Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Implementación y análisis comparativo de diferentes sistemas de control de un cuadricóptero de fabricación propia

GRADO DE INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Trabajo de Fin de Grado

JULIO, 2024

Autor: Rubén Sahuquillo Redondo

Tutor: Santiago Torres Álvarez

AGRADECIMIENTOS

Agradezco profundamente a mi tutor Santiago Torres Álvarez, por su orientación, paciencia y dedicación durante todo este proyecto, su orientación, explicaciones y sugerencias han sido imprescindibles para mejorar y finalizar este trabajo.

También quisiera agradecer a mi familia y pareja por su amor, apoyo incondicional, esfuerzo, sacrificio y resiliencia, por poner siempre buena cara pese a las adversidades y poder disfrutar y compartir cada día a su lado.

Por último, quisiera agradecer a la Universidad de La Laguna por permitirme desarrollar este proyecto y por los recursos proporcionados para su realización

Índice

Índice de ilustraciones:

Índice de tablas:

1. Glosario de términos, acrónimos.

Acrónimos

- UAV (*Unmanned Aerial Vehicle*): Vehículo aéreo no tripulado.
- BLDC (*Brushless DC Motor*): Motor DC sin escobillas.
- ESC (*Electronic Speed Controller*): Controlador electrónico de velocidad.
- MEMS (*Microelectromechanical system*): Sistemas microelectromecánicos. Se refiere a la tecnología electromecánica de dispositivos de tamaño microscópico.
- LQR (*Lineal Quadratic Regulador*): Controlador Cuadrático Lineal.
- PID (*Proportional – Integral - Derivative*): Controlador Proporcional – Integral – Derivativo.

Definición de variables

$m \epsilon R$: masa del cuadricóptero.

$I \epsilon R^{3x3}$: tensor de inercias del cuadricóptero.

$\vec{r} \epsilon R^3$: vector de posición del centro de masas del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$$\vec{r} = [x, y, z]^T$$

$\vec{\omega} \epsilon R^3$: vector de velocidad angular del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$$\vec{\omega} = [\omega_\phi, \omega_\theta, \omega_\Psi]^T$$

$\vec{\zeta} \epsilon R^3$: vector de velocidad angular del cuadricóptero expresado respecto al sistema de referencia del cuadricóptero {D}.

$$\vec{\zeta} = [p, q, r]^T$$

$\vec{v} \epsilon R^3$: vector de velocidad lineal del centro de masas del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$$\vec{v} = [v_x, v_y, v_z]^T$$

$\vec{d} \epsilon R^3$: vector de velocidad lineal del centro de masas del cuadricóptero expresado respecto al sistema del cuadricóptero {D}.

$$\vec{d} = [d_x, d_y, d_z]^T$$

$\vec{\eta} \epsilon R^3$: vector de orientación del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$$\vec{\eta} = [\Phi, \theta, \Psi]^T$$

$R \epsilon SO(3)$: Matriz de rotación ortonormal. Relaciona las posiciones y velocidades lineales entre ambos sistemas de referencia (inercial y no inercial).

$W \epsilon SO(3)$: Matriz de rotación que relaciona $\vec{\zeta}$ y $\vec{\omega}$.

$F_i \epsilon R$: Empuje generado por el i – ésimo rotor.

$M_i \epsilon R$: Par generado por el i – ésimo rotor.

$F \epsilon R$: Empuje total generado por la suma del empuje individual de cada uno de los rotores.

$\vec{\tau} \epsilon R^3$: Vector de pares que contiene los pares y fuerzas generados por el cuadricóptero

$$\vec{\tau} = [F, \tau_x, \tau_y, \tau_z]^T$$

$L \epsilon N$: Distancia del centro de masa del cuadricóptero al centro de masa del rotor.

2. Resumen / Abstract

Este trabajo de fin de grado se enfoca en el desarrollo completo de un UAV de tipo cuadricóptero, desde su conceptualización y modelización matemática y diseño hasta su implementación en una plataforma de cuadricóptero operativa que permita la realización de ensayos.

Inicia con el desarrollo de modelos matemáticos complejos no lineales que describen el comportamiento cinemático y dinámico del cuadricóptero, seguido de la linealización de estos modelos en torno a un punto de equilibrio estable de vuelo estacionario.

El estudio y evaluación de controladores en simulación constituye una parte importante de este trabajo. Esta evaluación busca identificar aquellos métodos más adecuados, en base a las sintonizaciones establecidas, para controlar el movimiento y orientación del cuadricóptero de manera robusta ante perturbaciones.

Por otro lado, el diseño y desarrollo de una plataforma de cuadricóptero operativa, tanto en hardware como en software se desarrolla y describe de forma completa, con el objetivo de permitir la realización de ensayos y flexibilidad en la implementación de algoritmos en microcontroladores de bajo coste.

This final degree project focuses on the complete development of a quadcopter-type UAV, from its conceptualization and mathematical modeling and design to its implementation in an operational quadcopter platform that allows the performance of tests.

It begins with the development of complex nonlinear mathematical models that describe the kinematic and dynamic behavior of the quadcopter, followed by the linearization of these models around a stable equilibrium point of hovering.

The study and evaluation of controllers in simulation is an important part of this work. This evaluation seeks to identify the most appropriate methods, based on the established tunings, to control the movement and orientation of the quadcopter in a robust way in the face of disturbances.

On the other hand, the design and development of an operational quadcopter platform, both in hardware and software, is developed and described in a complete way, with the aim of allowing the performance of tests and flexibility in the implementation of algorithms in low-cost microcontrollers.

3. Introducción

3.1. Estado del Arte

Un cuadricóptero es una aeronave no tripulada de cuatro rotores. Estas aeronaves han ganado popularidad debido a su flexibilidad y usos en multitud de aplicaciones que abarcan desde el uso recreativo, pasando por la fotografía, agricultura y vigilancia hasta el campo de la policía, bomberos y fuerzas armadas [1]

Los sistemas de control presentes en los cuadricópteros y UAV han evolucionado significativamente [2] debido, principalmente, a los avances en campos como:

- Electrónica y comunicaciones: los avances en estas áreas están permitiendo la miniaturización aún mayor de componentes electrónicos y el aumento de la velocidad de procesamiento de datos, así como comunicaciones más rápidas y fiables, mejorando la robustez frente a interferencias, ataques cibernéticos, protección de datos, etc.

- Sensores: los avances en esta área han permitido contar con IMUs y otros sensores basados en tecnología MEMS (*MicroelectroMechanical System*). Esta tecnología en auge consiste en sistemas electromecánicos de tamaño microscópico capaces de generar señales en base a el movimiento de elementos mecánicos internos. Estos sensores son extremadamente pequeños y en algunos casos extremadamente precisos. Otros avances como en cámaras de alta resolución y velocidad han permitido desarrollar control visual en estos vehículos y sensores láser como los LIDAR permiten un mapeo y percepción del entorno más exacta.

- Redes neuronales y *Machine Learning*: avances en estas áreas están permitiendo mejorar la respuesta de este tipo de vehículos para realizar maniobras más precisas, en entornos de vuelo cambiantes incluso de forma colaborativa con otros UAV. [3]

Los métodos de control basados en el control PID siguen siendo los métodos predominantes en aeronaves de este tipo, debido a su efectividad y simplicidad, si bien también métodos como el control LQR, adaptativo o redes neuronales están ganando popularidad en aplicaciones que requieren un control preciso y optimizado, así como en aplicaciones de robótica colaborativa.

El surgimiento de plataformas de código abierto tales como ArduPilot o PixHawk han abierto este mundo a una gran comunidad de aficionados. Estas plataformas se han vuelto muy populares debido a su robustez, potencia, fiabilidad e integración de software y hardware en un único dispositivo. Además, su coste es menor a plataformas propietarias y es compatible con la mayoría del hardware existente.

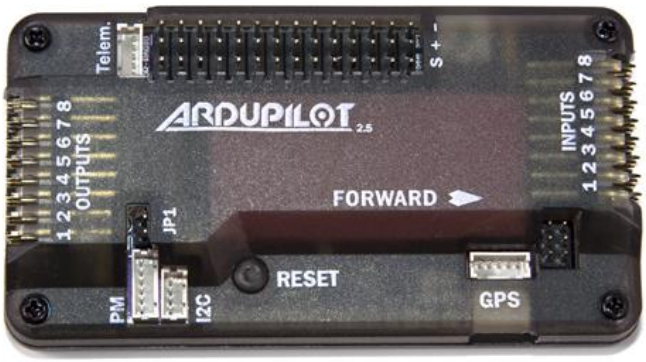


Ilustración 1: ArduPilot Mega [https://www.ardupilot.co.uk/]

Estas plataformas emplean principalmente algoritmos de control PID y variantes de estos, como PID adaptativos, aunque estos códigos pueden ser modificados por el usuario.

De igual manera, los avances en microcontroladores, ha propiciado la aparición de plataformas de propósito general, de bajo costo y alto rendimiento, muy superiores a los clásicos Arduinos, en soluciones del Internet de las Cosas (IoT), tales como el potente ESP32, y Arduinos de elevadas prestaciones como el Arduino Nano RP2040 y otros microcontroladores de elevada velocidad como el Teensy 4.0. Estos avances los convierte en dispositivos interesantes, potentes, flexibles y sencillos de utilizar en los que se puede desarrollar numerosos sistemas embebidos, siendo aún más baratos que plataformas como ArduPilot.

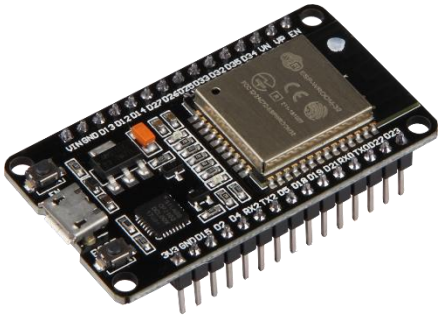


Ilustración 2: NodeMCU-ESP32 [https://www.joy-it.net/en/products/SBC-NodeMCU-ESP32]

3.2. Objetivos

Los objetivos marcados para este trabajo son los siguientes:

- Desarrollo de modelos del comportamiento de los cuadricópteros, incluyendo su dinámica completa, así como la de sus actuadores, a fin de establecer una herramienta de simulación fiable y robusta.

- Diseño de sistemas de control en simulación para el análisis de sus rendimientos.

- Diseño de una plataforma de cuadricóptero operativa, con la cual se puedan probar los diferentes algoritmos.

- Desarrollo del software necesario para la implementación de las diferentes técnicas de control, así como la comunicación con dispositivos de captación y almacenamiento de datos del rendimiento del cuadricóptero en tiempo real para microcontroladores de bajo coste tipo Arduino.

- Implementación real del sistema de control y práctica de vuelo en interiores para la confirmación y ajuste de los sistemas de control previamente testeados en simulación.

4. Modos de vuelo de UAV. Movimiento de un cuadricóptero

Un UAV se trata de un sistema parcialmente subactuado, es decir, algunas de las variables no pueden ser controladas directamente, sino que son consecuencia de controlar otra variable derivada.

Las variables de control de un UAV se pueden dividir en:

- **Variables que se pueden controlar de forma directa:**
    - **Alabeo (*Roll*):** El ángulo de alabeo se controla directamente con la diferencia del empuje ejercido entre los motores laterales de la aeronave.
    - **Cabeceo (*Pitch*):** El ángulo de cabeceo se controla directamente con la diferencia del empuje ejercido entre los motores delanteros y traseros de la aeronave.
    - **Guiñada (*Yaw*):** El ángulo de guiñada se controla directamente con la diferencia del par o torque ejercido entre los motores diagonales.
    - **Z:** La posición en Z se controla directamente con el empuje total que ejercen los motores.
- **Variables que son controladas de forma indirecta:**
    - **X:** La posición en X es consecuencia del ángulo de cabeceo de la aeronave.
    - **Y:** La posición en Y es consecuencia del ángulo de alabeo de la aeronave.

Los sistemas y algoritmos de control a desarrollar dependen del modo de vuelo que se quiera llevar a cabo. Dependiendo de la gama del UAV, pueden incorporar uno o varios de los siguientes modos de vuelo [4]:

Modos en los que interviene el operador activamente (control manual):

1. **Modo Acrobático:** En este modo el operador tiene control pleno sobre la aeronave, permitiéndole ejecutar maniobras complejas con mínima asistencia.

    a. *Variables de control*
        i. Velocidad de Ascenso (eje z)
        ii. Velocidades Angulares de Guiñada, Cabeceo y de Alabeo

    b. *Sensores mínimos requeridos:*
        i. Acelerómetros y Giroscopios
        ii. Altímetros

    c. *Procesamiento:*
        i. Un microcontrolador moderno es capaz de realizar los cálculos necesarios para mantener la estabilidad y alcanzar las consignas marcadas por el operador del dron.

2. **Modo Estable:** En este modo el operador tiene control manual sobre la aeronave, pero añade un control automático de la estabilidad, facilitando el control de la aeronave. Este es el modo que vamos a implementar para el estudio.

    a. *Variables de control*
        i. Velocidad de Ascenso (eje z)
        ii. Velocidad Angular de Guiñada
        iii. Ángulos de Cabeceo y de Alabeo

    b. *Sensores mínimos requeridos:*
        i. Acelerómetros y Giroscopios
        ii. Altímetros

    c. *Procesamiento:*
        i. Un microcontrolador moderno es capaz de realizar los cálculos necesarios para mantener la estabilidad y alcanzar las consignas marcadas por el operador del dron.

Modos en los que interviene el operador mínimamente (control automático)

1. **Modo de Posicionamiento:**
   a. *Variables de control*
      i. Posición en X, Y, Z

   b. *Sensores requeridos (uno o varios):*
      i. Acelerómetro, Giroscopio, Magnetómetro (IMU completa)
      ii. GPS
      iii. Altímetro

   c. *Procesamiento*
      i. Requiere un ordenador de vuelo para funciones de alto nivel (cálculo de posiciones, trayectorias, gestión de comunicaciones, procesamiento de lectura de sensores, etc.)
      ii. Requiere un microcontrolador para funciones de bajo nivel (control de actuadores, control de estabilidad, inclinación, etc.)

2. **Modo de Seguimiento:**
   a. *Variables de control*
      i. Posición en X, Y, Z

   b. *Sensores requeridos (uno o varios):*
      i. Acelerómetro, Giroscopio, Magnetómetro (IMU completa)
      ii. GPS
      iii. Altímetro
      iv. Cámaras, cámaras térmicas u otros sensores dependiendo del seguimiento.

   c. *Procesamiento*
      i. Requiere un ordenador de vuelo para funciones de alto nivel (cálculo de posiciones, trayectorias, gestión de comunicaciones, procesamiento de lectura de sensores, etc.)
      ii. Requiere un microcontrolador para funciones de bajo nivel (control de actuadores, control de estabilidad, inclinación, etc.)

Estos modos pueden implementar algoritmos de detección de obstáculos, y algoritmos de detección y actuación frente a fallo de GPS, fallo de algún rotor, etc.

Otros modos de vuelo:
Además de los mencionados, los UAV pueden incorporar algunos submodos autónomos adicionales, que pueden ser activados por el operador:

1. Vuelta a casa o "*Return to Home*" (RTH): permite la vuelta del cuadricóptero al lugar de partida.

2. Altitud estable o "*Altitude Hold*" (AH): permite al cuadricóptero mantener una altura constante.

3. Modo sin cabeza o "*Headless Mode*" (HM): permite al cuadricóptero desvincular la orientación de el mismo de los controles direccionales.

5. Modelado físico del cuadricóptero

A continuación, se incluyen los modelos cinemático y dinámico de un cuadricóptero, a *[5]*

5.1. Modelo cinemático

El modelo cinemático describe y relaciona las posiciones, velocidades y aceleraciones de un sistema sin considerar las fuerzas y momentos que actúan en dicho sistema.

Es necesaria para relacionar magnitudes entre distintos sistemas de referencia.

La orientación del cuadricóptero {D} con respecto al sistema de referencia {0} se obtiene por medio de la matriz de rotación R, la cual se parametriza empleando los ángulos de alabeo **Φ**, cabeceo Θ y guiñada **Ψ**. Estos representan la rotación respecto a los ejes X, Y, Z respectivamente. Siguiendo la convención Z, X, Y, la matriz R se define como:

$$R = R_{Z,\Psi} \cdot R_{X,\Phi} \cdot R_{Y,\theta}$$

$$R_{X,\Phi} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Phi) & -sen(\Phi) \\ 0 & sen(\Phi) & \cos(\Phi) \end{pmatrix}$$

$$R_{Y,\theta} = \begin{pmatrix} \cos(\theta) & 0 & sen(\theta) \\ 0 & 1 & 0 \\ -sen(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R_{Z,\Psi} = \begin{pmatrix} \cos(\Psi) & -sen(\Psi) & 0 \\ sen(\Psi) & \cos(\Psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Para obtener cada una de esas matrices, se proyecta cada uno de los ejes del sistema de referencia no inercial del dron {D} sobre el sistema de referencia inercial {0}.

$$R = \begin{pmatrix} cos\Psi cos\theta - sen\Phi sen\Psi sen\theta & -cos\Phi sen\Psi & cos\Psi sen\theta + sen\Phi sen\Psi cos\theta \\ sen\Phi cos\Psi sen\theta + sen\Psi cos\theta & \cos\Phi\cos\Psi & sen\Psi sen\theta - sen\Phi cos\Psi cos\theta \\ -cos\Phi sen\theta & sen\Phi & \cos\Phi\cos\theta \end{pmatrix}$$

Además, R, al ser ortonormal, cumple que:

$$R^{-1} = R^T$$

Para las magnitudes angulares la relación dada por la matriz de rotación R no es válida. Para relacionar las medidas angulares entre el sistema no inercial {B} y el inercial {I] se define la matriz W.

$$W = R^{-1} \cdot \dot{R}$$

$$W = \dot{\theta} i_2 + \dot{\Phi}[R_{Y,\theta}]^T i_1 + \dot{\Psi}[R_{Y,\theta}]^T [R_{X,\Phi}]^T i_3$$

$$W = \begin{pmatrix} cos\theta & 0 & -cos\Phi sen\theta \\ 0 & 1 & sen\Phi \\ sen\theta & 0 & \cos\Phi\cos\theta \end{pmatrix}$$

Sin embargo, W no es ortonormal:

$$W^{-1} \neq W^T$$

El modelo cinemático resulta en:

$$\vec{v} = \mathrm{R} \cdot \dot{\vec{r}}$$

$$\vec{\zeta} = \mathrm{W} \cdot \dot{\vec{\eta}}$$

siendo:
$\vec{r} \in R^3$: vector de posición del centro de masas del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$\vec{\zeta} \in R^3$: vector de velocidad angular del cuadricóptero expresado respecto al sistema de referencia del cuadricóptero {D}.

$\vec{v} \in R^3$: vector de velocidad lineal del centro de masas del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

$\vec{\eta} \in R^3$: vector de orientación del cuadricóptero expresado respecto al sistema de referencia inercial {0}.

5.2. Modelo dinámico del cuadricóptero

El modelo dinámico describe y relaciona las fuerza y momentos que actúan sobre un sistema con sus aceleraciones, velocidades y posiciones [5].

5.2.1. Modelo dinámico no lineal

Las ecuaciones empleadas que describen la dinámica vienen dadas por el método de Newton – Euler, a partir de las siguientes suposiciones:

- Asumiremos que se trata de un sólido rígido, por tanto, no deformable, por lo que los esfuerzos internos son nulos.
- Asumiremos que los momentos se aplican en el centro de masas del cuadricóptero

$$2ª\ Ley\ de\ Newton \sum \vec{F} = m \cdot \vec{a}$$

$$2ª\ Ley\ de\ Newton\ para\ la\ rotación \rightarrow \sum \vec{M} = I \cdot \dot{\vec{\zeta}} + \vec{\zeta} \times I \cdot \vec{\zeta}$$

$\vec{F}$: vector de fuerzas aplicadas [N]

m: masa del cuadricóptero [kg]

$\vec{a}$: vector de aceleraciones $\left[\frac{m}{s^2}\right]$

$\vec{M}$: vector de momentos aplicados [Nm]

I: tensor de inercias $[kg \cdot m^2]$

$\vec{\zeta}$: vector de velocidad angular $\left[\frac{rad}{s}\right]$

$\dot{\vec{\zeta}}$: vector de aceleración angular $\left[\frac{rad}{s^2}\right]$

$\vec{\zeta} \times I \cdot \vec{\zeta}$: término que recoge los efectos giroscópicos y fuerzas de Coriolis

5.2.1.1. Dinámicas expresadas respecto a los sistemas de referencia {I} y {D}

Dinámica Traslacional:

Desarrollando la segunda ley de Newton, expresando las aceleraciones respecto al sistema de referencia inercial {I}, usando para ello la matriz de rotación R:

$$\vec{v} = \mathrm{R} \cdot \dot{\vec{r}}$$

Resulta en la ecuación matricial:

$$\{I\} \quad \rightarrow \quad m \cdot \ddot{\vec{r}} = \vec{P} + R \cdot \vec{F}$$

Siendo:

- $\vec{P}$: vector de peso del UAV $\quad \vec{P} = \begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix}$
- $\vec{F}$: vector de empuje ejercido por los rotores del UAV $\quad \vec{F} = \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix}$
  - Para un cuadricóptero:

$$\sum F_i = F_1 + F_2 + F_3 + F_4$$

Desarrollando el producto $R \cdot \vec{F}$:

$$\{I\} \quad \rightarrow \quad m \cdot \ddot{\vec{r}} = \begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix} + \begin{bmatrix} sin(\theta)\, cos(\Psi) + sin(\Phi)\, cos(\theta)\, sin\,(\Psi) \\ sin(\theta)\, sin(\Psi) - sin(\Phi)\, cos(\theta)\, cos\,(\Psi) \\ cos(\Phi)\, cos\,(\theta) \end{bmatrix} \cdot \sum F_i$$

Dinámica Rotacional:

Desarrollando la segunda ley de Newton para rotaciones, expresando las magnitudes angulares respecto al sistema de referencia no inercial {D} resulta en la ecuación matricial:

$$\{D\} \quad \rightarrow \quad I \cdot \dot{\vec{\zeta}} = -\vec{\zeta} \times I \cdot \vec{\zeta} + \overrightarrow{\tau_0}$$

Desarrollando el producto matricial y cada término:

$$\{D\} \quad \rightarrow \quad I \cdot \dot{\vec{\zeta}} = \begin{bmatrix} -qp[I_{zz} - I_{yy}] \\ -pr[I_{zz} - I_{xx}] \\ pq[I_{xx} - I_{yy}] \end{bmatrix} + \begin{bmatrix} T_{1x} \\ T_{2y} \\ T_{3z} \end{bmatrix}$$

Donde los pares ejercidos respecto a cada eje se expresan como:

$$T_{1X} = L\sum \pm F_i$$
$$T_{2Y} = L\sum \pm F_i$$
$$T_{3Z} = \sum \pm M_i$$

En particular para un cuadricóptero:

$$T_{1X} = L(F_1 - F_2 - F_3 + F_4)$$
$$T_{2Y} = L(F_1 + F_2 - F_3 - F_4)$$
$$T_{3Z} = M_1 - M_2 + M_3 - M_4$$

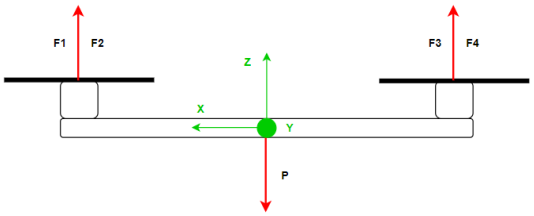


Ilustración 3: Esquema ilustrativo de fuerzas desde el plano xz (elaboración propia)

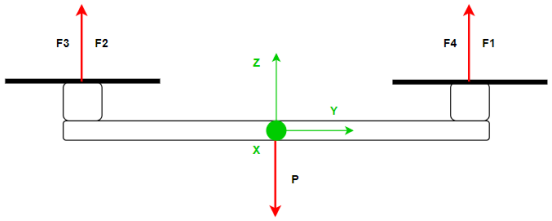


Ilustración 4:Esquema ilustrativo de fuerzas desde el plano yz (elaboración propia)

