadricóptero.

[1] El diseño de cada componente y las simulaciones realizadas por software fueron llevadas a cabo haciendo uso del software Simulink (MathWorks, s.f.-b) de Matlab (MathWorks, s.f.-a).

[2] Estrictamente hablando, los estados del cuadricóptero son las aceleraciones, velocidades y posiciones, por otra parte, en la Figura 3.1 se muestran únicamente las variables de estado que son utilizadas por el sistema de control desarrollado.

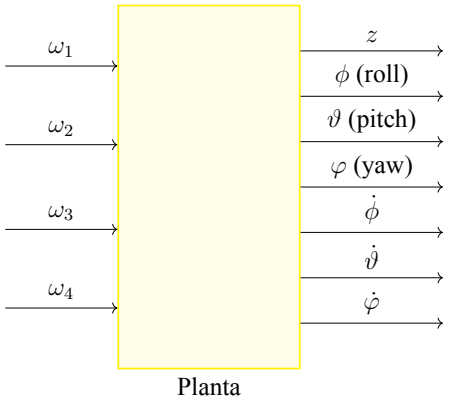


Figura 3.1: Modelo de la planta, en Simulink

Ahora bien, en la Figura 3.2 se puede observar cómo está compuesto internamente el bloque de la planta y de qué manera se relacionan las magnitudes físicas que la conforman.

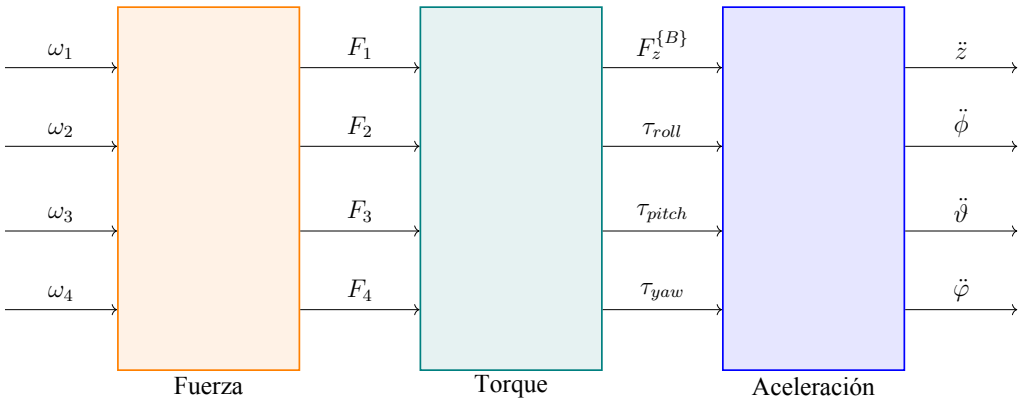


Figura 3.2: Modelo interno de la planta, en Simulink

3.1.1.1 Modelo de la fuerza

Este bloque está diseñando con base en la ecuación 2.1 y se encarga de generar las fuerzas producidas a partir de la velocidad angular de cada motor.

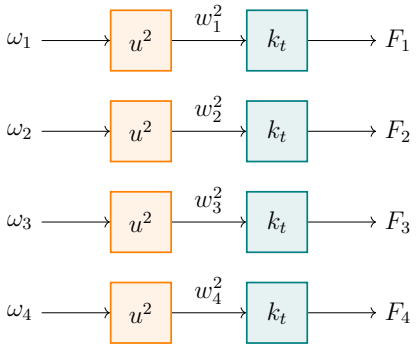


Figura 3.3: Modelo de la fuerza, en Simulink

3.1.1.2 Modelo del torque

En la Figura 3.4 se visualizan los subsistemas encargados de convertir las fuerzas generadas por cada motor F_1, F_2, F_3, F_4, en torques $\tau_{roll}, \tau_{pitch}, \tau_{yaw}$, y en la fuerza de empuje $F_z^{\{B\}}$ (ecuación 2.1), en dirección $z > 0$.

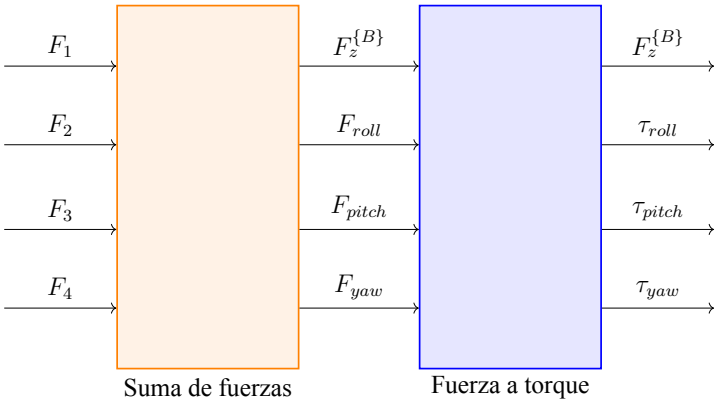


Figura 3.4: Modelo del torque, en Simulink

Luego, en la Figura 3.5 se encuentra diagramado el primer sub-bloque denominado *Suma de fuerzas*. Este se encarga de realizar la sumatoria de la fuerza producida por cada motor, en función del ángulo en cuestión $(roll, pitch, yaw)$ haciendo uso de la ecuación 2.3, Adicionalmente de la fuerza de empuje $F_z^{\{B\}}$.

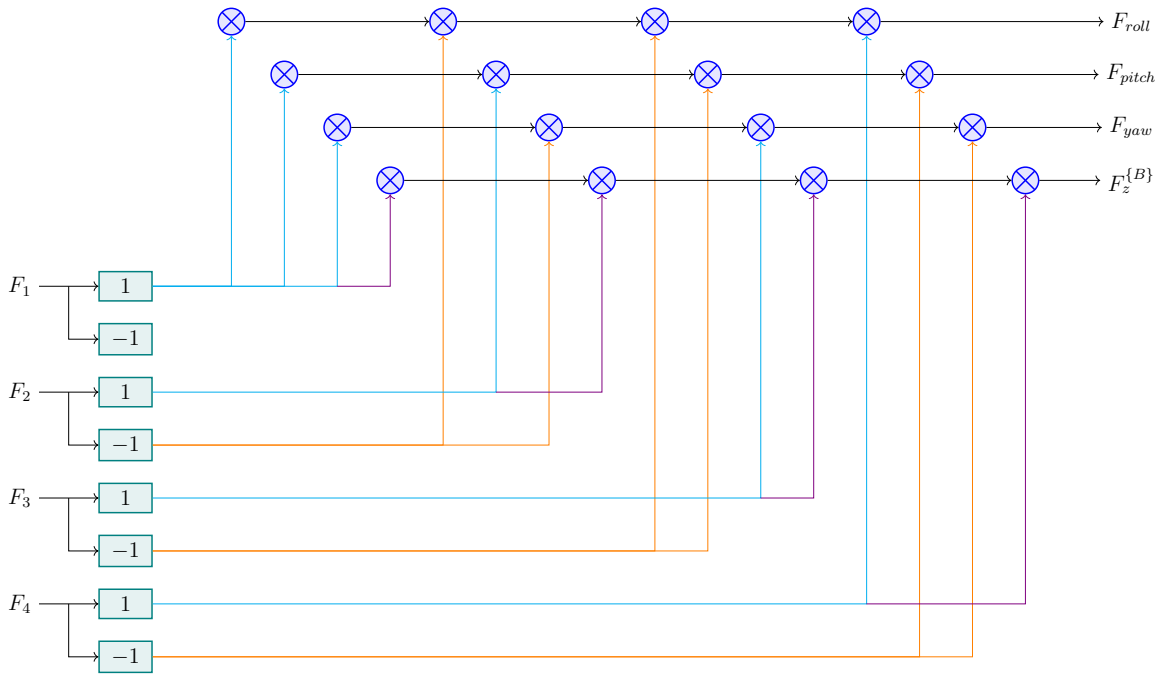


Figura 3.5: Primer sub-bloque, Suma de fuerzas, del modelo del torque, en Simulink

Ahora bien, en la Figura 3.6 se puede observar el segundo subbloque *Fuerza a torque*, que utiliza productos para convertir fuerza en torque para cada ángulo (ecuación 2.3), donde l es la

distancia desde el centro del cuadricóptero hacia cualquiera de sus 4 motores y $45°$ es el ángulo entre los ejes (x, y) y cada brazo (ver Figura A.1). Por otra parte, se nota que este subbloque no realiza ninguna modificación sobre la fuerza de empuje $F_z^{\{B\}}$.

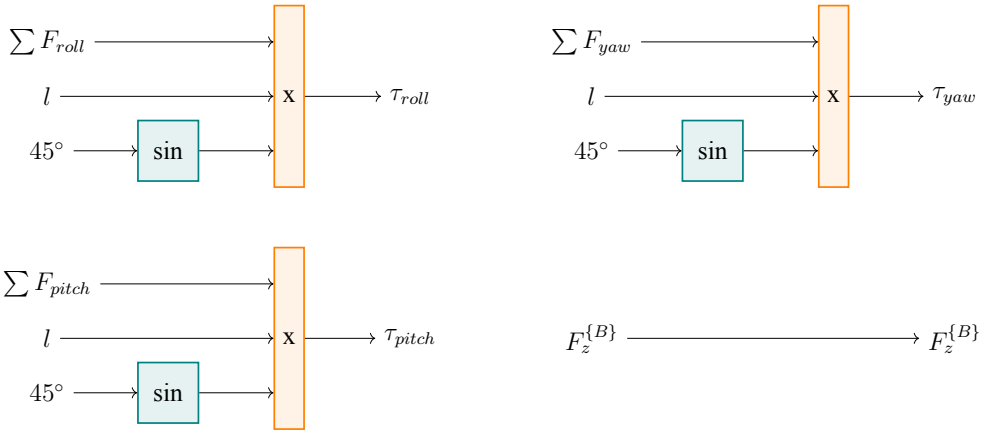


Figura 3.6: Segundo sub-bloque, Fuerza a torque, del modelo del torque, en Simulink

3.1.1.3 Modelo de la aceleración

Con el objetivo de simplificar el diseño de los controladores, se ha realizado una modificación a los resultados obtenidos en la ecuación 2.4. Esencialmente, esto consiste en tratar cada estado como una variable independiente del resto de estados[3]. Es decir, las aceleraciones

$$\ddot{\phi} = f(\omega_1, \omega_2, \omega_3, \omega_4, j_{xx}, \dot{\vartheta}, \dot{\varphi})$$
$$\ddot{\vartheta} = g(\omega_1, \omega_2, \omega_3, \omega_4, j_{yy}, \dot{\phi}, \dot{\varphi})$$
$$\ddot{\varphi} = h(\omega_1, \omega_2, \omega_3, \omega_4, j_{zz}, \dot{\vartheta}, \dot{\phi})$$

quedan simplificadas en

$$\ddot{\phi} = f(\omega_1, \omega_2, \omega_3, \omega_4, j_{xx}) = \frac{\tau_{roll}}{j_{xx}}$$
$$\ddot{\vartheta} = g(\omega_1, \omega_2, \omega_3, \omega_4, j_{yy}) = \frac{\tau_{pitch}}{j_{yy}}$$
$$\ddot{\varphi} = h(\omega_1, \omega_2, \omega_3, \omega_4, j_{zz}) = \frac{\tau_{yaw}}{j_{zz}}$$

En la Figura 3.7 se observa el diagrama del modelo de la aceleración, que utiliza un bloque divisor para modelar las ecuaciones simplificadas de cada estado. Por otra parte, los valores de cada momento de inercia j_{xx}, j_{yy}, j_{zz} son constantes que dependen de la estructura del cuadricóptero[4]. Además, adicionalmente a las aceleraciones, la manera de obtener las velocidades y posiciones es integrando cada variable mediante los bloques $\frac{1}{s}$, en el dominio de Laplace.

[3]Estos nuevos modelos para las aceleraciones angulares se aproximan a los originalmente planteados, siempre y cuando las velocidades angulares mantengan valores pequeños tal que se pueda despreciar su efecto en el resto de ángulos.

[4]En la sección 3.4 se encuentra detallado el proceso de obtención de estos parámetros.

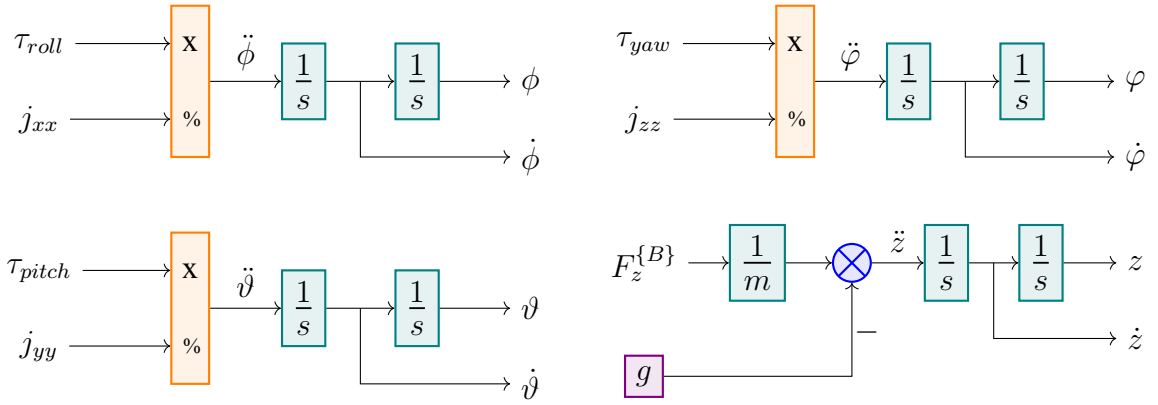


Figura 3.7: Modelo de la aceleración, en Simulink

La modelación de la aceleración vertical, $\ddot{z}$, se basa en la ecuación 2.2. No obstante, es importante notar que se ha asumido un punto de operación en el cual los valores máximos que alcanzan los ángulos son de algunos grados, con el fin de que la aproximación $\cos(x) \approx 1$ sea válida.

3.1.2 Modelo de los motores

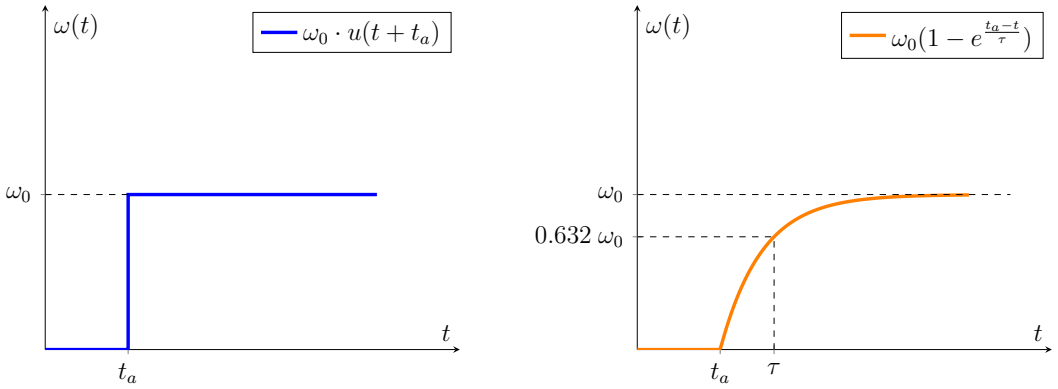
Del estudio de las máquinas eléctricas, se sabe que los motores no alcanzan la velocidad instantáneamente cuando se les aplica una tensión. Por tal motivo, y con el fin de que el modelo del cuadricóptero se asemeje lo más posible a la realidad, es necesario contemplar esta característica.

(a) Modelo ideal de un motor

(b) Modelo realista de un motor

Figura 3.8: Modelo ideal VS. modelo realista de un motor

Luego, desde un punto de vista gráfico, las situaciones que se quieren diferenciar son las graficadas en la Figura 3.9. En ambas situaciones se aplica un pulso, a través de una señal PWM, al motor en el instante t_a, sin embargo, en la Figura 3.9a el motor alcanza la velocidad w_0 instantáneamente, mientras que en la Figura 3.9b lo hace exponencialmente, es decir, le lleva cierto tiempo.



(a) Respuesta de un motor ideal (b) Respuesta de un motor realista

Figura 3.9: Respuesta de un motor ideal VS. respuesta de un motor realista

Esencialmente, se deja de modelar al motor con una función de Heaviside o escalón unitario $u(t)$, para modelarlo a través de una respuesta exponencial. Ahora bien, es necesaria la *función transferencia* que expresa la evolución del sistema, en el tiempo, frente a un pulso PWM. Para ello, hay que llevar la ecuación $w(t) = \omega_0(1 - e^{\frac{t_a-t}{\tau}})$ al dominio de Laplace, utilizando la *transformada de Laplace*.

Los detalles del cálculo matemático se encuentran en el *Apéndice D: Modelo de los motores en el dominio de Laplace*, no obstante, la función de transferencia hallada es

$$T(s) = \frac{1}{\tau s + 1}$$

donde τ es el tiempo, en segundos, que tarda el motor en alcanzar el $63.2\,\%$ de ω_0[5].

Finalmente, en la Figura 3.10 se puede observar el diagrama del modelo de los motores, en Simulink.

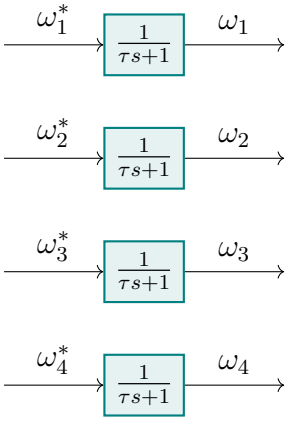


Figura 3.10: Modelo de los motores, en Simulink

[5]Puesto que los componentes utilizados en (Quan, 2017, Sec. 6.3.4.3, ec. 6.48) son los mismos que se usan en este proyecto, se utlizará el mismo valor para τ que aquel allí detallado.

3.1.3 Modelo de los sensores

El modelo de los sensores podría ser simplemente un cable; sin embargo, puesto que estos se encuentran sujetos a ruido de diversa índole, debe tenerse en cuenta para garantizar un modelo con mayor precisión y semejanza a la realidad.

En la Figura 3.11 se puede apreciar el diagrama del modelo del ruido, en Simulink. Este consiste en sumar ruido blanco Gaussiano[6] a los estados que envía la planta.

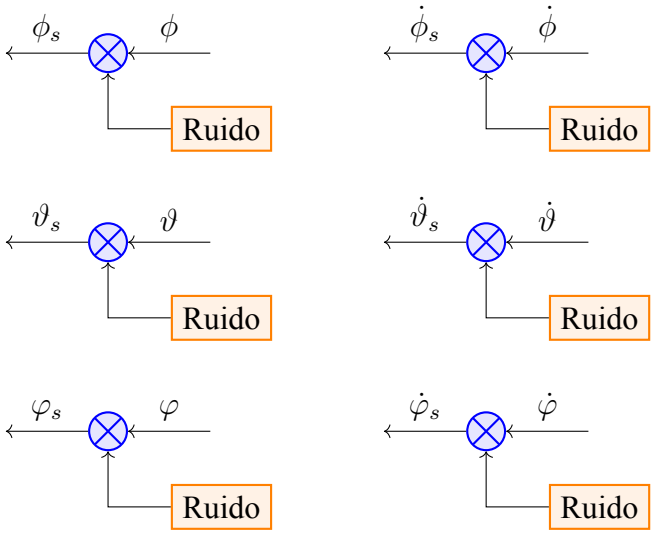


Figura 3.11: Modelo del ruido, en Simulink

3.2 Desarrollo

En esta sección se detallará el diseño y la elección de los modelos para cada algoritmo desarrollado a nivel firmware, es decir, los controladores para cada estado, el bloque "Motor Mixing Algorithm" que combina la salida de los controladores para obtener los ciclos de trabajo que definen la velocidad angular de cada motor y transformaciones que permiten mapear una variable de trabajo en otra. Luego, se analizará el modelo del sistema completo que integra tanto los componentes físicos como aquellos desarrollados por firmware. Por último, se expondrán los tipos de transmisores desarrollados para establecer una comunicación con el cuadricóptero.

3.2.1 Controladores

Con varias iteraciones de simulación y experimentación se llegó a la conclusión de que un único lazo de control para la posición como aquel explicado en la *Sección 2.3.4: Controlador PID* resulta lento, a tal punto que la acción de control generada no es lo suficientemente efectiva como para estabilizar al cuadricóptero. Por tal motivo, se implementó un algoritmo de control que regule tanto la posición como la velocidad.

En este caso, la referencia al controlador de velocidad no será enviada por el usuario, sino que será función del controlador de posición. Es decir, mientras más desviada esté la posición

[6] En Simulink, el bloque se denomina "Band-Limited White Noise" y se deben de especificar la energía asociada al ruido y el tiempo de muestreo del bloque.

real con respecto a la referencia especificada por el usuario, mayor será el impacto del controlador de velocidad en el sistema. Esto se puede ver gráficamente en la Figura 3.12, donde x_r es la referencia de posición y x_s es la posición medida o estimada[7], tal que la diferencia entre estos produce el error e_x. Luego, este último ingresa al controlador de posición PID(x) cuya acción de control resultante, $\dot{x}_r$, se interpreta como la *referencia* para el controlador de velocidad. Entonces, la diferencia entre esta última y la velocidad medida o estimada, $\dot{x}_s$, da lugar al error $e_{\dot{x}}$. Finalmente, este entra al controlador de velocidad, PID($\dot{x}$), quien se encarga de generar una acción de control u, que es la que ingresará al bloque *Motor Mixing Algorithm*, o MMA por sus siglas en inglés, detallado más adelante.

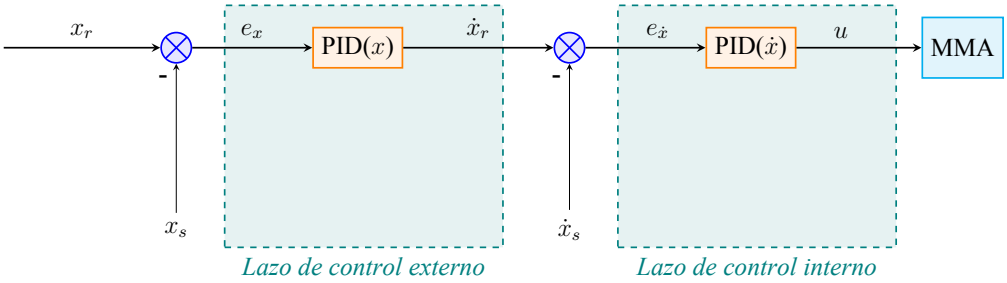


Figura 3.12: Diagrama en bloques del controlador para los ángulos

Existen diversos métodos para sintonizar un lazo de control, no obstante, el método que se explicará a continuación será el que funcionó tanto en las simulaciones como en el sistema real.

Dado que el controlador consta de dos lazos PID, uno interno y otro externo, lo primero que se debe garantizar es que el lazo interno siga correctamente la referencia. Asimismo, como se explicó previamente, la referencia del lazo interno, $\dot{x}_r$, se calcula automáticamente en función del lazo de control externo, tal que no depende directamente del usuario. Sin embargo, para sintonizarlo individualmente, se debe desconectar la referencia $\dot{x}_r$ y conectar una referencia que se ajustará manualmente denominada $\dot{m}_r$, como se puede visualizar en la Figura 3.13.

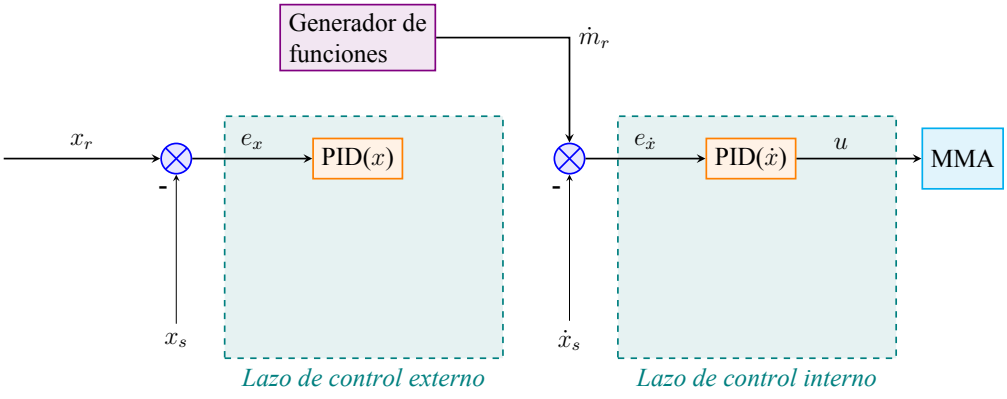


Figura 3.13: Sintonización del lazo de control interno

Por otra parte, se observa que la nueva referencia es un generador de funciones, seno, coseno, escalón, rampa, entre otras. Así, la función utilizada dependerá en gran parte del estado a

[7]La realidad es que no todos los estados son una medición directa de los sensores. En la práctica, algunos estados se pueden obtener directamente a través de los sensores mientras que otros requieren de una medición indirecta (ver *Sección 2.3.5: Filtro complementario*).

controlar. Es decir, la forma de onda de las posiciones y velocidades angulares se asemejan más a formas de onda sinusoidales, de manera que es una buena idea que la referencia sea una señal sinusoidal del estilo, $A\sin(2\pi ft)$. En contraste, para la altura un escalón o una rampa pueden ser buenas opciones.

Luego, se eligen las características de la señal a utilizar en base a cómo se desea que se comporte el cuadricóptero en su estado normal de operación. Entonces, se comienza únicamente con un control de tipo proporcional P, incrementando su ganancia $k_{p\{int\}}$ de a poco hasta alcanzar una respuesta lo suficientemente rápida y precisa. Una vez que esto suceda, se puede quitar el generador y volver a conectar el controlador del lazo externo, empezando por un control de tipo proporcional P e incrementando su ganancia $k_{p\{ext\}}$ de manera gradual.

Por último, el uso de las acciones I/D es evaluado de manera empírica[8], observando la respuesta del sistema. En un entorno de simulación, como Simulink, esto es sencillo y no radica en ningún inconveniente. Por otra parte, para probarlo en la práctica es necesario contener al cuadricóptero en algún tipo de estructura como la detallada en la *Sección 6.3: Estructura de prueba*.

3.2.2 Motor Mixing Algorithm

Para diseñar el algoritmo que controlará la velocidad de los motores, se hará uso de la Figura 1.1. Además, también será de utilidad mencionar algunas cuestiones.

- En la Figura 1.1, las flechas posicionadas sobre cada motor representan el sentido de giro de estos.
  - w_1 ↻
  - w_2 ↺
  - w_3 ↻
  - w_2 ↺
- El torque generado por cada hélice es opuesto al sentido de giro del respectivo motor. Es decir, si el sentido de giro de un motor es horario ↻, entonces el torque generado tendrá sentido antihorario ↺, y viceversa.
  - w_1 ↻ $\Rightarrow$ τ_1 ↺
  - w_2 ↺ $\Rightarrow$ τ_2 ↻
  - w_3 ↻ $\Rightarrow$ τ_3 ↺
  - w_2 ↺ $\Rightarrow$ τ_4 ↻
- La dirección positiva de cada estado $(z, roll, pitch, yaw)$ se define en la Figura 1.1

Continuando con el algoritmo, para elevar al cuadricóptero es necesario que los 4 motores giren a la misma velocidad, por tanto, la acción de control en z debe tener el mismo signo en los 4 motores.

[8] Se recomienda utilizar el software detallado en la *Sección 5.1: Software de monitoreo en tiempo real mediante USB*, para la visualización en tiempo real de la respuesta del cuadricóptero frente a variaciones en las acciones PID y sus ganancias.

$$w_1 = u_z$$
$$w_2 = u_z$$
$$w_3 = u_z$$
$$w_4 = u_z$$

Para generar movimiento positivo en roll el par de motores $(1, 4)$ tiene que girar más rápido que el par $(2, 3)$.

$$w_1 = u_z + u_{roll}$$
$$w_2 = u_z - u_{roll}$$
$$w_3 = u_z - u_{roll}$$
$$w_4 = u_z + u_{roll}$$

Para generar movimiento positivo en *pitch* el par de motores $(1, 2)$ tiene que girar más rápido que el par $(3, 4)$.

$$w_1 = u_z + u_{roll} + u_{pitch}$$
$$w_2 = u_z - u_{roll} + u_{pitch}$$
$$w_3 = u_z - u_{roll} - u_{pitch}$$
$$w_4 = u_z + u_{roll} - u_{pitch}$$

Para generar movimiento positivo en yaw el par de motores $(1, 3)$ tiene que girar más rápido que el par $(2, 4)$.

$$w_1 = u_z + u_{roll} + u_{pitch} + u_{yaw}$$
$$w_2 = u_z - u_{roll} + u_{pitch} - u_{yaw}$$
$$w_3 = u_z - u_{roll} - u_{pitch} + u_{yaw}$$
$$w_4 = u_z + u_{roll} - u_{pitch} - u_{yaw}$$

Ahora bien, a modo de ejemplo se puede suponer que todas las acciones de control son nulas excepto u_{roll}, tal que el par de motores que producía movimiento positivo era el $(1, 4)$, y movimiento negativo el par $(2, 3)$. Luego, si se dejan las ecuaciones como están ahora mismo, un torque positivo en $roll$ será equivalente a $\tau_{roll} = \alpha(\omega_1^2 + \omega_4^2 - \omega_2^2 - \omega_3^2) = \alpha(u_{roll}^2 + u_{roll}^2 - [-u_{roll}]^2 - [-u_{roll}]^2)$, siendo $\alpha = k_t(\frac{\sqrt{2}}{2})l$.

No obstante, este algoritmo da lugar a posibles errores, como los detallados debajo.

1. No existe torque neto ya que $\alpha(u_{roll}^2 + u_{roll}^2 - [-u_{roll}]^2 - [-u_{roll}]^2) = \alpha(2u_{roll}^2 - 2u_{roll}^2) = 0$.
2. La acción de control u_{roll} tiene efecto doble.

Para solucionar el primer inconveniente, se recuerda de la sección *Sección 2.1.3: Dinámica* que la velocidad angular de los motores es siempre positiva, $w_i \geq 0$. Por tal motivo, el sistema de control utiliza un bloque saturador que limita su valor mínimo en 0 (en vez de pedirle a los motores que cambien su sentido de giro, el controlador los apaga). Entonces, la ecuación quedaría $\alpha(u_{roll}^2 + u_{roll}^2 - [-u_{roll}]^2 - [-u_{roll}]^2) = \alpha(2u_{roll}^2 - 0^2 - 0^2) = \alpha 2u_{roll}^2$.

Con esto se soluciona el primer problema, pero ahora el segundo resulta evidente. La acción de control de cualquier controlador se termina duplicando, es decir, es como si tuviese un efecto doble. Por ello, con el fin de distribuir equitativamente cada acción de control, se deben multiplicarlas por $\frac{1}{2}$[9], como sugiere la Figura 3.14.

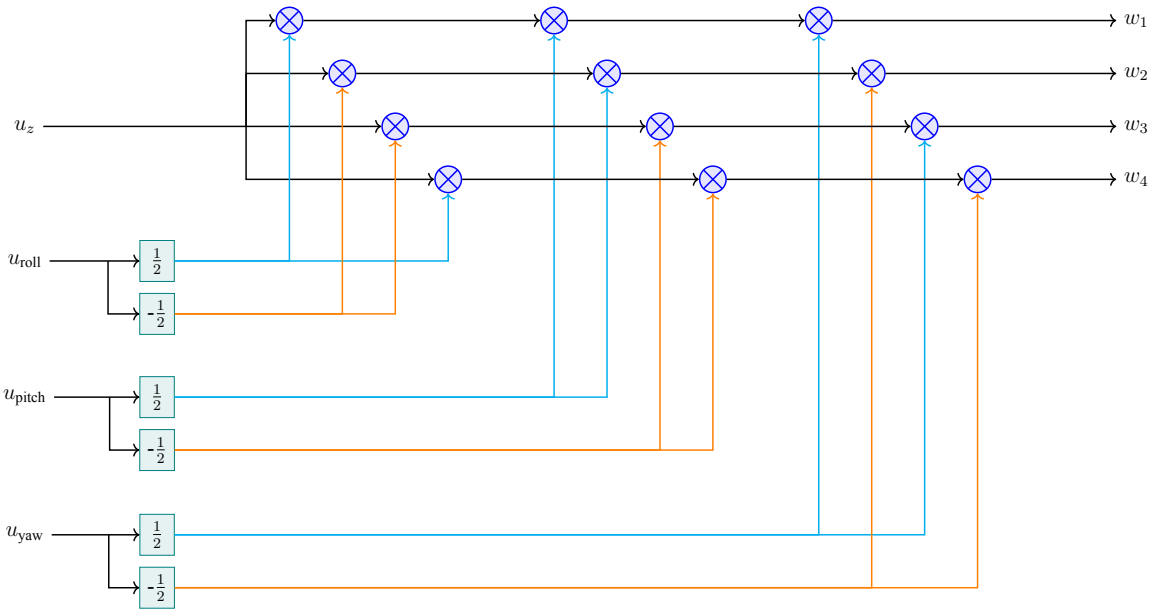


Figura 3.14: Modelo del bloque ”Motor Mixing Algorithm”, en Simulink

Finalmente, el algoritmo resultante queda de la siguiente manera.

$$w_1 = u_z + \frac{1}{2}(+u_{roll} + u_{pitch} + u_{yaw})$$
$$w_2 = u_z + \frac{1}{2}(-u_{roll} + u_{pitch} - u_{yaw})$$
$$w_3 = u_z + \frac{1}{2}(-u_{roll} - u_{pitch} + u_{yaw})$$
$$w_4 = u_z + \frac{1}{2}(+u_{roll} - u_{pitch} - u_{yaw})$$

3.2.3 Transformaciones

El objetivo de esta sección es encontrar las transformaciones que permitan diseñar la transición entre bloques (Quan, 2017, Sec. 6.1.4), como se puede apreciar en la Figura 3.15.

Figura 3.15: Diagrama en bloques de las transformaciones

En el cuadricóptero real el controlador no afecta directamente la velocidad angular de los motores, sino que la *variable manipulada* son las señales PWM, que luego ingresan a los ESC

[9] No es necesario multiplicar u_z por $\frac{1}{2}$, ya que esta aparece una única vez en las ecuaciones de las velocidades angulares w_i.

(controladores de velocidad) y terminan finalmente convirtiéndose en velodidades angulares. Por ello, el objetivo de esta sección es encontrar las transformaciones adecuadas para mapear una acción de control u a velocidad angular ω.

En primera instancia, como se observa en la Figura 3.16, las acciones de control calculadas por los controladores representan los *duty cycle* (ciclo de trabajo) de las señales PWM. Puesto que los ESC tienen asociados un duty cycle mínimo y máximo, se representan con DC_1 y DC_2, respectivamente.

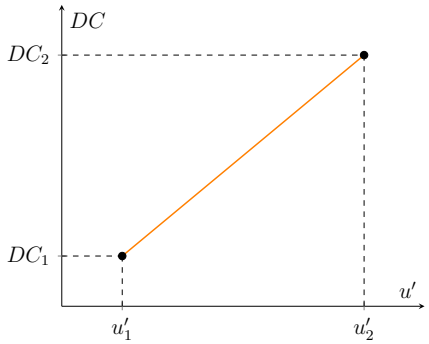


Figura 3.16: Transformación - Acción de control a ciclo de trabajo

Así, la transformación $u' \longrightarrow DC$[10] queda definida como

$$DC(u') = \left(\frac{DC_2 - DC_1}{u'_2 - u'_1}\right)u' + b$$

Para hallar el valor de la ordenada al origen, b, se reemplaza el punto (u'_1, DC_1) y se encuentra que

$$DC_1 = \left(\frac{DC_2 - DC_1}{u'_2 - u'_1}\right)u'_1 + b \Longleftrightarrow b = DC_1 - \left(\frac{DC_2 - DC_1}{u'_2 - u'_1}\right)u'_1$$

Ahora bien, dada la arquitectura del cuadricóptero, no es necesario que los motores giren en sentido opuesto al definido, para modificar la orientación de este. Por tanto, el mínimo valor que los controladores deberían de intentar asignarles a los motores tendría que ser un "motor apagado", es decir, $u'_1 = 0$. Luego,

$$b = DC_1 - \left(\frac{DC_2 - DC_1}{u'_2 - 0}\right)(0) = DC_1$$

y la transformación resultantes es entonces

$$DC(u') = \left(\frac{DC_2 - DC_1}{u'_2}\right)u' + DC_1$$

Continuando, hay que encontrar una transformación que permita modelar al controlador de velocidad o ESC. Este recibe una señal PWM y devuelve velocidad angular ω, es decir, se quiere $DC \longrightarrow \omega$. La Figura 3.17 muestra, gráficamente, como es el mapeo.

[10] El cambio de notación de u a u' se debe a la normalización de las transformaciones, que se detalla al final de la sección.

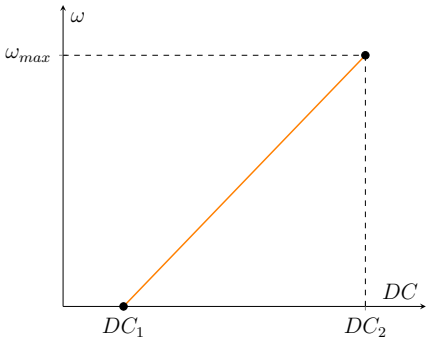


Figura 3.17: Transformación - Ciclo de trabajo a velocidad angular

$$\omega(DC) = \left(\frac{\omega_{max}}{DC_2 - DC_1}\right) DC + b$$

tal que reemplazando el punto $(DC_1, 0)$ en la ecuación se obtiene

$$b = -\left(\frac{\omega_{max}}{DC_2 - DC_1}\right) DC_1$$

Así, la transformación para pasar de PWM a velocidad angular es la siguiente,

$$\omega(DC) = \left(\frac{\omega_{max}}{DC_2 - DC_1}\right) DC - \left(\frac{\omega_{max}}{DC_2 - DC_1}\right) DC_1$$

Por otra parte, para obtener la velocidad angular en función de la acción de control, $\omega(u')$, se tiene que encontrar la ecuación definida por, g compuesta con f, es decir, $(g \circ f)(u')$, donde g y f se definen como,

$$f = DC(u')$$
$$g = \omega(DC)$$

$$\begin{aligned}\omega(u') = (g \circ f)(u') = g[f(u')] = g\left[\left(\frac{DC_2 - DC_1}{u_2'}\right) u' + DC_1\right] = \\ = \left(\frac{\omega_{max}}{DC_2 - DC_1}\right)\left[\left(\frac{DC_2 - DC_1}{u_2'}\right) u' + DC_1\right] - \left(\frac{\omega_{max}}{DC_2 - DC_1}\right) DC_1 = \\ = \frac{\omega_{max}}{u_2'} u' + \frac{\omega_{max} DC_1}{DC_2 - DC_1} - \frac{\omega_{max} DC_1}{DC_2 - DC_1} = \frac{\omega_{max}}{u_2'} u'\end{aligned}$$

Se observa que la transformación de acción de control a velocidad angular tiene una ganancia neta igual a $\frac{\omega_{max}}{u_2'}$. Por otra parte, descomponiendo la acción de control se tiene que $u' = k_p e(t) + k_i \int e(t)dt + k_d \frac{de}{dt}$. Entonces, multiplicar ambas da como resultado

$$\omega(e) = \frac{\omega_{max}}{u_2'}\left(k_p e(t) + k_i \int e(t)dt + k_d \frac{de}{dt}\right)$$

Ahora bien, los parámetros de las transformaciones son variables y se pueden cambiar en cualquier momento por otros que representen mejor la realidad, en tal caso se pueden plantear dos escenarios a modo de ejemplo.

$$1^{\underline{er}} \text{ caso: } \frac{1047}{100}\left(k_p e(t) + k_i \int e(t)\,dt + k_d \frac{de}{dt}\right) = C_1 \quad \begin{cases} \omega_{\max} = 1047\ \frac{\text{rad}}{s} \\ u_2' = 100 \end{cases}$$

$$2^{\underline{do}} \text{ caso: } \frac{860}{100}\left(k_p e(t) + k_i \int e(t)\,dt + k_d \frac{de}{dt}\right) = C_2 \quad \begin{cases} \omega_{\max} = 860\ \frac{\text{rad}}{s} \\ u_2' = 100 \end{cases}$$

Hay que notar que para un error dado, e, modificar los parámetros de las transformaciones da distintos resultados, $C_1 \neq C_2$. Esto representa un inconveniente, ya que el comportamiento del controlador se ve alterado y, en consecuencia, la sintonización de las ganancias k_p, k_i, k_d para C_1 puede no funcionar para C_2. Este motivo impulsa a normalizar la transformación, quitando la ganancia neta $\frac{\omega_{max}}{u_2'}$. Luego, lo que se busca es

$$\omega(u') = u'$$

Para ello, simplemente se tiene que agregar un nuevo bloque que anule la ganancia neta[11], es decir,

$$\frac{\omega_{max}}{u_2'} G_u = 1, \text{ tal que}$$

$$G_u = \frac{u_2'}{\omega_{max}}$$

Finalmente, en la Figura 3.18 se puede visualizar el diagrama en bloques con la ganancia normalizada.

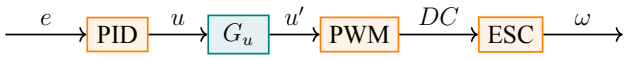


Figura 3.18: Diagrama en bloques de las transformaciones, con la ganancia normalizada

3.2.4 Modelo completo

En la Figura 3.19 se puede apreciar el diagrama de todos los componentes que conforman al cuadricóptero y cómo se conectan entre sí. El diseño del sistema de control utilizado permite mantenerlo a una cierta altura y orientación con respecto al sistema inercial $\{E\}$, a través de los controladores PID[12]. Entonces, si los valores medidos por los sensores son distintos a las referencias definidas por el usuario (sp_i), habrá un error, e_i, dinstinto de 0. Luego, los controladores realizarán el cálculo apropiado para así obtener una acción de control u_i. Esta última, ingresa al bloque *Motor Mixing Algorithm* encargado de combinar las acciones de cada controlador y determinar el valor apropiado para las velocidades angulares de cada motor. Continuando, el bloque *Mapeos* se encarga de modelar las señales PWM y los ESC, para transformar acciones de control en velocidades angulares, ω_i. Por último, se agregó el modelo de los motores para que la respuesta obtenida sea más realista y finalmente la planta que junto con los sensores brindan los estados del sistema.

[11] Si bien el efecto de G_u es anular el impacto de la ganancia neta sobre el controlador, es importante notar que su ecuación resultante va a depender de las transformaciones. Por tanto, en caso de modificar estas últimas es necesario volver a calcular G_u.

[12] Se han reducido los controladores a un simple bloque para simplificar el diagrama, no obstante, su funcionamiento y modelo completo es el mismo al explicado en la *Sección 3.2.1: Controladores*.

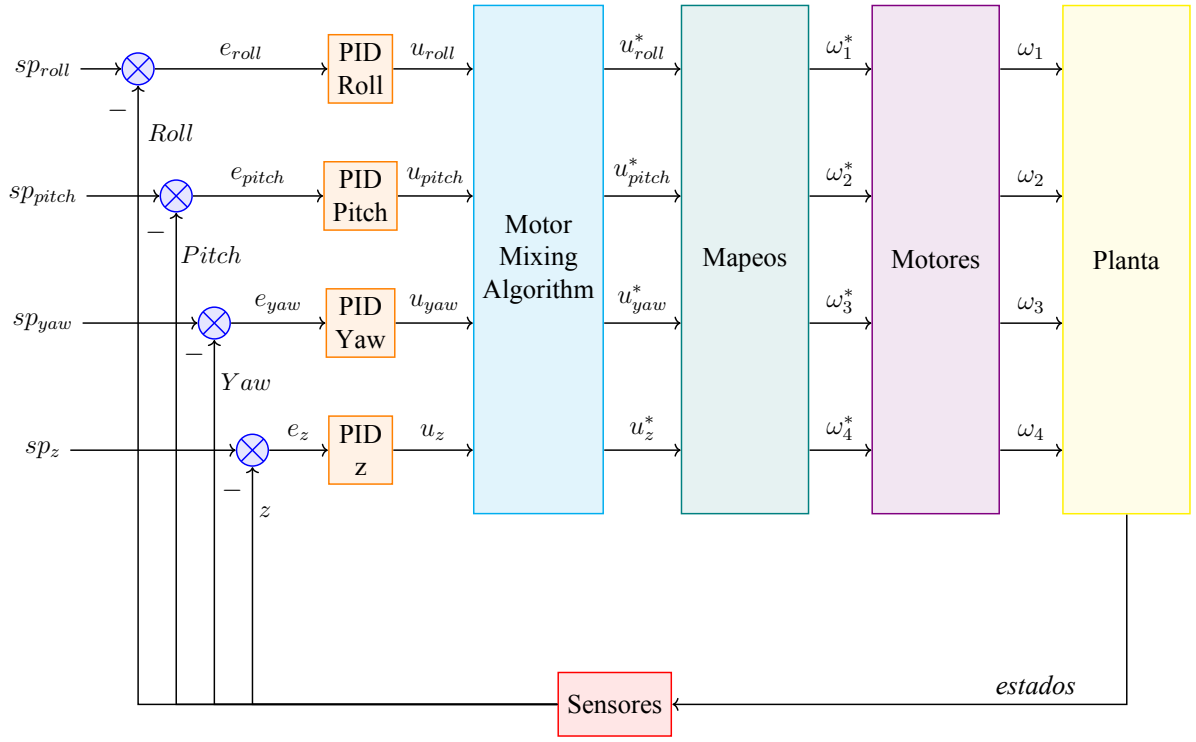


Figura 3.19: Diagrama del modelo completo del cuadricóptero, en Simulink

3.3 Simulaciones

En esta sección se mostrarán las ganancias obtenidas durante la sintonización de cada controlador, así como también los gráficos resultantes de las simulaciones realizadas.

3.3.1 Sintonización de Roll

Para este estado, se utiliza un controlador interno de tipo PD y otro externo de tipo P. En la Tabla VIII se encuentran detallados los valores finales de las ganancias asociados a cada controlador.

	PID interno (velocidad)	**PID externo (posición)**
P	300,0	100,0
I	0,0	0,0
D	80,0	0,0

Tabla VI: Ganancias de los controladores de roll

La Figura 3.20 muestra la respuesta, simulada en 10 segundos, de roll frente a una referencia sinusoidal de amplitud A igual a $10° = \left(\frac{\pi\ rad}{180°}\right) 10° = 0.17453\ rad$ y velocidad angular $w = \frac{2\pi\ rad}{4(1)\ \text{seg}}$.

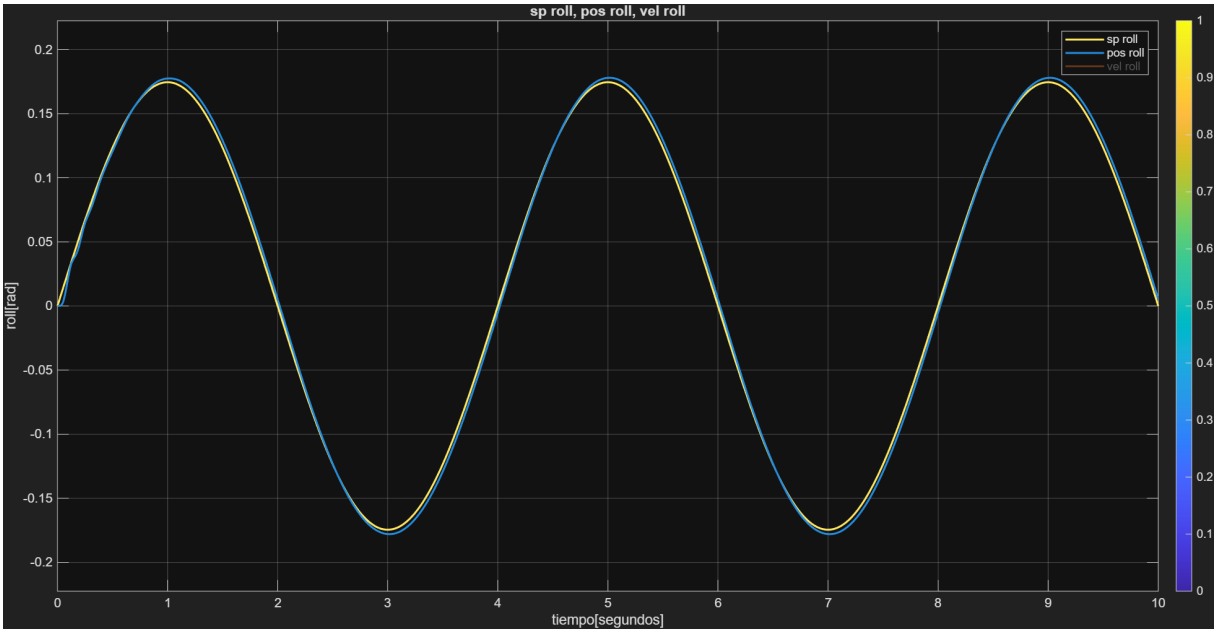


Figura 3.20: Simulación de 10 segundos de roll frente a una referencia sinusoidal, en Simulink

Es decir, se le está pidiendo al cuadricóptero que rote $10°$ sobre su eje x en 1 segundo.

3.3.2 Sintonización de Pitch

Puesto que se han simplificado los modelos de las aceleraciones angulares, correspondientes a cada ángulo, para que no dependan del resto de estados, no es de extrañar que la respuesta de pitch sea muy similar a la de roll ya que poseen la misma dinámica.

En la Tabla VII se encuentran los valores finales de las ganancias de cada controlador.

	PID interno (velocidad)	**PID externo (posición)**
P	300.0	100.0
I	0.0	0.0
D	80.0	0.0

Tabla VII: Ganancias de los controladores de pitch

Al igual que en roll, en la Figura 3.21 se vuelven a simular 10 segundos con la respuesta de pitch frente a una referencia sinusoidal de amplitud A igual a $10° = \left(\frac{\pi\ rad}{180°}\right) 10° = 0.17453\ rad$ y velocidad angular $w = \frac{2\pi\ rad}{4(1)\ \text{seg}}$.