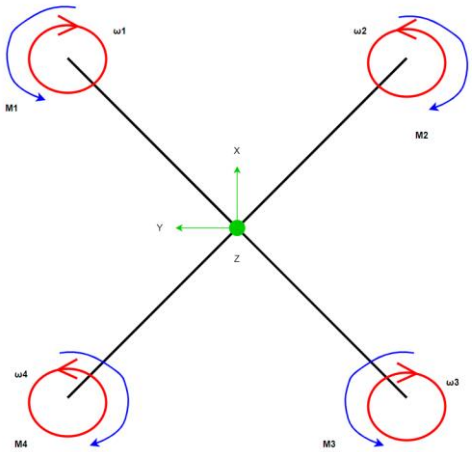


Ilustración 5: Esquema ilustrativo de pares desde el plano yz (elaboración propia)

5.2.1.2. Dinámicas expresadas respecto al sistemas de referencia {I}

Dinámica Traslacional:

$$\{I\} \quad \rightarrow \quad m \cdot \ddot{\vec{r}} = \vec{P} + R \cdot \vec{F}$$

Desarrollando:

$$\{I\} \quad \rightarrow \quad m \cdot \ddot{\vec{r}} = \begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix} + \begin{bmatrix} sin(\theta)\, cos(\Psi) + sin(\Phi)\, cos(\theta)\, sin\,(\Psi) \\ sin(\theta)\, sin(\Psi) - sin(\Phi)\, cos(\theta)\, cos\,(\Psi) \\ cos(\Phi)\, cos\,(\theta) \end{bmatrix} \cdot \sum F_i$$

Dinámica Rotacional:

Desarrollando la ley de Euler, expresando las magnitudes angulares respecto al sistema de referencia inercial {I}, por medio de la matriz W:

$$\vec{\zeta} = \mathrm{W} \cdot \dot{\vec{\eta}}$$

Y para la derivada (aplicando la derivada de un producto):

$$\dot{\vec{\zeta}} = \dot{\mathrm{W}} \cdot \dot{\vec{\eta}} + \mathrm{W} \cdot \ddot{\vec{\eta}}$$

Resulta en la ecuación matricial:

$$\{I\} \quad \rightarrow \quad I \cdot [\dot{\mathrm{W}} \cdot \dot{\vec{\eta}} + \mathrm{W} \cdot \ddot{\vec{\eta}}] = -\mathrm{W} \cdot \dot{\vec{\eta}} \times I \cdot \mathrm{W} \cdot \dot{\vec{\eta}} + \overrightarrow{\tau_0}$$

$$\{I\} \quad \rightarrow \quad I_r \cdot \ddot{\vec{\eta}} = \vec{K} + \overrightarrow{\tau_0}$$

Donde:

- $I_r = \begin{bmatrix} I_{xx}\cos(\theta) & 0 & -I_{xx}\cos(\Phi)\sin(\theta) \\ 0 & I_{yy} & I_{yy}\sin(\Phi) \\ I_{zz}\sin(\theta) & 0 & I_{zz}\cos(\Phi)\cos(\theta) \end{bmatrix}$
- $\vec{K} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix}$, donde los términos K_1, K_2 y K_3 son:

$$K_{11} = \left(I_{xx} + I_{yy} - I_{zz}\right)\dot{\Phi}\dot{\theta}sen(\theta) + \left(-I_{xx} + I_{yy} - I_{zz}\right)\dot{\Phi}\dot{\Psi}sen(\Phi)sen(\theta) + \left(I_{xx} + I_{yy} - I_{zz}\right)\dot{\theta}\dot{\Psi}\cos(\Phi)\cos(\theta) + \left(I_{yy} - I_{zz}\right)\dot{\Psi}^2 sen(\Phi)\cos(\Phi)\cos(\theta)$$

$$K_{22} = \left[-I_{yy} + (-I_{xx} + I_{zz})\cos(2\theta)\right]\dot{\Phi}\dot{\Psi}\cos(\Phi) + (-I_{xx} + I_{zz})\left(\dot{\Phi}^2 - \dot{\Psi}^2\cos(\Phi)^2\right)sen(\theta)\cos\,(\theta)$$

$$K_{33} = \left(I_{xx} - I_{yy} - I_{zz}\right)\dot{\Phi}\dot{\theta}\cos(\theta) + \left(I_{xx} - I_{yy} + I_{zz}\right)\dot{\Phi}\dot{\Psi}sen(\Phi)\cos\,(\theta) + \left(-I_{xx} + I_{yy} + I_{zz}\right)\dot{\theta}\dot{\Psi}\cos(\Phi)\,sen(\theta) + \left(-I_{xx} + I_{yy}\right)\dot{\Psi}^2 sen(\Phi)\cos(\Phi)\,\mathrm{sen}(\theta)$$

Sin embargo, podemos obtener una representación para la dinámica de rotación análoga a la de un robot manipulador, separando las inercias de los efectos giroscópicos y de Coriolis, definiendo la ecuación:

$$\overrightarrow{\tau_0} = M_o(\eta)\ddot{\eta} + C_o(\eta, \dot{\eta})\dot{\eta}$$

Donde:
$M_o(\eta)$: Matriz de inercias

$$M_o(\eta) = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

Y sus términos:

$m_{11} = I_{xx}\cos(\theta)^2 + I_{zz}\sin(\theta)^2$

$m_{12} = 0$

$m_{13} = (-I_{xx} + I_{zz})\cos(\Phi)\sin(\theta)\cos(\theta)$

$m_{21} = 0$

$m_{22} = I_{yy}$

$m_{23} = I_{yy}\sin(\Phi)$

$m_{31} = (-I_{xx} + I_{zz})\cos(\Phi)\sin(\theta)\cos(\theta)$

$m_{32} = I_{yy}\sin(\Phi)$

$m_{33} = I_{xx}\cos(\theta)^2\sin(\theta)^2 + I_{yy}\sin(\Phi)^2 + I_{zz}\cos(\Phi)^2\cos(\theta)^2$

$C_o(\eta, \dot{\eta})$: Matriz de Coriolis y fuerzas centrífugas

$$C_o(\eta, \dot{\eta}) = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

Y sus términos:

$$c_{11} = [-I_{xx} + I_{zz}]\dot{\theta}\sin(\theta)\cos(\theta)$$

$$c_{12} = [-I_{xx} + I_{zz}]\dot{\Phi}\sin(\theta)\cos(\theta) + \frac{1}{2}[-I_{xx} + I_{zz}]\dot{\Psi}\cos(\Phi)\cos(2\theta) - \frac{1}{2}I_{yy}\dot{\Psi}\cos(\Phi)$$

$$c_{13} = \frac{1}{2}[-I_{xx} + I_{zz}]\dot{\theta}\cos(\Phi)\cos(2\theta) - \frac{1}{2}I_{yy}\dot{\theta}\cos(\Phi) + I_{xx}\dot{\Psi}\sin(\Phi)\cos(\Phi)\sin(\theta)^2 - I_{yy}\dot{\Psi}\sin(\Phi)\cos(\Phi) + I_{zz}\dot{\Psi}\sin(\Phi)\cos(\Phi)\cos(\theta)^2$$

$$c_{21} = [I_{xx} - I_{zz}]\dot{\Phi}\sin(\theta)\cos(\theta) + \frac{1}{2}[I_{yy}\dot{\Psi}\cos(\Phi) + [I_{xx} - I_{zz}]\dot{\Psi}\cos(\Phi)\cos(2\theta)]$$

$$c_{22} = 0$$

$$c_{23} = \frac{1}{2}\left[I_{yy}\dot{\Phi}\cos(\Phi) + [I_{xx} - I_{zz}]\dot{\Phi}\cos(\Phi)\cos(2\theta)\right] + [-I_{xx} + I_{zz}]\dot{\Psi}\cos(\Phi)^2\sin(\theta)\cos(\theta)$$

$$c_{31} = [I_{xx} - I_{zz}]\dot{\Phi}\sin(\Phi)\sin(\theta)\cos(\theta) + \frac{1}{2}\left[[-I_{xx} + I_{zz}]\dot{\theta}\cos(\Phi)\cos(2\theta) + I_{yy}\dot{\theta}\cos(\Phi)\right] - I_{xx}\dot{\Psi}\sin(\Phi)\cos(\Phi)\sin(\theta)^2 + I_{yy}\dot{\Psi}\sin(\Phi)\cos(\Phi) - I_{zz}\dot{\Psi}\sin(\Phi)\cos(\Phi)\cos(\theta)^2$$

$$c_{32} = \frac{1}{2}\left[I_{yy}\dot{\Phi}\cos(\Phi) + [-I_{xx} + I_{zz}]\dot{\Phi}\cos(\Phi)\cos(2\theta)\right] + [I_{xx} - I_{zz}]\dot{\Psi}\cos(\Phi)^2\sin(\theta)\cos(\theta)$$

$$c_{33} = -I_{xx}\dot{\Phi}\sin(\Phi)\cos(\Phi)\sin(\theta)^2 + I_{yy}\dot{\Phi}\sin(\Phi)\cos(\Phi) - I_{zz}\dot{\Phi}\sin(\Phi)\cos(\Phi)\cos(\theta)^2 + [I_{xx} - I_{zz}]\dot{\theta}\cos(\Phi)^2\sin(\theta)\cos(\theta)$$

5.2.2. Modelo dinámico linealizado

Para simplificar la dinámica del modelo y poder aplicar algoritmos de control lineal, se requiere la linealización del modelo en torno a un punto de equilibrio.

Este punto de equilibrio se corresponde con el vuelo estacionario del cuadricóptero. El modelo linealizado será válido en las inmediaciones de esta condición. Por tanto, asumiendo ángulos de inclinación pequeños:

$$\begin{aligned} \sin(\theta) &\approx \theta \qquad \cos(\theta) \approx 1 \\ \sin(\Phi) &\approx \Phi \qquad \cos(\Phi) \approx 1 \end{aligned}$$

Y asumiendo un ángulo de guiñada constante y un vuelo estacionario en el plano xy:

$$\Psi = \Psi_{cte} \qquad \sum F_{i(xy)} = mg$$

estas aproximaciones, resultan en:

Dinámica Traslacional:

$$\{I\} \quad \rightarrow \quad m \cdot \ddot{\vec{r}} = \begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix} + \begin{bmatrix} g[\theta\, cos(\Psi_{cte}) + \Phi\, sin(\Psi_{cte})] \\ g[\theta\, sin(\Psi_{cte}) - \Phi\, cos(\Psi_{cte})] \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}$$

Dinámica Rotacional:

$$I_r = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \qquad \vec{K} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Y la dinámica de orientación expresada respecto al sistema inercial {I} y el no inercial {D} se simplifica significativamente e iguala ya que:

$$\vec{\zeta} = \mathrm{W} \cdot \dot{\vec{\eta}} \quad \rightarrow \quad \vec{\zeta} \approx \mathrm{I} \cdot \dot{\vec{\eta}} \quad \rightarrow \quad \vec{\zeta} \approx \dot{\vec{\eta}}$$

resultando en:

$$\{I\} \approx \{\mathrm{D}\} \quad \rightarrow \quad \ddot{\vec{\eta}} = {I_r}^{-1} \overrightarrow{\tau_0}$$

y desarrollando los términos:

$$\{I\} \approx \{\mathrm{D}\} \quad \rightarrow \quad \ddot{\vec{\eta}} = \begin{bmatrix} 1/I_{xx} & 0 & 0 \\ 0 & 1/I_{yy} & 0 \\ 0 & 0 & 1/I_{zz} \end{bmatrix} \cdot \begin{bmatrix} L(F_1 - F_2 - F_3 + F_4) \\ L(F_1 + F_2 - F_3 - F_4) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$

Observamos que, con la linealización del modelo, para ángulos pequeños, las dinámicas de orientación y velocidad en Z quedan desacopladas entre sí.

5.3. Modelo dinámico de los actuadores

Los actuadores de un cuadricóptero están conformados por:

- **Motores:** Elementos encargados de transformar la tensión aplicada en una velocidad de giro
- **Hélices:** Elementos encargados de transformar la velocidad de giro en dos componentes:
  - Fuerza de empuje vertical F_i
  - Par de arrastre M_i
- **Controladores ESC / Puente en H:**
  - ESC: Circuito electrónico encargado de conmutar y controlar los motores sin escobillas o BLDC.
  - Puente en H: Circuito electrónico encargado de controlar los motores DC.

5.3.1. Modelo de los motores

Los motores empleados generalmente en drones pueden ser:

- Motores sin escobillas o *BLDC Motor*
- Motores DC con escobillas

Emplearemos motores del tipo BLDC debido a sus excelentes prestaciones. La dinámica de estos motores es compleja [6] [7] pero se puede aproximar a la dinámica de un motor DC típico:

Dinámica eléctrica:

La dinámica eléctrica de un motor DC se aproxima a la de un circuito RL añadiendo una tensión contraelectromotriz o BEMF:

$$V_i = R_i i_i + L\frac{di_i}{dt} + e_i$$

Siendo la expresión de la tensión contraelectromotriz dependiente de la velocidad de rotación del rotor:

$$e_i = K_e \cdot \omega_i$$

Resulta en:

$$V_i = R_i i_i + L\frac{di_i}{dt} + K_e \cdot \omega_i$$

Y despejando la derivada:

$$\frac{di_i}{dt} = \frac{1}{L}[V_i - R_i i_i - K_e \cdot \omega_i]$$

Dinámica mecánica:
La dinámica mecánica de un motor DC se puede calcular a partir de la segunda ley de Newton rotacional, anulando los términos giroscópicos:

$$\sum \vec{M} = I \cdot \dot{\vec{\zeta}}$$

**(en adelante la inercia I del motor se nombrará J para no inducir a errores con la intensidad de corriente)*

$$T_{m,i} - T_{roz,i} - T_{carga,i} = J \cdot \frac{d\omega_i}{dt}$$

Siendo el torque generado por los motores proporcional a la corriente consumida y el rozamiento dinámico proporcional a la velocidad de rotación:

$$K_m i_i - B_v \omega_i - T_{carga,i} = J \cdot \frac{d\omega_i}{dt}$$

Si despreciamos el par producido por la carga (la hélice en nuestro caso):

$$K_m i_i - B_v \omega_i = J \cdot \frac{d\omega_i}{dt}$$

Y despejando la derivada:

$$\frac{d\omega_i}{dt} = \frac{1}{J}[K_m i_i - B_v \omega_i]$$

Dinámica completa:

$$\frac{di_i}{dt} = \frac{1}{L}[V_i - R_i i_i - K_e \cdot \omega_i]$$
$$\frac{d\omega_i}{dt} = \frac{1}{J}[K_m i_i - B_v \omega_i]$$

5.3.2. Modelo de las hélices

El modelo de las hélices responde a una relación algebraica que relaciona la velocidad de giro del rotor con la fuerza y par que producen por medio de parámetros aerodinámicos y características de esta (densidad del fluido, coeficiente de sustentación, coeficiente de arrastre, diámetro de la hélice...) [8]:

Fuerza de empuje
La fuerza generada por la hélice se expresa de forma cuadrática y en función de la velocidad de rotación como:

$$F_i = C_f \rho D^4 \omega^2 \quad \rightarrow \quad F_i = K_f \omega^2$$

Par de arrastre
El par generado por la hélice se expresa de forma cuadrática y en función de la velocidad de rotación como:

$$M_i = C_q \rho D^4 \omega^2 \quad \rightarrow \quad F_i = K_q \omega^2$$

$$F_i = C_f \rho D^4 \omega^2 \quad \rightarrow \quad F_i = K_f \omega^2$$
$$M_i = C_q \rho D^4 \omega^2 \quad \rightarrow \quad F_i = K_q \omega^2$$

siendo:
F_i: empuje generado por el rotor i-ésimo [N]

M_i: par generador por el rotor i-ésimo [Nm]

C_f: coeficiente de sustentación [N/rps^2]

C_q: coeficiente de arrastre o *"drag"* [Nm/rps^2]

ρ: densidad del aire [kg/m^3]

ω: velocidad de rotación del rotor [rps]

5.3.3. Modelo de los controladores electrónicos de velocidad (ESC)

Estos circuitos electrónicos se encargan de regular la velocidad de giro del motor BLDC según la señal PWM de entrada, además de conmutar sus fases mediante transistores de potencia. Se encargan de convertir el comando PWM [0:1] en una tensión proporcional a la tensión de la batería. [9]

La tensión de salida que alimenta el motor se expresa como:

$$V_{mi} = kV_e - I_e R_e$$

La corriente que llega al ESC se expresa como:

$$I_e = kI_m$$

Y la tensión de entrada al ESC:

$$V_e = V_b - I_b R_b$$

Y la corriente proporcionada por la batería:

$$I_b = 4I_e + I_{elec}$$

Combinando las ecuaciones resulta en:

$$V_{mi} = 4k^2 I_m R_b + k[V_b - I_{elec} R_b - I_m R_e]$$

Y si despreciamos las resistencias R_b y R_e:

$$V_{mi} = kV_b$$

siendo:
R_b: resistencia de salida de la batería [Ω]

R_e: resistencia de los cables de entrada de los ESC [Ω]

I_{elec}: corriente consumida por la electrónica [A]

I_b: corriente entregada por la batería [A]

I_e: corriente consumida por el ESC [A]

I_m: corriente consumida por el motor [A]

V_b: tensión de la batería [V]

V_{mi}: tensión entregada por el ESC al motor [V]

V_e: tensión de entrada al ESC [V]

k: comando PWM enviado al ESC desde el microcontrolador, valores entre 0 y 1

5.4. Reducción de la dinámica del sistema de propulsión.

Los elementos que poseen características dinámicas en nuestro sistema de propulsión son los motores.

Podemos reducir la dinámica del modelo del motor DC a una de primer orden, ya que los efectos debidos a la dinámica eléctrica del motor DC son despreciables frente a la dinámica mecánica, pues la constante de tiempo eléctrica τ_e es mucho menor que la constante de tiempo mecánica τ_m (algunos órdenes de magnitud menor, dependiendo del motor).

El objetivo de este apartado será obtener una función de transferencia de primer orden para la velocidad de giro de un motor DC. Para ello se propone la identificación del modelo del sistema de propulsión en su conjunto de forma experimental.

5.4.1. Metodología y características de los ensayos

La metodología empleada se basa en utilizar un banco de pruebas donde se miden los empujes producidos por los diferentes motores, así como una estimación precisa de las velocidades angulares y constantes de tiempo teniendo en cuenta la respuesta espectral obtenida del movimiento de los motores. Se propone la siguiente metodología con el fin establecer las relaciones entre variables en nuestro sistema de propulsión:

1. Montaje del banco de pruebas.

2. Montaje del motor, ESC, batería y dispositivos de medida.

3. Cargar programa "*TESTBENCH.ino*" al microcontrolador y ejecutar. Este programa envía los comandos de PWM de 1000 a 2000 µs en orden ascendente. Se toman varias lecturas de empuje producido para cada velocidad de giro de la hélice y se obtiene una media para cada una.

4. Grabación del sonido generado por el rotor y recopilación de los valores devueltos del banco de pruebas y equipos de medida.

5. Tratamiento de los datos obtenidos en Excel. Calcular las medias de cada grupo de medidas.

6. Obtención de la relación velocidad angular – empuje, velocidad angular – torque, %PWM – velocidad angular, %PWM – intensidad y %PWM – RPS.

7. Obtención de la relación entre el comando PWM introducido y la intensidad de corriente eléctrica consumida.

8. Tratamiento de la grabación de audio para generar el espectrograma de la señal grabada. En primer lugar, se recortará la señal y se pasará a formato .WAV. El filtrado de frecuencias no deseadas y espectrograma se realizará ejecutando el script desarrollado en Python con nombre "*motor_code_dron.py*".
9. Análisis del espectrograma para obtener la constante de tiempo y las velocidades angulares

Para conocer las prestaciones reales de nuestros rotores y obtener las relaciones entre empuje, velocidad angular, intensidad, par, PWM, etc., crearemos un banco de

pruebas para realizar ensayos a los rotores. Este banco constará de los siguientes elementos:

- 1 vatímetro o en su defecto 2 multímetros.
- 1 microcontrolador
- 1 celda de carga de 1kg
- 1 amplificador para celda de carga HX711
- 1 ESC
- Grabadora de sonido
- Motor y hélice para probar

Variables de interés a medir o estimar:

- Tensión de la batería
- Corriente total consumida
- Empuje ejercido por el rotor
- Sonido del rotor. De esta forma podremos estimar la velocidad de giro en base a un estudio de frecuencias

Esquema eléctrico desarrollado para la realización de ensayos en el banco de pruebas:

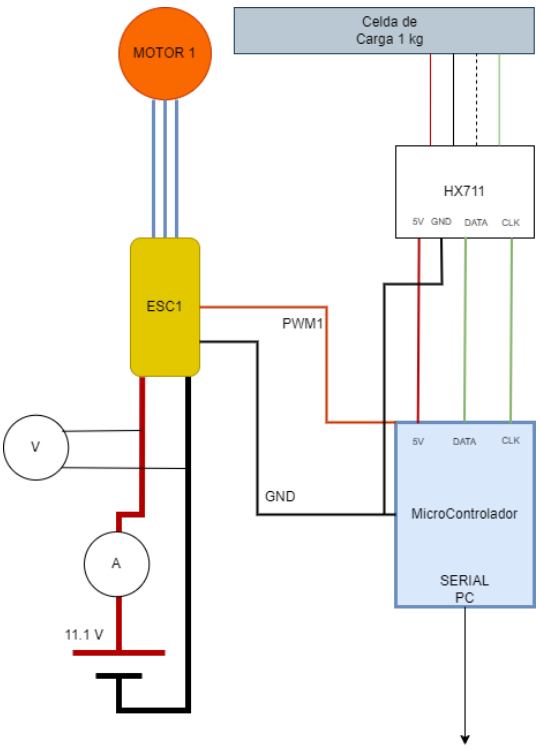


Ilustración 6: Esquema eléctrico del banco de pruebas de motores (elaboración propia)

Para medir la fuerza de empuje capaz de ejercer el rotor se hará uso de una celda de carga de 1kg (1.5 kg admisibles de forma segura). Se trata de un transductor que convierte la fuerza o el peso en una pequeña señal eléctrica que posteriormente es amplificada y convertida a digital gracias a un ADC.

Esta celda de carga funciona como un puente de Wheatstone, un circuito electrónico empleado típicamente con sensores resistivos para medir el desbalance de tensiones entre sus ramas debido a la variación de alguna(s) de las resistencias. En nuestro caso las resistencias que pueden variar con la deformación producida al ejercer fuerza con un objeto son galgas extensiométricas, pero también se emplea con resistencias PTC, NTC y RTD, resistencias diseñadas para variar significativamente con la temperatura.
Este desbalance de tensiones es medido y amplificado por un amplificador de instrumentación y digitalizado empleando un ADC de 24 bits (ambos incorporados por el HX711).