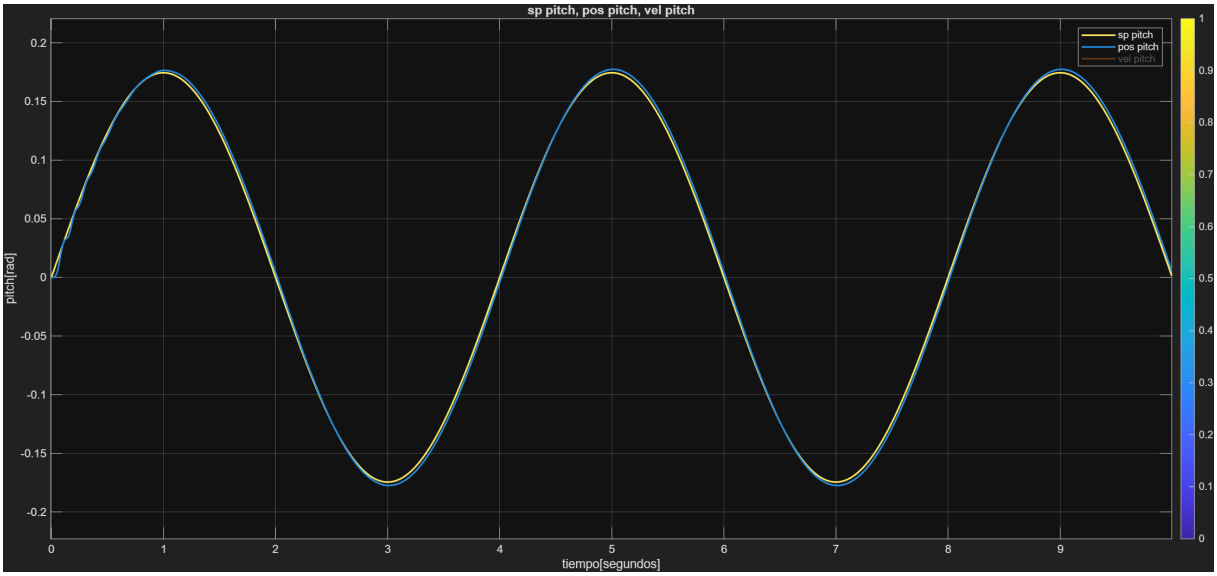


Figura 3.21: Simulación de 10 segundos de pitch frente a una referencia sinusoidal, en Simulink

3.3.3 Sintonización de Z

Para controlar la altura, se utilizan ambos controladores, tanto interno como externo, de tipo PID. En la Tabla VIII se encuentran los valores finales de las ganancias asociados a cada controlador.

	PID interno (velocidad)	**PID externo (posición)**
P	800,0	60,0
I	20,0	0,8
D	300,0	4,0

Tabla VIII: Ganancias de los controladores de z

La Figura 3.22 muestra la respuesta, simulada en 10 segundos, de z frente a una combinación de varios escalones.

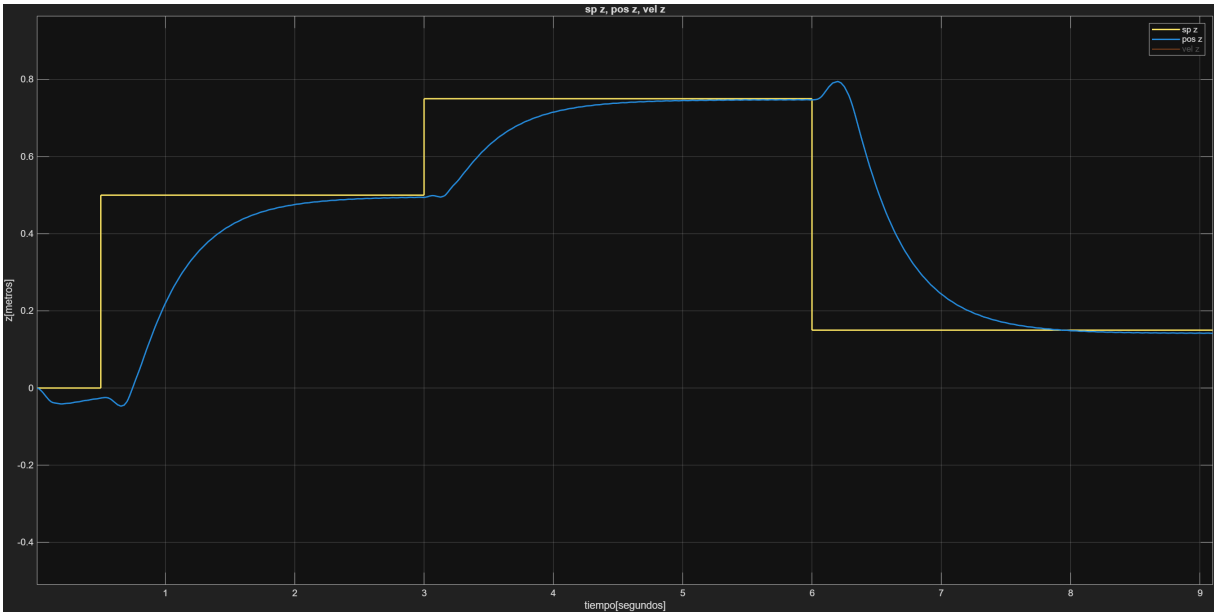


Figura 3.22: Simulación de 10 segundos de z frente a una combinación de escalones, en Simulink

Con el fin de evitar que los setpoint recibidos sean demasiado bruscos, se implementa un filtro pasa bajos cuya función es “suavizar” la señal de entrada. Esto se ve gráficamente en la Figura 3.23, que expresa una clara mejora en la variable de salida.

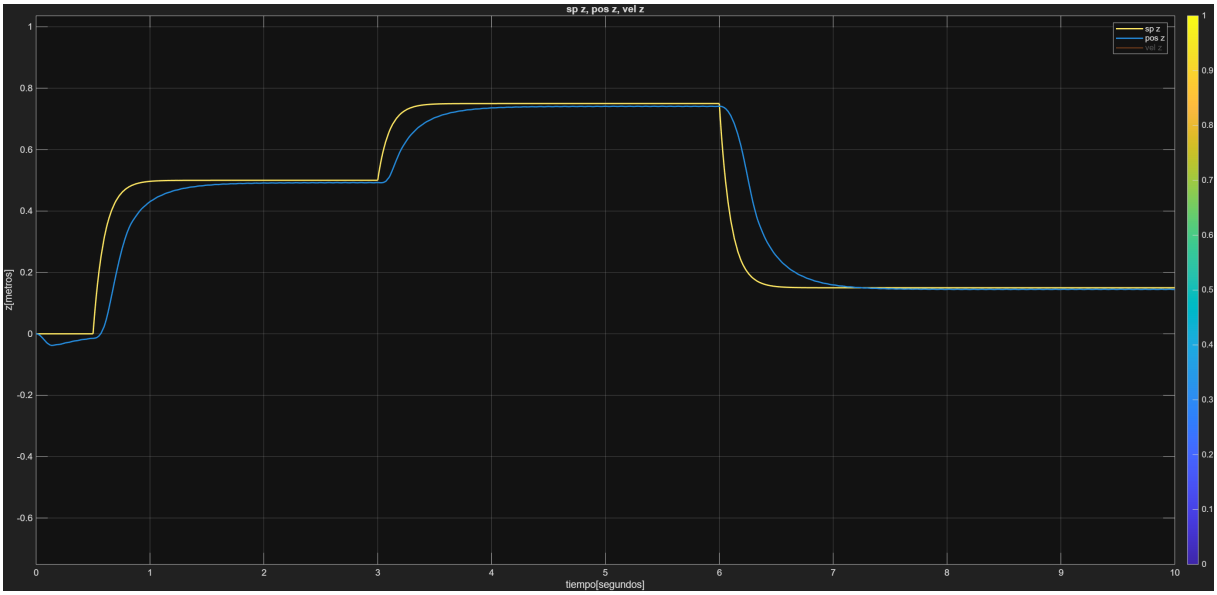


Figura 3.23: Simulación de z con el setpoint filtrado, en Simulink

3.4 Identificacion de sistema

El objetivo de esta sección es exponer los métodos utilizados para identificar los parámetros físicos de los modelos desarrollados sobre los distintos componentes del cuadricóptero.

3.4.1 Momentos de inercia

Los primeros parámetros a analizar serán los momentos de inercia principales, j_{xx}, j_{yy}, j_{zz}, según (Quan, 2017, Sec. 6.3.3). Estos se pueden determinar a través de un péndulo bifilar, como el de la Figura 3.24 donde,

- m_0 es la masa del objeto en cuestión.
- g es la aceleración gravitacional.
- El objeto se suspende a través de dos finos cables, de igual longitud.
- La distancia entre cables es de d metros.
- La longitud de cada cable es de L metros.

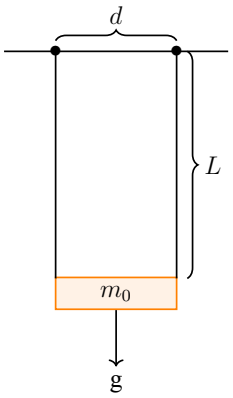


Figura 3.24: Péndulo bifilar

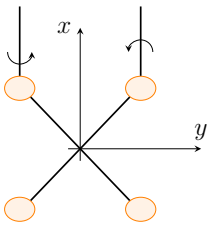
El período de oscilación se puede calcular como

$$T_0 = 4\pi\sqrt{\frac{jL}{m_0 g d^2}}$$

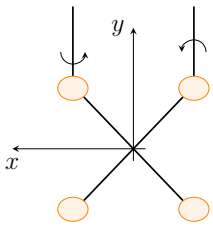
y despejando la inercia j resulta,

$$j = \frac{m_0 g d^2}{L}\left(\frac{T_0}{4\pi}\right)^2.$$

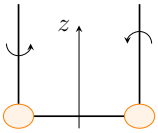
La Figura 3.25 es la analogía del péndulo bifilar aplicada a cada eje del cuadricóptero. Luego se detalla el procedimiento para medir el valor aproximado de cada inercia.



(a) Medición de j_{xx}



(b) Medición de j_{yy}



(c) Medición de j_{zz}

Figura 3.25: Medición de cada momento principal de inercia

1. Conectar dos vértices del cuadricóptero a dos agarres (gancho o similar), a través de dos finos cables y esperar hasta que deje de balancearse.

2. Asegurarse que los cables sean paralelos al eje que corresponda y medir su longitud L. Asimismo, también asegurarse que la distancia entre cables d sea el doble de la distancia de cada cable al eje, y medirla.

3. Rotar levemente[13], menor a $10°$, el cuadricóptero alrededor del eje en cuestión y medir el tiempo, T, que tarda en alcanzar 50 períodos. Entonces, el período del péndulo bifilar se expresa como $T_0 = \frac{T}{50}$.

El proceso será el mismo para los 3 casos y consistirá en medir el período de oscilación cuatro veces consecutivas y calcular su promedio. Luego, con este valor se procede al cálculo del período del péndulo bifilar, T. Por último, se usa este valor para obtener la inercia en cuestión.

Inercia en el eje x (j_{xx})

Los parámetros medidos son, $L = 0,51m, d = 0,34m$ y $g = 9,81\frac{m}{s^2}$. Luego,

$$T = \frac{1}{50}\sum_{i=1}^{n=4}\frac{T_i}{4} = \frac{41,54s + 41,53s + 41,34s + 41,42s}{4(50)} = \frac{165,83s}{200} = 0,82915s$$

tal que,

$$j_{xx} = \frac{(1,112kg)(9,81\frac{m}{s^2})(0,34m)^2}{0,51m}\left(\frac{0,82915s}{4\pi}\right)^2 = 0,010765m^2kg \approx 0,011m^2kg$$

Inercia en el eje y (j_{yy})

Los parámetros medidos son, $L = 0,525m, d = 0,34m$ y $g = 9,81\frac{m}{s^2}$. Luego,

$$T = \frac{1}{50}\sum_{i=1}^{n=4}\frac{T_i}{4} = \frac{43,79s + 43,75s + 43,575s + 43,585s}{4(50)} = \frac{174,7s}{200} = 0,8735s$$

tal que,

$$j_{yy} = \frac{(1,112kg)(9,81\frac{m}{s^2})(0,34m)^2}{0,525m}\left(\frac{0,87345s}{4\pi}\right)^2 = 0,011604m^2kg \approx 0,012m^2kg$$

Inercia en el eje z (j_{zz})

Los parámetros medidos son, $L = 0,5325m, d = 0,085m$ y $g = 9,81\frac{m}{s^2}$. Luego,

[13] Si bien la Figura 3.25 muestra un sentido de giro antihorario, este no es relevante. Por tanto, se puede rotar al cuadricóptero tanto en sentido horario como antihorario.

$$T = \frac{1}{50}\sum_{i=1}^{n=4}\frac{T_i}{4} = \frac{114,42s + 109,2s + 112,67s + 113,46s}{4(50)} = \frac{449,75s}{200} = 2,24875s$$

tal que,

$$j_{zz} = \frac{(1,112kg)(9,81\frac{m}{s^2})(0,085m)^2}{0,5325m}\left(\frac{2,24875s}{4\pi}\right)^2 = 0,00474m^2kg \approx 0,005m^2kg$$

Finalmente, en la Figura 3.26 se puede observar la estructura de prueba que utilizamos para la medición de cada período de oscilación T.

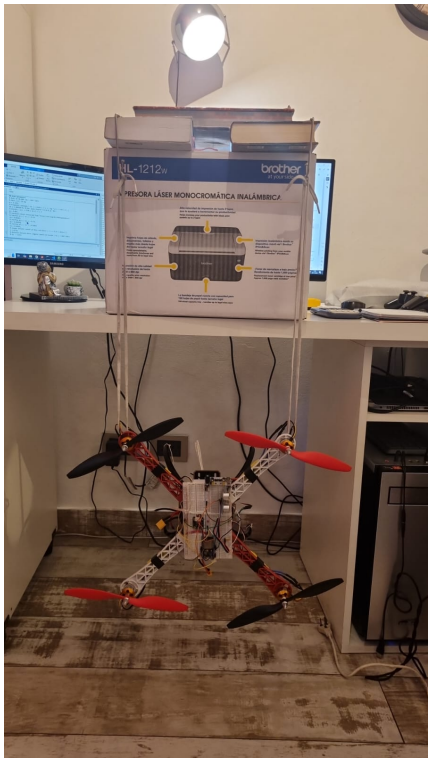


Figura 3.26: Estructura de prueba para medir los momentos de inercia del cuadricóptero

3.4.2 Motores

Otro parámetro importante para identificar son los parámetros de los motores, para poder tener una idea de sus capacidades y limitaciones.

Se planteó y ejecutó el ambiente de prueba mostrado en la figura 3.27 para analizar la velocidad de giro y la corriente consumida en función del ancho de pulso de la señal de control de los controladores electrónicos de velocidad, medida como porcentaje de ciclo de trabajo.

La metodología de prueba fue

- 1. Alimentar la ESP32 y los motores, y colocar el dron en modo “Calibración de hélices” que permite aumentar la velocidad de los motores desde el mínimo hasta el máximo calibrado en los controladores de velocidad, con incrementos discretos del 10 %.
- 2. Aumentar la velocidad un 10 %.

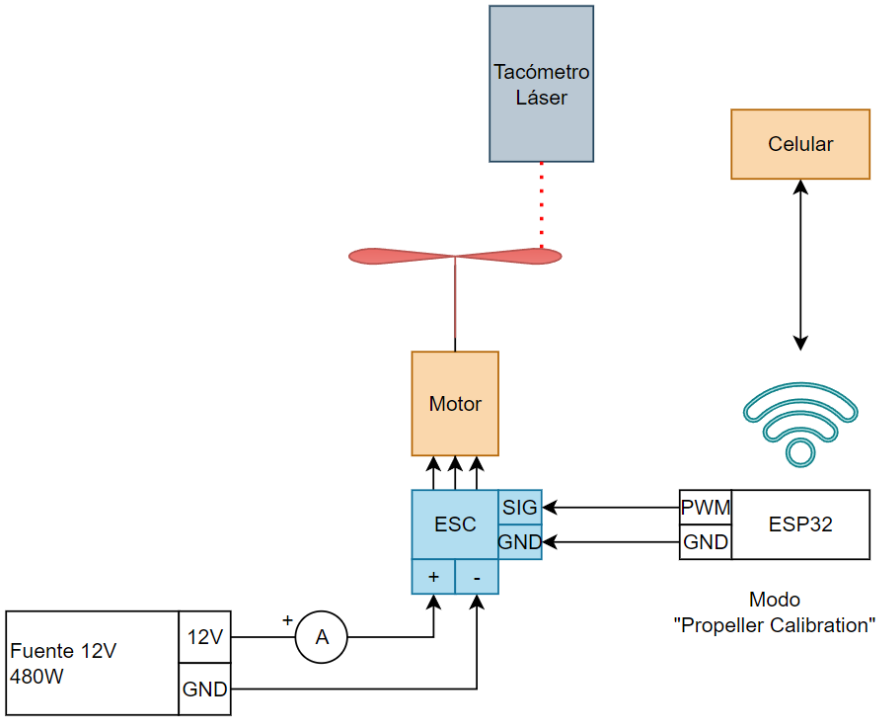


Figura 3.27: Diagrama orientativo para la medición de corriente y velocidad

- 3. Tomar los datos de corriente medida por el amperímetro, velocidad con el tacómetro láser y ciclo de trabajo.
- 4. Repetir 2 y 3

Con esta metodología se obtuvieron los resultados indicados en la tabla IX

Tabla IX: Comparación de fabricantes de *PCB*

Ciclo de trabajo	Corriente [A]	Revoluciones por minuto
0,0613	0,550	1499
0,0655	0,857	2300
0,0697	1,4	3172
0,0739	2,3	4200
0,0781	3,4	4500
0,0823	4,8	5000
0,0865	6,4	5730
0,0907	8,05	6216
0,0949	9,7	6751
0,0970	10,6	10575

Los gráficos resultantes son los siguientes:

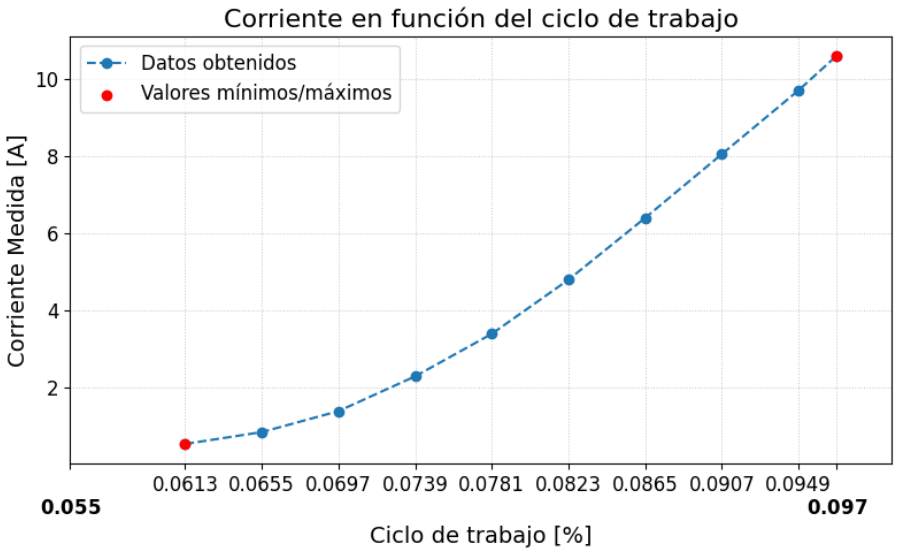


Figura 3.28: Gráfico de la corriente en función del ciclo de trabajo de los controladores de velocidad

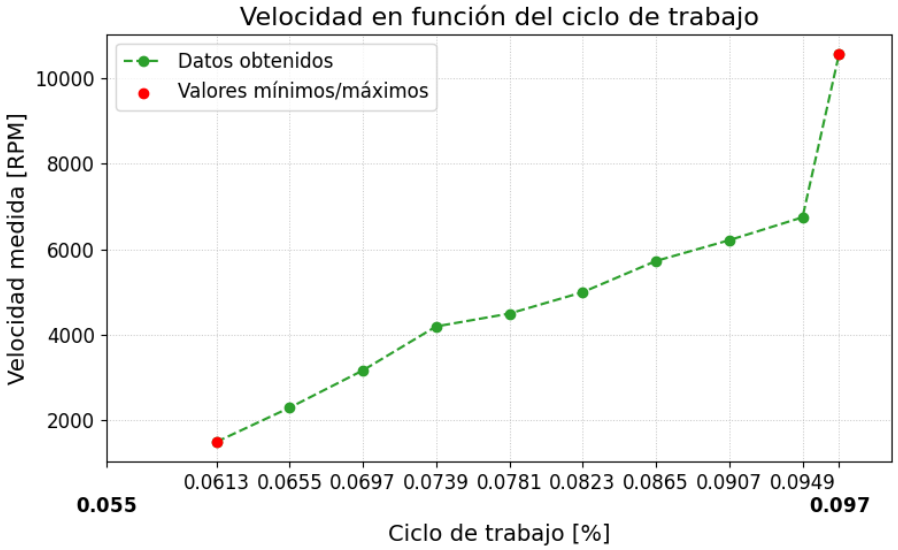


Figura 3.29: Gráfico de la velocidad de giro en función del ciclo de trabajo de los controladores de velocidad

Interpolando ambos gráficos, se obtiene el gráfico de la curva característica “Corriente-Velocidad de giro” del motor usado:

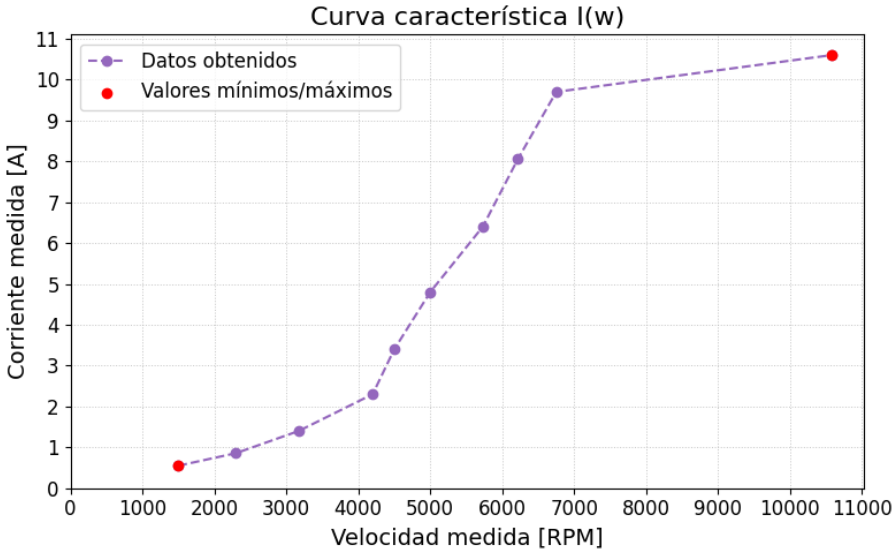


Figura 3.30: Curva característica I-W del motor utilizado a partir de los gráficos obtenidos.

3.4.3 Peso del sistema

En esta sección, se expondrán los pesos de cada componente incluido en el cuadricóptero.

Componente	Cantidad [UN]	Peso [g]
Motor BLDC	4	48
ESC	4	37
PCB (completo)	1	30
Hélice	4	15
Batería	1	200
Frame	1	270
Total	-	900

Tabla X: Peso de cada componente

Capítulo 4

Firmware

En esta sección se detallarán las herramientas utilizadas para desarrollar el firmware del cuadricóptero, así como también la estructura y de los componentes que lo conforman.

4.1 Herramientas necesarias

El firmware se desarrolló utilizando la herramienta Visual Studio Code, de aquí en adelante abreviado como "VSCode", ("Visual Studio Code", s.f.) junto con la extensión ESP-IDF, de espressif (la instalación de la misma se detalla en el *Apéndice B: Instalación de la extensión ESP-IDF en VSCode*).

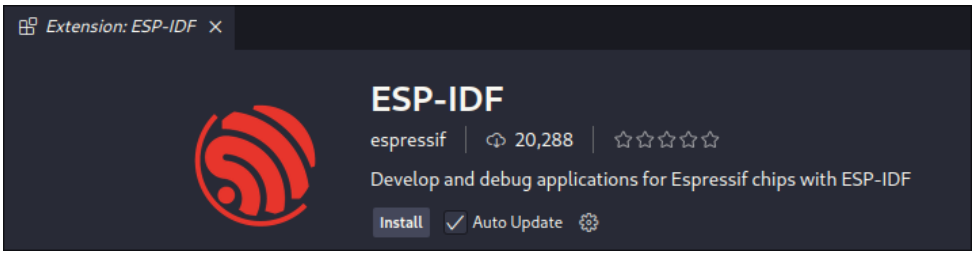


Figura 4.1: Extensión de ESP-IDF, en vscode

4.2 Estructura del código fuente

La estructura generada por defecto al crear un nuevo proyecto con la opción *template-app* es la siguiente.

```
├── CMakeLists.txt
├── main/
│   ├── CMakeLists.txt
│   └── main.c
└── README.md
```

Figura 4.2: Estructura por defecto de un proyecto en ESP-IDF

No obstante, se la modificó de la siguiente manera

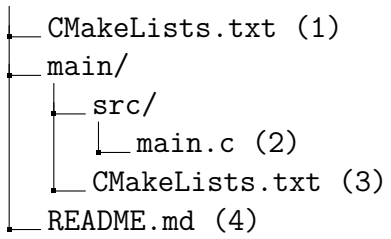


Figura 4.3: Estructura modificada de un proyecto en ESP-IDF

En definitiva, cada componente contiene una carpeta *include* (el componente main no tiene ya que no contiene ningún *.h*) en la cual se almacenan todos los *.h*, y otra carpeta *src* donde se encuentran todos los *.c*.

- (1) CmakeLists.txt: Contiene la versión mínima requerida de cmake, la ruta en el sistema a cmake y el nombre del proyecto, tal como se puede apreciar en el Listing 4.1

```
cmake_minimum_required(VERSION 3.5)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(Drone)

```

Listing 4.1: CMakeLists.txt del proyecto

- (2) main/main.c: Es el código principal.
- (3) main/CMakeLists.txt: Contiene todos los archivos fuente (*.c*) utilizados por el componente, los directorios de todos los encabezados (*.h*) utilizados[1], y los componentes requeridos.

```
idf_component_register(SRCS "src/main.c"
                       INCLUDE_DIRS "."
                       REQUIRES "tasks" "drone")

```

Listing 4.2: CMakeLists.txt del componente main

- (4) README.md: Es un archivo utilizado para describir brevemente las principales características del proyecto o trabajo en cuestión. Luego, su uso es recomendable pero no mandatorio.

Retomando la modificación de la estructura de base del proyecto, en la Figura 4.4 se muestra la estructura de un componente a modo de ejemplo.

[1] Exceptuando main, para el resto de componentes quedaría, INCLUDE DIRS ”include”.

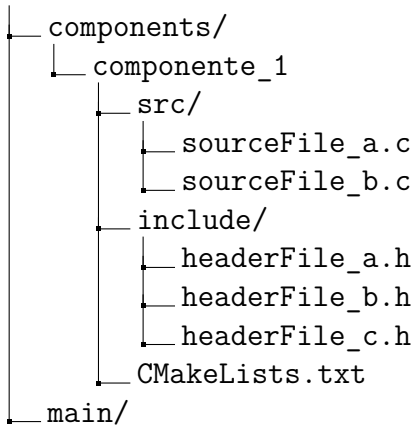


Figura 4.4: Estructura de un componente

Entonces, el CMakeLists.txt del componente “componente_1” sería el indicado en el Listing 4.3.

```
idf_component_register(
    SRCS "src/sourceFile_a.c" "src/sourceFile_b.c"
    INCLUDE_DIRS "include"
    REQUIRES "required_component_1" "required_component_2" "required_component_n"
)
```

Listing 4.3: CMakeLists.txt de un componente

Por último, es necesario incluir componentes externos cuando se utiliza cualquiera de sus encabezados. Es decir, en caso de que se incluya el archivo motores.h, del componente “motores”, en el componente_1, entonces se tiene que poner *REQUIRES “motores”*.

4.3 Capas de abstracción en los controladores de sensores

En la Figura 4.5 se observa el diseño estructural del controlador, o driver, utilizado por cada sensor del cuadricóptero. Fundamentalmente, el objetivo es separar el firmware asociado a la manipulación de registros, del firmware asociado a las tareas que tienen un nivel macro.

Luego, la manipulación de registros está asociada a la modificación de los bits de los distintos registros del sensor. Esto es necesario para configurarlo, obtener mediciones útiles a partir de las lecturas de su ADC, reiniciarlo, entre otras. En cambio, las operaciones a nivel macro hacen referencia a, por ejemplo, ejecutar todas las funciones necesarias para inicializar el sensor o leer aquellos registros donde el sensor almacena los valores leidos del ADC y convertirlos en mediciones útiles.

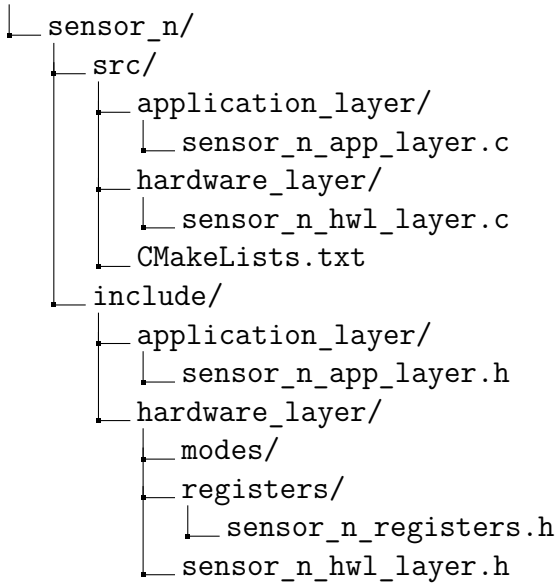


Figura 4.5: Estructura del controlador de cada sensor

4.4 Arquitectura general del firmware

Para este proyecto se decidió utilizar un patrón de diseño denominado *Singleton* debido a la simplicidad del mismo, y a las ventajas cruciales que ofrece para sistemas embebidos. Entre estas, destacan la optimización de memoria, al evitar la creación de múltiples instancias, la consistencia de estado, asegurando que todos los módulos interactúen con la misma instancia de hardware; y la facilidad de acceso a las variables necesarias.

El *singleton* implementado es *drone*, que contiene todos los demás componentes y algunos métodos generales que se detallan en las siguientes figuras. Como *drone* contiene dentro de sí otras estructuras, si se quiere limitar el acceso de un componente o método solo pasando la dirección de memoria de lo que se desea que se tenga acceso. Por ejemplo, el transmisor *Tx* no necesita tener acceso a todo *drone*, sino solo a sus variables globales o al estado de los botones.

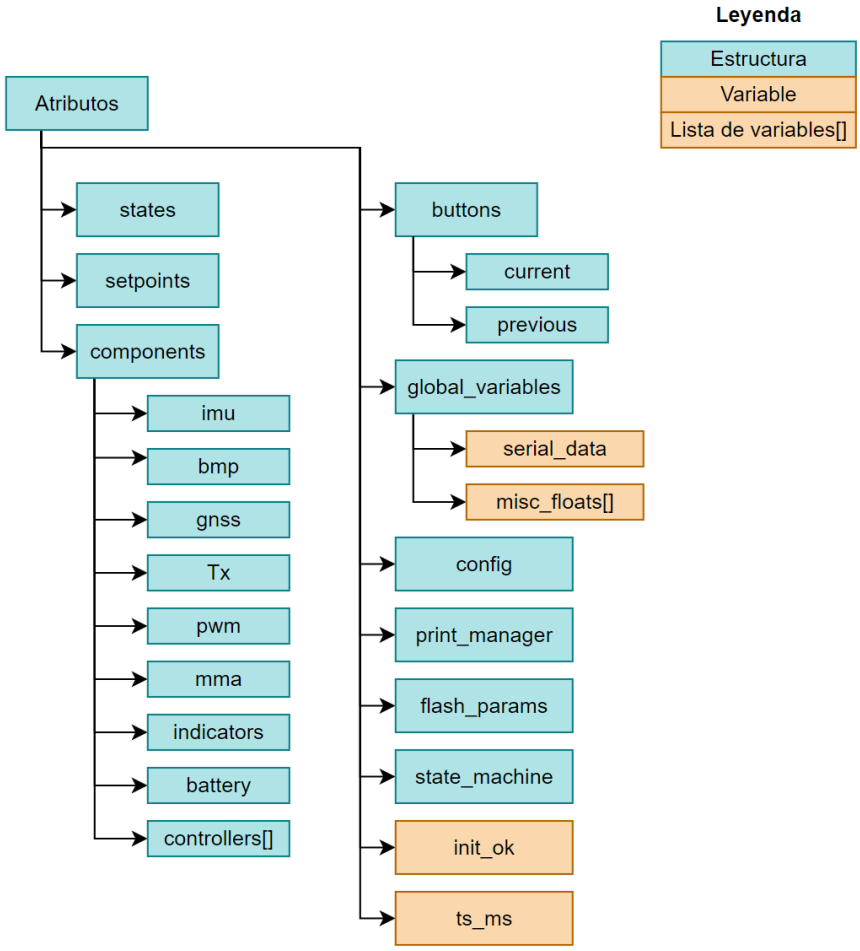


Figura 4.6: Diagrama de los atributos principales del firmware

Cada componente a su vez posee métodos y atributos. Los métodos toman como argumento principal no la dirección de memoria del *drone*, sino la del objeto al que pertenecen, es decir, tienen alcance limitado. Esto permite segregar el alcance de cada función para evitar errores o modificaciones de variables no deseadas, y le da un orden lógico a la ubicación de cada variable.

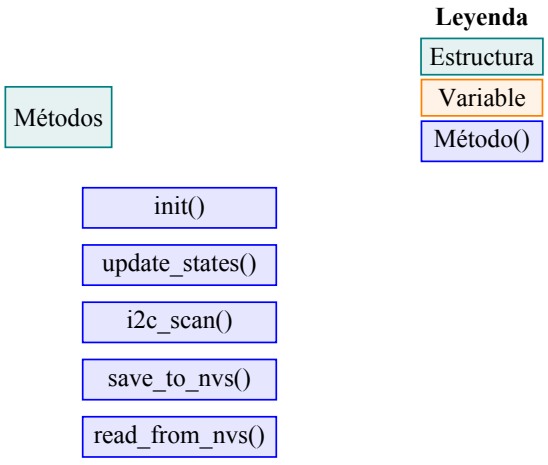


Figura 4.7: Diagrama de los métodos principales del firmware

4.5 Implementación de tareas concurrentes

Como se ha explicado en la Sección 2.6, para suplir la principal necesidad a nivel firmware de reducir el tiempo entre muestras lo máximo posible, se utilizó el driver FreeRTOS.

La ESP32 cuenta con 2 núcleos de los cuales el usuario dispone para ejecutar hilos en paralelo. Por tanto, las tareas se distribuyeron de forma tal que aquellas de mayor relevancia se ejecuten en un núcleo y el resto en otro.

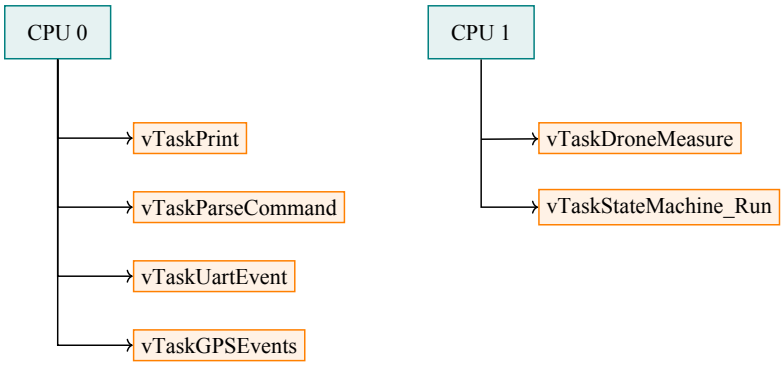


Figura 4.8: Tareas de FreeRTOS

Como se puede observar, las principales tareas se ejecutan en el núcleo 1 mientras que las tareas secundarias lo hacen en el núcleo 0.

- **vTaskPrint**: Envía el valor de las variables más relevantes mediante puerto serie, para poder visualizarlas en el Serial Plotter (ver *Sección 5.1: Software de monitoreo en tiempo real mediante USB*) en tiempo real. Además, se aprovechó esta tarea de baja prioridad para manejar otras funcionalidades como prender o apagar los leds y medir la carga restante de la batería
- **vTaskParseCommand**: Se encarga de ejecutar la función asociada al comando recibido por Bluetooth o UART. A continuación se listan las funcionalidades implementadas según el Listing 4.4.
    - Actualizar el algoritmo (la función) que utiliza cada acción P/I/D de cada controlador específico.
    - Actualizar una variable almacenada en la memoria NVS del cuadricóptero.
    - Actualizar los setpoints.
    - Guardar los valores actuales de las variables alojadas en RAM del dron en la memoria NVS.
    - Modificar por comando los estados de los botones, esto permite depurar y probar el código desde la interfaz serie, sin necesidad de comunicarse mediante un transmisor real.

```
1  static cmd_function_t cmd_function_array[] = {
2      {.cmd_name = "pid actions", .func = &PidActionsCmdFunc},
3      {.cmd_name = "var update",  .func = &VarsUpdateCmdFunc},
4      {.cmd_name = "sp update",   .func = &SpUpdateCmdFunc},
5      {.cmd_name = "nvs_store",   .func = &NvsStoreCmdFunc },
```

```
        {.cmd_name = "tx",          .func = TxCmdFunc}
    };

```

Listing 4.4: Comandos disponibles

Las ventajas que otorga esta definición, son la simplicidad para agregar nuevos comandos o quitar existentes, y la modularidad respecto al procesamiento de estos ya que el hecho de agregar o quitar alguno no provoca que el programa deje de funcionar.

- **vTaskUartEvent**: Detecta eventos de recepción de datos en el puerto UART0, destinado a propísito general, y almacenar la información recibida en el buffer de mensajes del dron.
- **vTaskGPSEvent**: Detecta eventos de recepción de datos en el puerto UART1, destinado a GPS, y ejecutar el método *measure* del módulo GNSS con la información recibida.
- **vTaskDroneMeasure**: Se encarga de leer los registros de cada sensor y actualizar los estados del cuadricóptero haciendo uso de los nuevos valores medidos.
- **vTaskStateMachine_Run**: Esta tarea es la encargada de decidir qué tiene que hacer el cuadricóptero en función de los eventos ocurridos, tal como se explicó en la *Sección 4.6: Lógica de comportamiento principal del dron* y, debido a su relevancia y cantidad de operaciones, es la única tarea que se ejecuta en el procesador número 1 del MCU.

  Lo que es interesante destacar de esta tarea, en esta sección, es la manera en la cual está compuesta. Por un lado, se define la *matriz de transición* como aquella que contiene en cada fila, el estado actual, el evento ocurrido y el estado futuro.

$$\begin{bmatrix} estado\ actual & evento\ ocurrido & estado\ futuro \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{bmatrix}$$

  Se puede pensar a esta matriz como un mapa de “donde estoy, qué sucedió y a dónde tengo que ir”.

  El último paso, consiste en recorrer la *matriz de transición* con un bucle y verificar el estado actual, el evento ocurrido y a cuál estado hay que transicionar ejecuntando su respectiva función. En el *Apéndice C: Implementación en firmware de la máquina de estados* se explica, con mayor énfasis en el código, como se interconecta entre sí cada parte del programa.

4.6 Lógica de comportamiento principal del dron

Nuestra prioridad al momento de diseñar la forma de trabajo del cuadricóptero fue tener la certeza en todo momento de qué es lo que está haciendo y qué es lo que hará a partir de un determinado evento.

Por ello, se optó por modelar su comportamiento en base a una máquina de estados. En términos generales, una máquina de estados es una forma de representar el comportamiento de un sistema frente a diversos eventos. Todos los estados son consecuencia de un evento dado, incluyendo a “sin cambios” como un evento. Luego, los estados se representan mediante círculos y se conectan entre sí a través de flechas. Asimismo, cada flecha hace referencia a un evento.

La ventaja de este tipo de representación es que permite definir con exactitud cada posible estado de un sistema, así como también qué acción hay que tomar a partir de un evento dado.

En la Figura 4.9 se puede apreciar el diagrama de la máquina de estados que rige el comportamiento del cuadricóptero.

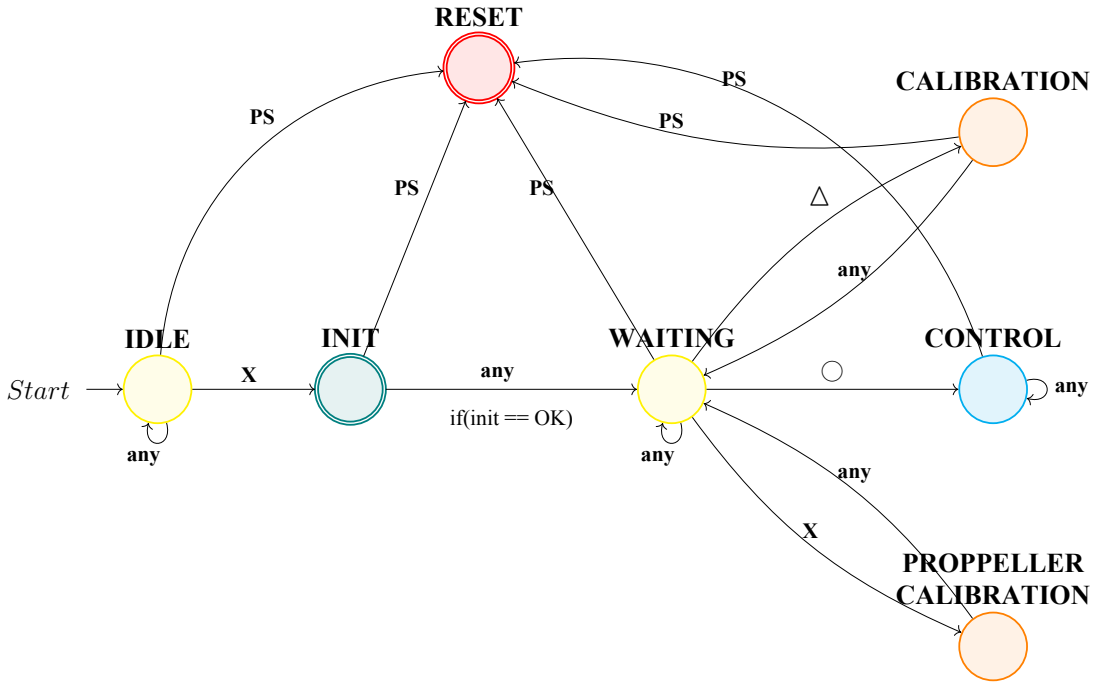


Figura 4.9: Diagrama de la máquina de estados del cuadricóptero

A continuación se detalla cada estado y sus respectivas definiciones.

- **IDLE**: Es el estado al cual el sistema transiciona inmediatamente después de encender el microcontrolador. Además, en este estado no se realiza ninguna acción, solo se espera hasta que el usuario decida inicializar el cuadricóptero.
- **INIT**: En este estado se inicializan todos los componentes necesarios para garantizar el correcto funcionamiento del cuadricóptero.
- **WAITING**: Este estado garantiza que todos los componentes del cuadricóptero fueron inicializados correctamente y se encuentran disponibles para ser utilizados por el resto del firmware. Luego, el cuadricóptero se mantiene a la espera sin realizar ninguna operación.
- **CALIBRATION**: Consiste en la calibración automática de los ESC (ver rutina de calibración en Hobbywing, s.f.). Al finalizar, se retorna al estado WAITING.
- **CONTROL**: En este estado el cuadricóptero se encuentra constantemente actualizando las acciones de cada controlador en base a los valores medidos por los sensores. Por otra parte, nótese que no es posible transicionar a otro estado más que a RESET, en caso de que algo falle.
- **PROPPELLER CALIBRATION**: Este estado permite calibrar manualmente las hélices a partir de los datos de vibración que estas arrojan. Al finalizar se retorna al estado WAITING nuevamente.

- **RESET**: Se utiliza en caso de errores y consiste en apagar los motores y luego resetear el microcontrolador. Así, el objetivo de este estado es devolver el sistema a un estado seguro.

El usuario puede calibrar los ESC y las hélices cuantas veces quiera, sólo hasta que el cuadricóptero transicione al estado de CONTROL. Se entiende que una vez iniciado el vuelo, el cuadricóptero ya se encuentra calibrado y con la configuración que corresponda. Por tal motivo, no es posible salir de este último estado en condiciones normales de operación.

Finalmente, se detallan los posibles eventos.

- X: Botón X del transmisor.
- any: Significa que no ha sucedido ningún evento. Es decir, al finalizar todas las tareas de un estado en particular, automáticamente se transiciona al siguiente.
- $\triangle$: Botón triángulo del transmisor.
- $\bigcirc$: Botón círculo del transmisor.

4.7 Implementación de algoritmos de control

En esta sección se explicarán los algoritmos asociados a la etapa de control, desde que entra el error al sistema hasta que se determinan los ciclos de trabajo. Es importante mencionar que el código mostrado se simplificará lo máximo posible, haciendo foco en transmitir la idea del diseño para cada algoritmo. Por tanto, las funciones y/o líneas de código pueden diferir del firmware real implementado en el MCU, en caso de requerir el código original ver (Scoflich Lautaro y Grandinetti Juan, 2025).

En primera instancia, lo primero que sucede es el ingreso del error en los controladores configurados en cascada.

```
C_pos = pidUpdate(pos_actual, sp);
C_vel = pidUpdate(vel_actual, C_pos);
```

Listing 4.5: Controladores en cascada

Examinando el Listing 4.5 se puede identificar que, C_pos es la salida del controlador de posición mientras que C_vel es la salida del controlador de velocidad, cuyo setpoint es C_pos tal como se explicó previamente en la *Sección 3.2.1: Controladores*. Por otra parte, este proceso se repite para los 4 estados $z, roll, pitch, yaw$.

Luego, el objetivo fundamental de la función pidUpdate es calcular cada acción P/I/D y sumar los resultados para obtener la acción de control resultante. En formato de código sería algo así como el Listing 4.6.

```
float pidUpdate(float process_value, float sp) {
    error = sp - process_value;

    float pAction = pFunc(error);
    float iAction = iFunc(error);
    float dAction = dFunc(error);

    float pid_out = pAction + iAction + dAction;

    return saturate(pid_out);
}
```

Listing 4.6: pidUpdate

Los aspectos más relevantes a destacar son la modularización de las acciones PID, y la limitación de la salida del controlador para evitar la saturación de los motores. Para el primer caso, el SerialPlotter desarrollado permite modificar las funciones que utiliza cada acción en tiempo de ejecución, de manera que en un ambiente de pruebas es sumamente útil para verificar el comportamiento del sistema sin tener que volver a flashear el código en el MCU.

Una vez calculadas las acciones de control para cada estado, estas ingresan al bloque MMA.

```
mma.input[C_ROLL]  = CRolld;
mma.input[C_PITCH] = CPitchd;
mma.input[C_YAW]   = CYawd;
mma.input[C_Z]     = CZd;

mma.compute(dc_min, dc_max);
```

Listing 4.7: Motor Mixing Algorithm

Se observa, en base al Listing 4.7, que las entradas al componente *mma* son los elementos de un vector con los estados del cuadricóptero, y el método “compute” simplemente calcula las sumas y restas que se obtienen en la *Sección 3.2.2: Motor Mixing Algorithm* y actualiza sus valores de salida con los resultados.

Por último, se actualizan los ciclos de trabajo de los 4 motores.

```
for(int i = 0; i < ((sizeof(drone->attributes.components.mma.output)) / (sizeof(drone->
    attributes.components.mma.output[0]))); i++) {

    drone->attributes.components.pwm[i].set_pwm_dc(
        &drone->attributes.components.pwm[i],
        drone->attributes.components.mma.output[i]
    );
}
```

Listing 4.8: Actualización de los ciclos de trabajo

4.8 Metodología de desarrollo y versionado

Para el desarrollo del firmware y el control de versiones del código fuente se adoptó el sistema Git, una herramienta de control de versiones distribuido muy utilizada en el ámbito del desarrollo de software. Esto permitió gestionar de manera eficiente los cambios, facilitar la colaboración y mantener un historial completo del proyecto. La estrategia de ramificación definida para el proyecto se compone de tres ramas principales con propósitos específicos.

La rama denominada *main* alberga exclusivamente las versiones finales y estables del firmware. Cada *commit* en esta rama representa un hito de desarrollo probado y funcional, correspondiente a una versión específica del dron que está lista para ser desplegada o distribuida. El historial de esta rama es lineal y se mantiene limpio, conteniendo únicamente los estados de lanzamiento oficiales.

La rama identificada como *dev* sirve como la rama de integración principal para el desarrollo continuo. En esta rama se encuentra todo el trabajo de nuevas funcionalidades y posee la última versión del código en estado de desarrollo. El flujo de trabajo dicta que toda nueva característica, ya sea individual o grupal, debe ser desarrollada en una rama de características o *feature branch* que se bifurca directamente desde *dev*. Una vez que el desarrollo de una característica se completa y verifica mediante pruebas básicas, su rama correspondiente se fusiona de vuelta a *dev*. Este proceso de integración continua asegura que los cambios sean consolidados de manera progresiva. Solo cuando el estado de la rama *dev* alcanza un nivel de estabilidad y funcionalidad satisfactorio, tras un ciclo de pruebas, se procede a su fusión con la rama *main*.

Para la fase de pruebas en el dron real se emplea una rama dedicada llamada *test*. Esta rama tiene un propósito operativo distinto a las anteriores, ya que su historial no mantiene una coherencia lineal, y funciona como un entorno de preparación para pruebas de campo. Antes de una sesión de pruebas, el estado de la rama *dev* o, en algunos casos, el de una rama de características específica, es copiado mediante un *commit* de fusión o *reset* sobre la rama test. Esta metodología permite disponer rápidamente del código que se desea evaluar en el hardware real sin alterar el historial de desarrollo de las ramas *dev* o *main*. Los resultados de las pruebas realizadas con el código en *test* informan las decisiones para realizar ajustes y correcciones, que son luego implementados en sus respectivas ramas de origen (*dev* o la rama de características), cerrando así el ciclo de desarrollo y validación.

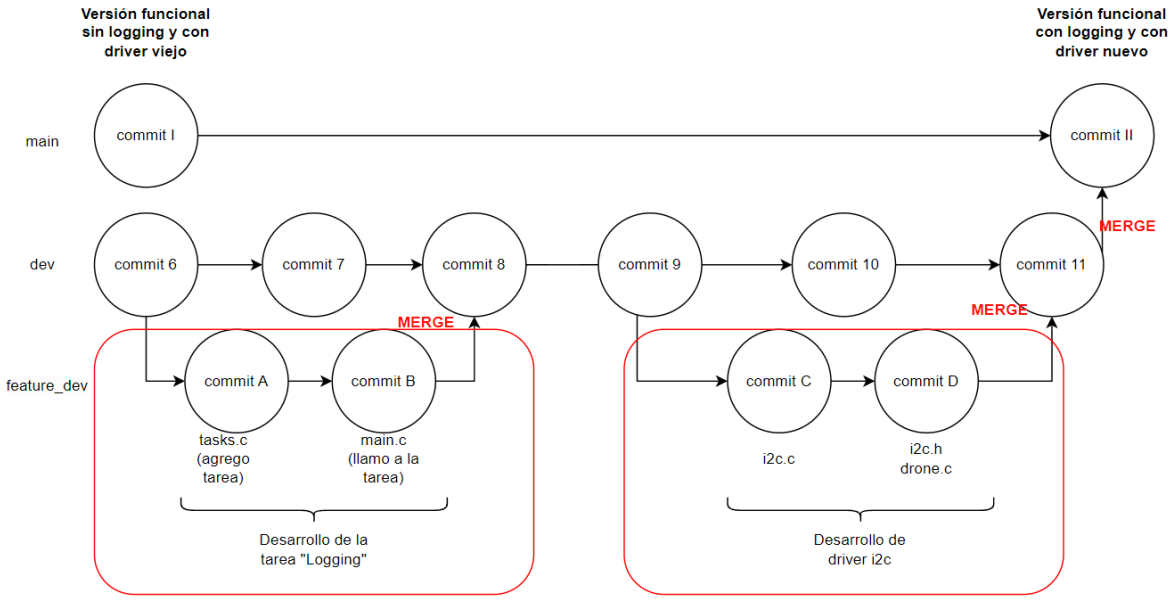


Figura 4.10: Ejemplo del flujo de desarrollo con *Git*.