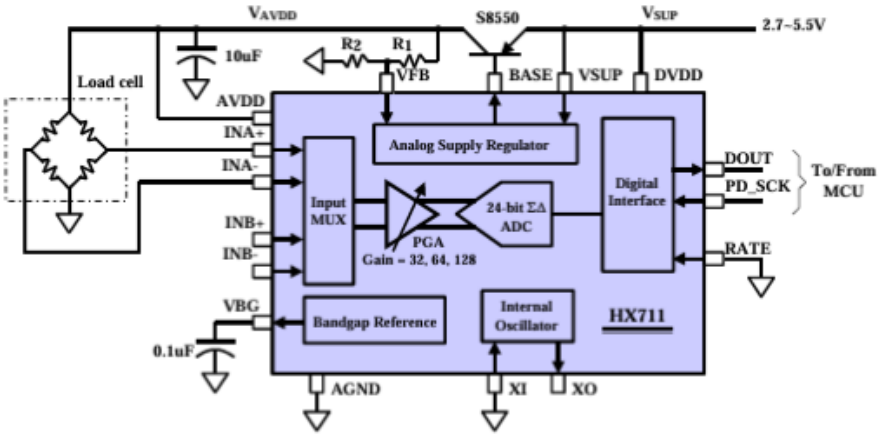


Ilustración 7: Diagrama de electrónico / bloques del módulo HX711 (datasheet)

Algunas características de la celda de carga y del HX711 son:

Característica	**HX-711**
Resolución	ADC 24 bits
Alimentación	2.7 – 5 V
Consumo	< 1.5 mA
Velocidad de muestreo	10 SPS o 80 SPS

Tabla 1: Tabla de características del módulo amplificador HX711

**SPS: Samples per Second* (muestras por segundo)

Característica	Celda de carga de 1 kg
Rango de Medición	0 - 1 kg
Precisión de Medición	0.03% del FS
Salida Nominal	1.0 ± 0.15 mV/V
Repetibilidad	0.03% FS
Impedancia de Entrada	1115 ± 10% Ω
Impedancia de Salida	1000 ± 10% Ω
Tasa de Sobrecarga Segura	150% FS
Tasa de Sobrecarga Final	200% FS
Rango de Temperatura	-20 a +60 °C
Voltaje de Funcionamiento	3VDC ~ 14 VDC
Material	Aleación de aluminio

Tabla 2: Tabla de características de la celda de carga de 1 kg

*FS: Fondo de Escala (1 kg)

Ilustración 8: Celda de carga de 1 kg empleada (establecimiento el Faro)

Ilustración 9: Módulo amplificador HX711(establecimiento el Faro)

5.4.2. Resultados de la identificación del modelo del sistema de propulsión.

Las relaciones obtenidas son válidas para nuestro sistema de propulsión, conformado por los siguientes componentes:

- Motor BLDC A2212/10T de 1400 KV
- ESC XDD de 30 A
- Hélice APC 7x5 E
- Batería de 3 celdas de Li-Po, de tensión nominal 11.1 V, cargada a 12.6 V

A continuación, se grafican las relaciones obtenidas. Los datos completos pueden encontrarse en el anexo:

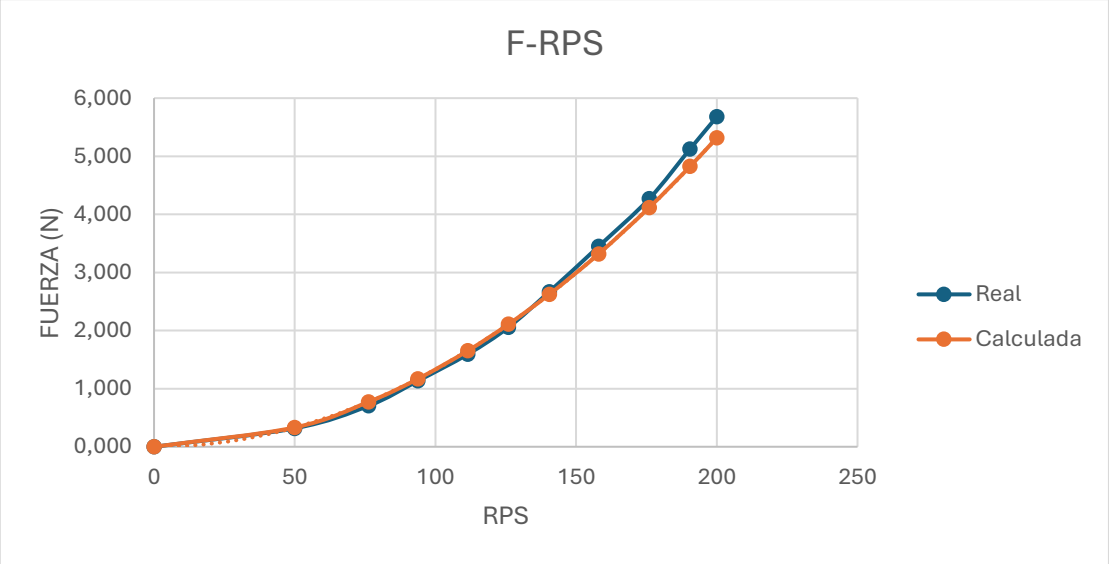


Ilustración 10: Gráfica. En eje y la fuerza generada por el rotor [N] y en eje x la velocidad de rotación [rps]

Esta curva muestra como la fuerza de empuje generada varía con la velocidad de giro del rotor. La relación es cuadrática y coincide bastante bien con la ecuación teórica que describe el comportamiento de la hélice. La curva difiere a velocidades altas, pues el coeficiente de sustentación (que nosotros asumimos constante al calcular la media de todos los valores de este coeficiente) no es constante, sino que depende de otras variables aerodinámicas.

La media de todos los $\boldsymbol{K_f}$ resulta en:

$$K_f = 1.3294 \cdot 10^{-4} \qquad [\frac{N}{rps^2}]$$

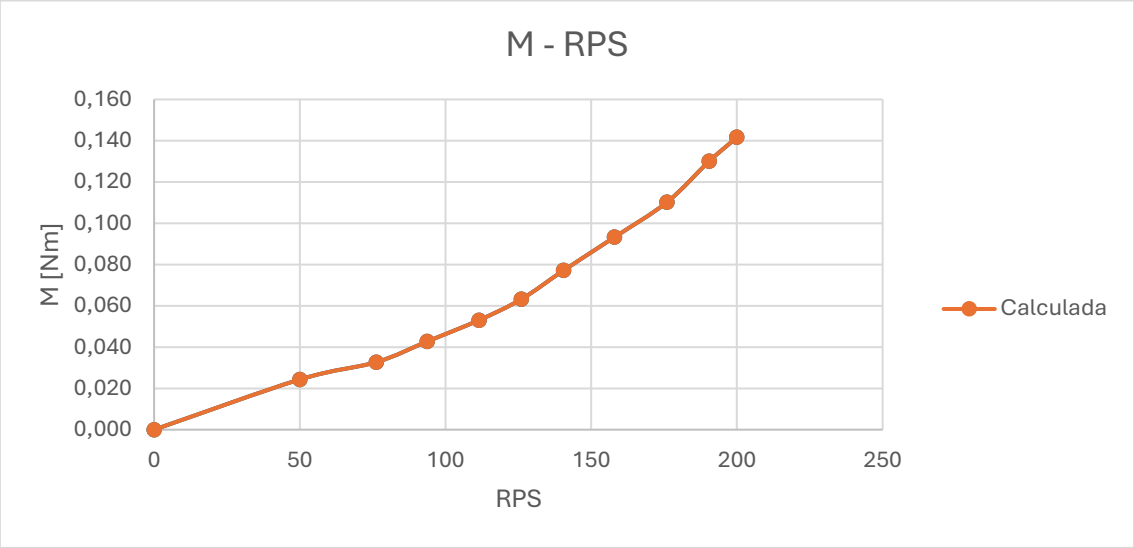


Ilustración 11:Gráfica. En eje y el par estimado generado por el rotor [Nm] y en eje x la velocidad de rotación [rps]

Para estimar el par generado por el rotor, emplearemos la siguiente ecuación que relaciona el par con el rendimiento del sistema y el consumo:

$$P = M \cdot \omega \cdot \eta$$

Entendiendo el rendimiento máximo del motor (estimado en un 80%):

$$M = \frac{V \cdot I}{\omega \cdot 0.8}$$

Si graficamos esta curva para los valores de $\omega, V\ e\ I$ resulta en la gráfica superior.
Observamos que la relación entre el par generado y la velocidad de giro aplicada es cuadrática.

Al igual que antes obtenemos la constante de arrastre a partir de las medias de las constantes de arrastre en cada medida:
La media de todos los $\boldsymbol{K_q}$ resulta en:

$$K_q = 4.67946 \cdot 10^{-6} \quad \frac{Nm}{rps^2}$$

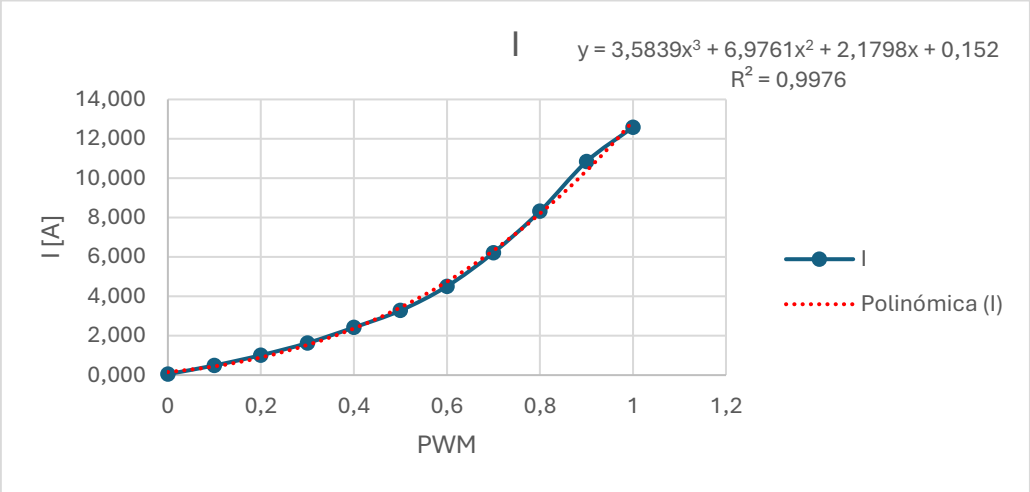


Ilustración 12: Gráfica. En eje y la corriente consumida por el rotor [A] y en eje x el comando PWM aplicado [0-100%]

Esta curva indica cómo varía la corriente eléctrica consumida por el motor con el ancho de pulso PWM en porcentaje. La curva se puede ajustar a una expresión polinómica de segundo o tercer orden, sin perder demasiada precisión.

$$I = 3{,}5839\omega^3 + 6{,}9761\omega^2 + 2{,}1798\omega + 0{,}152$$

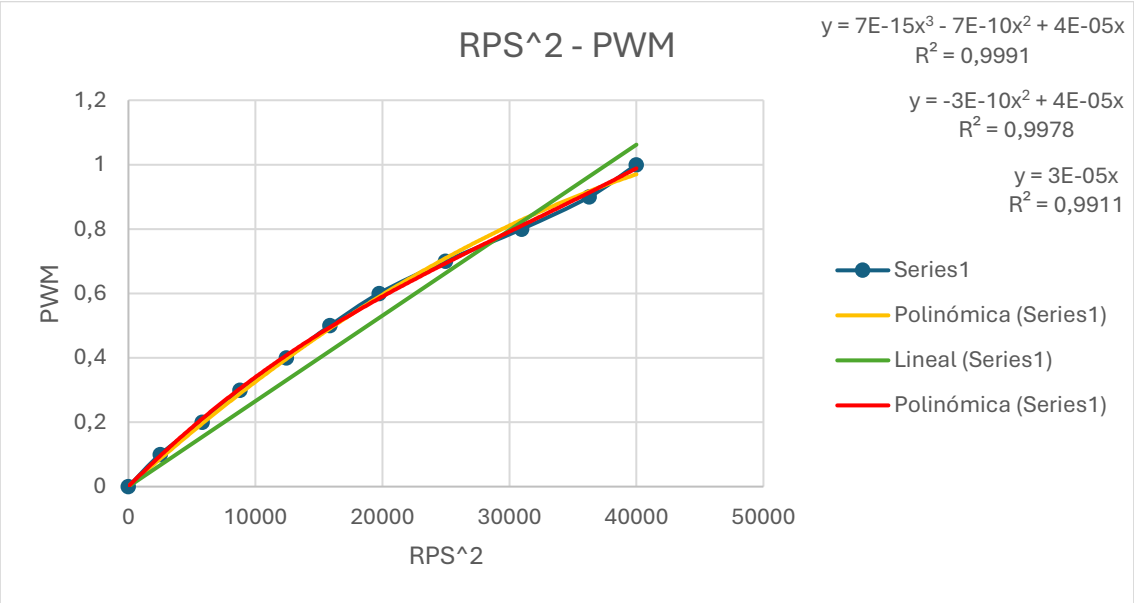


Ilustración 13: Gráfica. En eje y el comando PWM [0-100%] y en eje x la velocidad de rotación al cuadrado [rps^2]

La curva superior muestra la relación entre el % de PWM y la velocidad de giro al cuadrado. De esta forma podemos obtener una relación directa entre % PWM y RPS^2.

$PWM_i = 7 \cdot 10^{-15}(\omega_i^2)^3 - 7 \cdot 10^{-10}(\omega_i^2)^2 + 4 \cdot 10^{-5}(\omega_i^2)$ (error tiende a 0)
$PWM_i = -3 \cdot 10^{-10}(\omega_i^2)^2 + 4 \cdot 10^{-5}(\omega_i^2)$ (error aceptable)
$PWM_i = 3 \cdot 10^{-5}(\omega_i^2)$ (error elevado)

A partir del espectrograma de la señal podemos conocer el tiempo transitorio del sistema en cambiar de una velocidad a otra.

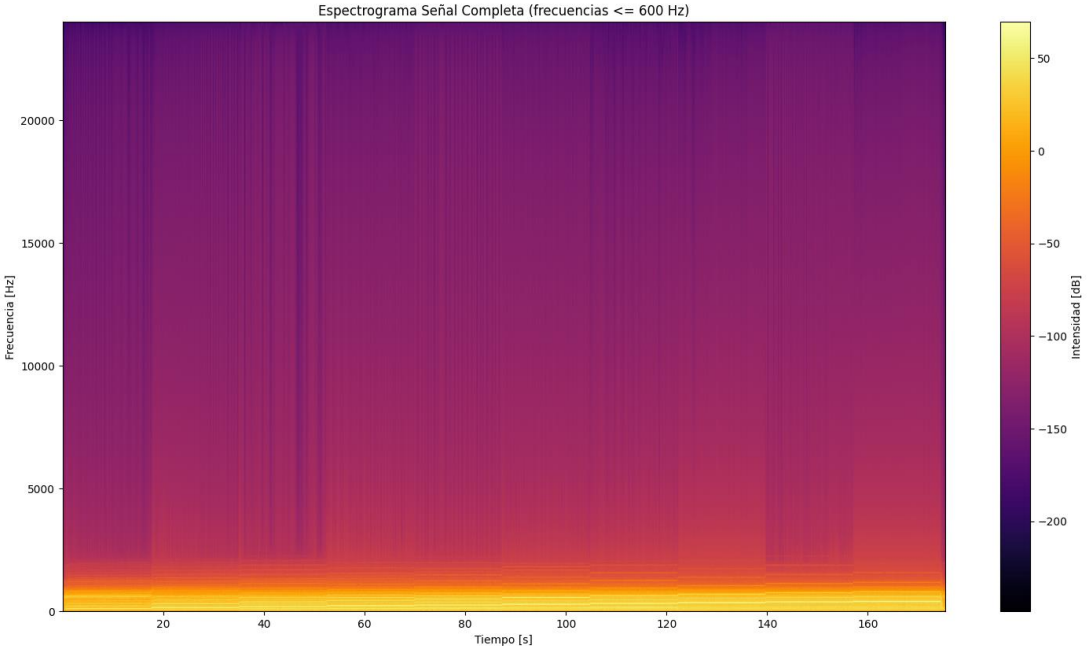


Ilustración 14: Espectrograma de la señal de audio grabada en el banco de pruebas generada por el giro del motor

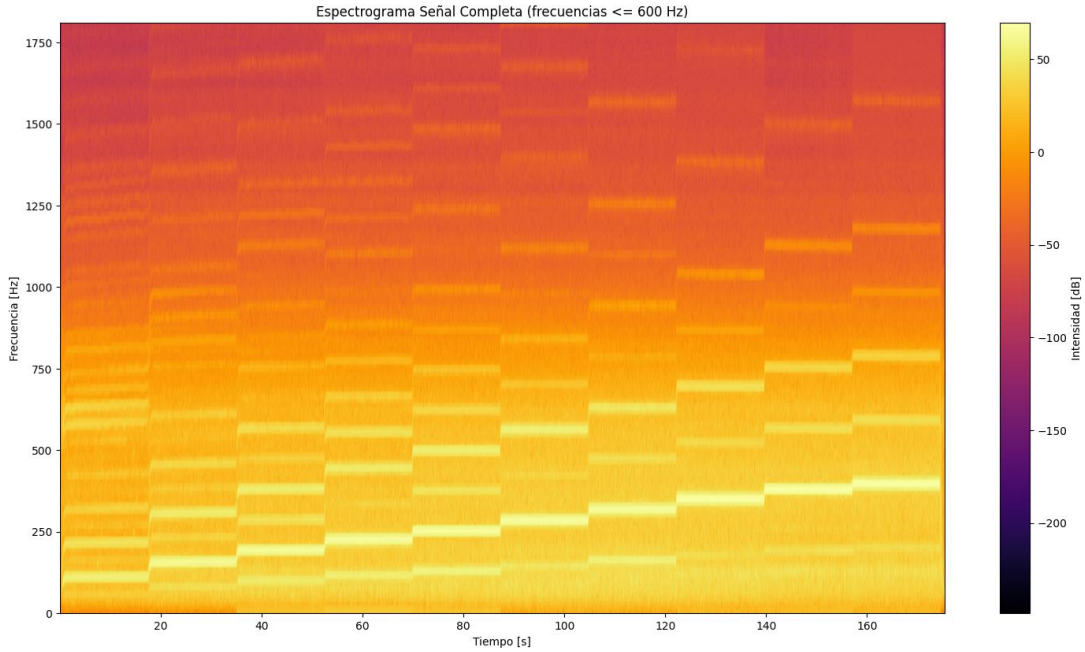


Ilustración 15: Espectrograma de la señal de audio grabada (detalle)

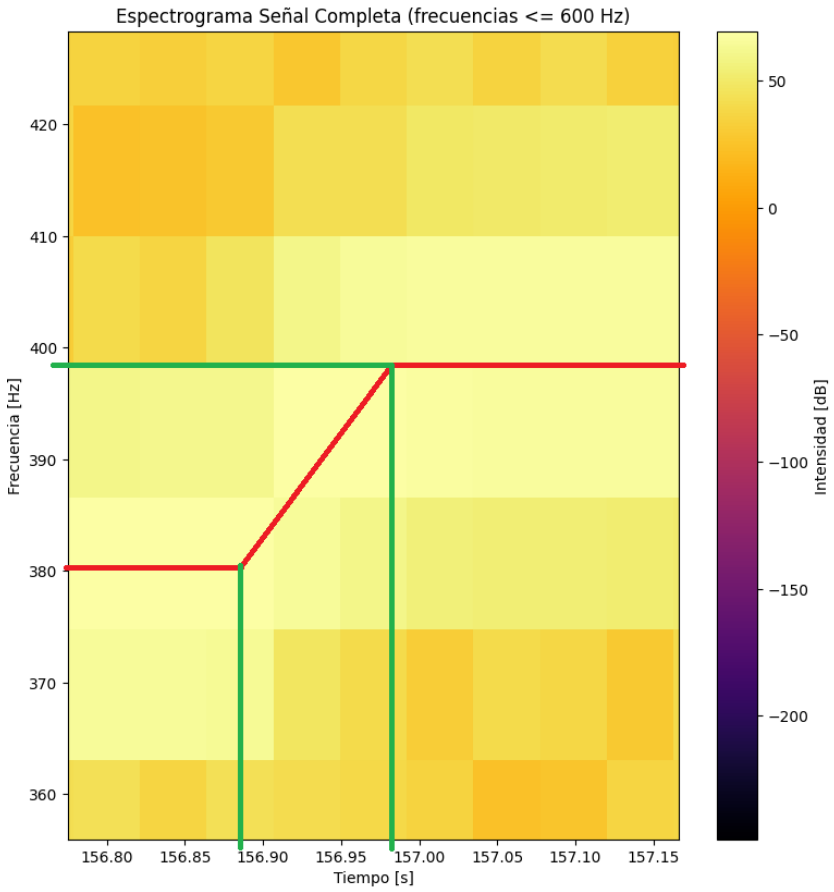


Ilustración 16: Detalle de la estimación del tiempo transitorio del salto de velocidades correspondiente a comandos PWM de 1900 µs a 2000 µs

En la figura se aprecia el último periodo de $1900\ \mu s - 2000\ \mu s$ (el resto se adjunta en el anexo) y las referencias aproximadas empleadas para estimar el tiempo transitorio de este salto de velocidades.

Para calcular la constante de tiempo se obtuvo la media de las constantes de tiempo medidas en cada cambio de velocidad, desde $1000\ \mu s - 2000\ \mu s$, cada $100\ \mu s$
Media de las constantes de tiempo estimadas:

$$4\tau = 0.13\ s \quad \rightarrow \quad \tau \approx 0.03$$

Idealmente sabemos que la velocidad máxima de giro será:

$$1400\ KV\ \cdot 11{,}1V = 15.540\ rpm = 259\ Hz$$

La frecuencia del sonido a máxima velocidad sería cercana (y como máximo) a 518 Hz (ya que la hélice acoplada al eje del motor consta de 2 aspas).

Debido a las pérdidas, la velocidad máxima que puede alcanzar el motor será menor, y por tanto su frecuencia sonora también. En el espectrograma que la frecuencia más cercana a la calculada es 400 Hz. Podemos estimar así la velocidad de giro máxima:

$$\frac{400\ Hz}{2} = 200\ Hz = 12000\ rpm$$

Finalmente podemos expresar la dinámica del motor DC reducida a primer orden como:

$$\omega(s) = \frac{1}{\tau_m s + 1} \cdot \frac{\omega}{s} \rightarrow \omega(s) = \frac{1}{0.03s + 1} \cdot \frac{\omega}{s}$$

Donde la fuerza y par generados por la hélice se definen como:

$$F = F_1 + F_2 + F_3 + F_4 = K_f \cdot (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$$
$$T_{1X} = L(F_1 - F_2 - F_3 + F_4) = L \cdot K_f \cdot (\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2)$$
$$T_{2Y} = L(F_1 + F_2 - F_3 - F_4) = L \cdot K_f \cdot (\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$$
$$T_{3Z} = M_1 - M_2 + M_3 - M_4 = K_q \cdot (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)$$

O de forma matricial como:

$$\begin{pmatrix} F \\ T_{1X} \\ T_{2Y} \\ T_{3Z} \end{pmatrix} = \begin{pmatrix} K_F & K_F & K_F & K_F \\ LK_F & -LK_F & -LK_F & LK_F \\ LK_F & LK_F & -LK_F & -LK_F \\ K_M & -K_M & K_M & -K_M \end{pmatrix} \cdot \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix}$$

Donde:

$$K_f = 1.3294 \cdot 10^{-4} \quad [\frac{N}{rps^2}] \qquad K_q = 4.67946 \cdot 10^{-6} \quad \frac{Nm}{rps^2}$$

Y en función de la relación entre %PWM y ω_i^2 escogida:

$PWM_i = 7 \cdot 10^{-15}(\omega_i^2)^3 - 7 \cdot 10^{-10}(\omega_i^2)^2 + 4 \cdot 10^{-5}(\omega_i^2)$ (error tiende a 0)
$PWM_i = -3 \cdot 10^{-10}(\omega_i^2)^2 + 4 \cdot 10^{-5}(\omega_i^2)$ (error aceptable)
$PWM_i = 3 \cdot 10^{-5}(\omega_i^2)$ (error elevado)

6. Diseño del cuadricóptero. Componentes

6.1. Sensores

6.1.1. IMU

Una unidad de medida inercial (IMU) es el sensor más importante de cualquier UAV.
Se trata de un elemento electrónico capaz de determinar las velocidades angulares, aceleraciones lineales y orientación del cuerpo sobre el que se ubique. Existen diferentes tipos de IMU en función de los ejes que pueda medir:

- IMU de 3 ejes (3DoF): incluyen acelerómetro (generalmente):
  - Permite medir las aceleraciones en los 3 ejes del espacio
  - Permite estimar los ángulos de alabeo y cabeceo.

- IMU de 6 ejes (6DoF): incluyen acelerómetro y giroscopio.
  - Permite medir las aceleraciones en los 3 ejes del espacio
  - Permite medir las velocidades angulares en los 3 ejes del espacio
  - Permite calcular de forma precisa los ángulos de alabeo y cabeceo.
  - Mediante algoritmos de filtrado como Madgwick, Mahony o Kalman se puede afinar las lecturas de cabeceo y alabeo y/o estimar la guiñada.