Capítulo 5

Software

5.1 Software de monitoreo en tiempo real mediante USB

Para facilitar el análisis del comportamiento del sistema de control del dron durante las pruebas, se desarrolló una aplicación de escritorio en Python, utilizando las bibliotecas PyQt5 para la interfaz gráfica y PyQtGraph para la visualización en tiempo real de los datos provenientes del dron. Esta herramienta permite al usuario observar, registrar y gestionar de manera eficiente la información relevante durante el proceso de desarrollo y calibración.

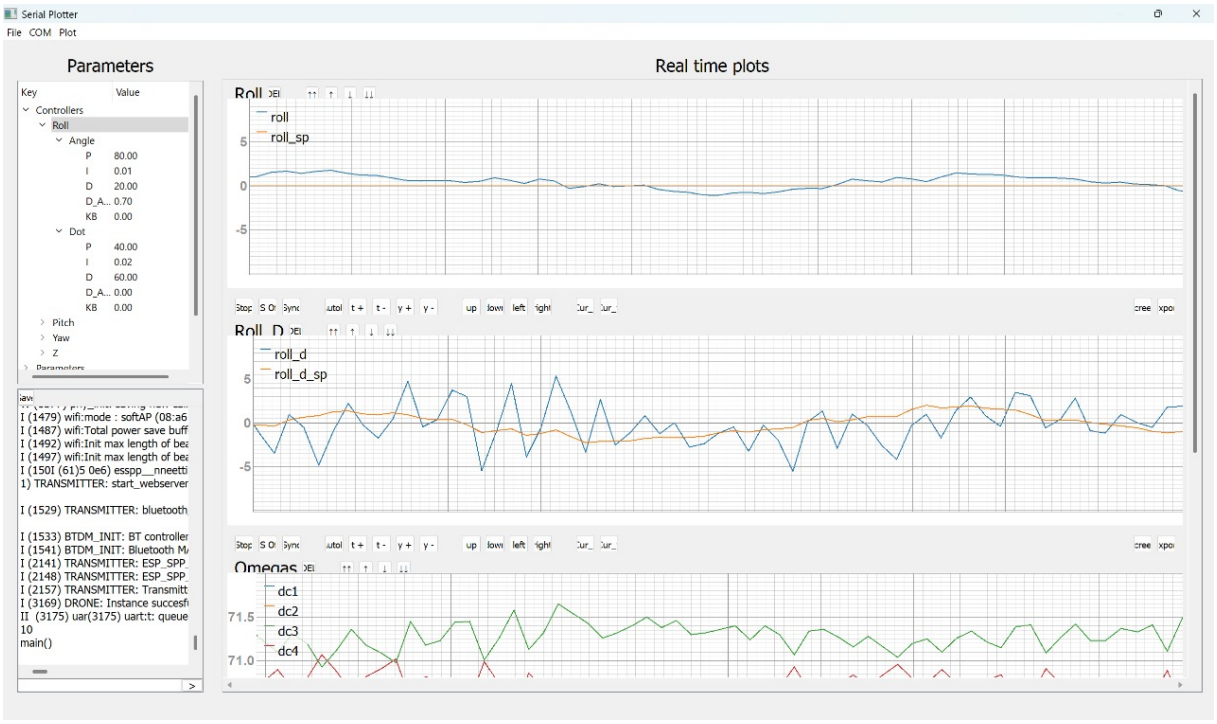


Figura 5.1: Vista general de la aplicación

La aplicación se compone de distintos módulos con los que el usuario puede interactuar:

Esto permite agregar cuantas variables se necesiten y en el orden que se deseen para enviar, y separarlas según sean variables en tiempo real o variables a mostrarse en el árbol de variables estáticas.

5.1.1 Gráficos en tiempo real

Cada gráfico puede crearse y configurarse de manera independiente, y mostrar hasta 8 variables distintas simultáneamente, facilitando el análisis de correlaciones entre señales como ángulos de orientación, errores del controlador, revoluciones por minuto de motores, *buffers* de integración, entre otros.

Los gráficos permiten varias funciones, algunas propias de un osciloscopio, como por ejemplo:

- Zoom interactivo y desplazamiento de tiempo y señal
- Visualización de coordenadas del puntero y activar o desactivar punteros en tiempo y señal
- Pausa, reinicio, sincronización de tiempo
- Exportar datos de las señales presentes en un gráfico particular
- Sacar una captura al gráfico y copiarla en el portapapeles

De igual modo, se pueden exportar los datos de todos los gráficos en su totalidad como .csv para un análisis posterior desde la interfaz principal.

La aplicación permite al usuario guardar configuraciones personalizadas de visualización, denominadas perfiles de gráficos. Cada perfil define un conjunto de gráficos, junto con las variables específicas que se muestran en cada uno. Esto resulta especialmente útil para cambiar rápidamente entre diferentes contextos de prueba o modos de análisis, sin necesidad de volver a configurar manualmente los gráficos cada vez.

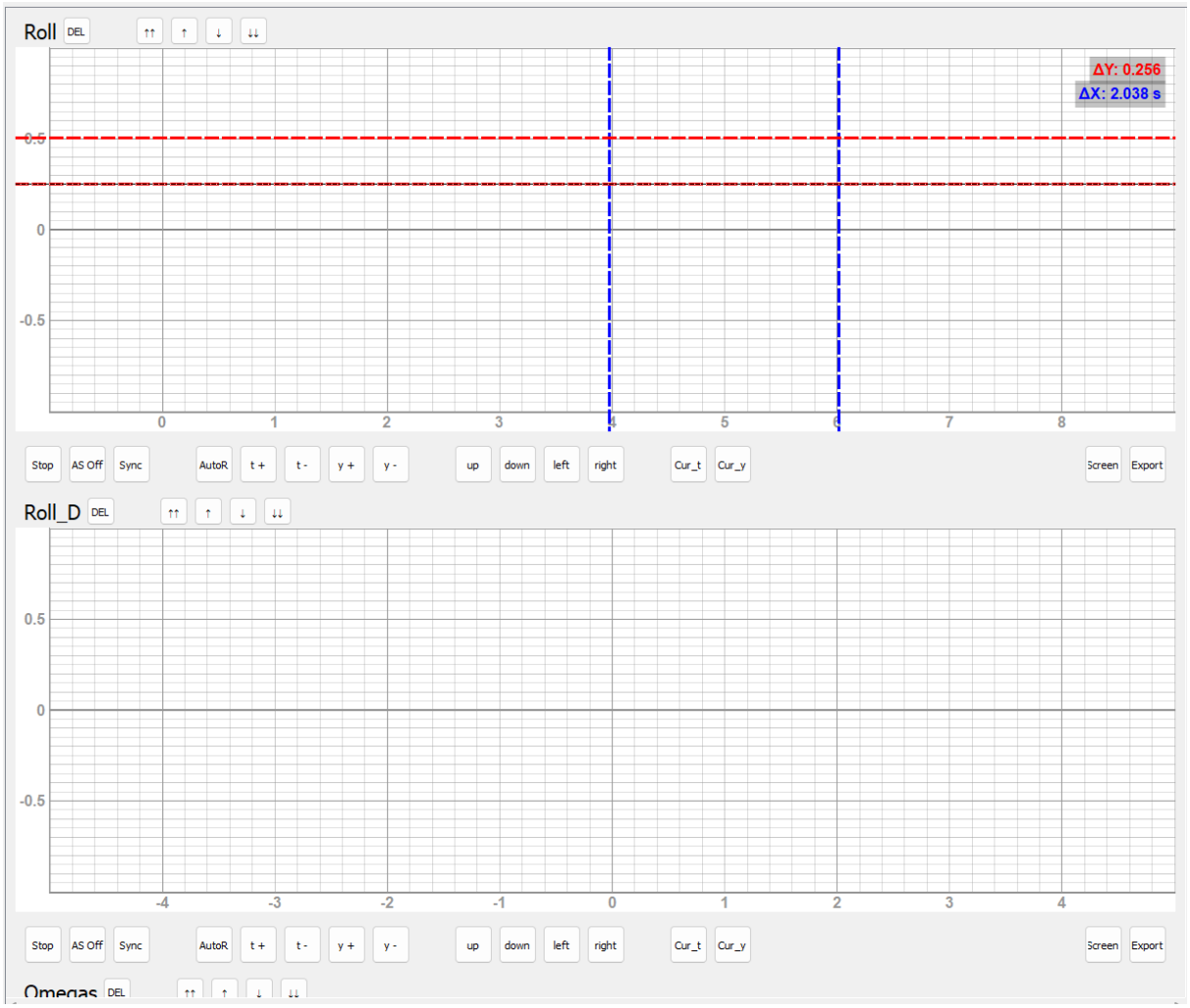


Figura 5.2: Plantilla de gráficos. El primero con los cursores de tiempo y señal activados.

Una función muy práctica es la de sacar una captura de la vista actual del gráfico con un botón, que permite generar imágenes como la mostrada en la Figura 5.3

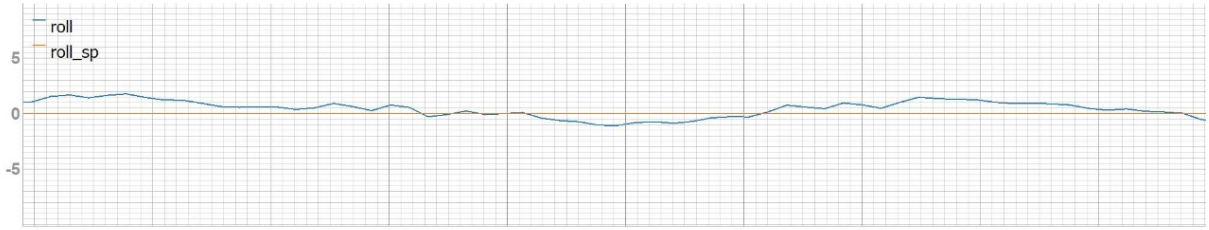


Figura 5.3: Captura del gráfico generada por la aplicación *Serial Plotter*

Por su parte, la Figura 5.4 enseña el diálogo que se muestra al exportar los datos exitosamente.

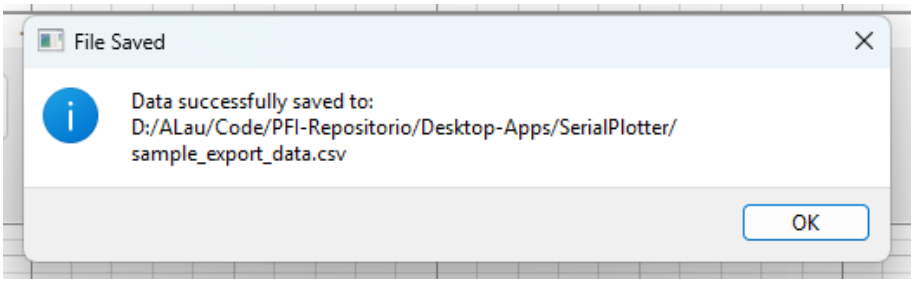


Figura 5.4: Diálogo de confirmación de exportación de datos

5.1.2 Consola de mensajes

La consola muestra todo lo enviado mediante serie por el dispositivo que no sea un mensaje con variables estáticas o para graficar, como mensajes de estado, errores o confirmaciones que provienen de RTOS o del propio firmware. Es fundamental para el seguimiento de eventos en tiempo real que no se muestran en los gráficos o variables estáticas, especialmente durante la ejecución de comandos críticos. Asimismo, permite enviar mensajes al microcontrolador, permitiendo interactuar mediante comandos personalizados para variar el setpoint, simular acciones del transmisor o cambiar el controlador a utilizar para determinado lazo de control.

Figura 5.5: Consola de mensajes de la aplicación

5.1.3 Árbol de variables estáticas

Se implementó una estructura en forma de árbol que permite visualizar y editar variables estáticas como constantes del sistema, parámetros de control (PID, coeficientes de filtro, etc.) e

incluso cambiar el algoritmo de control a utilizar, permitiendo hacer pruebas de manera dinámica sin necesidad de recompilar y grabar un nuevo firmware. La edición se realiza simplemente mediante un diálogo de entrada que se activa con un doble clic sobre el valor deseado.

Las variables a mostrar en el árbol estático y su posición dentro del mismo son totalmente configurables mediante un archivo .json que permite indicar el nombre con el que es enviada por el controlador de vuelo, el nombre con el que se muestra en el árbol, si es un variable modificable o no y la posición dentro del árbol.

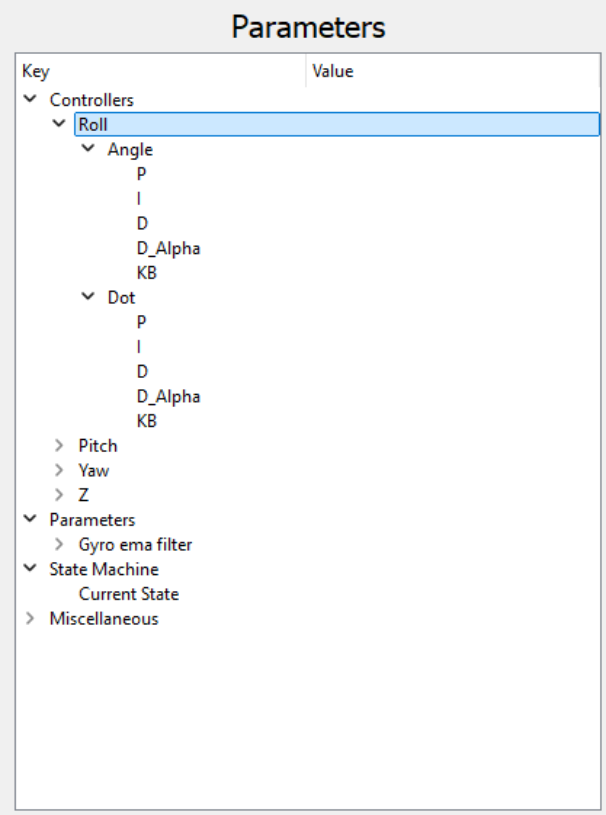


Figura 5.6: Árbol de variables estáticas

5.1.4 Protocolo de comunicación

El dron envía la información a graficar por el *Serial Plotter* mediante su UART con el siguiente formato:

printer:nombre1,valor1|nombre2,valor2\nstatic:nombre4,valor4|nombre5,valor5\n

5.2 Transmisor por WiFi

Para ahorrar en complejidad y costo del hardware, se decidió desarrollar la opción de comunicarse con el dron mediante un transmisor WEB, que corre en un navegador web de un teléfono celular y permite controlar completamente y realizar maniobras con el dron. La implementación se basa en una solución web embebida que aprovecha las capacidades WiFi del microcontrolador ESP32 para establecer un enlace de comunicación robusto y de baja latencia.

La ESP32 se configura para operar en modo *Access Point* (AP), creando una red WiFi independiente. Esta configuración permite el establecimiento de comunicaciones en entornos donde

no existe conectividad a internet o redes preexistentes, con un muy bajo alcance pensado para pruebas y desarrollo.

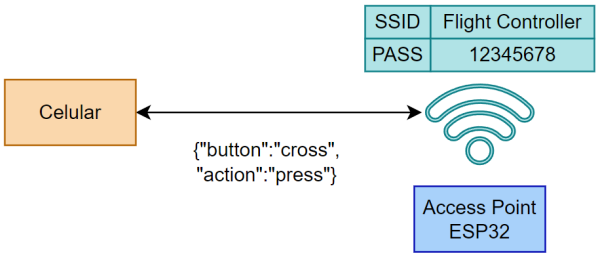


Figura 5.7: Diagrama del sistema de transmisor web

El firmware embebido incluye procedimientos de autenticación, asignación dinámica de direcciones IP mediante DHCP, y monitoreo continuo para detectar conexiones y desconexiones de clientes.

El microcontrolador integra un servidor web liviano que gestiona las peticiones HTTP entrantes y proporciona tanto contenido estático (en este caso, el archivo HTML de la figura 5.8 y Adicionalmente *endpoints* dinámicos para la comunicación de datos de control. La página web está guardada en la memoria Flash de la placa de desarrollo, y al establecerse una conexión cliente, el servidor entrega automáticamente la página principal de la interfaz, que incluye todos los recursos necesarios para el funcionamiento del controlador virtual.

La comunicación se basa en el formato JSON (JavaScript Object Notation) para el intercambio de datos, indicando el botón y la acción (presionar o soltar). Para los *sticks* analógicos, cuando la posición cambia se envía un mensaje donde se indica el *stick* como izquierdo o derecho y un valor de -100 a 100, proporcional a la posición actual.

Figura 5.8: Vista del transmisor web en un celular

Capítulo 6

Hardware

6.1 Placa de control de vuelo

6.1.1 Objetivo del diseño

Tanto para pruebas como para el producto final, es necesario montar todo sobre una placa de circuito impreso con el fin de simplificar el armado del cuadricóptero, reducir la cantidad de cables y proveer un ensamblado firme de los componentes, evitando desconexiones y falsos contactos durante el vuelo.

El objetivo es entonces diseñar una placa de control de vuelo compacta, que reúna las funcionalidades típicas de una placa de control de vuelo comercial, pero que además permita ciertas flexibilidades y dé la posibilidad de medir señales importantes, siendo ambas características indispensables durante el desarrollo y las pruebas.

Nace entonces como resultado la placa *Flight Controller Test Kit* que se detalla en esta sección. Esta versión prioriza la accesibilidad de la medición, la modularidad y el bajo costo educativo por sobre la miniaturización final. En versiones futuras se integrarán los ESCs en la misma placa y se migrará a una carcasa dedicada impresa en 3D, reduciendo cableado y peso.

6.1.2 Esquemático completo

En la Figura 6.1 se muestra el esquemático completo de la placa de control de vuelo. En él se integran todas las funciones principales, como la etapa de potencia, entradas y salidas, microcontrolador, conectores, y periféricos auxiliares.

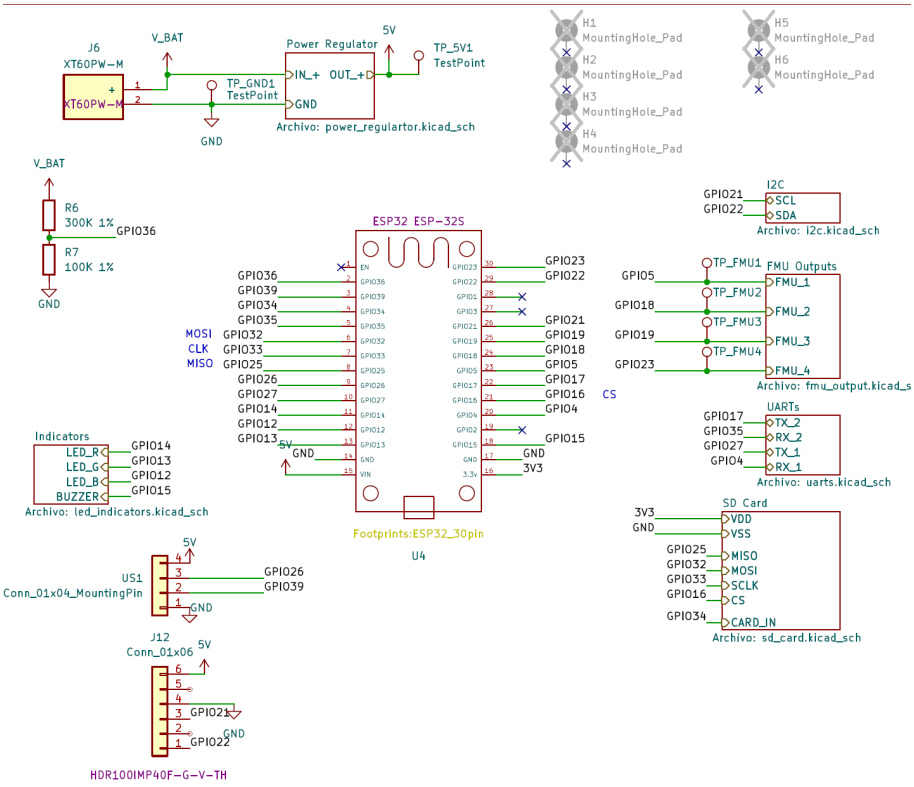


Figura 6.1: Esquemático completo de la placa

6.1.3 Diseño del PCB

La Figura 6.2 muestra la vista superior e inferior del diseño de PCB. Se puede observar la distribución compacta para mantener las dimensiones dentro de los límites de bajo costo de la mayoría de fabricantes de PCB, y de fácil colocación en drones pequeños.

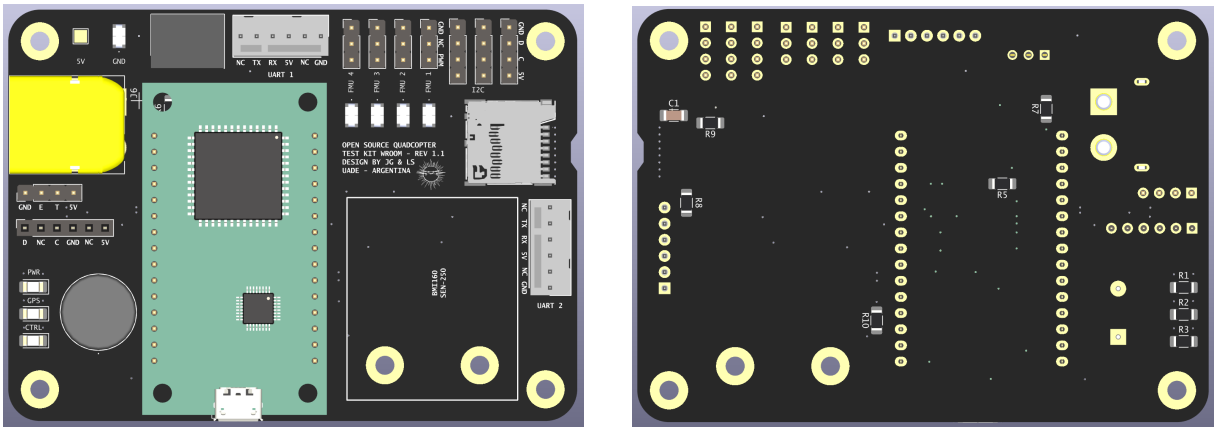


Figura 6.2: Vista superior e inferior del PCB

6.1.3.1 Bloque de potencia

Este bloque incluye el regulador de 5 Volts, el divisor resistivo para la medición de la batería, y el conector de la batería. El regulador 173010578 utilizado integra un capacitor dentro

del empaquetado, proveyendo fluctuaciones de hasta 25 miliVolts para 5 Volts junto con varias protecciones térmicas y eléctricas, reduciendo en gran medida el área necesaria fuera del regulador.

6.1.3.2 Conectores

Se incorporan conectores pin macho de 3 posiciones para salidas PWM (para ESCs), de 4 posiciones para interfaces I2C y para un sensor ultrasónico, 2 puertos UART estándar FTDI molex y un conector pin hembra de 6 posiciones para un módulo de sensor barométrico, que deberá tener sus pines soldados a 90 grados para un montaje vertical. Los conectores están etiquetados pin a pin y organizados por función para facilitar el cableado.

6.1.3.3 Bloque de indicadores

Se incluyeron tres LEDs de estado (POWER, CONTROLLER y GPS) y un buzzer. Los LEDs permiten indicar estados de los distintos sistemas, en este caso se colocó un LED para la alimentación (encendido, baja batería), uno para el transmisor (buscando, encontrado, perdido) y otro para el GPS (fijación, baja precisión, señal perdida). El zumbador puede ser usado como alarma acústica o señal de ubicación.

6.1.3.4 Montaje de la unidad inercial

Como este desarrollo tiene un gran componente de pruebas y reiteraciones, se aprovechó el espacio disponible en la placa para permitir el montaje de una placa de evaluación de unidad inercial de distintos fabricantes, haciendo el diseño independiente de la unidad inercial seleccionada y otorgando flexibilidad.

Esto permite fijar el módulo directamente sobre la placa de control de vuelo usando cinta bifaz como atenuador de vibraciones, logrando así una sujeción firme que minimiza vibraciones indeseadas en el sensor. Esta solución mejora la calidad de la medición inercial al reducir las vibraciones mecánicas recibidas por la unidad inercial.

6.1.3.5 Puntos de prueba

Debido al carácter de prueba y posteriormente educativo del producto, se colocaron unos puntos de prueba para osciloscopio en GND y las salidas PWM de los motores para poder monitorizar con un osciloscopio y verificar que se estén generando correctamente.

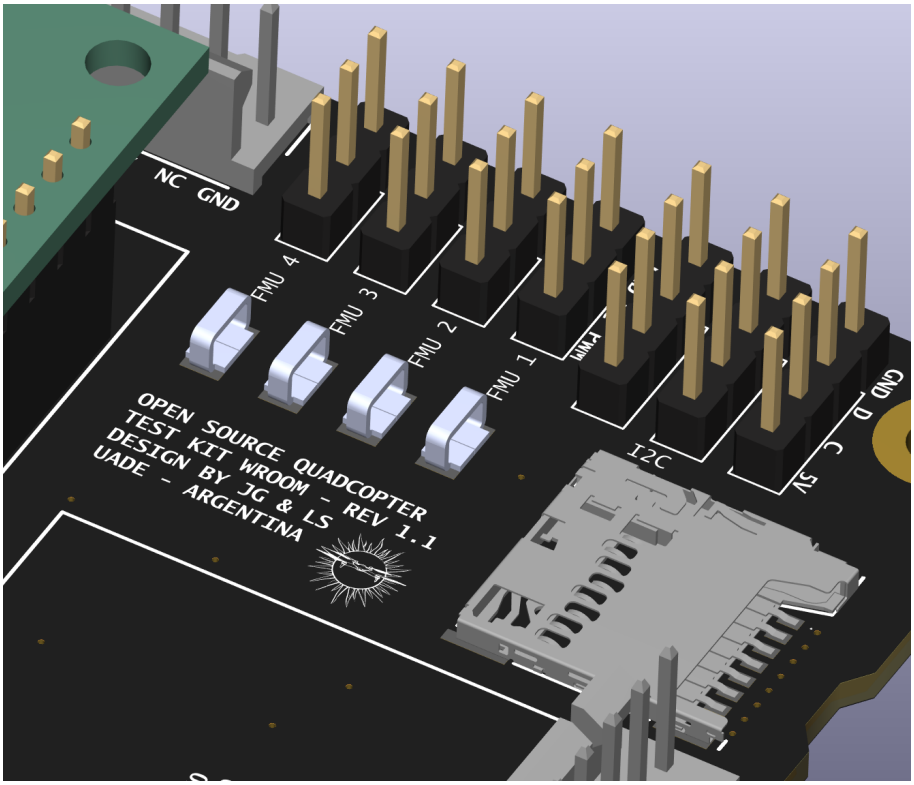


Figura 6.3: Puntos de prueba para osciloscopio

6.1.3.6 Slot para tarjeta MicroSD

Las memorias MicroSD permiten ser operadas mediante un protocolo de comunicación propietario o SPI. Por simplicidad y requerimientos de baja velocidad de transmisión de datos, en este proyecto se optó por utilizar SPI. La Figura 6.4 muestra el esquemático con las conexiones necesarias para el correcto funcionamiento de la memoria SD, que consta de resistencias de Pull-Up en cada pin de comunicación y una resistencia de Pull-Up adicional en un pulsador mecánico que integra el componente para detectar cuando se coloca una memoria microSD. Además se colocó un capacitor cercano al pin de alimentación para filtrar posibles ruidos en la línea.

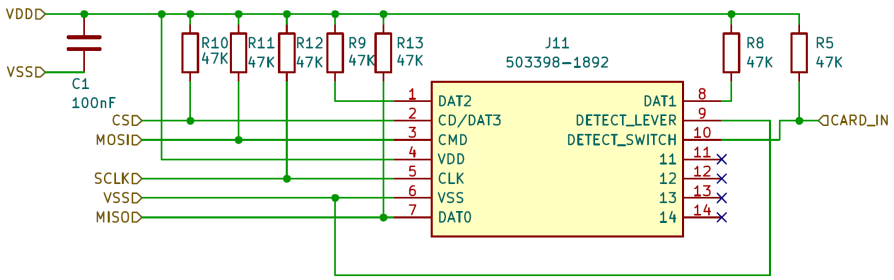


Figura 6.4: Esquemático de slot para tarjeta MicroSD

6.1.4 Lista de materiales

Tabla XI: Lista de componentes de la placa *Flight Controller Test Kit V1.1*

Referencia	Número de parte	Cantidad	Precio total
C1	C1206C104K5RACTU	1	$0,024
D1	SML-S13UTT68	1	$0,692
D2	SML-S13PTT68	1	$0,692
D3	SMLS14BNTT68	1	$0,858
J1,J2	22-11-2062	2	$1,38
J6	XT60PW-M	1	$0,675
J9	HDR100IMP40M-G-V-TH	1	$0,58
J11	503398-1892	1	$2,62
J12	HDR100IMP40F-G-V-TH	1	$1,38
LS1	CMI-1275-05TH	1	$0,99
R1,R2	ESR18EZPJ750	2	$0,122
R3	SFR18EZPJ4R7	1	$0,055
R5,R8,R9,R10,R11,R12,R13	SFR18EZPJ473	7	$0,616
R6	KTR18EZPF3003	1	$0,105
R7	CRCW1206100KFKEBC	1	$0,047
TP_1,TP_2,TP_3,TP_4	5019	5	$1,35
U2	173010578	1	$7,29
U4	ESP32 ESP-32S	1	$10,52
US1	HDR100IMP40M-G-RA-TH	1	$0,58

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de julio de 2025.

Esto da un costo de componentes de US$30,58.

6.2 Adaptación para kit F450

Al momento de montar el PCB se encontró el inconveniente de que la placa superior del Kit F450 contiene una gran cantidad de ranuras y una geometría irregular, factores que no permitían colocar el PCB *Flight Controller Test Kit* de manera fija sobre la misma.

Esto supuso un problema crítico debido a las vibraciones propagadas desde la carcasa a la unidad inercial y a la pobre sujeción del PCB. Por eso se optó por diseñar una adaptación y fabricarla mediante manufactura aditiva. Este diseño propio imita la forma de la placa superior pero sin ranuras, con un solo hueco para poder pasar cables hacia la parte inferior de la carcasa. Además, se hizo más gruesa para compensar la menor resistencia del material, y se agregaron 4 topes con agujeros de montaje M3 que permiten fijar el PCB *Flight Controller Test Kit* o cualquier PCB de 80x55mm con agujeros de montaje a 5 milímetros de los bordes.

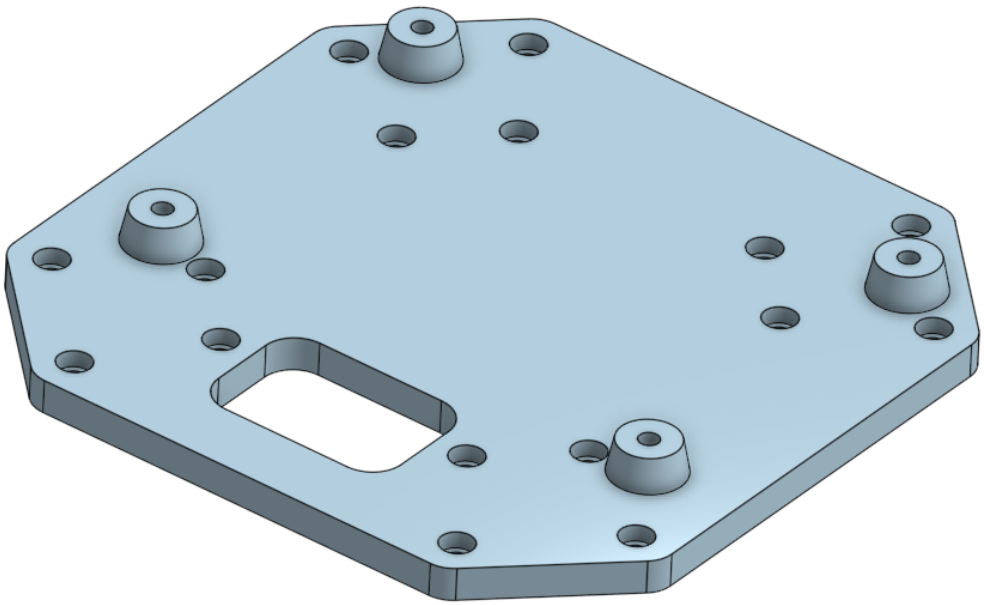


Figura 6.5: Vista isométrica del diseño del adaptador para *Flight Controller Test Kit* del Kit F450

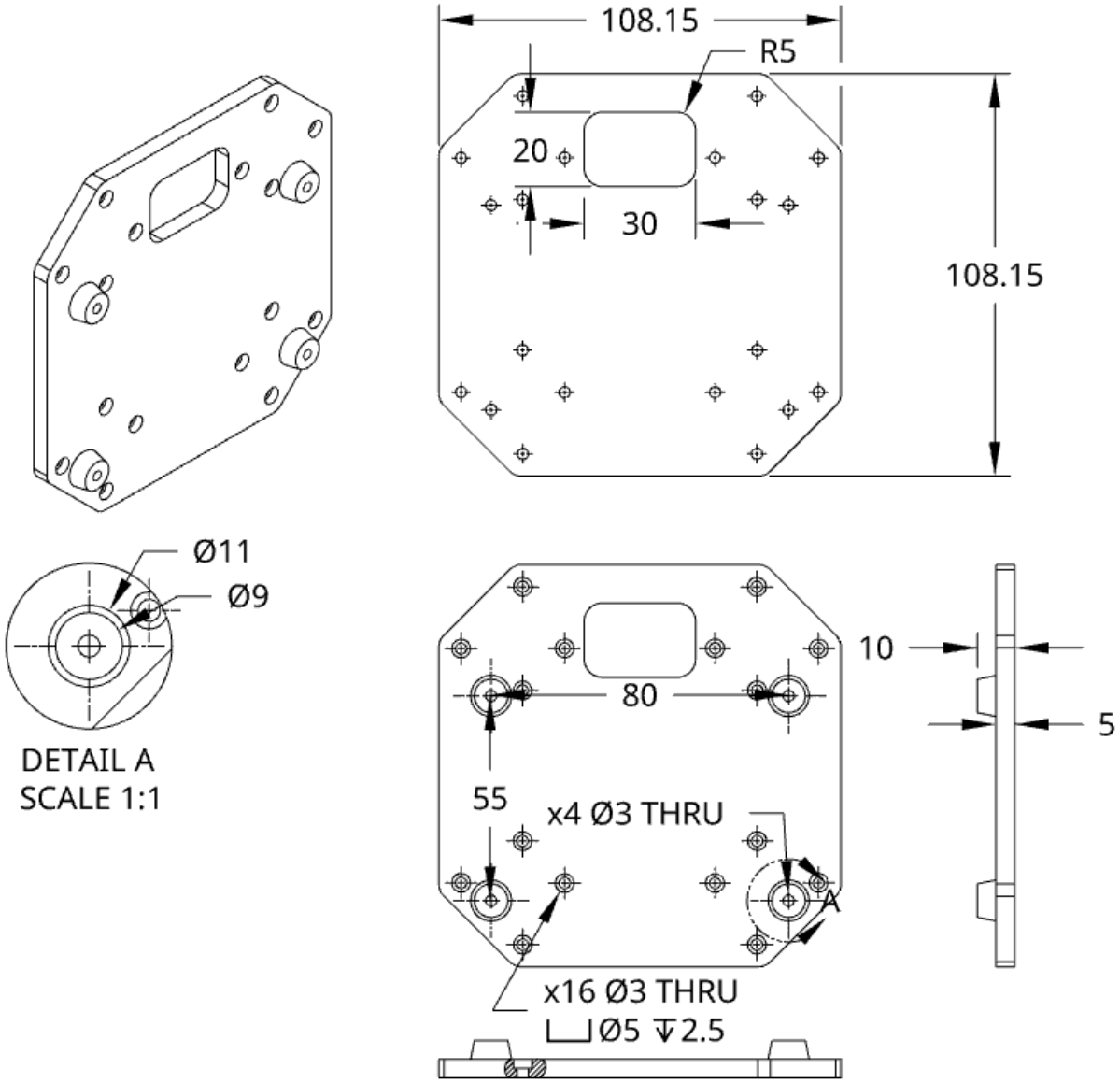


Figura 6.6: Planos del adaptador para *Flight Controller Test Kit* del Kit F450

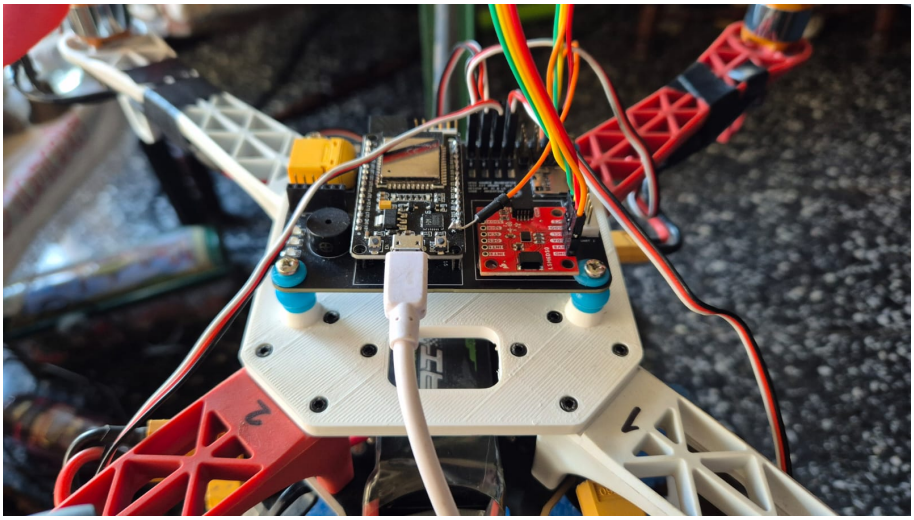


Figura 6.7: Placa *Flight Controller Test Kit* montada sobre el adaptador en el dron

6.3 Estructura de prueba

Durante el desarrollo del proyecto, se evidenció como necesidad imperiosa el probar el controlador de vuelo restringiendo los grados de libertad del cuadricóptero, de manera de no dañarlo y de aislar el problema de control a sólo un ángulo en vez de los 3 ángulos y la altura en simultáneo. Para eso se construyó una estructura de prueba que consiste en una base de hierro en U, con dos agujeros en cada arista vertical por donde puede pasar una varilla. Esta varilla calza en un cilindro hueco y puede asegurarse ajustando un bulón.

Es necesario agregar a la parte inferior de la carcasa del drone una tubería de 1/4 de pulgada con dos tapones con agujeros de diámetros similares al diámetro de la varilla. Esto logra que el drone quede fijo a la estructura en altura, rotación sobre el eje Z y sobre el eje perpendicular al eje de la varilla, y permite solo la rotación, de forma segura, en el eje paralelo a la varilla.

Por otra parte, posee una T de altura ajustable debajo de donde se monta el cuadricóptero que permite limitar el ángulo máximo de rotación sobre el ángulo a probar. En la figura 6.8 se puede ver cómo queda el dron montado sobre la estructura.

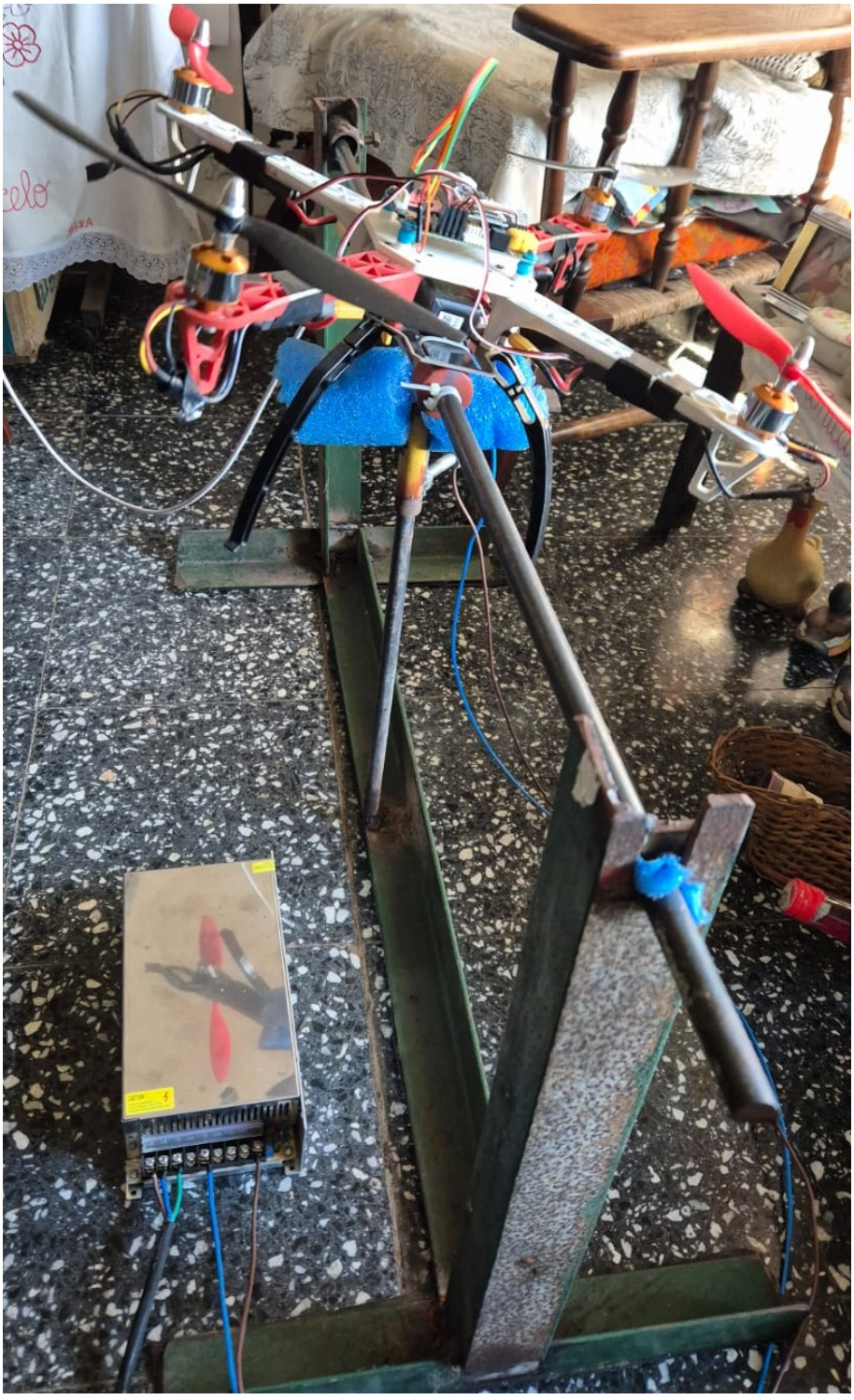


Figura 6.8: Estructura de prueba. Debajo está la fuente de alimentación para no usar la batería.

Capítulo 7

Pruebas y resultados experimentales

7.1 Objetivo de las pruebas

El objetivo de esta sección es la verificación del correcto funcionamiento del controlador de vuelo implementado en la placa "*Flight Controller Test Kit*". Las pruebas consistirán en probar la secuencia de arranque, la máquina de estados y pruebas de control de vuelo en una estructura de prueba.

Las pruebas experimentales fueron realizadas sobre una estructura de sujeción que restringe grados de libertad, por razones de seguridad, repetibilidad de ensayo y protección del hardware. El objetivo de estas pruebas no es demostrar vuelo libre, sino validar experimentalmente el lazo interno de estabilización de actitud (roll/pitch) y contrastar su desempeño real con el modelo teórico y las simulaciones desarrolladas en el Capítulo 3.

Los datos de la realidad se obtuvieron como archivo *csv* exportándolos desde el software *Serial Plotter*, desarrollado específicamente para este proyecto.

7.2 Pruebas realizadas

7.2.1 Estabilidad angular

La primera prueba a ejecutar es el mantenimiento del ángulo *Roll* alrededor del *setpoint* de 0°.

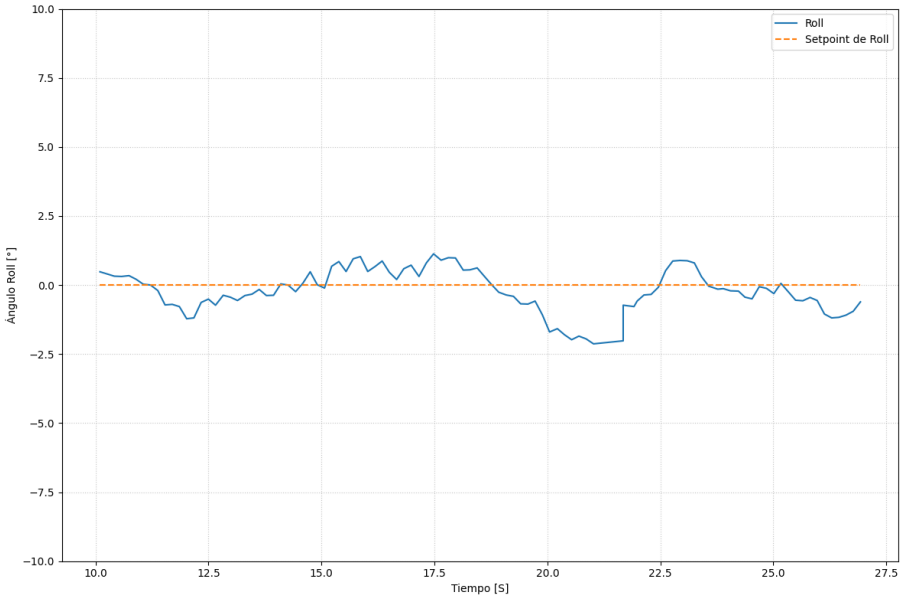


Figura 7.1: Prueba de estabilidad alrededor del 0

7.2.2 Respuesta al escalón

Luego, se prueba la respuesta al escalón del sistema, que permite analizar la velocidad de respuesta, el sobrepaso, la oscilación y la presencia de error de estado estacionario.

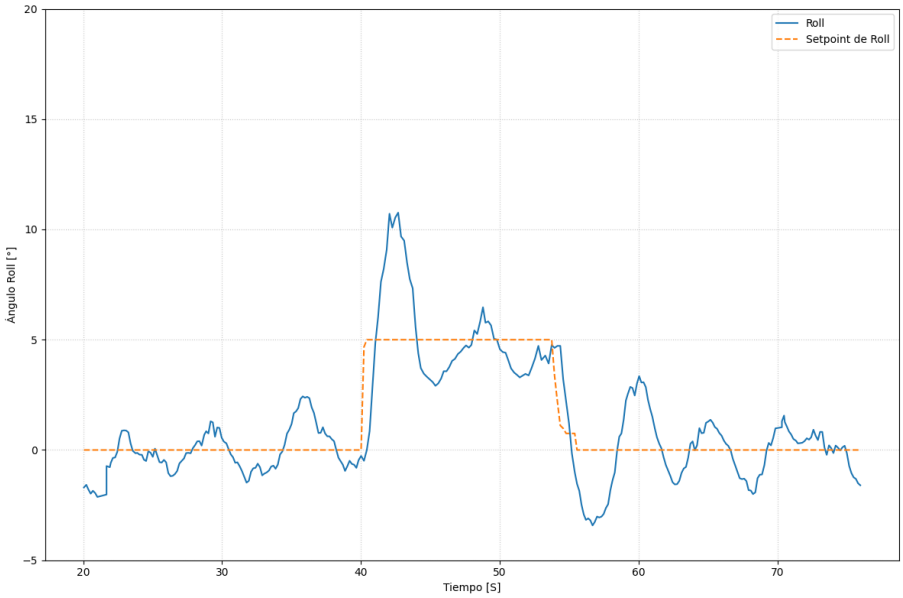


Figura 7.2: Prueba de respuesta al escalón

7.2.3 Solapamiento de órdenes de control en estructura

También es importante probar la estabilidad angular cuando se solapa junto a otra orden de control, como puede ser un aumento de la orden de control de altura, que causa que todos los motores giren más rápido.

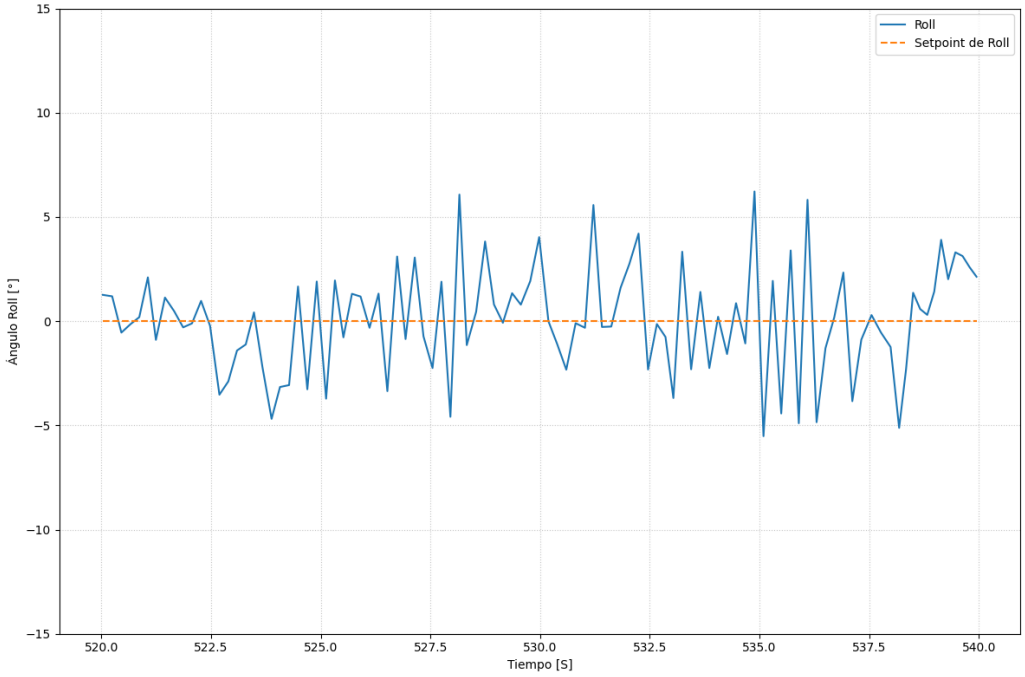


Figura 7.3: Prueba de estabilidad alrededor del 0 con la velocidad de todos los motores al 75 %

Se puede ver que la desviación es mayor, en parte por la mayor vibración de los motores y en parte por la interferencia mecánica del dron con la estructura de prueba, ya que la tubería que tiene acoplada choca contra la varilla de la estructura.