- IMU de 9 ejes (9DoF): incluyen acelerómetro, giroscopio y magnetómetro.
  - Permite medir las aceleraciones en los 3 ejes del espacio
  - Permite medir las velocidades angulares en los 3 ejes del espacio
  - Permite calcular de forma precisa los ángulos de alabeo, cabeceo y guiñada.
  - Mediante algoritmos de filtrado como el filtro de Kalman se puede afinar las lecturas de cabeceo, alabeo y guiñada

Una IMU completa, define perfectamente la inclinación y orientación del cuerpo sobre el que se ubica en los 3 ejes del espacio.

Los acelerómetros y giroscopios electrónicos modernos basan su principio de funcionamiento en la tecnología MEMS (*MicroelectroMechanical System*). Esta tecnología consiste en sistemas electromecánicos microscópicos en los cuales una microscópica estructura mecánica sujeta por resortes puede moverse debido a la acción de alguna fuerza externa y volver a su posición de equilibrio. Este movimiento se aprovecha para hacer variar el valor de alguna magnitud electrónica. Esta tecnología en auge se emplea tanto en microsensores como en actuadores microscópicos [10] [11].

Los acelerómetros y giroscopios MEMS más usuales son de tipo capacitivo. En este tipo de sensores varía la capacitancia debido a la variación de la distancia entre las placas o "dedos" que hacen de condensadores, al moverse la masa de prueba a la que están conectados.

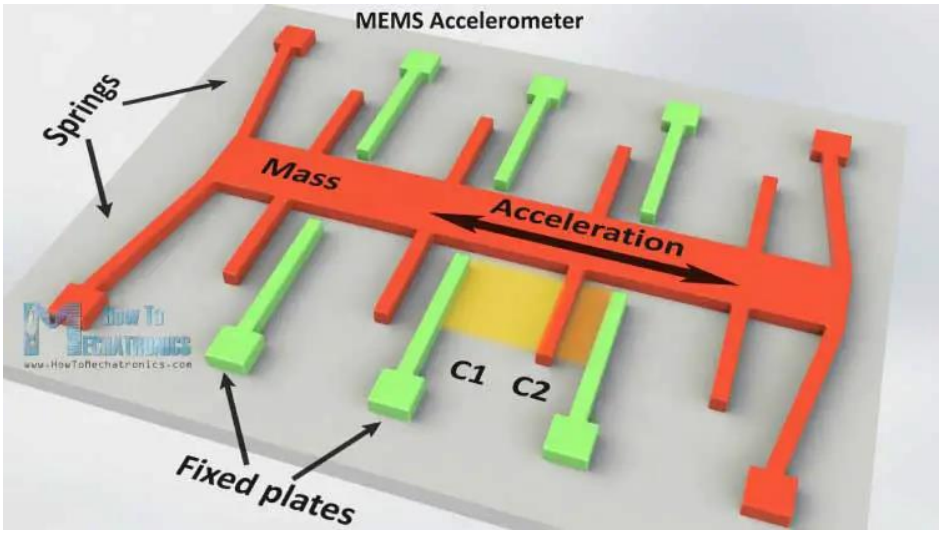


Ilustración 17: Esquema de funcionamiento de acelerómetro MEMS [https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/]

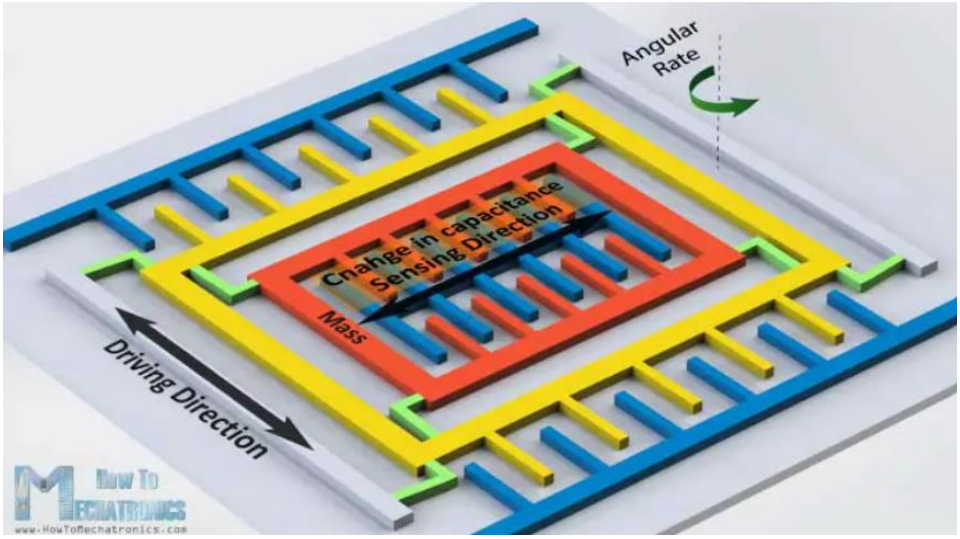


Ilustración 18: Esquema de funcionamiento de giroscopio MEMS [https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/]

Debido a su constitución, los acelerómetros introducen bastante ruido de medida en sus lecturas, mientras que el giroscopio muestra una deriva a lo largo del tiempo, pero proveen lecturas con poco ruido. Este es el motivo por el que es necesario aplicar filtros paso bajos o pasos altos a las lecturas y en algunos casos, aplicar algoritmos de predicción y corrección.

Las IMU completas incluyen también magnetómetro. Se constituyen por conductores especialmente sensibles a variaciones en al campo magnético que les afecta (magnetorresistencias). Estas magnetorresistencias se orientan en cada eje cartesiano, de tal manera que puede medirse el campo magnético en cada eje del espacio.

Son sensores muy sensibles a interferencias y a la presencia de metales ferromagnéticos cercanos, por lo que requieren de calibraciones previas, filtros y algoritmos correctivos.

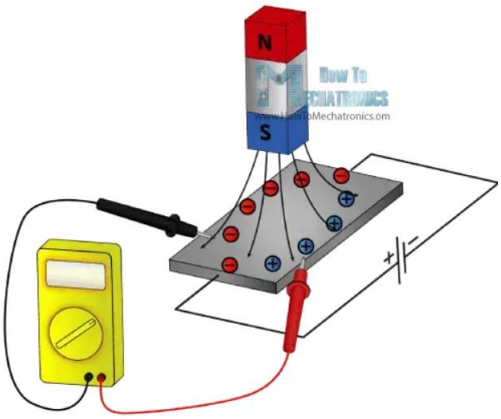
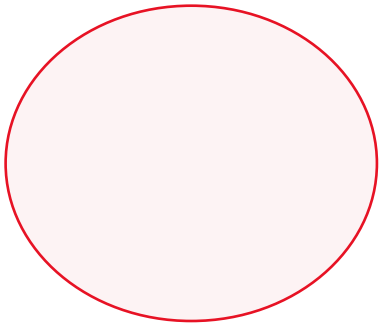


Ilustración 19: Esquema de funcionamiento de magnetómetro [https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/]

Es de suma importancia combinar las lecturas de estos sensores para obtener lecturas precisas de la orientación del cuerpo y minimizar ruidos, interferencias y derivas mediante algoritmos de predicción y corrección como el que se presenta en la ilustración 20 [11].

En nuestro caso nos interesa obtener lecturas precisas de los ángulos de cabeceo y de alabeo, además de la velocidad angular de guiñada, por lo que aplicaremos la parte superior del método propuesto. No necesitamos la guiñada, ya que controlaremos la velocidad angular de guiñada y no el ángulo de guiñada.



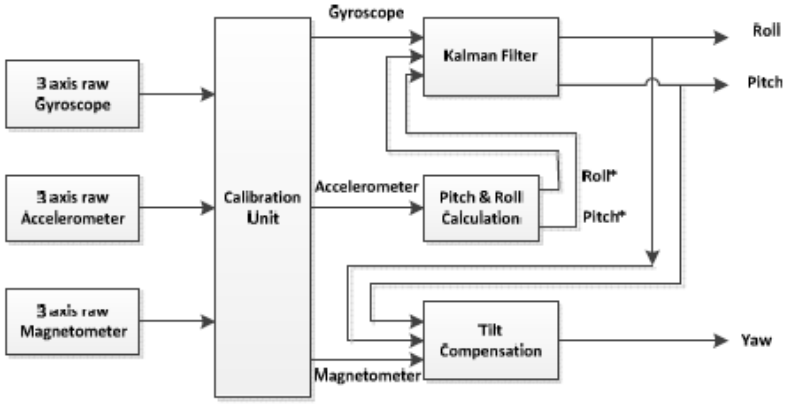


Ilustración 20:Algoritmo de fusión 3D para sensores MEMS [11]

Algunas opciones disponibles en el mercado son:

Característica	**MPU-6050**	**MPU-9250**	**LSM6DOX**
Grados de libertad	6 ejes	9 ejes	6 ejes
Rango del Giroscopio	±250 º/s ±500 º/s ±1000 º/s ±2000 º/s	±250 º/s ±500 º/s ±1000 º/s ±2000 º/s	±125 º/s ±250 º/s ±500 º/s ±1000 º/s ±2000 º/s
Resolución del Giroscopio	16 bits	16 bits	16 bits
Sensibilidad del Giroscopio	131 LSB/(º/s) 65.5 LSB/(º/s) 32.8 LSB/(º/s) 16.4 LSB/(º/s)	131 LSB/(º/s) 65.5 LSB/(º/s) 32.8 LSB/(º/s) 16.4 LSB/(º/s)	4.375 mdps/LSB 8.75 mdps/LSB 17.5 mdps/LSB 35 mdps/LSB 70 mdps/LSB
Rango del acelerómetro	±2 g ±4 g ±8 g ±16 g	±2 g ±4 g ±8 g ±16 g	±2 g ±4 g ±8 g ±16 g
Resolución del Acelerómetro	16 bits	16 bits	16 bits
Sensibilidad del Acelerómetro	16,384 LSB/g 8,192 LSB/g 4,096 LSB/g 2,048 LSB/g	16,384 LSB/g 8,192 LSB/g 4,096 LSB/g 2,048 LSB/g	0,061 mg/LSB 0,122 mg/LSB 0,244 mg/LSB 0,488 mg/LSB
Rango del Magnetómetro	N/A	±4800 µT	N/A
		13 bits	N/A

Característica	MPU-6050	MPU-9250	LSM6DOX
Resolución del Magnetómetro	N/A		
Sensibilidad del Magnetómetro	N/A	0.6 µT/LSB	N/
Comunicación	I2C	I2C	SPI/I2C
Consumos	Giroscopio: 3.6 mA Acelerómetro: 0.5 mA	Giroscopio: 3.2 mA Acelerómetro: 0.45 mA Magnetómetro: 0.28 mA	Giroscopio: 0.65 mA Acelerómetro: 0.42 mA
Alimentación	3.3V - 5V	3.3V - 5V	1.71V - 3.6V
Otros	DMP FIFO Filtros	DMP FIFO Filtros	Machine Learning FIFO Filtros
Precio	3 - 10 €	7 - 20 €	5 - 15 €

Tabla 3: Tabla de características de diferentes IMU

Haremos uso del LSM6DOX. Se trata de una IMU de 6 ejes (acelerómetro y giroscopio) programable, que integra varios núcleos con algoritmos de *Maching Learning* para la detección de patrones de movimiento.
De igual manera incorpora filtros programables, tanto como paso alto como filtros paso bajo, tanto para el giroscopio como para el acelerómetro.

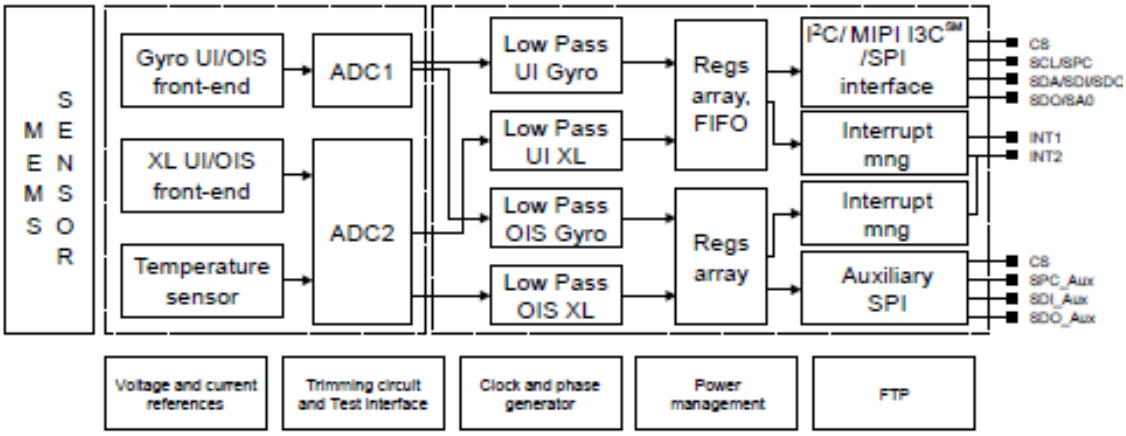


Ilustración 21: Diagrama de bloques del LSM6DOX [datasheet]

Emplearemos la librería que Arduino ofrece para este sensor. Sin embargo, se realizarán algunos cambios dentro de esta para obtener datos filtrados, con

rangos de medición más adecuados, con menos ruido de medida y un muestreo más adecuado.

6.1.2. Altímetro

Un altímetro se trata de un sensor capaz de medir, de forma directa o indirecta, la altura de un cuerpo respecto a una superficie, nivel del mar, etc.

Es necesario para estimar la altura de nuestro UAV y la velocidad en Z.

Existen diferentes formas de medir la altitud:

- **Sensores de Ultrasonidos**: Estos sensores miden la altura relativa a la superficie más cercana. Funcionan emitiendo pulsos de ultrasonido por un emisor y recibiéndolos por un receptor. Sabiendo el tiempo transcurrido desde que se envían los pulsos hasta que se reciben, es posible calcular la distancia.

- **Barómetros:** Estos sensores miden la presión atmosférica. Como la presión atmosférica es función de la altura es posible calcular la altura de un cuerpo respecto del nivel del mar.

- **Sensores láser:** Estos sensores funcionan de forma análoga a los sensores de ultrasonidos. Son mucho más precisos y rápidos que estos, ya que la luz láser viaja a la velocidad de la luz.

Característica	BMP 280	VL53L0X	HC-SR04
Tipo	Barómetro	Láser	Ultrasonido
Rango presiones / distancia	300 - 1100 hPa	50 -1200 mm (2000 mm en condiciones óptimas)	2 - 400 cm
Resolución presiones / distancia	0.16 Pa	…	3 mm
Precisión presiones / distancia	±1 hPa	±3 - 5 %	±0.3 cm
Rango temperaturas	-40 a +85 °C	-20 a 70 ºC	N/A
Comunicación	I2C/SPI	I2C	Digital
Consumo	2.7 µA	5 µA	15 mA
Alimentación	1.71 V - 3.6 V	2,6 V – 5 V	5 V
Dimensiones	2.0 x 2.5 mm	10.5 x 13.3 mm	45 x 20 x 15mm
Coste	0 - 5 €	10 – 15 €	1 - 5 €

Tabla 4: Tabla de características de altímetros

Elegiremos el sensor láser VL53L0X debido a su precisión y rapidez, limitando la altura de vuelo a 1.2 m. Como alternativa sería recomendable el barómetro, pues no depende de la altura, pero su precisión es limitada.

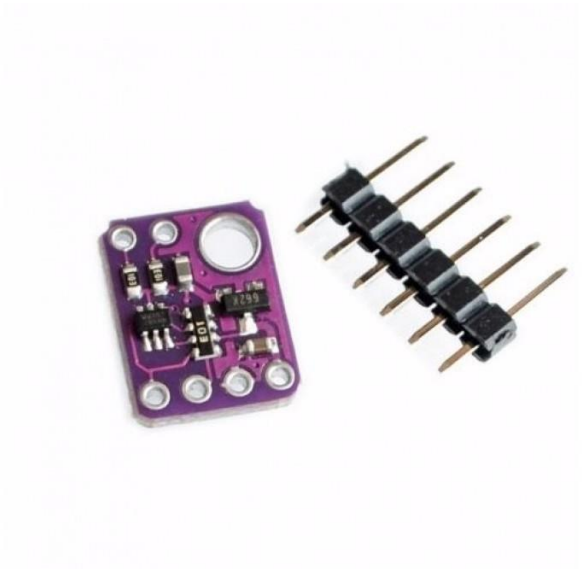


Ilustración 22: Módulo láser VL53L0X (establecimiento el Faro)

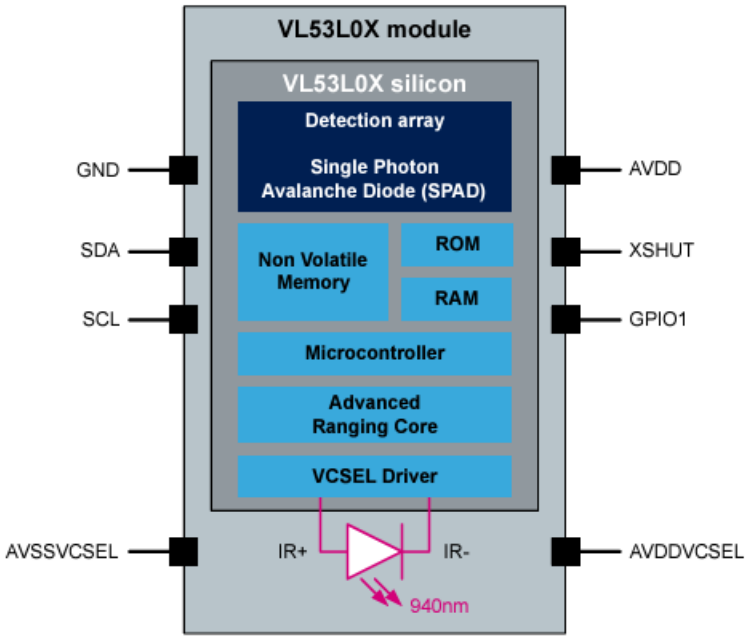


Ilustración 23: Diagrama de bloques del sensor VL53L0X (datasheet)

6.2. Actuadores

6.2.1. Motores

Los motores serán los elementos encargados de producir la velocidad de giro para que las hélices puedan transformarla en fuerza de empuje.
La elección de estos, por tanto, dependerá del peso a mover y de la hélice empleada. Se han de tener en cuenta algunos requerimientos antes de elegir cualquier motor para UAV:

- **KV (constante de velocidad):** este parámetro representa la velocidad angular máxima que puede alcanzar un motor por cada voltio aplicado, sin tener en cuenta rozamientos de ningún tipo.

$$\omega = KV \cdot V$$

  - **KV elevado:** Estos motores pueden girar a mayores velocidades y están constituidos por menos devanados. Este tipo de motores no requieren de hélices tan grandes para generar la fuerza de sustentación necesaria para desplazar el dron, ya que giran más rápido. Se emplean en drones pequeños.

  - **KV pequeño:** Estos motores pueden girar a velocidades menores y están constituidos por más devanados. Este tipo de motores requieren de hélices más grandes para generar la fuerza de sustentación necesaria para desplazar el dron, ya que giran más despacio. Se emplean en drones grandes o que cargan mucho peso.

- **Relación peso / empuje:** Se trata de una medida del rendimiento del motor. Relaciona el empuje capaz de ejercer un motor con relación a su propio peso.

$$RPE = \frac{P}{F}$$

- **Corriente vacío:** Corriente mínima a aplicar al motor sin carga para vencer la fricción e inercias iniciales.

- **Máxima corriente:** Esta es la corriente o potencia máxima que puede soportar el motor.

- **Resistencia de los devanados:** Si bien es pequeña, es especialmente relevante cuando circulan corrientes elevadas, ya que provocarán el calentamiento del motor.

- **Tamaño:** Viene representado por 4 dígitos de la forma XXYY. Las dimensiones del estátor están directamente relacionadas con las

características del motor. Un mayor tamaño del estátor se relaciona con mayor potencia y menor KV. Los motores más grandes son adecuados para drones más grandes y pesados.

- Los 2 primeros dígitos indican el diámetro del estátor en milímetros.
- Los 2 últimos dígitos indican la altura del estátor en milímetros.

Característica	Motor DC	Motor BLDC
Mantenimiento	Elevado	Mínimo
Rendimiento	Bueno	Excelente
Conmutación	Mecánica (Escobillas) Dentro del motor	Electrónica (ESC) Fuera del motor
Control	Sencillo Circuito requerido: Puente H	Complejo Circuito requerido: ESC
Precio	Bajo	Elevado
Durabilidad	Menor, debido al desgaste de escobillas	Mayor, sin partes en contacto
Eficiencia Energética	Menor	Mayor
Par Motor	Elevado a bajas velocidades	Consistente en un rango amplio de velocidades
Aplicaciones comunes	Drones pequeños	Drones medios / grandes
Respuesta dinámica	Rápida	Rápida

Tabla 5: Tabla comparativa de características de motores DC y BLDC [12]

Optaremos por los motores BLDC. Son motores fiables que requieren poco mantenimiento y permiten hacer ensayos sin peligro de desgaste temprano, con un buen rendimiento (superior al 80 % en algunos casos).

Opciones disponibles:

- Motor XXD A2212/10T de 1400 KV

- Motor XXD A2212/10T de 1000 KV

KV	Batería	Máximas RPM
1400	LiPo: 2s (7,4 V)	10360 rpm
1400	LiPo: 3s (11,1 V)	15540 rpm
1000	LiPo: 2s (7,4 V)	7400 rpm
1000	LiPo: 3s (11,1 V)	11100 rpm

Tabla 6: Tabla de velocidades de rotación en función de los KV del motor y de la batería de Li-Po

Optamos por el motor A2212/10T de 1400 KV: permite alcanzar mayor velocidad de giro que el de 1000 KV empleando la misma tensión, haciendo algo menor el tamaño de la hélice necesaria.

Ilustración 24: Motor BLDC A2212/10T de 1400 KV (establecimiento el Faro)

Característica	Descripción / Valor
KV	1400
Corriente para rendimiento > 75%	6 - 12 A
Corriente sin carga	0,5 - 0,7 A
Corriente máxima	16 A durante 60 s
ESC recomendado	30 A
Resistencia devanado	65 mΩ
Inductancia	…
Polos	14
Potencia máxima	180 W
Eficiencia	75 % - 80 %
Celdas recomendadas de la batería	Li-Po: 2s – 3s Ni-Cd: 6s - 10s
Inercia	…
Peso	36 - 47 g

Tabla 7: Tabla de características del motor A2212/10T de 1400 KV

6.2.2. ESC

El ESC (Controlador Electrónico de Velocidad) es un dispositivo electrónico que permite efectuar el control en motores sin escobillas. Se encarga de generar las secuencias PWM para activar o desactivar cada una de las fases del motor y regular la velocidad de giro de éste. El dispositivo consta de:

- **Entrada de Alimentación DC.**
- **Entrada de Control PWM:** A través de esta entrada regulamos la velocidad del motor. Las características de la PWM que admite dependen de cada ESC.
- **BEC (Circuito eliminador de batería):** Se trata de un pequeño circuito que incorpora un regulador a 5V. Permite alimentar otros componentes del dron sin necesidad de incluir otras baterías adicionales.
- **Microprocesador** se encarga de regular la velocidad de giro del motor a partir de la PWM introducida.
- **Inversor**: Se trata de un circuito electrónico capaz de convertir la corriente continua a trifásica gracias a la activación y desactivación de transistores siguiendo cierta secuencia marcada por el microprocesador.
- **Salidas (3 Fases)** que alimentan los bobinados del motor de acuerdo con la señal de control de entrada.