7.3 Comparación de resultados experimentales y simulaciones

En la Figura 7.4 se graficaron las respuestas del cuadricóptero físico y la simulación de este, frente a un escalón de 5° para roll. Además, nótese que se agregó ruido a la señal medida en la simulación para simular el ruido que introducen los sensores físicos.

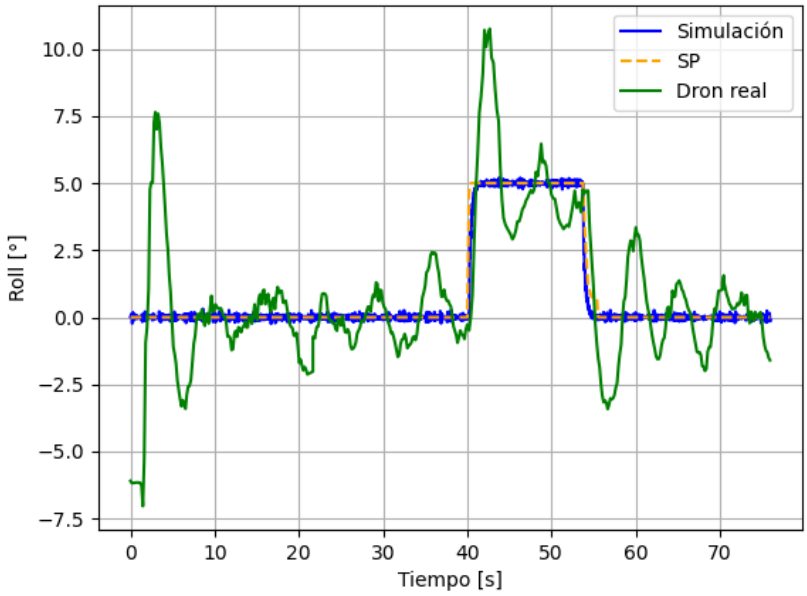


Figura 7.4: Comparativa de roll entre la simulación y el resultado experimental

Con el objetivo de lograr una comparación objetiva, se definieron ciertos parámetros típicos de las señales de control. Estos son útiles porque permiten caracterizarlas y dan una idea fundada del rendimiento del sistema de control.

- **Tiempo de establecimiento**: Se define como el tiempo que tarda la señal controlada en mantenerse dentro de un rango delimitado por $[y_{min}, y_{max}]$.

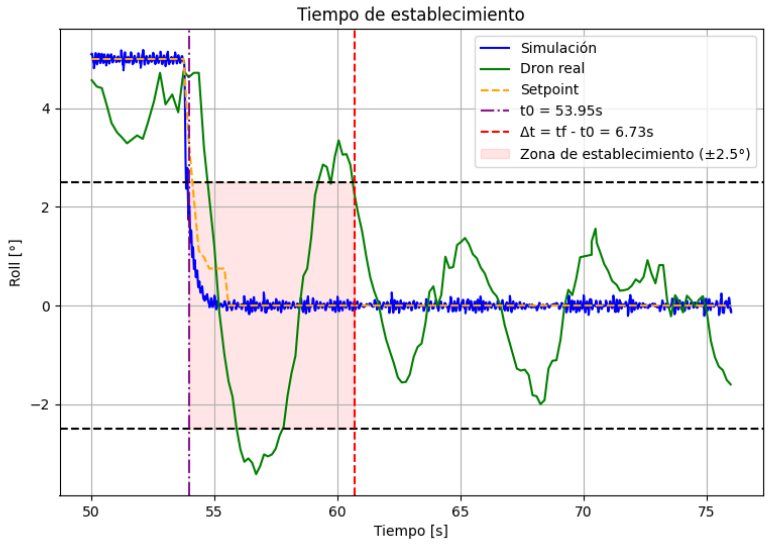


Figura 7.5: Tiempo de establecimiento de roll

Realizando pruebas y ajustando las ganancias del controlador, el mejor rango obtenido

fue de ±2.5°. Luego, en el artículo (Berrios et al., 2017, Fig. 11) se puede apreciar que estos valores son cercanos a los típicos valores esperados en un vuelo libre. Así, el tiempo de establecimiento queda definido como el tiempo que tarda Roll[1] en mantenerse dentro del rango $[-2,5°, 2,5°]$, en el instante en que la referencia transiciona del estado ALTO al estado BAJO. El valor final es de $\Delta t = 6,73s$.

- **Sobreimpulso**: Se define como la máxima diferencia, positiva o negativa, entre la respuesta del sistema y el valor final de la referencia, es decir,

$$x_{os} = |x - sp|$$

donde x_{os} es el sobreimpulso, x es el estado a controlar y sp es la referencia. Una forma alternativa de representar el sobreimpulso es de forma porcentual

$$x_{os(\%)} = \frac{|x - sp|}{sp} \cdot 100\,\%$$

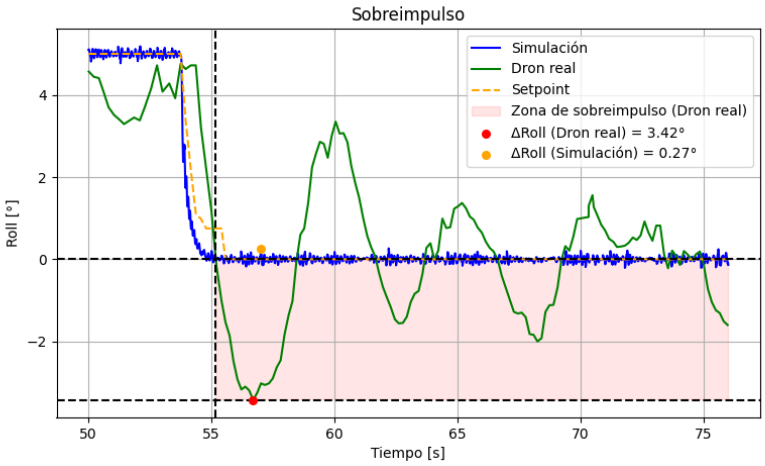


Figura 7.6: Sobreimpulso de roll

Para este caso, lo único que se debe hacer es buscar el máximo valor, una vez que la respuesta alcanzo la referencia, en la lista de datos. Según la Fgiura 7.6, el sobreimpulso de Roll es de $x_{os} = |-3,42° - 0°| = 3,42°$.

- **Error de estado estacionario**: Se define como la diferencia entre el estado controlado y la referencia, una vez que la señal a controlar ya se estableció dentro de un rango delimitado por $[y_{min}, y_{max}]$. En base a la Figura 7.7, se define al error de estado estacionario como

$$e_0 = \frac{1}{N} \sum_{0}^{N} |x - sp|$$

[1]Puesto que la respuesta de la simulación no presenta oscilaciones, no es relevante determinar este parámetro y se lo puede aproximar al caso ideal de 0s.

El resultado es un error promedio de $0.812°$ para Roll real y $0.064°$ para la simulación.

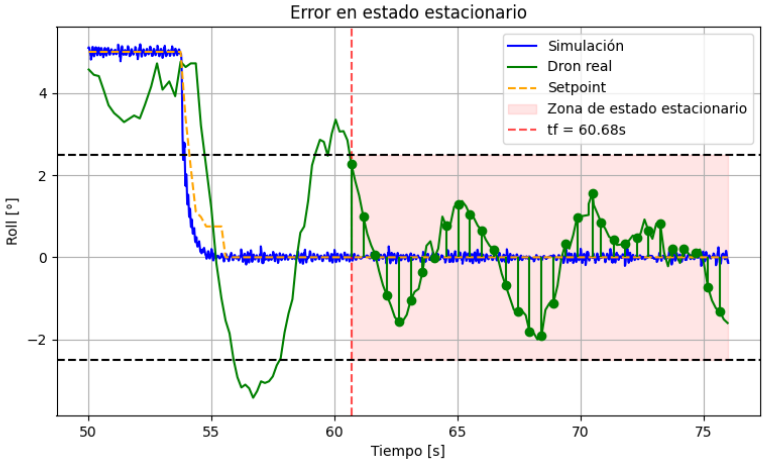


Figura 7.7: Error de establecimiento de roll

7.4 Conclusiones de las pruebas

Los resultados demuestran que el controlador implementado es capaz de comandar al dron y estabilizar el ángulo de *roll* de manera efectiva. Sin embargo, se observaron discrepancias entre la respuesta del sistema simulado y el comportamiento real del cuadricóptero. Estas diferencias, principalmente en la dinámica del movimiento de *roll*, son atribuibles a varios factores inherentes al sistema físico que no son capturados en su totalidad por el modelo matemático idealizado.

El sistema real está sujeto a vibraciones generadas por los motores y las hélices, las cuales son transmitidas a la estructura y a la unidad de medición inercial (IMU). Estas vibraciones introducen ruido de alta frecuencia en las mediciones de los giroscopios y acelerómetros, afectando la estimación de la orientación y, por lo tanto, también el desempeño del controlador.

Otro causante es que el modelo de simulación asume una respuesta lineal e ideal de los motores y controladores de velocidad (ESC). En la práctica, estos componentes presentan no linealidades como tiempos de respuesta asimétricos entre el aumento y la disminución de velocidad, saturación y dinámicas propias que el lazo de control debe compensar.

Asimismo, el ensamblaje manual del *frame*, así como ligeros desbalances en las hélices y los motores, introducen asimetrías en el sistema. Estas imperfecciones generan acoplamientos no modelados entre los ejes y momentos de inercia ligeramente diferentes a los utilizados en la simulación.

También, los sensores de la vida real presentan deriva, offsets y retardos de fase mínimos que deben ser filtrados y compensados mediante software. El modelo de simulación, en cambio, utiliza datos de sensores ideales.

Capítulo 8

Análisis Económico

8.1 Objetivo del análisis económico

En este capítulo se dispondrá al desarrollo del análisis económico del proyecto. Al ser un trabajo de tipo Open Source sin fines de lucro, el objetivo de este análisis no es determinar un precio de venta o la rentabilidad comercial, sino cuantificar y presentar el coste de implementación de la solución. De esta manera, se busca proporcionar a un aficionado, una universidad o una institución con interés en replicarlo una estimación de la inversión requerida para su desarrollo y despliegue.

8.2 Costo unitario

En esta sección se estudiará el costo total de fabricación de un Cuadricóptero que emplee el controlador de vuelo Cuadricóptero Open Source. El costo total está dado por el costo del PCB, los componentes y conectores que lo integran, y la carcasa del cuadricóptero junto a sus sensores y actuadores.

8.2.1 Costos de fabricación

Se detalla a continuación el costo de fabricación y ensamble de la placa *Flight Controller Test Kit* obtenido de cotizaciones a distintos fabricantes y la lista de materiales XI. Para la fabricación, se tomaron como referencia dos empresas conocidas de China (JLCPCB y PCBWay) y dos empresas importantes que se dedican a la fabricación de PCBs en Argentina (Ernesto Meyer S.A. y PCB Factory)

Tabla XII: Comparación de fabricantes de PCB

Fabricante	Precio	Tiempo de fabricación
JLCPCB	2	2–3 días
PCBWay	5	3–5 días
Ernesto Meyer	120	14 días
PCB Factory	150+25	14 días

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de julio de 2025. Los fabricantes de Argentina poseen c□ostos básicos,□ indicados con un +.

Por lo que si se suma el costo de US$30,56 de componentes a los US$2 de fabricación de, por ejemplo, JLCPCB, se obtiene un costo de materiales por unidad final de US$32,56.

8.2.2 Lista de materiales

A continuación, se detalla un ejemplo de los materiales necesarios para confeccionar un Cuadricóptero Open Source funcional que pueda ser empleado con fines de desarrollo y educativos.

Es importante notar que el resto de los componentes, tales como la carcasa del dron, los conjuntos motores/hélices y los controladores de velocidad, no tienen por qué coincidir con los mismos utilizados en este proyecto para un correcto funcionamiento, y se pueden obtener reemplazos a distintos precios.

Un buen ejemplo de esto es la carcasa del dron, la cual un aficionado avanzado o una institución educativa pueden diseñar o encontrar en internet una variante para fabricarla mediante manufactura 3D aditiva. Esto no solo disminuirá sustancialmente los costos, sino que agregará otra línea de investigación y desarrollo interdisciplinares al proyecto, lo cual puede resultar de gran interés para una institución educativa.

Habiendo hecho esta aclaración, se desarrollarán estos costos de implementación teniendo en cuenta dos casos de uso: perfil programador y perfil integral o educativo.

8.2.2.1 Perfil programador

Este perfil de usuario final apunta a individuos u organizaciones con interés únicamente en el firmware o sistemas de control. Se trata de investigadores, desarrolladores independientes o estudiantes avanzados cuyo interés principal es probar, modificar y validar algoritmos de control.

Hipotéticamente, este perfil no necesita invertir tiempo ni recursos en rediseñar hardware, porque su valor agregado está en el software. Por eso, los costos son más bajos en infraestructura y más altos en la flexibilidad para iterar en el firmware.

Tabla XIII: Lista de productos con sus cantidades y precios para perfil programador.

Producto	**Cantidad**	**Precio total**
Kit Frame F450	1	21
Kit x4 Motor + Hélice + ESC	1	50,39
Módulo LSM6DSO	1	17
Módulo GNSS	1	14
Módulo BMP390	1	16
Placa *Flight Controller Test Kit*	5	2
Componentes para placa *Flight Controller Test Kit*	1	30,57

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de Agosto de 2025. El precio de la placa es por 5 unidades (mínimo) del *PCB* sin componentes

Esto da un total de **US$150,96**, compuesto por US$62,57 de componentes y placa *Flight Controller Test Kit*, y US$88,39 correspondientes al resto del *hardware* que compone al dron en sí.

Con esta alternativa, un estudiante puede validar en un dron real lo aprendido en simulación de laboratorios de materias de control o de programación en sistemas embebidos, con una inversión mínima. Incluso puede que se utilice este perfil en una institución educativa, aprovechando

la baja inversión, y que este perfil sea el punto de partida para escalar posteriormente hacia el perfil integral.

8.2.2.2 Perfil integral o educativo

Este perfil tiene en cuenta a un usuario final que está interesado en diseñar los componentes y desarrollar su producto más allá del firmware, denominado perfil integral o educativo. Está orientado a instituciones educativas, laboratorios de investigación o empresas que buscan no solo usar el dron, sino formar personas en todas las áreas: electrónica, mecánica y software.

Este perfil presenta las siguientes consideraciones particulares:

- **Se trabajará en equipos multidisciplinares**, por lo que cada uno podrá centrarse en tareas de su área de dominio y necesitará realizar pruebas con hardware exclusivo. Debido a esto serán necesarias 10 unidades.
- **Ahorrará en la compra de la carcasa**, diseñando el suyo propio y fabricándolo mediante manufactura aditiva o corte de *Medium Density Fibreboard* (MDF).
- **Rediseñará la placa *Flight Controller Test Kit*** para simplemente integrar los sensores a la misma, y ahorrar el costo de los módulos de los sensores.

Tabla XIV: Lista de productos con sus cantidades y precios para perfil integral o educativo.

Producto	**Cantidad**	**Precio total**
Kit x4 Motor + Hélice + ESC	10	503,9
Sensor LSM6DSO + componentes pasivos	10	80
Sensor BMP390 + componentes pasivos	10	80
Módulo GNSS	10	140
Placa *Flight Controller Test Kit*	10	4
Componentes para placa *Flight Controller Test Kit*	10	305,7

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de Agosto de 2025.

Esto da un total de **US$1113,6**, compuesto por US$469,57 de componentes y placa *Flight Controller Test Kit*, y US$644,03 de demás *hardware* que hacen al dron en sí, con una capacidad de reducirse aún más si se consideran alternativas de menor costo.

Si se hubiesen comprado 10 conjuntos como los planteados en el perfil programador, el costo hubiese sido de US$1463,6. Teniendo en cuenta las consideraciones especiales planteadas para este perfil integral o educativo, se puede generar fácilmente un ahorro de aproximadamente el **23 %**, por lo que se recomienda invertir el tiempo extra en desarrollo para aprovecharlo.

Con este trabajo se genera a su vez conocimiento y experiencias transferibles a la industria en aquellos integrantes que se hayan dedicado al rediseño. Además, el trabajo cerca de personas de otras ramas, conocimientos y responsabilidades en el proyecto acerca a los integrantes a un esquema de trabajo en investigación y desarrollo empresarial. Todo esto es un gran motivador pedagógico para realizar estas modificaciones y elegir este perfil siendo una institución.

8.2.3 Firmware y software

Cualquier uso de este proyecto va a necesitar tanto modificar el firmware del controlador de vuelo grabado en la memoria de programa como usar o modificar el software desarrollado durante este proyecto para configurar el dron o ver datos en tiempo real. Todas estas herramientas son gratuitas, y a continuación se detalla el nombre y la licencia de cada una.

Tabla XV: Lista de herramientas de programación

Herramienta	Precio	Licencia de uso	Licencia de código
VSCode	Gratis	Microsoft	MIT
Plugin ESP-IDFf	Gratis	APACHE 2.0	APACHE 2.0
Python	Gratis	MIT	MIT
PyQt5	Gratis	PSF License 2.0	PSF License 2.0
Serial Plotter (propio)	Gratis	MIT	MIT

8.3 Costos de desarrollo del proyecto

En esta sección se detallan, a modo de referencia, los costos de realización del proyecto Cuadricóptero Open Source tanto económicos como temporales. De esta manera, se espera que sea útil para que individuos, empresas o universidades interesadas en realizar proyectos de similar envergadura tengan acceso a datos históricos obtenidos de la realidad para la evaluación y pronóstico de costos de desarrollo del proyecto. De igual modo, se busca mostrar el ahorro de horas de ingeniero y dinero que se logra si se basan nuevos desarrollos en la plataforma Cuadricóptero Open Source.

8.3.1 Costos económicos

A continuación se detallan los costos económicos de desarrollo del proyecto:

Etapa	Objeto	Precio
Desarrollo de Hardware	Fabricación de PCB (x3)	100
Desarrollo de Hardware	Batería	70
Desarrollo de Hardware	x4 Kit ESC+hélices+motores	140
Desarrollo de Hardware	x4 ESC Programable	40
Desarrollo de Hardware	x4 ESC Programable de repuesto	40
Desarrollo de Hardware	Compra de frame	62
Desarrollo de Hardware	Módulo sensor LSM6DSO	17
Desarrollo de Hardware	Módulo sensor BMI160	17
Desarrollo de Hardware	Módulo sensor GPS NeoV7	14
Desarrollo de Hardware	Módulo sensor HCSR-04	10
Desarrollo de Hardware	Módulos sensores BMP	40
Desarrollo de Hardware	Módulos lector micro SD	3
Desarrollo de Hardware	Topes antivibración	16
Desarrollo de Hardware	Impresión de placa adaptadora	7
Total	-	**576**

Tabla XVI: Gastos de desarrollo

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de julio de 2025. **No tienen en cuenta gastos de envío ni pago de impuestos nacionales**

Esto da un total de **US$576** en conceptos de gastos para investigación pruebas y desarrollo del proyecto.

8.3.2 Costos temporales

Al ser un proyecto *freelance* sin una estructura temporal rígida, la planificación se centra en el desglose de tareas y la estimación de esfuerzo, más que en un cronograma con fechas específicas. El siguiente cuadro cuantifica las horas de trabajo totales por actividad, realizadas de manera flexible, con el objetivo de generar datos que puedan resultar valiosos para la planificación de proyectos análogos.

Etapa	Tarea	Horas/ingeniero
Investigación	Modelo dinámico del cuadricóptero	48
Investigación	Sensores y actuadores a utilizar	24
Investigación	Sistema de mando a utilizar	6
Investigación	Sistemas de control a utilizar	24
Modelado y Simulación	Simulación y depuración de modelo dinámico	32
Modelado y Simulación	Diseño y reiteración de controladores	24
Desarrollo de Firmware	Arquitectura general del programa	52
Desarrollo de Firmware	Algoritmos de control	14
Desarrollo de Software	Aplicación “Serial Plotter”	24
Desarrollo de Hardware	Diseño de *PCBs*	24
Desarrollo de Hardware	Armado de *PCBs*	2
Desarrollo de Firmware	Drivers propios para subsistemas electrónicos	12
Desarrollo de Firmware	Integración de subsistemas electrónicos	10
Pruebas	Depuración de firmware basado en pruebas	48
Pruebas	Tuneo basado en maniobras en estructura	140
Total	–	**484**

Tabla XVII: Lista de tareas

8.4 Conclusiones del análisis económico

En conclusión, el análisis económico muestra que el proyecto Cuadricóptero Open Source es accesible tanto para desarrolladores individuales como para instituciones educativas o empresas. El perfil programador ofrece una alternativa de bajo costo para validar algoritmos de control, mientras que el perfil integral o educativo, aunque exige una inversión inicial mayor, proporciona ahorros significativos en volumen y genera experiencias de formación interdisciplinaria con impacto académico e industrial. Esto confirma que el proyecto no solo es técnicamente viable, sino también económicamente atractivo para distintos contextos de aplicación.

Por otra parte, los gastos totales de desarrollo se pueden resumir en la siguiente tabla:

Etapa	Costo total	Horas/ingeniero estimadas
Investigación	0	102
Modelado y Simulación	0	56
Desarrollo de Firmware	0	88
Desarrollo de Software	0	24
Desarrollo de Hardware	576	26
Pruebas	0	188
Total	**576**	**484**

Tabla XVIII: Resumen de costo y tiempo

Nota: Los precios están expresados en dólares estadounidenses y corresponden al mes de julio de 2025. No tienen en cuenta gastos de envío ni pago de impuestos nacionales

Capítulo 9

Conclusión

Para concluir el presente trabajo, se confirma que se han completado los objetivos iniciales de la siguiente manera: Entendiendo el problema de control, estudiando la bibliografía sobre el tema y construyendo distintos modelos para representar los componentes, realizando ensayos sobre los sistemas reales para obtener sus parámetros, diseñando y simulando distintos sistemas de control hasta encontrar el óptimo, desarrollando un controlador de vuelo con diversas funcionalidades que implemente estos sistemas de control, y finalmente dándole vida mediante un PCB y un kit de drones "Hágalo usted mismo".

La culminación de este proyecto otorga a la comunidad una base de código, hardware y documentación a integrar en la base de conocimiento de individuos, empresas e instituciones, aportando material de consulta para nuevos trabajos. Esto con el objetivo de reducir costos, tiempos de desarrollo, iteración y salida al mercado.

Se logró, adicionalmente a lo propuesto, implementar un software de monitoreo en tiempo real, diseñar una placa de circuito impreso que integra una placa de desarrollo ESP32 junto a conectores y demás componentes útiles, así como adaptarla mecánicamente al kit *F450* rediseñando la placa superior e imprimiéndola en 3D.

También vale la pena mencionar que este es un proyecto vivo, en constante desarrollo y en búsqueda de colaboradores tanto individuales como organizaciones. El repositorio del firmware se encuentra abierto en GitHub (Scoflich Lautaro y Grandinetti Juan, 2025), y se planea aceptar mejoras y funcionalidades propuestas por colaboradores.

Finalmente, las futuras líneas principales de las que se beneficiaría este proyecto son:

- Desarrollo de carcasa de dron 3D personalizada para este proyecto, de manera de optimizar el espacio para componentes, disminuir altamente los costos y generar experiencia en el área de diseño y manufactura.
- Diseño y manufactura de una placa de control de vuelo que integre a los controladores de velocidad en la misma, de manera de disminuir costos y mejorar el desempeño del dron, a la vez de generar experiencia en el área de diseño de controladores de velocidad y circuitos híbridos.
- Integrar la solución propuesta con una computadora de vuelo comercial, e incluso desarrollar una computadora de vuelo propia, que permita trabajar a un mayor alto nivel apoyándose en el bajo nivel del controlador desarrollado en este proyecto. Esto haría que Cuadricóptero Open Source sea una solución completa, y generaría experiencia en el área

de sistemas embebidos y ciencias de la computación, y ayudaría a aplicar conceptos de aprendizaje automático y procesamiento de imágenes y video.

En síntesis, este trabajo sienta las bases técnicas y conceptuales para el desarrollo de un ecosistema de drones abierto, modular y accesible, que combina hardware personalizado, software libre y potencial de crecimiento colaborativo. La plataforma desarrollada fue validada en banco, mostró coincidencia con las simulaciones y constituye una base sólida para futuros desarrollos de vuelo libre y control avanzado.

El principal resultado de este trabajo no es un único prototipo puntual, sino una plataforma abierta y documentada que permite reproducir, auditar, modificar y extender cada bloque del sistema: dinámica modelada e identificada, controladores en cascada implementados en firmware en tiempo real, hardware de control propio con puntos de prueba, herramientas de monitoreo y ajuste en línea, y validación experimental comparada con simulación. Esta plataforma queda disponible como base para líneas futuras, incluyendo integración de ESCs en la placa principal, carcasa optimizada, computadora de misión con visión/IA y vuelo autónomo multieje.

Apéndice A

Torque en los ejes (x, y)

De la *Sección 2.1.3: Dinámica*, se desprende que los torques en cada eje x, y, z se calculan como $\tau_i = F_i l$. Ahora bien, el objetivo de esta sección es explicar porqué en τ_{roll} y τ_{pitch}, la distancia l se reemplaza por $\frac{\sqrt{2}}{2}l$.