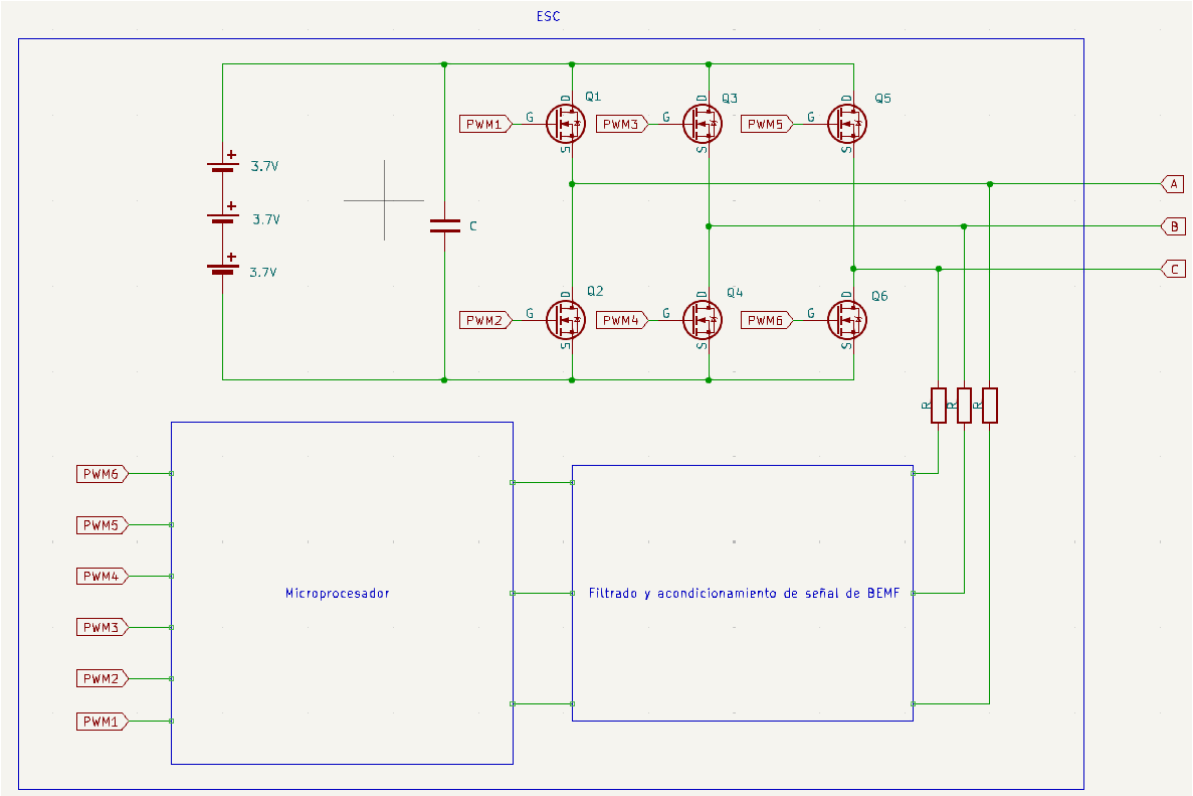


Ilustración 25: Esquema de las principales partes de un ESC (elaboración propia)

Tal y como se mencionó, en la mayoría de los ESC, la secuencia de conmutación se controla mediante un microprocesador, que “abre” o “cierra” cada uno de los transistores en función de la señal de realimentación recibida y de la velocidad consignada. Esta señal de realimentación es la fuerza contraelectromotriz (BEMF) producida por el motor al girar y depende de la posición del rotor en cada instante de tiempo.

De esta manera el microprocesador conoce en cada instante la posición del rotor y generar así las secuencias y el cambio de estado de los transistores.

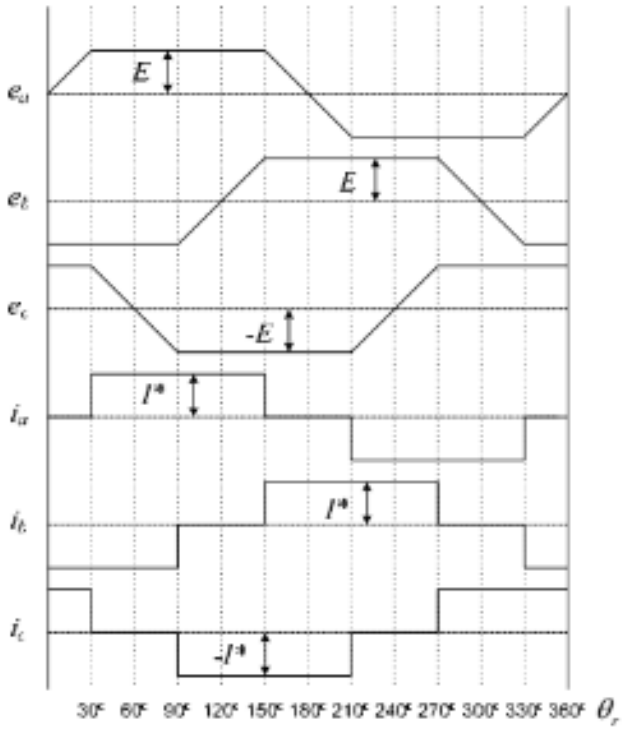


Ilustración 26: Gráfica de las tensiones de BEMF y corrientes de fase respecto a la posición angular geométrica del rotor [6]

La elección del ESC depende en gran medida del motor elegido, pues debe de soportar con holgura la intensidad admisible por éste y poder conmutar los transistores lo suficientemente rápido como para alcanzar velocidades de giro elevadas.

Requerimientos:

- Intensidad de 30 A
- Tensión 7,4 - 14,8 V
- Circuito BEC

Elegiremos el ESC XXD HW30A:

Característica	Descripción / Valor
Corriente máxima	30 A
BEC	Corriente máxima: 2 A Tensión: 5V
Tensión de entrada	2 - 3 celdas Li-Po
Resistencia interna	0,0050 Ω
Número de transistores FET	12
Temperatura Máxima	110 º C
Frecuencia de funcionamiento	8 kHz
Programas	Programa de Calibración Programa de Configuración

Tabla 8: Tabla de características del ESC XDD 30ª

Ilustración 27: Controlador electrónico de velocidad ESC (establecimiento el Faro)

6.2.3. Hélices

Son los elementos que transforman la velocidad de giro del motor en fuerza empuje. Son elementos complejos estudiados por la aerodinámica. Antes de elegir las hélices correctas, es necesario familiarizarse con algunos conceptos y nomenclatura de estas.

Nomenclatura de las hélices:

- Las hélices se nombran de acuerdo con su diámetro y paso, mediante dos cifras separadas por el carácter 'x', en pulgadas. De esta manera una hélice 8 x 5 indica que se trata de una hélice de diámetro 8 pulgadas y paso de 5 pulgadas.

- Pueden tener letras como sufijo, indicando el tipo o característica de esta:

Tipo de hélice	Uso / Característica
E	"Thin-Electric" o "Electric": Hélices diseñadas para emplearse con motores eléctricos
F	"Fold": Hélices plegables
MR	"MultiRotor" Similares a las E, pero están especialmente diseñadas para motores eléctricos de multirotores
R	"Reversible ESC" Hélices para girar en doble sentido solo para motores eléctricos
SF	"Slow Fly", Hélices diseñadas para generar una gran sustentación a bajas velocidad de giro. Para motores eléctricos de bajo KV
CW	"Clockwise" Hélices diseñadas para avanzar girando en sentido horario
CCW	"Counter Clockwise" Hélices diseñadas para avanzar girando en sentido antihorario

Tabla 9: Tabla de sufijos de hélices

Las características aerodinámicas de la hélice están ligadas a la geometría de esta y a parámetros propios de la mecánica de fluidos. Es necesario un estudio aerodinámico si se desea profundizar más sobre la eficiencia y el diseño de hélices.

Cada aspa de la hélice se compone de varios perfiles alares inclinados en diferentes ángulos a lo largo de esta para optimizar su rendimiento. Por lo general los ángulos de ataque de los perfiles cercanos al centro de la hélice son mayores ya que la velocidad lineal de la hélice es menor en esa zona por lo que para generar sustentación se necesita mayor ángulo. Para secciones más externas el ángulo de ataque es menor, ya que su velocidad lineal en mayor y pueden generar sustentación con ángulos de ataque más pequeños.

Muchas hélices están conformadas por uno o varios perfiles alares normalizados, siendo los más comunes los perfiles normalizados NACA y otros como los Clark y los E. La forma de estos perfiles alares determinará los coeficientes de sustentación y arrastre (entre otros) de la hélice. Estas características se pueden estimar mediante ensayos además de los datos que pueda proveer el fabricante.

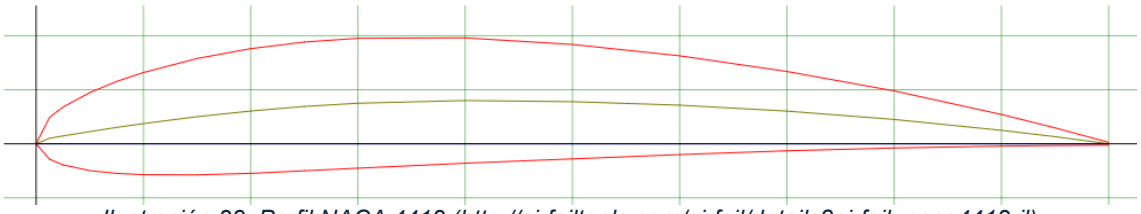


Ilustración 28: Perfil NACA 4412 (http://airfoiltools.com/airfoil/details?airfoil=naca4412-il)

En esta ocasión elegiremos hélices de la empresa APC. Tienen gran variedad de hélices y proporcionan datos tanto de su geometría cómo de rendimiento, empuje, par, etc. Gracias a estos parámetros podemos elegir aquella hélice que mejor se adecúe a nuestras necesidades y objetivos

Algunos parámetros que describen geométricamente la hélice son:

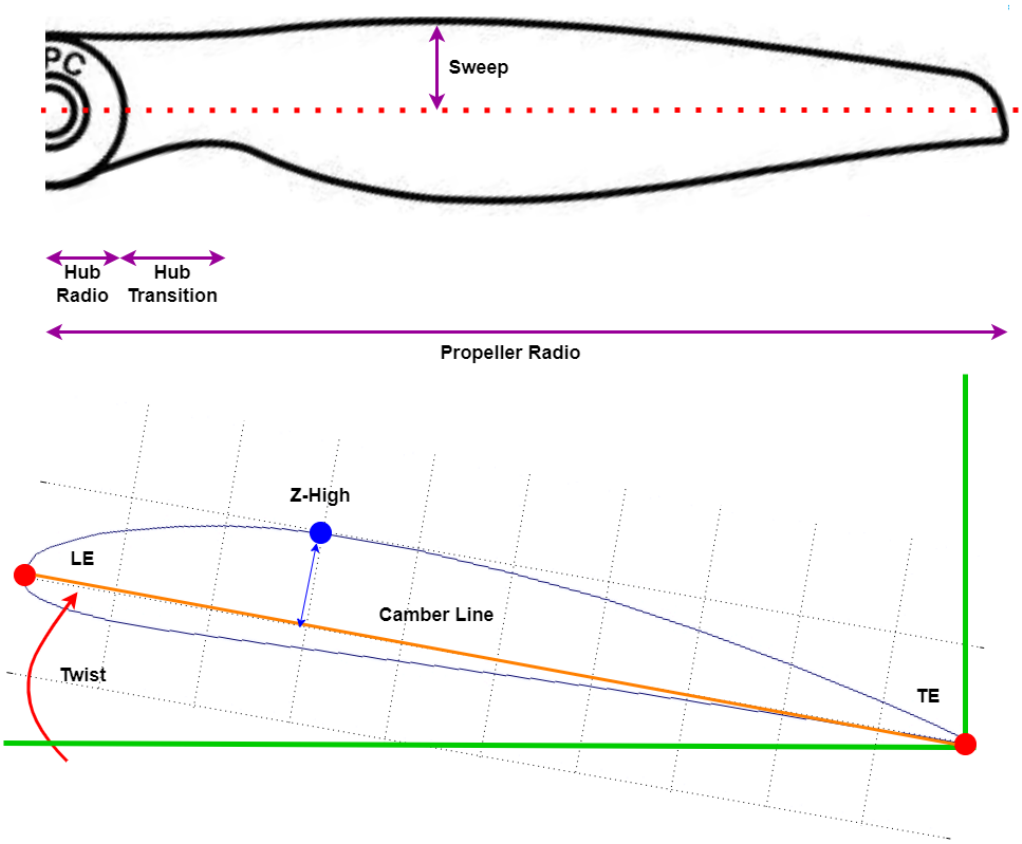


Ilustración 29: Esquema con algunas de las partes que caracterizan geométricamente a una hélice (elaboración propia)

- **Cubo "*Hub*":** Es la parte central de la hélice. Une las aspas con el eje del rotor.
- **Barrido "*Sweep*":** Inclinación de las aspas respecto al plano longitudinal de la hélice.
- **Paso geométrico "*Twist*":** Ángulo de las aspas de la hélice respecto al plano de rotación.
- **Línea de curvatura "*Camber Line*":** Línea recta que une el borde de ataque (LE) de la hélice con el opuesto (TE).
- **Cuerda "*Chord*":** Cada una de las secciones que conforma el aspa de la hélice.

El objetivo es poder levantar el cuadricóptero y moverlo cómodamente sin que el motor tenga que llegar al límite de sus capacidades.

Se estima el peso del cuadricóptero en un kilo, por lo que para mantener el cuadricóptero estático en vuelo será necesario una fuerza de:

$$P = m \cdot g = 1 \cdot 9.81 = 9.81\ N$$

Y por motor:

$$F_{motor} = \frac{P}{4} = \frac{9.81}{4} = 2.453\ N \qquad F_{motor} = \frac{2.453}{9.81} = 0.25\ kg = 250\ g$$

Como norma general, se recomienda escoger una hélice que, entre los 4 motores a velocidad máxima, puedan levantar 2 veces el peso del dron.

En base a nuestros requisitos y ensayos realizados (ver punto X), optaremos por las hélices APC 7 x 5 E.

Ilustración 30: Hélice APC 7x5E (tienda RC Innovations)

6.3. Controlador de vuelo

La elección del microcontrolador es crucial debido a las necesidades de procesamiento, conectividad y compatibilidad con los sensores y los actuadores. Algunos requisitos buscados son:

1. Procesamiento: procesador potente para manejar las tareas de control de vuelo, conectividad, lectura de sensores y manejo de actuadores en tiempo real.

2. Memoria: Cantidad de memoria suficiente para poder almacenar variables y algoritmos de control.

3. Conectividad: que permita el envío y recepción de datos en tiempo real.

4. Pines Entrada / Salida: Suficientes pines para conectar todos los componentes.

Algunas opciones disponibles son:

Placa	Microcontrolador	Frecuencia	Memoria	Comunicación	Conectividad	Pines
ESP32	Xtensa Dual-Core 32-bit LX6	240 MHz	ROM: 448 KB SRAM: 520 KB Flash: 4-16 MB	UART I2C SPI USB	Wi-Fi BLE LoRa	34 digitales 34 PWM 16 ADC 2 DAC
Arduino Nano RP2040	RP2040 Dual-Core ARM Cortex M0+	133 MHz	Flash: 16 MB SRAM: 264 KB	UART I2C SPI USB	Wi-Fi BLE	22 digitales 20 PWM 8 ADC
Raspberry Pi Pico	RP2040 Dual-Core ARM Cortex M0+	133 MHz	Flash: 16 MB SRAM: 264 KB	UART I2C SPI USB	Depende de la versión de la placa	26 digitales 16 PWM 3 ADC
Arduino Uno	ATmega328P	16 MHz	Flash: 32 KB SRAM: 2 KB EEPROM: 1 KB	UART I2C SPI USB	-	14 digitales 6 PWM 6 ADC
Arduino Mega 2560	ATmega2560	16 MHz	Flash: 256 KB SRAM: 8 KB EEPROM: 4 KB	UART I2C SPI USB	-	54 digitales 15 PWM 16 ADC
Teensy 4.0	ARM Cortex-M7	600 MHz	Flash: 2 MB SRAM: 1 MB	UART I2C SPI USB	-	40 digitales 31 PWM 14 ADC

Tabla 10: Tabla comparativa de microcontroladores

Optaremos por el Arduino Nano RP2040. Se trata de un microprocesador de tamaño reducido con excelentes prestaciones en cuanto a memoria, velocidad de procesamiento y conectividad. Además, integra el IMU LSDM6DOX mencionado.

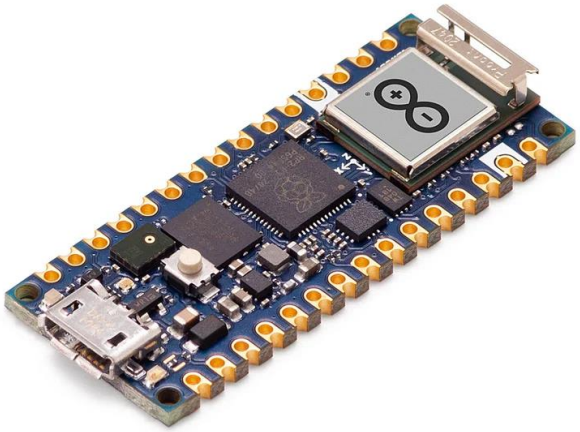


Ilustración 31:Arduino Nano RP2040 [https://store.arduino.cc/products/arduino-nano-rp2040-connect]

6.4. Chasis

El chasis será la estructura que contendrá y será la base de los componentes del cuadricóptero. Se plantea el diseño y la impresión 3D del mismo, siguiente una serie de requisitos que éste debe cumplir:

- **Ligereza:** Importante para mejorar la eficiencia y el tiempo de vuelo.
- **Resistencia a impactos:** que permita realizar pruebas y ensayos sin peligro de rotura.
- **Refrigeración:** Crucial para mantener temperaturas óptimas y prevenir el sobrecalentamiento de los componentes.
- **Material:** Al tratarse de un prototipo, barajamos las siguientes opciones:
    - **PLA:** Fácil de imprimir y económico, pero menos resistente al calor.
    - **ABS:** Más resistente y duradero además de mejor comportamiento a altas temperaturas.
    - **Fibra de Carbono:** Extremadamente ligera y muy resistente a impactos. Ideal para aplicaciones donde el peso y la resistencia son críticos, aunque es más costosa.
- **Dimensiones de la estructura:** La estructura se dimensiona de acuerdo con el diámetro de la hélice escogida minimizando el tamaño de esta, para obtener las mejores prestaciones [9]:

$$R = \frac{R_{max}}{sin(\frac{180º}{n_{rotores}})}$$

Sabiendo que el diámetro de la hélice escogida es de 7 pulgadas:

$$D_{hélice} = 7\ in = 17.78\ cm \ \rightarrow \ R_{hélice} = 3.5\ in = 8.89\ cm$$

Podemos calcular R_{max} según (7):

$$R_{max} = 1.05 \sim 1.2 \cdot R_{hélice}$$

$$R_{max} = 1.05 \cdot R_{hélice} = 1.05 \cdot 8.89 = 9.3345\ cm$$

$$R_{max} = 1.2 \cdot R_{hélice} = 1.2 \cdot 8.89 = 10.668\ cm$$

Con R = 9.3345 cm, el radio de la estructura sería:

$$R = \frac{R_{max}}{sin(\frac{180º}{n_{rotores}})} = \frac{9.3345}{sin(\frac{180º}{4})} = 13.2\ cm$$

Con R = 10.668 cm, el radio de la estructura sería:

$$R = \frac{R_{max}}{sin(\frac{180º}{n_{rotores}})} = \frac{10.668}{sin(\frac{180º}{4})} = 15.1\ cm$$

Se va a elegir un radio para el chasis de 15 cm y se imprimirá en ABS debido a sus excelentes prestaciones.

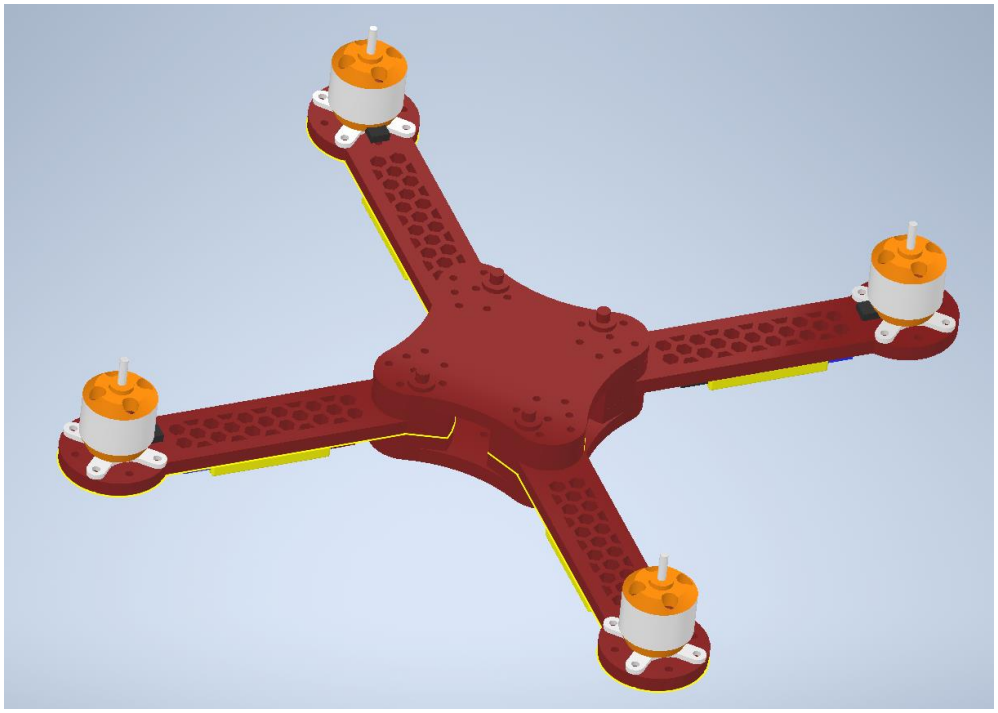


Ilustración 32: Modelo 3D del cuadricóptero desarrollado en Inventor

Los planos del chasis se adjuntan en el anexo.

6.5. Batería

La batería será el elemento encargado de suministrar la energía necesaria a todos los componentes del cuadricóptero.

Algunas características para considerar la batería son:

- **Capacidad** que permita la autonomía buscada sin elevar demasiado el peso de esta. Se mide en mAh (miliamperios hora).
- **Capacidad de descarga (C)** suficiente para poder alimentar a todos los componentes.
- **Tensión o número de celdas** adecuadas para proporcionar la tensión de funcionamiento requerida por los motores y ESC.
- **Precio y peso:** por lo general a mayor capacidad y número de celdas, mayor es el precio. De forma similar sucede con el peso de esta.

Cálculo de consumos, tensión y estimación del tiempo de vuelo: Antes de seleccionar una batería calcularemos el consumo total del cuadricóptero y lo verificaremos con la plataforma de FlyEval. Esta plataforma permite introducir las características de los componentes elegidos para el cuadricóptero y devuelve algunos parámetros estimados del dron, tales como tiempo de vuelo estimado, consumos máximos, mínimos y medios, empujes, etc.

Prácticamente todo el consumo del cuadricóptero es por parte de los motores. El consumo de la electrónica frente a estos es despreciable, en el orden de miliamperios.

- Consumo máximo pico del motor A2212: 16 A durante 60 s
- Consumo máximo pico de los 4 motores: $I = 16 \cdot 4 = 64\ A$

La batería debe ser capaz de proporcionar, al menos, 64 A. Para evitar sobrecalentamientos y evitar forzar la batería, es recomendable una que pueda seleccionar que pueda entregar esa corriente con holgura, sin llegar al máximo.

En cuanto a la tensión, se recomienda usar una batería entre 7,4 V y 14,8 V de tensión nominal:

- Una tensión demasiado baja implicará velocidades de rotación menores y por tanto menor empuje y menores consumos.
- Una tensión demasiado alta implicará velocidades de rotación mayores y por tanto mayor empuje y consumo de corriente.

Tipos de baterías más comunes empleadas para aplicaciones de radiocontrol:

Característica	**Li-Po**	**Li-Ion**	**Ni-MH**	**Ni-Cd**
Densidad de Energía	Alta	Alta	Moderada	Baja
Tasa de Descarga	Alta	Alta	Moderada	Moderada
Peso	Ligero	Ligero	Moderado	Moderado
Tensión nominal por celda	3.7 – 4.2 V	3.7 – 4.2 V	1.2 V	1.2 V
Capacidades Comunes	Cientos de mAh - Miles de mAh	Cientos de mAh - Miles de mAh	Miles de mAh	Cientos de mAh - Miles de mAh
Eficiencia	Buena	Buena	Moderada	Moderada
Peligrosidad	Moderada	Moderada	Baja	Baja
Resistencia a Impactos	Baja – Media	Baja – Media	Alta	Alta
Tiempo de recarga	Bajo	Bajo	Moderado	Moderado
Ciclos de vida	300 – 500	300 – 500	500 – 1000	500 - 1000
Precios	30 – 60 €	30 – 60 €	20 – 50 €	20 – 50 €

Tabla 11: Tabla comparativa de diferentes tipos de baterías

Optaremos por baterías Li-Po por su elevada densidad de energía en relación con su peso y elevado rendimiento.

En esta ocasión se consiguieron baterías **Li-Po 3S 3600 mAh 35C** y cargador para baterías.

En total esta batería puede entregar $3.6 \cdot 35 = 126\,A$, muy por encima de los 64 A requeridos.

Ilustración 33: Batería AeRobots 4u de 3s, 3600 mAh y 35 C (elaboración propia)

Muchos fabricantes también proporcionan las curvas SOC de las baterías, con las que se puede estimar el porcentaje de batería restante a partir de la medición de la tensión de esta.

De forma genérica una curva SOC típica sería:

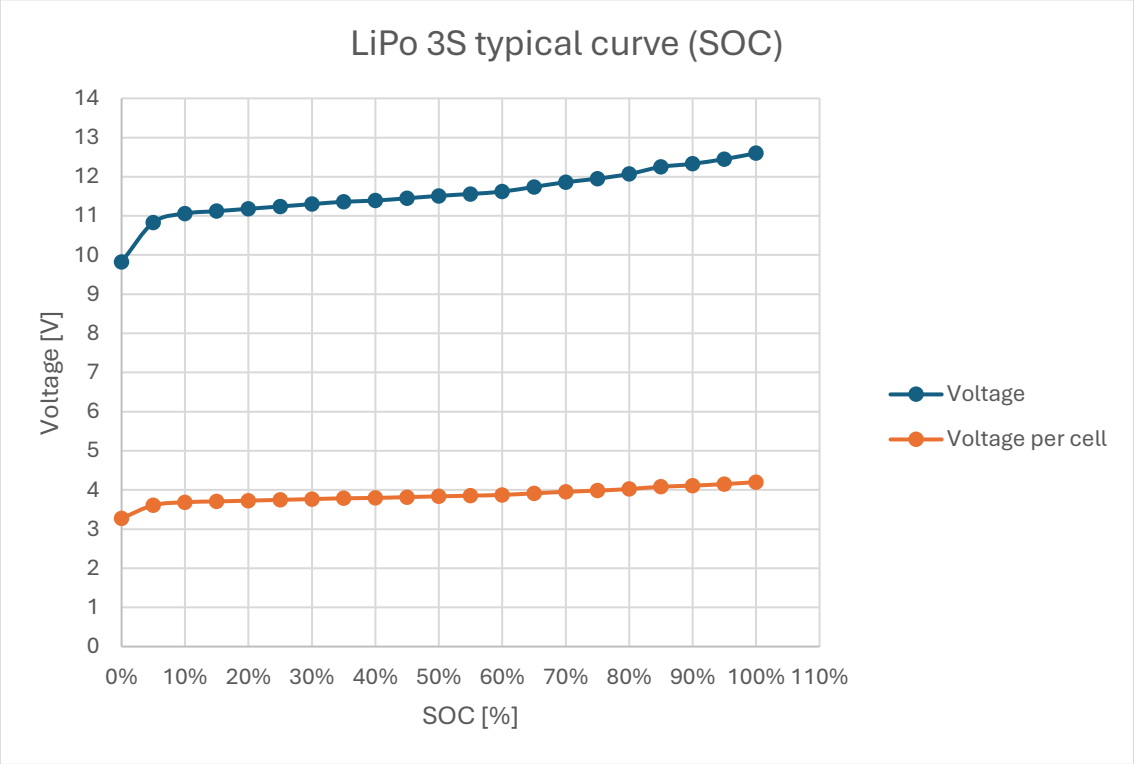


Ilustración 34: Curva extraída de [https://www.researchgate.net/figure/LiPo-Voltage-SOC-state-of-charge-table-SOC-Cell-Voltage-V-2-Cells-Voltage-V-3_tbl1_341375744] que muestra la curva SOC de una batería de Li-Po típica

Esta curva también se puede obtener en laboratorio, midiendo la tensión de la batería ante una determinada descarga constante.

Los resultados obtenidos por la plataforma FlyEval son:

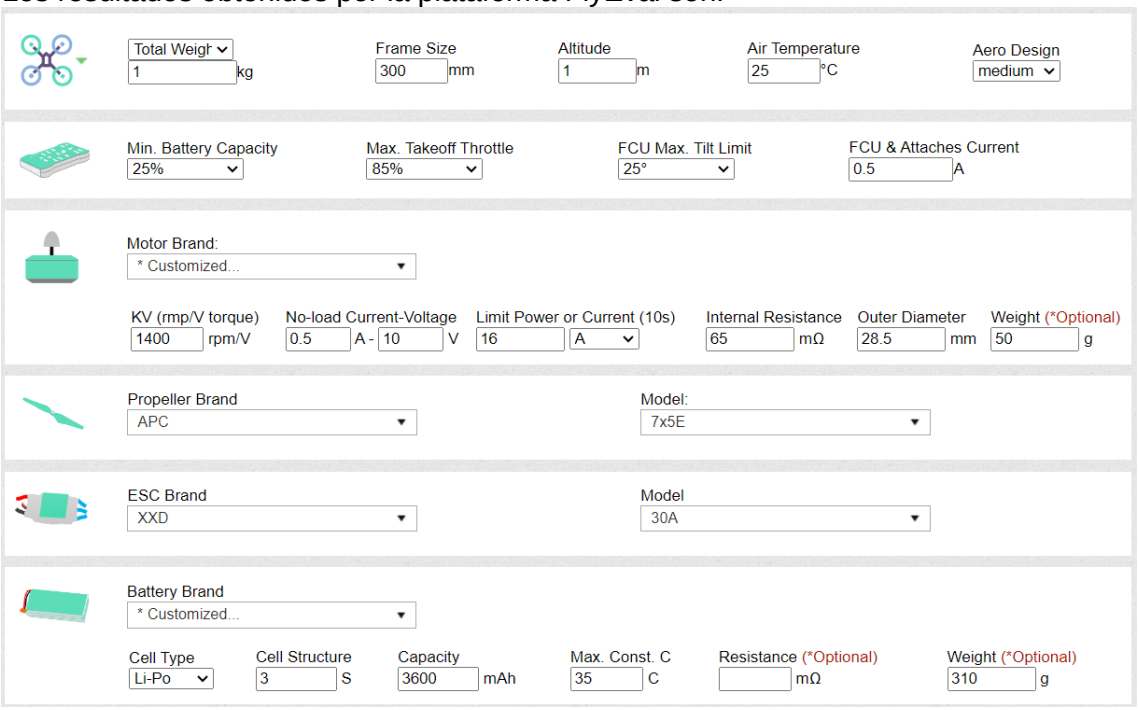


Ilustración 35: Parámetros introducidos a la plataforma web de FlyEval (https://flyeval.com/index.html)

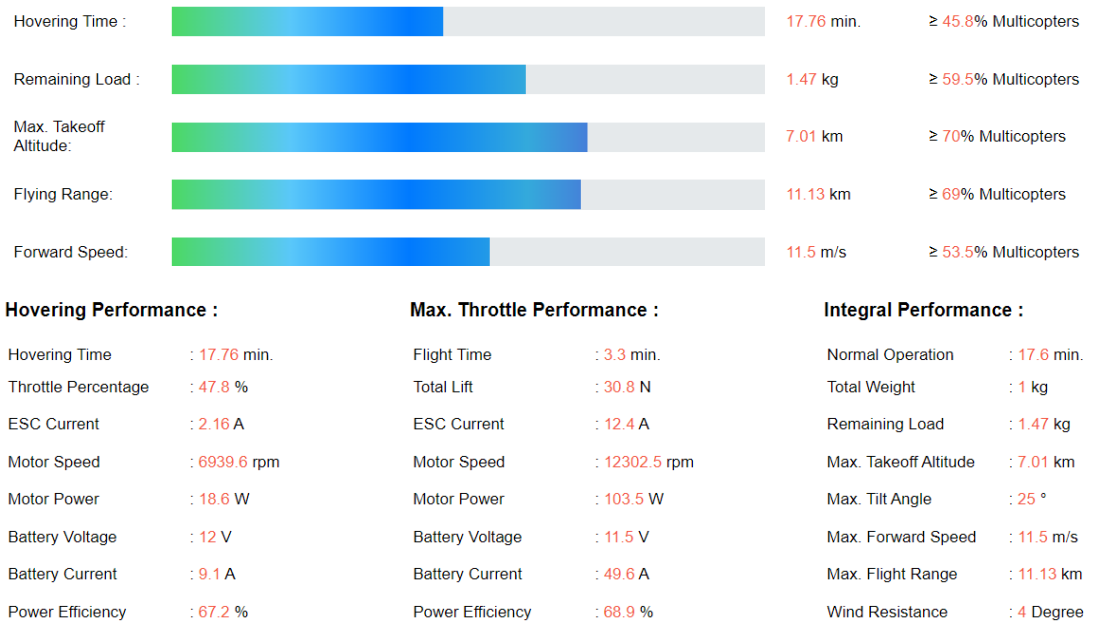


Ilustración 36: Parámetros obtenidos de la plataforma web de FlyEval (https://flyeval.com/index.html)

De la ilustración 36 podemos extraer información importante:

- La velocidad de giro máxima obtenida del análisis espectral es válida, por lo que las relaciones calculadas tienen sentido y una conexión directa con el comportamiento real de los actuadores

- El tiempo de vuelo promedio se estima en 17.76 minutos, con un consumo de 9.1 A

6.6. Esquema eléctrico del cuadricóptero

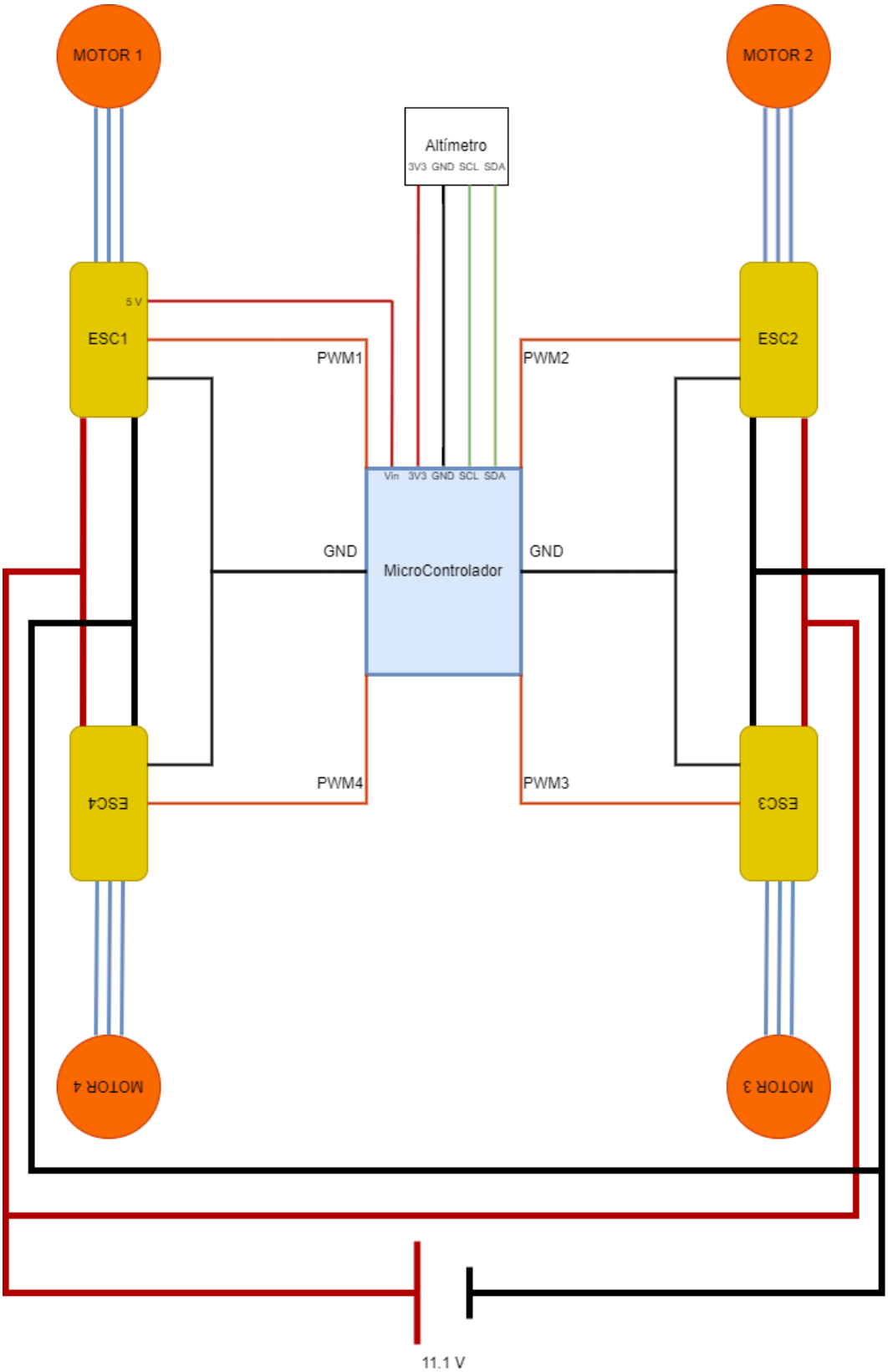


Ilustración 37: Esquema eléctrico del cuadricóptro (imagen propia)

6.6. Plataforma de pruebas

Se ha diseñado una plataforma de pruebas con el propósito de que permita realizar los ensayos y ajustar la sintonización de los controladores antes de probarlo en vuelo, preservando la integridad del dron, del operador y el resto de las personas.

La plataforma de pruebas consta de una rótula de bola unida a un poste vertical, que sujetará el cuadricóptero para impedir que su movimiento en los 3 ejes del espacio, permitiendo solamente 3 grados de libertad, que se corresponden con las inclinaciones o movimientos de alabeo, cabeceo y guiñada.

Para sujetar la batería se hará uso de bridas, un soporte de madera y un sargento.

Ilustración 38: Plataforma de pruebas desarrollada (imagen propia)

6.7. Comunicación

6.7.1. Emisora. Funciones principales

La emisora es la interfaz de control que permite al operador controlar el UAV. En este caso se ha desarrollado una aplicación móvil para usar éste como emisora.

Esta aplicación denominada “DRONE_TFG (v1.0)” se ha desarrollado mediante el IDE de Processing. Se trata de un lenguaje de programación estructurada de código abierto basada en Java (aunque incluye extensiones que permite programar en otros lenguajes como Python), ideado para la realización de proyectos interactivos y gráficos. Además, incorpora una extensión con librerías propias para programar y exportar aplicaciones a móviles con sistema operativo Android. La estructura de programación es similar a la empleada en cualquier programa Arduino, constando de una función de inicialización y un bucle como cuerpo principal de todos los programas. El código completo de la aplicación se adjunta en el anexo.

Las funciones que incluye la emisora diseñada son:

1. **Control de movimiento:** la aplicación consta de dos joysticks para controlar el cuadricóptero. El joystick izquierdo controlará la velocidad de ascenso / descenso con el eje vertical y el con eje horizontal se controlará la velocidad angular de guiñada. El joystick derecho controlará los ángulos de cabeceo (eje vertical) y de alabeo (eje horizontal).

2. **Activación / Desactivación de ejes de joystick** la aplicación, además, consta de cuatro botones que permiten al operador activar o desactivar alguno o varios de los ejes de los joysticks, para poder probar movimientos “puros” de ascenso / descenso, velocidad angular de guiñada, ángulo de cabeceo o de alabeo.

3. **Encender / Apagar:** Por último, la aplicación cuenta con un botón para encender / apagar el cuadricóptero.

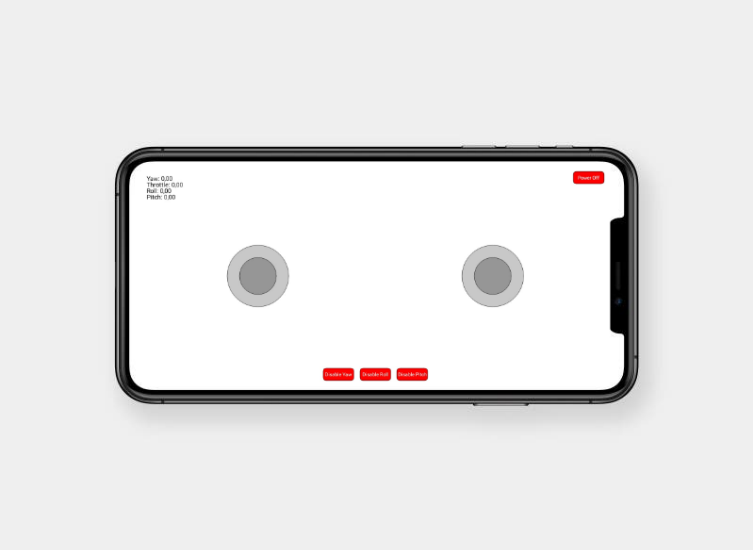


Ilustración 39: Imagen de la interfaz de la app que funciona a modo de emisora (elaboración propia)

6.7.2. Recepción de datos en tiempo real

Con el fin de poder comparar las simulaciones con la respuesta en tiempo real del sistema, se ha elaborado un script en Python para recibir paquetes de datos que contengan los ángulos de alabeo, cabeceo, velocidad en Z y la velocidad angular de guiñada. De esta forma podremos conocer en tiempo real el estado de las variables del dron, almacenarlas y graficarlas.

Al ejecutar este script, esperará a recibir datos. Una vez reciba datos, los mostrará por pantalla mientras los almacena en vectores dinámicos.

6.7.2. Método de Comunicación empleado

La comunicación entre la emisora móvil y el UAV se realiza a través de Wifi, empleando el protocolo UDP. De igual manera entre el ordenador y el UAV.

Este protocolo garantiza una transmisión rápida y en tiempo real, enviando paquetes con los comandos desde la aplicación al microcontrolador y los estados medidos al script de Python.

Este protocolo no precisa de confirmación de la recepción (a diferencia del protocolo TCP/IP que sí que lo precisa), lo cual acelera la velocidad de trasmisión de información, a costa de poder perder paquetes de datos durante el envío. Esto no es crítico en nuestra aplicación, pues se estarán enviando paquetes de forma constante, recuperando la información.

Las ilustraciones 40 y 41 muestran de forma esquemática el funcionamiento de la comunicación Wifi mediante UDP y la forma de los paquetes enviados y/o recibidos:

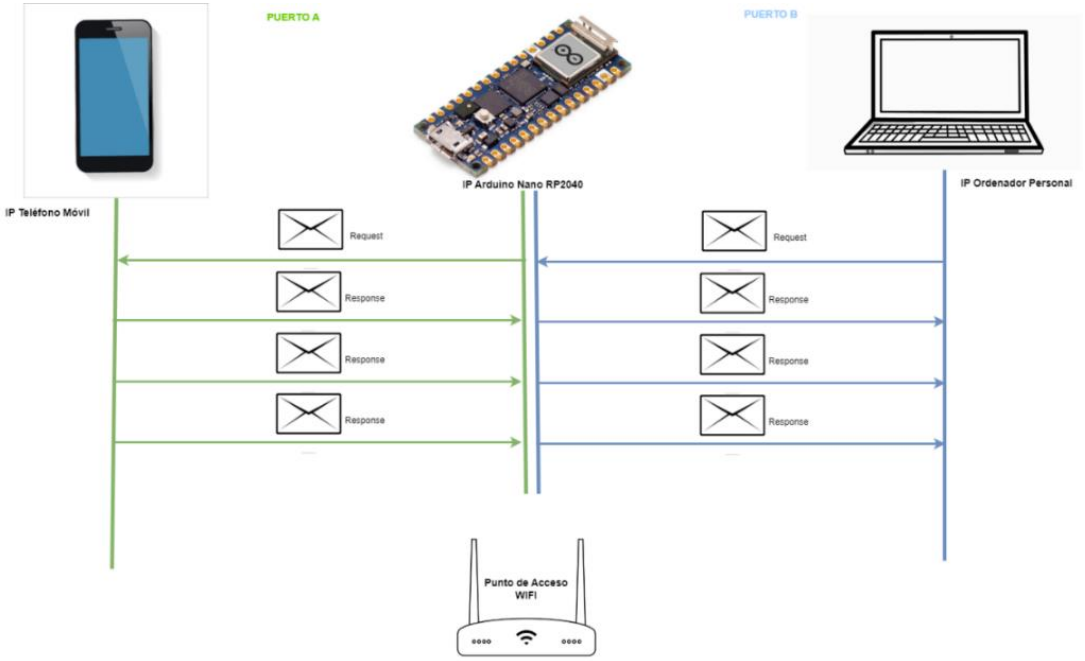


Ilustración 40:Esquema del protocolo de comunicación WIFI-UDP (Ilustración propia)

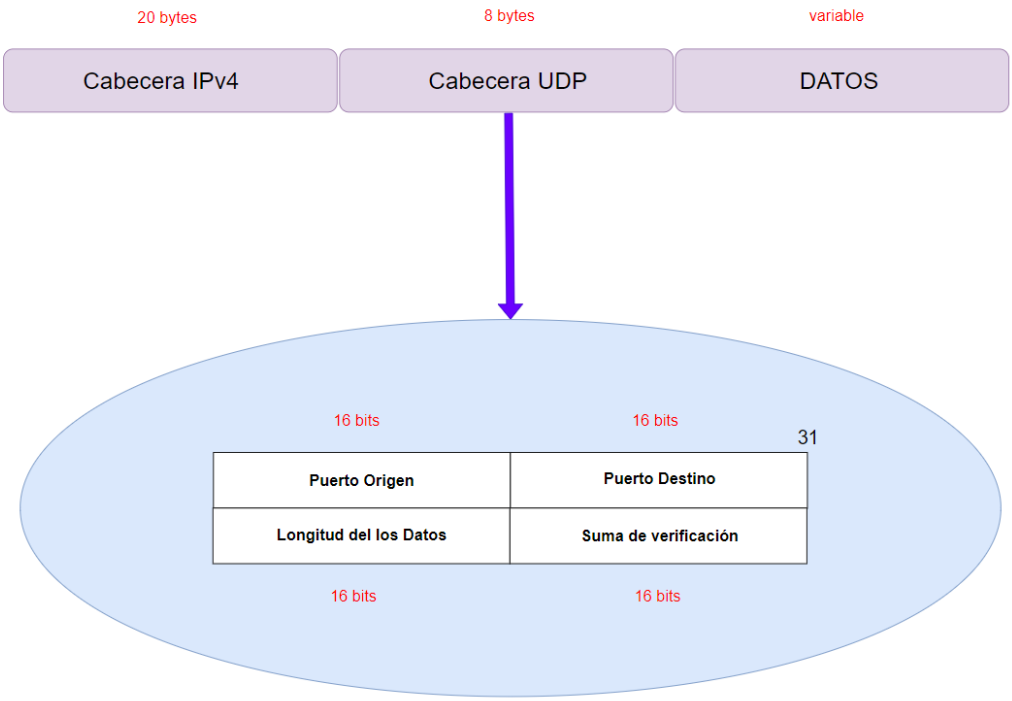


Ilustración 41:Esquema del datagrama IP y datagrama UDP (ilustración propia)

7. Estrategias de control lineal para el cuadricóptero

El control de cuadricópteros es una tarea compleja, debido a la dinámica no lineal y altamente acoplada en estos sistemas, tal y como observamos en el modelo no lineal. A fin de simplificar la dinámica de estos, se linealizó el modelo entorno a un punto de equilibrio. Este punto de equilibrio es aquel en el cual cuadricóptero se encuentra en vuelo estacionario, siendo el modelo válido para ángulos cercanos a cero.