Lo primero que a comprender es que la dirección de los torques debe estar alineada con los ejes del cuadricóptero. En la Figura A.1, se puede observar que esto no es así, ya que los ejes x e y pasan entremedio de los brazos de este, es decir, el ángulo entre cada brazo y cualquiera de los ejes es de $45°$.

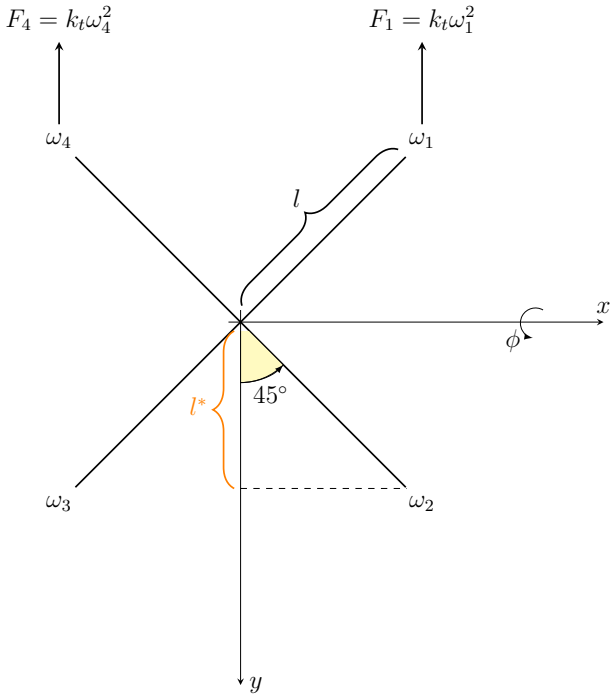


Figura A.1: Diagrama de torques en $roll$ y $pitch$

Por otra parte, nótese que al calcular el torque en x e y sin modificar nada, se estaría obteniendo el mismo a lo largo de los brazos del cuadricóptero es decir, en las direcciones $\omega_4 - \omega_2$ y $\omega_1 - \omega_3$. Entonces, dado que los torques son necesarios en las mismas direcciones que los ejes x e y, se debe proyectar la distancia l en estos.

La Figura A.1 expresa un torque en $x(\tau_x = \tau_{roll})$ positivo, donde se ha definido a l^* como la proyección de l sobre el eje y. Adicionalmente, aplicando trigonometría resulta relativamente sencillo relacionarlas,

$$\cos(45°) = \frac{adj}{h} = \frac{l^*}{l}$$

Despejando l^* se encuentra que,

$$l^* = \cos(45°)l = \frac{\sqrt{2}}{2}l$$

Finalmente, el torque resultante es

$$\tau_i = F_i l^* = F_i \frac{\sqrt{2}}{2}l$$

Apéndice B

Instalación de la extensión ESP-IDF en VSCode

Como se puede observar en la Figura B.1, seleccione la opción, debajo a la izquierda, $Advanced \longrightarrow ConfigureESP-IDFExtension$. Luego, elija la instalación $EXPRESS$.

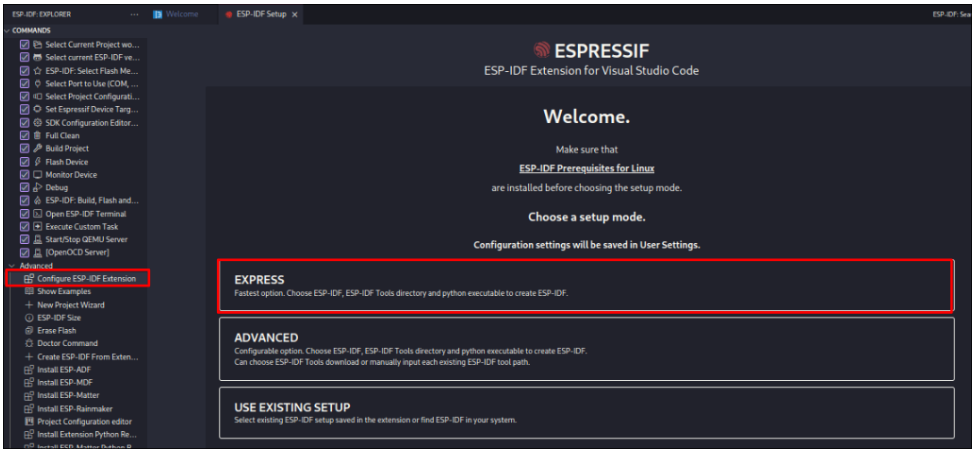


Figura B.1: ESP-IDF setup

Por último, seleccione la version de ESP-IDF a instalar así como también el directorio donde se instalará el software. Adicionalmente, como se observa en la Figura B.2, en la última opción se debe elegir la ruta donde se encuentra instalado Python[1].

[1]Es necesario tener instalado *pip* (Python, s.f.)

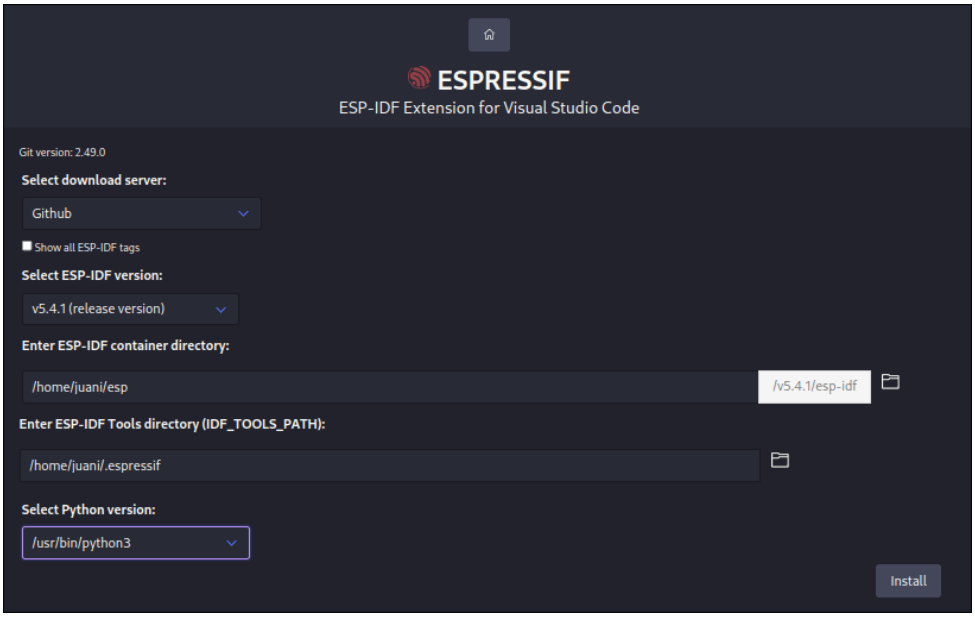


Figura B.2: ESP-IDF configuraciones de instalación

Apéndice C

Implementación en firmware de la máquina de estados

El objetivo de esta sección es explicar cómo se interconectan los distintos bloques de código, que trabajando en conjunto logran el funcionamiento de la máquina de estados.

Lo primero, es definir cómo estará compuesta la máquina de estados. En el Listing C.1 se puede apreciar que esta contiene el estado actual y el evento ocurrido.

```
typedef struct state_machine {
    /* Current state */
    sm_state_t curr_state;

    /* Occured event */
    sm_event_t event;

} sm_state_machine_t;
```

Listing C.1: Estructura de una máquina de estados

Siguiendo, se definen los posibles estados en los cuales se podría encontrar el cuadricóptero,

```
typedef enum state_machine_states {

    ST_IDLE,
    ST_INIT,
    ST_WAITING,
    ST_CALIBRATION,
    ST_CONTROL,
    ST_PROPELLER_CALIBRATION,
    ST_RESET

} sm_state_t;
```

Listing C.2: Definición de estados

así como también los eventos que podrían llegar a ocurrir.

```
typedef enum state_machine_events {

    EV_ANY,
    EV_CROSS,
    EV_TRIANGLE,
    EV_CIRCLE,
    EV_PS

} sm_event_t;
```

Listing C.3: Definición de eventos

A partir de estas estructuras, se define una transición entre estados como un suceso compuesto por el estado actual, el evento ocurrido y el próximo estado al cual se debe transiconar.

```
typedef struct state_transition_row {

    /* Current state */
    sm_state_t curr_state;

    /* Occured event */
    sm_event_t event;

    /* State to go next */
    sm_state_t next_state;

} state_trans_row_t;
```

Listing C.4: Definición de una transición de estado

Luego, se procede a armar una matriz que contenga todas las transiciones del sistema[1].

```
static const state_trans_row_t state_trans_matrix[] = {

/* From IDLE to ... */
{ .curr_state = ST_IDLE, .event = EV_ANY,   .next_state = ST_IDLE },
{ .curr_state = ST_IDLE, .event = EV_CROSS, .next_state = ST_INIT },
{ .curr_state = ST_IDLE, .event = EV_PS,    .next_state = ST_RESET },

... Rest of transitions
```

Listing C.5: Matriz de transiciones

Cada estado tiene una finalidad y, por tanto, cada uno tiene asociado una función encargada de llevar a cabo dicha finalidad. Nótese que el orden de cada elemento del array (Listing C.6) es el mismo en el cual se definieron los elementos del enum del Listing C.2.

```
static state_func_row_t state_function_array[] = {
    { .name = "ST_IDLE",                .func = &StIdleFunc },
    { .name = "ST_INIT",                .func = &StInitFunc },
    { .name = "ST_WAITING",             .func = &StWaitingFunc },
    { .name = "ST_CALIBRATION",         .func = &StCalibrationFunc },
    { .name = "ST_CONTROL",             .func = &StControlFunc },
    { .name = "ST_PROPELLER_CALIBRATION", .func = &StPropCalibrationFunc },
    { .name = "ST_RESET",               .func = &StResetFunc },
};
```

Listing C.6: Estados con sus respectivas funciones

Para utilizar la máquina de estados, se comienza por crear una instancia e inicializarla[2].

```
sm_state_machine_t state_machine;   // Make a new state machine
StateMachine_Init(&state_machine);  // Initialize the state machine
```

Listing C.7: Instancia e inicialización de una máquina de estados

El último paso es desarrollar un algoritmo encargado de detectar el estado actual, el evento ocurrido y definir cuál es el estado al cual se debe transicionar.

```
void StateMachine_RunIteration(sm_state_machine_t * state_machine, drone_t * drone) {
/* Loop through the entire transition matrix to match actual state and occurred event */
    for(int i = 0; i < sizeof(state_trans_matrix) / sizeof(state_trans_matrix[0]); i++) {

        /* If matched actual state */
        if(state_trans_matrix[i].curr_state == state_machine->curr_state) {

```

[1] Se muestra únicamente las transiciones del estado *IDLE*, ya que no es necesario que la matriz esté completa para la explicación.

[2] La inicialización consiste únicamente en setear el estado actual de la máquina de estados, en *IDLE*.

```
8                 /* If matched occurred event */
9                 if(state_trans_matrix[i].event == state_machine->event) {
10
11                    /* Go to the next state */
12                    state_machine->curr_state = state_trans_matrix[i].next_state;
13
14                    /* Run new actual state respective function */
15                    state_function_array[state_machine->curr_state].func(drone);
16                    break;
17                }
18            }
19        }
20    }
```

Listing C.8: Procesamiento de la máquina de estados

Fundamentalmente, lo que hace esta función es recorrer la *matriz de transición* a través de un bucle *for*, tal que las tareas en orden secuencial son las siguientes.

1. Verificar cuál es el estado actual de la máquina de estados.
2. Verificar qué evento ocurrió.
3. Si hubo una coincidencia entre estado y evento, entonces ya se habrá identificado la fila correspondiente en la *matriz de transición*, de manera que solo resta transicionar al próximo estado. Para ello, se actualiza el estado actual de la máquina de estados, con el valor *estado_futuro* de esa fila de la matriz.
4. Por último, se ejecuta la función asociada al nuevo estado[3].

[3]Es importante darse cuenta que, para que este algoritmo funcione, el array de estados-funciones del Listing C.6 tiene que tener el mismo orden que el enum de estados del Listing C.2. De lo contrario, la línea `state_function_array[state_machine->curr_state].func(drone);`, del Listing C.8, no funcionará según lo esperado.

Apéndice D

Modelo de los motores en el dominio de Laplace

El objetivo de esta sección es hallar una ecuación, en el dominio de Laplace, que permita modelar la respuesta de un motor de corriente continua sin escobillas, frente a un pulso PWM. Luego, se sabe que la ecuación que modela el comportamiento de los motores, en el dominio temporal, es

$$y(t) = \omega_0(1 - e^{\frac{t_a - t}{\tau}})$$

Entonces, aplicando la transformada de Laplace a ambos lados de la igualdad se tiene que

$$\mathscr{L}\{y(t)\} = \mathscr{L}\{\omega_0(1 - e^{\frac{t_a - t}{\tau}})\}$$

$$Y(s) = \omega_0\left[\mathscr{L}\{1\} - \mathscr{L}\{e^{\frac{t_a - t}{\tau}}\}\right] = \omega_0\left[A - B\right]$$

$$A : \mathscr{L}\{1\} = \frac{1}{s}$$

$$B : \mathscr{L}\{e^{\frac{t_a - t}{\tau}}\} = \mathscr{L}\{e^{\frac{t_a}{\tau}} \cdot e^{\frac{-t}{\tau}}\} = e^{\frac{t_a}{\tau}}\mathscr{L}\{e^{\frac{-t}{\tau}}\} = e^{\frac{t_a}{\tau}}\frac{1}{s + \frac{1}{\tau}}$$

Ahora bien, juntando ambos resultados

$$Y(s) = \omega_0\left[\frac{1}{s} - e^{\frac{t_a}{\tau}}\frac{1}{s + \frac{1}{\tau}}\right] = \omega_0\left[\frac{(s + \frac{1}{\tau}) - e^{\frac{t_a}{\tau}}s}{s(s + \frac{1}{\tau})}\right]$$

Ahora bien, el instante en el cual el motor recibe el pulso PWM, t_a, es arbitrario y se puede definir como $t_a = 0s$, tal que $e^{\frac{0}{\tau}} = e^0 = 1$. Luego, la ecuación queda de la siguiente forma

$$Y(s) = \omega_0\left[\frac{1}{s} - 1\frac{1}{s + \frac{1}{\tau}}\right] = \omega_0\left[\frac{(s + \frac{1}{\tau}) - s}{s(s + \frac{1}{\tau})}\right] = \omega_0\left[\frac{\frac{1}{\tau}}{s(s + \frac{1}{\tau})}\right]$$

Por otra parte, en los diagramas en bloques, la función transferencia $T(s)$ se define de la siguiente manera

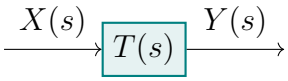


Figura D.1: Diagrama en bloques genérico de una función transferencia

Es decir, $Y(s) = X(s)T(s)$ y análogamente

$$Y(s) = \overbrace{\frac{\omega_0}{s}}^{X(s)} \underbrace{\frac{\frac{1}{\tau}}{s + \frac{1}{\tau}}}_{T(s)}$$

La Figura D.2 representa la forma de onda de una señal PWM genérica con período $T = t_{on} + t_{off}$.

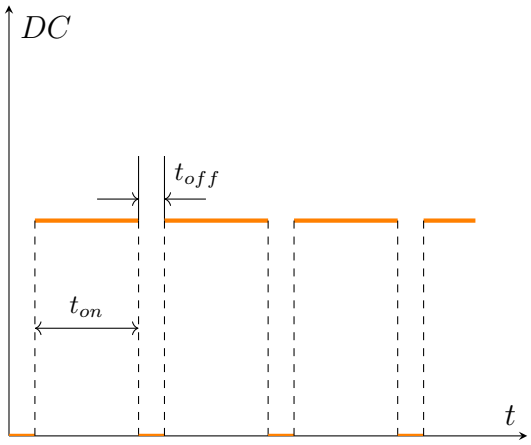


Figura D.2: Forma de onda de una señal PWM

Por otra parte, la transformada de Laplace de un escalón unitario es $\frac{1}{s}$, lo cual da lugar a interpretar el término $X(s)$ como la entrada al sistema (motor). Así, aplicando la transformada inversa de Laplace se obtiene,

$$\mathscr{L}^{-1}\{\frac{\omega_0}{s}\} = \omega_0$$

Esto significa que la señal PWM está enviando un pulso de tensión al motor, provocando un giro equivalente a ω_0. Luego, la forma en la cual el motor intentará alcanzar este valor se encuentra definida por $T(s)$.

Así, se concluye que la función de transferencia que modela el comportamiento de los motores es

$$T(s) = \frac{\frac{1}{\tau}}{s + \frac{1}{\tau}} = \frac{1}{\tau s + 1} \text{ , con } [\tau] = segundos. \tag{D.1}$$

Apéndice E

Obtención de las aceleraciones angulares

El objetivo de esta sección es exponer el procedimiento matemático para la obtención de las ecuaciones para las aceleraciones angulares. Comenzando por la ley de Euler del movimiento de cuerpos rígidos descrita en la sección 2.1.3 se tiene que,

$$\begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \begin{bmatrix}j_{xx} & 0 & 0\\ 0 & j_{yy} & 0\\ 0 & 0 & j_{zz}\end{bmatrix}\begin{bmatrix}\ddot{\phi}\\ \ddot{\vartheta}\\ \ddot{\varphi}\end{bmatrix} + \begin{bmatrix}\dot{\phi}\\ \dot{\vartheta}\\ \dot{\varphi}\end{bmatrix} \times \begin{bmatrix}j_x & 0 & 0\\ 0 & j_y & 0\\ 0 & 0 & j_z\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\vartheta}\\ \dot{\varphi}\end{bmatrix} =$$
$$= \begin{bmatrix}j_{xx}\ddot{\phi}\\ j_{yy}\ddot{\vartheta}\\ j_{zz}\ddot{\varphi}\end{bmatrix} + \begin{bmatrix}\dot{\phi}\\ \dot{\vartheta}\\ \dot{\varphi}\end{bmatrix} \times \begin{bmatrix}j_{xx}\dot{\phi}\\ j_{yy}\dot{\vartheta}\\ j_{zz}\dot{\varphi}\end{bmatrix} = A + B$$

Para calcular B simplemente se debe realizar una multiplicación vectorial,

$$B = \begin{vmatrix}\hat{\imath} & \hat{\jmath} & \hat{k}\\ \dot{\phi} & \dot{\vartheta} & \dot{\varphi}\\ j_{xx}\dot{\phi} & j_{yy}\dot{\vartheta} & j_{zz}\dot{\varphi}\end{vmatrix} = \begin{vmatrix}\dot{\vartheta} & \dot{\varphi}\\ j_{yy}\dot{\vartheta} & j_{zz}\dot{\varphi}\end{vmatrix}\hat{\imath} - \begin{vmatrix}\dot{\phi} & \dot{\varphi}\\ j_{xx}\dot{\phi} & j_{zz}\dot{\varphi}\end{vmatrix}\hat{\jmath} + \begin{vmatrix}\dot{\phi} & \dot{\vartheta}\\ j_{xx}\dot{\phi} & j_{yy}\dot{\vartheta}\end{vmatrix}\hat{k} =$$
$$= \begin{bmatrix}j_{zz}\dot{\vartheta}\dot{\varphi} - j_{yy}\dot{\vartheta}\dot{\varphi}\\ j_{xx}\dot{\phi}\dot{\varphi} - j_{zz}\dot{\phi}\dot{\varphi}\\ j_{yy}\dot{\vartheta}\dot{\phi} - j_{xx}\dot{\vartheta}\dot{\phi}\end{bmatrix} = \begin{bmatrix}\dot{\vartheta}\dot{\varphi}(j_{zz} - j_{yy})\\ \dot{\phi}\dot{\varphi}(j_{xx} - j_{zz})\\ \dot{\vartheta}\dot{\phi}(j_{yy} - j_{xx})\end{bmatrix}$$

Ahora que ya se ha obtenido una ecuación para el término B, se vuelve a reescribir la ecuación que describe los torques sobre cada eje como sigue,

$$\begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \begin{bmatrix}j_{xx}\ddot{\phi}\\ j_{yy}\ddot{\vartheta}\\ j_{zz}\ddot{\varphi}\end{bmatrix} + \begin{bmatrix}\dot{\vartheta}\dot{\varphi}(j_{zz} - j_{yy})\\ \dot{\phi}\dot{\varphi}(j_{xx} - j_{zz})\\ \dot{\vartheta}\dot{\phi}(j_{yy} - j_{xx})\end{bmatrix}$$

Finalmente, despejando el término que contiene las aceleraciones angulares y eliminando las variables que las multiplican, se obtiene el sistema de ecuaciones resultante para cada aceleración angular.

$$\begin{bmatrix}j_{xx}\ddot{\phi}\\ j_{yy}\ddot{\vartheta}\\ j_{zz}\ddot{\varphi}\end{bmatrix} = \begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} - \begin{bmatrix}\dot{\vartheta}\dot{\varphi}(j_{zz} - j_{yy})\\ \dot{\phi}\dot{\varphi}(j_{xx} - j_{zz})\\ \dot{\vartheta}\dot{\phi}(j_{yy} - j_{xx})\end{bmatrix}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\vartheta} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} \frac{\tau_x}{j_{xx}} \\ \frac{\tau_y}{j_{yy}} \\ \frac{\tau_z}{j_{zz}} \end{bmatrix} - \begin{bmatrix} \dot{\vartheta}\dot{\varphi}\frac{(j_{zz}-j_{yy})}{j_{xx}} \\ \dot{\phi}\dot{\varphi}\frac{(j_{xx}-j_{zz})}{j_{yy}} \\ \dot{\vartheta}\dot{\phi}\frac{(j_{yy}-j_{xx})}{j_{zz}} \end{bmatrix}$$

Bibliografía

Administración Nacional de Aviación Civil (A.N.A.C). (2025). *Resolución ANAC 550/2025*. Administración Nacional de Aviación Civil (A.N.A.C). https://www.argentina.gob.ar/anac/nuevo-marco-normativo-para-la-operacion-de-drones

Alan V. Oppenheim, S. H. N., Alan S. Willsky. (s.f.). *Señales y sistemas* (Vol. 2). Prentice Hall.

Ardupilot. (2025a). *Archivado: Placa de control de vuelo APM*. Ardupilot. https://ardupilot.org/copter/docs/common-apm25-and-26-overview.html

Ardupilot. (2025b). *Página oficial de Ardupilot*. Ardupilot. https://ardupilot.org/copter/index.html#

Berrios, M. G., Berger, T., Tischler, M. B., & Sanders, F. C. (2017). Hover Flight Control Design for UAS Using Performance-based Disturbance Rejection Requirements [May 2017]. *AHS International 73rd Annual Forum*.

Betaflight. (2025). *Características de placa de control de vuelo Airbot Fenix G4 35A AIO*. Betaflight. https://betaflight.com/docs/wiki/boards/current/AIRBOTG4AIO

Bosch Sensortec. (2021a, octubre). *BMP280 Digital Pressure Sensor Datasheet* [Document Number: BST-BMP280-DS001]. Ver. 1.26. Bosch Sensortec GmbH. https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf

Bosch Sensortec. (2021b, marzo). *BMP390 Digital Pressure Sensor Datasheet* [Document Number: BST-BMP390-DS002-07]. Ver. 1.7. Bosch Sensortec GmbH. https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp390-ds002.pdf

Cytron Technologies. (2013, mayo). *HC-SR04 User's Manual*. Ver. 1.0. Cytron Technologies. https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit?tab=t.0

Espressif Systems. (2023). *I2C Driver - ESP-IDF Programming Guide* [Documentación oficial del ESP-IDF v5.4.1]. Espressif Systems. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/i2c.html

Hobbywing. (s.f.). *Hobbywing Skywalker datasheet* (V2). Hobbywing. https://www.hobbywing.com/en/uploads/file/20230321/69381b562c41439ee4451c7152905f10.pdf

MathWorks. (s.f.-a). *Página principal de Matlab*. MathWorks. https://es.mathworks.com/

MathWorks. (s.f.-b). *Software Simulink de Matlab*. MathWorks. https://es.mathworks.com/products/simulink.html

Poder Ejecutivo Nacional. (1967). *Ley 17.285*. Poder Ejecutivo Nacional. https://www.argentina.gob.ar/transporte/anac/normativa/codigo-aeronautico

Poder Ejecutivo Nacional. (2007). *Decreto DNU 239 / 2007*. Poder Ejecutivo Nacional. https://www.argentina.gob.ar/normativa/nacional/decreto-239-2007-126444

PX4. (2025). *Características de placa de control de vuelo NXP MR-VMU-RT1176*. PX4. https://docs.px4.io/main/en/flight_controller/nxp_mr_vmu_rt1176.html

Python. (s.f.). *Documentation - Instalando módulos de Python*. Python.org. https://docs.python.org/es/3/installing/index.html

Quan, Q. (2017). *Introduction to multicopter design and control* (Vol. 10). Springer.

Scoflich Lautaro y Grandinetti Juan. (2025). *Flight-Controller* [Firmware implementado para el cuadricóptero]. https://github.com/Lautarosco/PFI-Firmware/tree/main/Flight-Controller

u-blox. (2018, junio). *u-blox 7 Receiver Description* [Document Number: GPS.G7-SW-12001-B1]. Ver. 14.00. u-blox. https://content.u-blox.com/sites/default/files/products/documents/u-blox7-V14_ReceiverDescriptionProtocolSpec_%28GPS.G7-SW-12001%29_Public.pdf

Valdez, J., & Becker, J. (2015). Understanding the I2C bus. *Texas instruments*, 8. https://www.ti.com/lit/an/slva704/slva704.pdf

Visual Studio Code. (s.f.). https://code.visualstudio.com/