En esta sección se propone el cálculo de las inercias y masa del cuadricóptero y el posterior análisis y estudio de 3 métodos de control:

- **Control PID:** se trata de los métodos de control más empleados gracias a su simplicidad y efectividad. La respuesta de este controlador se puede ajustar variando las ganancias proporcional, integral y derivativa que multiplican al error, integral del error y derivada del error respectivamente.

- **Control LQR:** este controlador ofrece una respuesta óptima al minimizar una función de costo que pondera la rapidez de la respuesta y los comandos de control empleados.

- **Control difuso**: este controlador emplea los principios de la lógica difusa para manejar cierta incertidumbre y no linealidades en la dinámica del cuadricóptero. Las salidas de este controlador se calculan mediante una serie de reglas heurísticas, derivadas del conocimiento del técnico.

7.1. Representación del modelo linealizado en variables de estado

El modelo linealizado del cuadricóptero obtenido:

Dinámica Traslacional:

$$\{I\} \quad \rightarrow \quad \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{1}{m} \cdot \left[\begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix} + \begin{bmatrix} g[\theta\, cos(\Psi_{cte}) + \Phi\, sin(\Psi_{cte})] \\ g[\theta\, sin(\Psi_{cte}) - \Phi\, cos(\Psi_{cte})] \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \right]$$

Dinámica Rotacional:

$$\{I\} \approx \{\mathrm{D}\} \quad \rightarrow \quad \begin{bmatrix} \ddot{\Phi} \\ \ddot{\theta} \\ \ddot{\Psi} \end{bmatrix} = \begin{bmatrix} 1/I_{xx} & 0 & 0 \\ 0 & 1/I_{yy} & 0 \\ 0 & 0 & 1/I_{zz} \end{bmatrix} \cdot \begin{bmatrix} L(F_1 - F_2 - F_3 + F_4) \\ L(F_1 + F_2 - F_3 - F_4) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$

Se optará por una representación del modelo en variables de estado para facilitar la implementación y análisis del sistema linealizado:

$$q_1 = x \quad \rightarrow \quad f_1 \rightarrow \dot{q_1} = \dot{x} = q_2$$

$$q_2 = \dot{x} \quad \rightarrow \quad f_2 \rightarrow \dot{q_2} = \ddot{x} = g q_9$$

$$q_3 = y \quad \rightarrow \quad f_3 \rightarrow \dot{q_3} = \dot{y} = q_4$$

$$q_4 = \dot{y} \quad \rightarrow \quad f_4 \rightarrow \dot{q_4} = \ddot{y} = -g q_7$$

$$q_5 = z \quad \rightarrow \quad f_5 \rightarrow \dot{q_5} = \dot{z} = q_6$$

$$q_6 = \dot{z} \quad \rightarrow \quad f_6 \rightarrow \dot{q_6} = \ddot{z} = \frac{1}{m}(F - mg)$$

$$q_7 = \phi \quad \rightarrow \quad f_7 \rightarrow \dot{q_7} = \dot{\phi} = q_8$$

$$q_8 = \dot{\phi} \quad \rightarrow \quad f_8 \rightarrow \dot{q_8} = \ddot{\Phi} = \frac{1}{I_{xx}} T_{1X}$$

$$q_9 = \theta \quad \rightarrow \quad f_9 \rightarrow \dot{q_9} = \dot{\theta} = q_{10}$$

$$q_{10} = \dot{\theta} \quad \rightarrow \quad f_{10} \rightarrow \dot{q_{10}} = \ddot{\theta} = \frac{1}{I_{yy}} T_{2Y}$$

$$q_{11} = \Psi \quad \rightarrow \quad f_{11} \rightarrow \dot{q_{11}} = \dot{\Psi} = q_{12}$$

$$q_{12} = \dot{\Psi} \quad \rightarrow \quad f_{12} \rightarrow \dot{q_{12}} = \ddot{\Psi} = \frac{1}{I_{zz}} T_{3Z}$$

De forma generar, un sistema lineal se puede expresar en variables de estado:

$$dQ_{12x1} = A_{12x12} Q_{12x1} + B_{12x4} U_{4x1} + W_{12x1}$$

Y las salidas del sistema se obtienen como:

$$Y_{12x1} = C_{12x12} Q_{12x1} + D_{12x4} U_{4x1}$$

En particular:

- Vector de entradas U

$$U = [F, T_{1X}, T_{2Y}, T_{3Z}]^T$$

- Vector de estados Q

$$Q = [q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}]^T$$

- Matriz dinámica A

$$A = \begin{bmatrix} \frac{df_1}{dq_1} & \cdots & \frac{df_1}{dq_{12}} \\ \vdots & \ddots & \vdots \\ \frac{df_{12}}{dq_1} & \cdots & \frac{df_{12}}{dq_{12}} \end{bmatrix}_{12x12}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g \cdot sin(\Psi_{cte}) & 0 & g \cdot \cos(\Psi_{cte}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g \cdot \cos(\Psi_{cte}) & 0 & g \cdot \sin(\Psi_{cte}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Matriz de entradas B

$$B = \begin{bmatrix} \frac{df_1}{du_1} & \cdots & \frac{df_1}{du_4} \\ \vdots & \ddots & \vdots \\ \frac{df_{12}}{du_1} & \cdots & \frac{df_{12}}{du_4} \end{bmatrix}_{12x4}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/I_{yy} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/I_{zz} \end{pmatrix}$$

- Vector de perturbaciones W

$$W = [0,0,0,0,0,-g,0,0,0,0,0,0]^T$$

- Matriz de observación o de salida C

$$C = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}_{12x12}$$

- Matriz de transmisión directa D

$$D = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}_{12x4}$$

7.2. Cálculo de masas e inercias.

Masa de los componentes:

Elemento	Cantidad	Masa [g]
Base superior	1	27
Base inferior	1	37
Brazo	4	21
Motor	4	50
Hélice	4	20
ESC	4	25
Batería	1	310
Cableado y tornillería	…	100
Electrónica y sensores	…	100
TOTAL		1038 g

Tabla 12: Tabla que recoge la masa de los componentes del cuadricóptero

Inercias:

Para simplificar los cálculos, asumiremos algunas simplificaciones:

- Asumimos el centro de masas del dron el centro de este (origen del sistema {D}), pues se trata de un cuadricóptero simétrico y sin desbalances de peso.
- Los momentos de inercia de los elementos estructurales y electrónicos centrales se consideran nulos (masa y distancias respecto al centro de masas del dron pequeñas).
- Se tendrán en cuenta la masa de los motores BLDC, ESC, brazos y batería del dron
- Se tendrá en cuenta la inercia de los motores, aproximada a cilindros macizos de masa uniforme cuyo eje Z coincide con el longitudinal de este.
- Se tendrá en cuenta la inercia de la batería, aproximada a un prisma rectangular de masa uniforme, cuyo eje longitudinal coincide con el lado más largo, orientado al eje X.

Inercia de los motores:

$$I_{xxmotores} = 4 \cdot \frac{1}{12} m(3r^2 + h^2) = \frac{4}{12} \cdot \frac{50+20}{1000}\left(3 \cdot \left(\frac{15}{100}\right)^2 + \left(\frac{2.7}{100}\right)^2\right) = 0.00159\ kgm^2$$

$$I_{yymotores} = 4 \cdot \frac{1}{12} m(3r^2 + h^2) = \frac{4}{12} \cdot \frac{50+20}{1000}\left(3 \cdot \left(\frac{15}{100}\right)^2 + \left(\frac{2.7}{100}\right)^2\right) = 0.00159\ kgm^2$$

$$I_{zzmotores} = 4 \cdot \frac{1}{2} mr^2 + 4I_{helice} = \frac{4}{2} \cdot \frac{50+20}{1000} \cdot \left(\frac{15}{100}\right)^2 + 4 \cdot 0.000012 = 0.00320\ kgm^2$$

Inercia de la batería:

$$I_{xxbateria} = \frac{1}{12} m(b^2 + c^2) = \frac{1}{12} \cdot \frac{310}{1000} \cdot \left(\left(\frac{3.1}{100}\right)^2 + \cdot \left(\frac{4}{100}\right)^2\right) = 0.00007\ kgm^2$$

$$I_{yybateria} = \frac{1}{12} m(a^2 + b^2) = \frac{1}{12} \cdot \frac{310}{1000} \cdot \left(\left(\frac{13.6}{100}\right)^2 + \left(\frac{3.1}{100}\right)^2\right) = 0.00050\ kgm^2$$

$$I_{zzbateria} = \frac{1}{12} m(c^2 + a^2) = \frac{1}{12} \cdot \frac{310}{1000} \cdot \left(\cdot \left(\frac{4}{100}\right)^2 + \left(\frac{13.6}{100}\right)^2\right) = 0.00052 kgm^2$$

- Aplicando el teorema de ejes paralelos:

$$I_{xx} = \sum (I_{CMx} + m{d_{ejeparalelo\ a\ x}}^2)$$
$$I_{yy} = \sum (I_{CMy} + m{d_{ejeparalelo\ a\ y}}^2)$$

$$I_{xx} = 0.00159 + 4 \cdot \frac{50+10}{1000} \cdot \left(\frac{15}{100}\right)^2 + 0 + 4 \cdot \frac{25}{1000} \cdot \left(\frac{7.5}{100}\right)^2 + 0 + 4 \cdot \frac{21}{1000} \cdot \left(\frac{7.5}{100}\right)^2 + 0.00007 = 0.00810\ kgm^2$$

$$I_{yy} = 0.00159 + 4 \cdot \frac{50+10}{1000} \cdot \left(\frac{15}{100}\right)^2 + I \cdot 4 \cdot \frac{25}{1000} \cdot \left(\frac{7.5}{100}\right)^2 + I \cdot 4 \cdot \frac{21}{1000} \cdot \left(\frac{7.5}{100}\right)^2 + 0.0050 = 0.01303\ kgm^2$$

- Aplicando el teorema de ejes perpendiculares

$$I_{zz} = I_{xx} + I_{yy}$$

$$I_{zz} = 0.00810 + 0.01303 = 0.02113\ kgm^2$$

7.3. Limitaciones impuestas a los actuadores

A continuación, se calcularán las limitaciones físicas de los actuadores (o aquellas que queramos imponer por alguna cuestión de diseño).

Estas condiciones serán los valores máximos y mínimos (o el rango de trabajo seleccionado) tanto de fuerza como de par que pueden proporcionar, y serán vitales para limitar la respuesta de nuestras acciones de control.

$$F_{max} = K_F \cdot (\omega_{1max}^2 + \omega_{2max}^2 + \omega_{3max}^2 + \omega_{4max}^2)$$
$$= 0.00013294 \cdot (200^2 + 200^2 + 200^2 + 200^2) = 21.27\ [N]$$

$$F_{min} = K_F \cdot (\omega_{1min}^2 + \omega_{2min}^2 + \omega_{3min}^2 + \omega_{4min}^2)$$
$$= 0.00013294 \cdot (0 + 0 + 0 + 0) = 0\ [N]$$

- Limitaremos los valores de F entre 0 y 20 N
- Si se aplica compensador de gravedad limitamos entre -10.183 y 9.817 N

$$T_{1Xmax} = LK_F(F_{1max} - F_{2min} - F_{3min} + F_{4max})$$
$$= \frac{0.15}{\sqrt{2}} \cdot 0.00013294 \cdot (200^2 + 0 + 0 + 200^2) = 1.128\ [Nm]$$

$$T_{1Xmin} = LK_F(F_{1min} - F_{2max} - F_{3max} + F_{4min})$$
$$= \frac{0.15}{\sqrt{2}} \cdot 0.00013294 \cdot (0 - 200^2 - 200^2 + 0) = -1.128\ [Nm]$$

- Limitaremos los valores de T_{1X} entre -1 y 1 Nm

$$T_{2Ymax} = LK_F(F_{1max} + F_{2max} - F_{3min} - F_{4min})$$
$$= \frac{0.15}{\sqrt{2}} \cdot 0.00013294 \cdot (200^2 + 0 + 0 + 200^2) = 1.128\ [Nm]$$

$$T_{2Ymin} = LK_F(F_{1min} + F_{2min} - F_{3max} - F_{4max})$$
$$= \frac{0.15}{\sqrt{2}} \cdot 0.00013294 \cdot (0 + 0 - 200^2 - 200^2) = -1.128\ [Nm]$$

- Limitaremos los valores de T_{2Y} entre -1 y 1 Nm

$$T_{3Zmax} = K_M \cdot (\omega_{1max}^2 - \omega_{2min}^2 + \omega_{3max}^2 - \omega_{4min}^2)$$
$$= 4.67946 \cdot 10^{-6} \cdot (200^2 - 0 + 200^2 - 0) = 0.374\ [Nm]$$

$$T_{3Zmin} = K_M \cdot (\omega_{1min}^2 - \omega_{2max}^2 + \omega_{3min}^2 - \omega_{4max}^2)$$
$$= 4.67946 \cdot 10^{-6} \cdot (0 - 200^2 + 0 - 200^2) = -0.374\ [Nm]$$

- Limitaremos los valores de T_{3Z} entre -0.3 y 0.3 Nm

7.4. Algoritmo de Mezcla de los Motores

A continuación, se muestra una relación algebraica que relaciona las velocidades angulares a las que debe girar el motor BLDC para generar los pares y fuerza de empuje dados por el controlador. A estas relaciones algebraicas se las conoce como Algoritmo de Mezcla de Motores, "*Motor Mixing Algorithm*" o "*MMA*".

A partir de las expresiones de empuje y par generados por las hélices y de los pares y empujes aplicados al cuadricóptero [13] y [14]:

$$\begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \begin{pmatrix} K_F & K_F & K_F & K_F \\ LK_F & -LK_F & -LK_F & LK_F \\ LK_F & LK_F & -LK_F & -LK_F \\ K_M & -K_M & K_M & -K_M \end{pmatrix}^{-1} \cdot \begin{pmatrix} F \\ T_{1X} \\ T_{2Y} \\ T_{3Z} \end{pmatrix}$$

$$\begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \begin{pmatrix} \frac{1}{4K_F} & \frac{1}{4K_F L} & \frac{1}{4K_F L} & \frac{1}{4K_M} \\ \frac{1}{4K_F} & -\frac{1}{4K_F L} & \frac{1}{4K_F L} & -\frac{1}{4K_M} \\ \frac{1}{4K_F} & -\frac{1}{4K_F L} & -\frac{1}{4K_F L} & \frac{1}{4K_M} \\ \frac{1}{4K_F} & \frac{1}{4K_F L} & -\frac{1}{4K_F L} & -\frac{1}{4K_M} \end{pmatrix} \cdot \begin{pmatrix} F \\ T_{1X} \\ T_{2Y} \\ T_{3Z} \end{pmatrix}$$

Y desarrollando cada término del vector de velocidad angulares:

$$\omega_1^2 = \frac{1}{4K_F} F + \frac{1}{4K_F L} T_{1X} + \frac{1}{4K_F L} T_{2Y} + \frac{1}{4K_M} T_{3Z}$$
$$\omega_2^2 = \frac{1}{4K_F} F - \frac{1}{4K_F L} T_{1X} + \frac{1}{4K_F L} T_{2Y} - \frac{1}{4K_M} T_{3Z}$$
$$\omega_3^2 = \frac{1}{4K_F} F - \frac{1}{4K_F L} T_{1X} - \frac{1}{4K_F L} T_{2Y} + \frac{1}{4K_M} T_{3Z}$$
$$\omega_4^2 = \frac{1}{4K_F} F + \frac{1}{4K_F L} T_{1X} - \frac{1}{4K_F L} T_{2Y} - \frac{1}{4K_M} T_{3Z}$$

Una vez conocida la velocidad angular a alcanzar por el motor, se puede calcular el comando PWM [0:1] a introducir a los ESC con las expresiones calculadas anteriormente.

7.5. Modelado de perturbaciones

Las principales perturbaciones que sufre un UAV en vuelo son [5]:

- Pares giroscópicos ejercidos por los rotores
- Pares de arrastre
- Aleteo de hélices
- Pares aerodinámicos debidos al viento.

En nuestro análisis vamos a considerar únicamente las perturbaciones debidas a ráfagas de viento y turbulencias:

- *Turbulencias:* estas perturbaciones son generadas por movimientos irregulares y aleatorios de aire.
    - Como primera aproximación, se pueden modelar como un ruido blanco.
- *Ráfagas de viento: estas perturbaciones son generadas por un* aumento repentino y breve en la velocidad del viento.
    - Como primera aproximación se pueden modelar como escalones repentinos de breve duración.

7.6. Metodología para el diseño de controladores y simulación

Para el diseño y comparación en simulación de diferentes sistemas de control en un cuadricóptero, se propone la siguiente metodología que se desarrollarán en el software de MATLAB / Simulink:

1. A partir del modelo linealizado del cuadricóptero, diseño de los controladores, mostrando los parámetros y ganancias seleccionados para los mismos, así como las configuraciones adoptadas para controladores:
   - PID
   - PID en cascada
   - LQR en cascada
   - Difuso (Fuzzy) (Empleado la herramienta de “ToolBox Fuzzy Logic” que provee MATLAB)

   Las sintonizaciones obtenidas y utilizadas en este estudio se determinarán mediante simulaciones haciendo un ajuste manual por prueba y error. En ningún caso representan necesariamente las configuraciones óptimas en todas las situaciones.

2. Analizar la respuesta ante una entrada escalón para cada una de las variables de control. Objetivos:
   - Comparar tiempos de establecimiento
   - Compara tiempos de subida
   - Sobrepasos
   - Errores en el estacionario
   - Consumo de recursos y comandos de control

3. Analizar la respuesta ante una entrada en rampa para cada una de las variables de control. Objetivos:
   - Errores de seguimiento
   - Consumo de recursos y comandos de control

4. Analizar la robustez del sistema ante perturbaciones. Objetivos:
   - Estabilidad
   - Consumo de recursos y comandos de control

5. Analizar la respuesta ante cambios en parámetros del modelo. Objetivos:
   - Estabilidad
   - Consumo de recursos y comandos de control

7.7. Diseño de controladores

7.7.2. Control PID. PID en cascada

Este controlador define la ley de control siguiente como la suma de 3 contribuciones:

$$u(t) = P(t) + D(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

- **Parte proporcional $P(t)$:** cuya salida es el resultado de multiplicar el error por una ganancia proporcional al mismo.

$$u = K_p e(t)$$

- **Parte integral $I(t)$:** cuya salida es el resultado la integral del error. Suma todos los errores pasados. Esta acción garantiza error 0 en el estacionario, bajando la fase 90 º (puede dar lugar a sistemas inestables).
    - Las dinámicas de nuestro modelo son integradoras puras, en el caso de las velocidades, o doble integradoras en el caso de los ángulos y posiciones. Es por ello por lo que no incluiremos el término integral, pues, teóricamente, se alcanzará la consigna ante una entrada escalón sin necesidad de incluir esta acción.

$$u = K_i \int_0^t e(t)dt$$

- **Parte derivativa $D(t)$:** cuya salida es el resultado de la derivada del error. “Pronostica” el error futuro en base a la tasa de variación del error. Esta acción suaviza la respuesta, y sube la fase 90º (puede estabilizar el sistema). No debe emplearse si existen ruido de medida en nuestro sistema.
    - Deberemos filtrar correctamente las lecturas del IMU para reducir su ruido de lectura, pues se trata del sensor con mayor sensibilidad y las vibraciones generadas al girar los motores, introducirán ruido de alta frecuencia en el sistema, llegando al punto de no poder ser rechazadas por el controlador, volviendo al sistema inestable.

$$u = K_d \frac{de(t)}{dt}$$

No se ha conseguido una sintonización satisfactoria y estable ante diferentes entradas para un controlador PID simple por lo que se ha optado por un PID en cascada. Además, se ha añadido un compensador de gravedad al control de la velocidad en Z.

Un control PID en cascada se basa en dos lazos de control PID, uno interno y otro externo. El lazo externo es más lento y genera las referencias para el lazo interno, más rápido. En nuestro caso:

- El lazo de control interno, el más rápido, controlará las velocidades angulares, así como la velocidad lineal en Z, generando los pares que deben entregar los motores.
    - Controla las velocidades angulares de pitch, roll y yaw, además de la velocidad lineal en Z.
    - Genera los pares que deben desarrollar los actuadores.
- El lazo de control externo, más lento:
    - Controla los ángulos de pitch y roll, asegurando que se alcancen las referencias de dichos ángulos.
    - Genera las referencias de velocidades angulares para los controladores del lazo interno.

El control PID en cascada teóricamente presenta algunas ventajas frente al control PID clásico, especialmente cuando se controlan variables que tienen influencia directa en otras:

- Mejora la estabilidad del sistema.
- Mejora en el rechazo a perturbaciones.
- Simplificación del diseño del controlador, flexibilizando el diseño y sintonización de este.
- Adaptabilidad a variaciones de parámetros de la planta o modelo.

Por otro lado, un compensador de gravedad se trata de un algoritmo de control que genera una fuerza de empuje constante para contrarrestar la fuerza gravitatoria.

$$F_{comp.gravedad=m\cdot g}$$

Este se aplicará al lazo de control de la velocidad en Z. Ventajas:

- Mejora en la estabilidad del sistema
- Reducción en la carga del controlador PID
- Mejora la respuesta del sistema

Diseño de controlador PID en cascada propuesto:

- **Lazo interno:**
  - ***Controlador PID propuesto para la Velocidad en Z con compensador de gravedad***

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} + mg$$

  - ***Controlador PID propuesto para la Velocidad Angular de Roll***

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

  - ***Controlador PID propuesto para la Velocidad Angular de Pitch***

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

  - ***Controlador PID propuesto para la Velocidad Angular de Yaw***

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

- **Lazo externo:**
  - ***Controlador PID propuesto para el ángulo de Roll***

$$u(t) = K_p e(t)$$

  - ***Controlador PID propuesto para el ángulo de Pitch***

$$u(t) = K_p e(t)$$

La sintonización de los controladores se ha realizado de forma manual aplicando una entrada escalón:

Controlador	Ganancia proporcional	Ganancia integral	Ganancia derivativa	Valor del filtro N
Controlador del lazo de velocidad en Z	25	0	1	100
Controlador del lazo de velocidad angular de alabeo	0.3	0	0.007	100
Controlador del lazo de velocidad angular de cabeceo	0.2	0	0.002	100
Controlador del lazo del ángulo de alabeo	10	0	0	0
Controlador del lazo del ángulo de cabeceo	5.5	0	0	0
Controlador del lazo de velocidad angular de guiñada	0.35	0	0.006	100

Tabla 13: Tabla que recoge las ganancias de los distintos controladores PID

Lazo de control

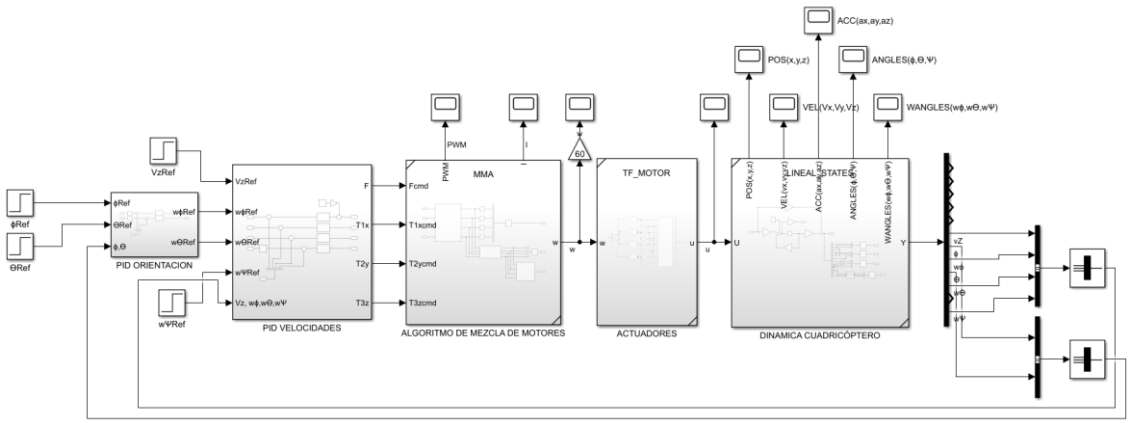


Ilustración 42: Lazo de control PID en cascada desarrollado

Estructura del controlador:

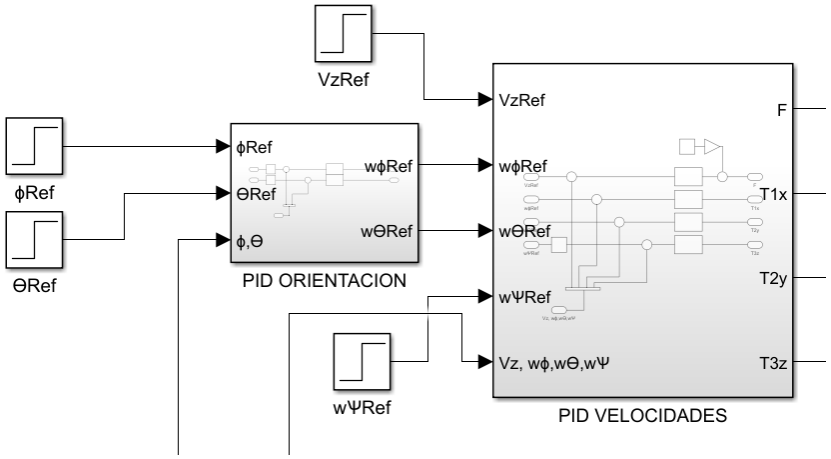


Ilustración 43: Vista de los bloques del controlador PID en cascada

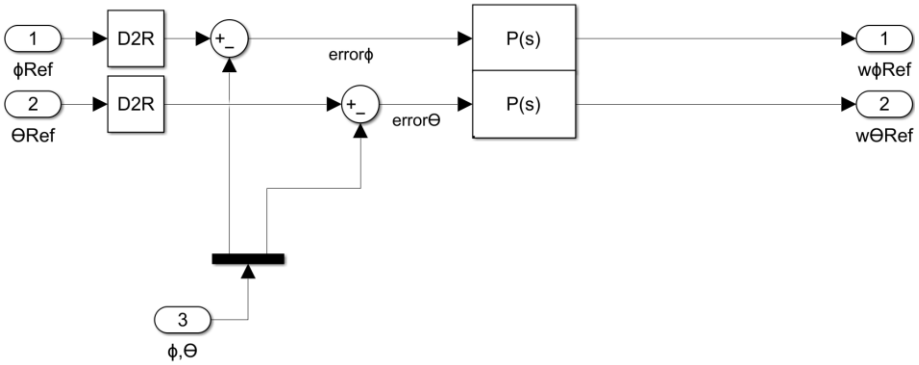


Ilustración 44: Detalle del controlador del lazo externo de orientación

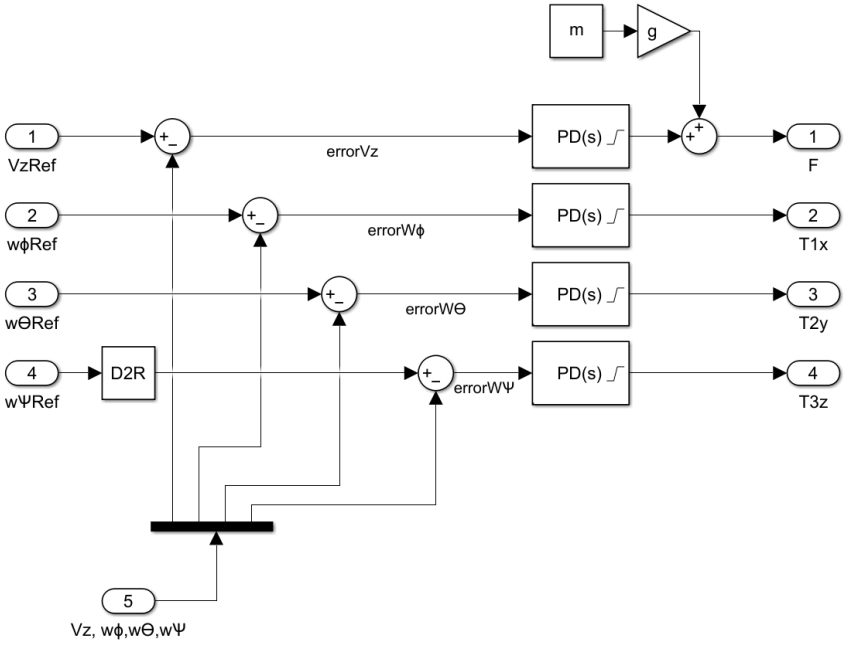


Ilustración 45: Detalle del controlador del lazo interno de velocidades

7.7.3. Control LQR. LQR en cascada

Este controlador define la ley de control:

$$u(t) = -Kq(t)$$

Siendo u las entradas del sistema, q los estados del sistema.

Los comandos aplicados por el controlador vienen de resolver la siguiente integral de coste J:

$$J = \int_{t_o}^{t_f} (q^T Q q + u^T R u) dt$$

cuya solución P, viene dada por Ricatti:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \rightarrow \ A^T P + PA - PBR^{-1}B^T P + Q = 0$$

Conformando las matrices A y B con aquellos estados que nos interesa realimentar.

Para calcular K:

$$K = R^{-1}B^T P$$

Donde las matrices Q y R ponderan la rapidez de respuesta del sistema y el coste en recursos de dicha respuesta:

$Q > R \rightarrow \ control\ más\ rápido\ pero\ menos\ optimizado$
$Q < R \rightarrow \ control\ más\ lento\ pero\ más\ optimizado$

Los elementos de la matriz Q se pueden definir como [15]:

$$Q_{i,i} = \frac{1}{máximo\ error\ aceptable\ de\ los\ estados^2}$$

Los elementos de la matriz R se pueden definir como:

$$R_{j,j} = \frac{1}{máximo\ valor\ aceptable\ de\ entrada^2}$$

Es recomendable el ajuste manual de las matrices R y Q para obtener la respuesta deseada.

No se ha conseguido una sintonización satisfactoria y estable ante diferentes entradas para un controlador LQR simple por lo que se ha optado por un LQR en cascada, presentando ventajas similares al control PID en cascada, con las características propias del control LQR. También se ha añadido un compensador de gravedad al control de la velocidad en Z.

Controlador LQR propuesto para Velocidades Angulares y en Z con compensador de gravedad

$$u(t) = -K_{4x4}\overline{q_{4x1}} + \begin{pmatrix} mg \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Controlador LQR propuesto para los Ángulos

$$u(t) = -K_{2x2}\overline{q_{2x1}}$$

Sintonización del controlador para velocidades angulares y velocidad en Z

$$R = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 \\ 0 & 0 & 0.75 & 0 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

$$Q = \begin{pmatrix} 400 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 \\ 0 & 0 & 0.011 & 0 \\ 0 & 0 & 0 & 0.35 \end{pmatrix}$$

$$K = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 0.0816 & 0 & 0 \\ 0 & 0 & 0.1211 & 0 \\ 0 & 0 & 0 & 0.2236 \end{pmatrix}$$

Sintonización del controlador para ángulos de cabeceo y alabeo

$$R = \begin{pmatrix} 0.6 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} 7 & 0 \\ 0 & 10 \end{pmatrix}$$

$$K = \begin{pmatrix} 3.4157 & 0 \\ 0 & 3.1623 \end{pmatrix}$$

Lazo de control

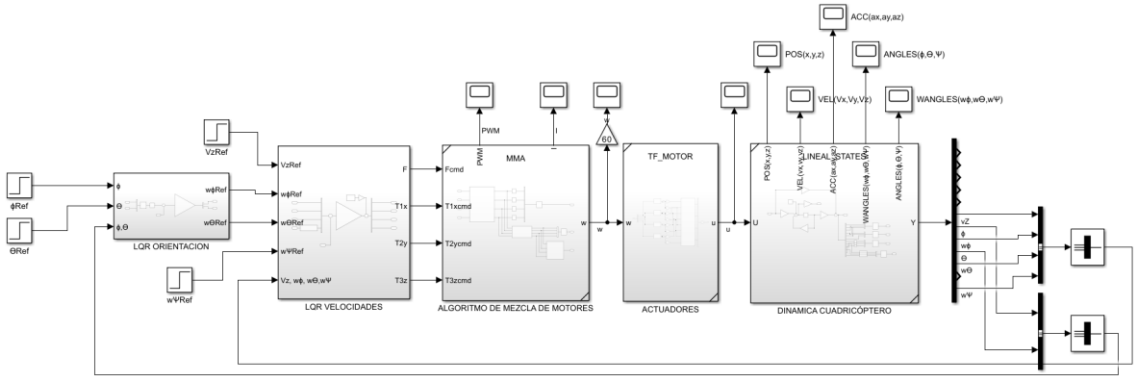


Ilustración 46: Lazo de control LQR en cascada desarrollado

Estructura del controlador:

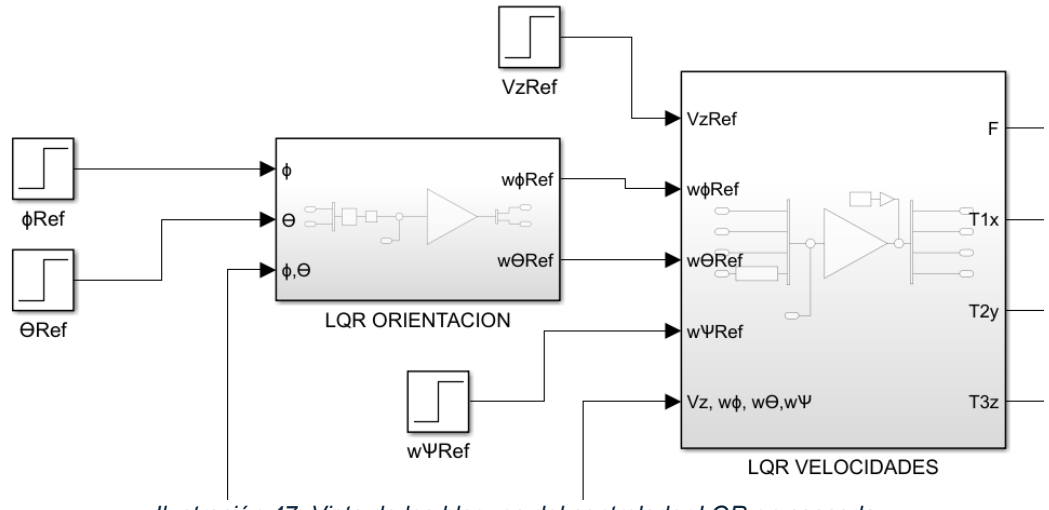


Ilustración 47: Vista de los bloques del controlador LQR en cascada

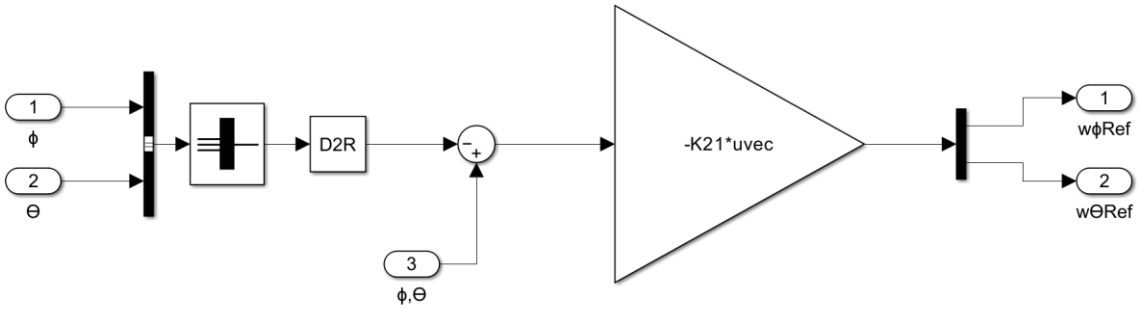


Ilustración 48: Detalle del controlador del lazo externo de orientación

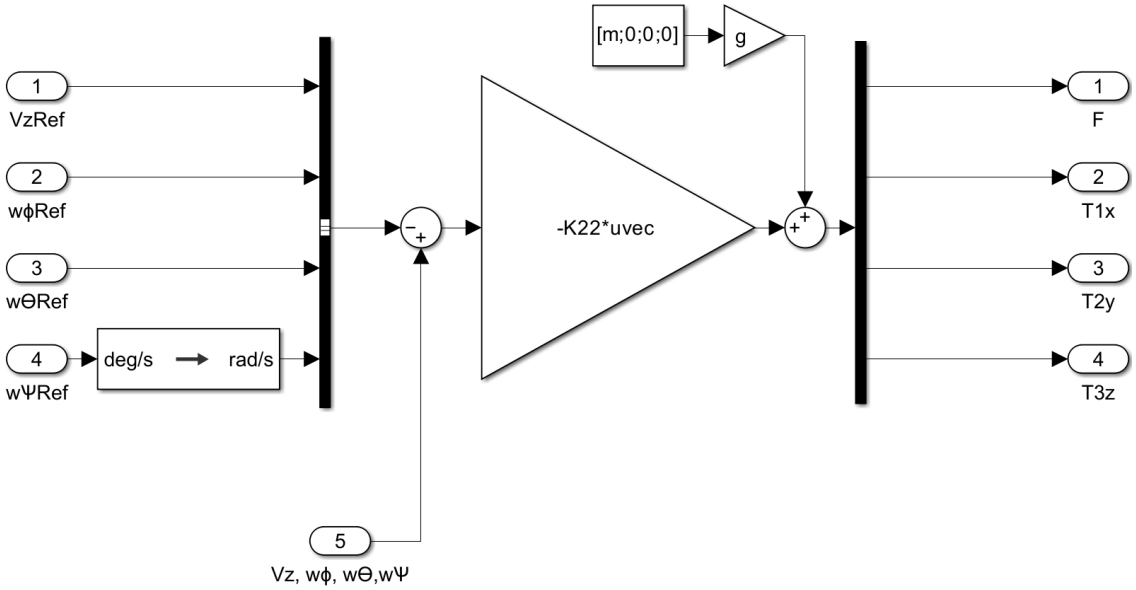


Ilustración 49: Detalle del controlador del lazo interno de velocidades

7.7.4. Control Difuso

Se trata de un controlador que emplea la metodología de la lógica difusa en sistemas de control reales. Se trata de un control especialmente útil en sistemas complejos, no lineales, en los que es difícil aplicar técnicas de control clásicas y se posee cierto conocimiento y experiencia en la planta a controlar [16]. Emplearemos el controlador difuso de tipo Mamdani.

El diseño de este controlador consiste en asociar a cada variable cuantitativa de entrada (en nuestro caso el error y la derivada del error) un término lingüístico con una cierta incertidumbre descrita por una función de pertenencia μ (fuzzyficación). A este término se le aplicarán una serie de reglas de inferencia lógicas (por ejemplo: Si A y B entonces C) a fin de obtener una salida lingüística (*inferencia*). Finalmente, esta salida lingüística es convertida a un valor numérico z, que se aplicará a los actuadores del cuadricóptero (defuzzyficación).

Ilustración 50: Esquema general de un controlador difuso [16]

Diseño del controlador

Se elegido funciones de pertenencia triangulares para todas las variables de nuestro controlador. A continuación, se muestran las variables lingüísticas y los punto que definen a las funciones de pertenencia, así como las reglas de inferencia empleadas.
Al igual que en los otros controladores, se ha añadido un compensador de gravedad al lazo de la velocidad en Z.

Fuzzyficación:

Variables lingüísticas y funciones de pertenencia para la entrada de error:

Variables lingüísticas	Velocidad en Z	Ángulo de alabeo	Ángulo de cabeceo	Velocidad angular guiñada
GN Grande Negativo	-1.732, -1.3, -0.8663	-0.349, -0.262, -0.1746	-0.349, -0.262, -0.1746	-2.318, -1.74, -1.16
MN Medio Negativo	-1.3, -0.8663, -0.4333	-0.262, -0.1746, -0.08733	-0.262, -0.1746, -0.08733	-1.74, -1.16, -0.58
PN Pequeño Negativo	-0.8663, -0.4333, 0	-0.1746, -0.08733, 0	-0.1746, -0.08733, 0	-1.16, -0.58, 0
ZE Cero	-0.4333, 0, 0.4333	-0.08733, 0, 0.08733	-0.08733, 0, 0.08733	-0.58, 0, 0.58
PP Pequeño Positivo	0, 0.4333, 0.8663	0, 0.08733, 0.1746	0, 0.08733, 0.1746	0, 0.58, 1.16
MP Medio Positivo	0.4333, 0.8663, 1.3	0.08733, 0.1746, 0.262	0.08733, 0.1746, 0.262	0.58, 1.16, 1.74
GP Grande Positivo	0.8663, 1.3 1.733	0.1746, 0.262, 0.3493	0.1746, 0.262, 0.3493	1.16, 1.74, 2.32

Tabla 14: Tabla que recoge las variables lingüísticas y la forma de las funciones de pertenencia del error

Variables lingüísticas y funciones de pertenencia para la derivada del error:

Variables lingüísticas	Velocidad en Z	Ángulo de alabeo	Ángulo de cabeceo	Velocidad angular guiñada
GN Grande Negativo	-66.7, -50, -33.33	-2.403, -1.8, -1.2	-2.403, -1.8, -1.2	-66.75, -50, -33.33
MN Medio Negativo	-50, -33.35, -16.66	-1.8, -1.2, -0.5998	-1.8, -1.2, -0.5998	-50, -33.33, -16.66
PN Pequeño Negativo	-33.35, -16.66, 0	-1.2 -0.5998, 0	-1.2 -0.5998, 0	-33.33, -16.66, 0
ZE Cero	-16.66, 0, 16.66	-0.5998, 0, 0.5998	-0.5998, 0, 0.5998	-16.66, 0, 16.66
PP Pequeño Positivo	0, 16.66, 33.35	0, 0.5998, 1.2	0, 0.5998, 1.2	0, 16.66, 32.32
MP Medio Positivo	16.66 33.35, 50	0.5998, 1.2, 1.8	0.5998, 1.2, 1.8	16.66, 33.32, 50
GP Grande Positivo	33.35, 50, 66.7	1.2, 1.8, 2.403	1.2, 1.8, 2.403	33.32, 50, 66.75

Tabla 15: Tabla que recoge las variables lingüísticas y la forma de las funciones de pertenencia de la derivada del error

Variables lingüísticas y funciones de pertenencia de salida:

Variables lingüísticas	Empuje total	Par de alabeo	Par de cabeceo	Par de guiñada
FNG Fuerza Grande Negativo	-13.08, -9.813, -6.542	-0.533, -0.3998, -0.2666	-0.6663, -0.4998, -0.3332	-0.3999, -0.2999, -0.2
FNM Fuerza Medio Negativo	-9.813, -6.542, -3.275	-0.3998, -0.2666, -0.1334	-0.4998, -0.3332, -0.1668	-0.2999, -0.2, -0.1
FNP Fuerza Pequeño Negativo	-6.542, -3.275, 0	-0.2666, -0.1334, 0	-0.3332, -0.1668, 0	-0.2, -0.1, 0
FZE Fuerza Cero	-3.275, 0, 3.275	-0.1334, 0, 0.1334	-0.1668, 0, 0.1668	-0.1, 0, -0.1
FPP Fuerza Pequeño Positivo	0, 3.275, 6.544	0, 0.1334, 0.2666	0, 0.1668, 0.3333	0, 0.1, 0.2
FPM Fuerza Medio Positivo	3.275, 6.544, 9.817	0.1334, 0.2666, 0.4	0.1668, 0.3333, 0.5	0.1, 0.2, 0.3
FPG Fuerza Grande Positivo	6.544, 9.817, 13.09	0.2666, 0.4, 0.5334	0.3333, 0.5, 0.6668	0.2, 0.3, 0.4002

Tabla 16: Tabla que recoge las variables lingüísticas y la forma de las funciones de pertenencia de salida

Los valores de cada celda describen una función de pertenencia triangular, de la forma:

$$\mu\,(1er\ numero_{error\ min}) = \ 0$$
$$\mu(2º\ número_{error}) = 1$$
$$\mu(3er\ número_{error\ max}) = 0$$

El primer número representa el mínimo error, derivada del error, o salida, para el cual la que la función de pertenencia vale 0.

El segundo número representa el error, derivada del error o salida, para el cual la que la función de pertenencia vale 1.

El tercer número representa el máximo error, derivada del error o salida, para el cual la que la función de pertenencia vale 0

Inferencia:
Emplearemos la siguiente tabla de reglas para establecer nuestras reglas de inferencia. Se trata de una configuración típica para sistemas con 7 variables lingüísticas de entrada y de salida:

errorDVz **errorVz**	*GN Grande Negativo*	*MN Medio Negativo*	*PN Pequeño Negativo*	*ZE Cero*	*PP Pequeño Positivo*	*MP Medio Positivo*	*GP Grande Positivo*
GN Grande Negativo	FNG	FNG	FNG	FNG	FNM	FNP	FZE
MN Medio Negativo	FNG	FNG	FNG	FNM	FNP	FZE	FPP
PN Pequeño Negativo	FNG	FNG	FNM	FNP	FZE	FPP	FMP
ZE Cero	FNG	FNM	FNP	FZE	FPP	FMP	FGP
PP Pequeño Positivo	FNM	FNP	FZE	FPP	FMP	FGP	FGP
MP Medio Positivo	FNP	FZE	FPP	FMP	FGP	FGP	FGP
GP Grande Positivo	FZE	FPP	FMP	FGP	FGP	FGP	FGP

Tabla 17: Tabla de reglas de inferencia empleada para cada uno de los controladores

Por tanto, las reglas de inferencia establecidas son:

1. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = GN) \rightarrow (salida = FNG)$
2. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = MN) \rightarrow (salida = FNG)$
3. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = PN) \rightarrow (salida = FNG)$
4. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = ZE) \rightarrow (salida = FNG)$
5. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = PP) \rightarrow (salida = FNM)$
6. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = MP) \rightarrow (salida = FNP)$
7. $Si\ (errorEntrada = GN)\ y\ (errordEntrada = GP) \rightarrow (salida = FZE)$
8. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = GN) \rightarrow (salida = FNG)$
9. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = MN) \rightarrow (salida = FNG)$
10. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = PN) \rightarrow (salida = FNG)$
11. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = ZE) \rightarrow (salida = FNM)$
12. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = PP) \rightarrow (salida = FNP)$
13. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = MP) \rightarrow (salida = FZE)$
14. $Si\ (errorEntrada = MN)\ y\ (errordEntrada = GP) \rightarrow (salida = FPP)$
15. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = GN) \rightarrow (salida = FNG)$
16. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = MN) \rightarrow (salida = FNG)$
17. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = PN) \rightarrow (salida = FNM)$
18. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = ZE) \rightarrow (salida = FNP)$
19. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = PP) \rightarrow (salida = FZE)$
20. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = MP) \rightarrow (salida = FPP)$
21. $Si\ (errorEntrada = PN)\ y\ (errordEntrada = GP) \rightarrow (salida = FMP)$
22. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = GN) \rightarrow (salida = FNG)$
23. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = MN) \rightarrow (salida = FNM)$
24. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = PN) \rightarrow (salida = FNP)$
25. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = ZE) \rightarrow (salida = FZE)$
26. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = PP) \rightarrow (salida = FPP)$
27. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = MP) \rightarrow (salida = FMP)$
28. $Si\ (errorEntrada = ZE)\ y\ (errordEntrada = GP) \rightarrow (salida = FGP)$
29. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = GN) \rightarrow (salida = FNM)$
30. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = MN) \rightarrow (salida = FNP)$
31. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = PN) \rightarrow (salida = FZE)$
32. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = ZE) \rightarrow (salida = FPP)$
33. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = PP) \rightarrow (salida = FMP)$
34. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = MP) \rightarrow (salida = FGP)$
35. $Si\ (errorEntrada = PP)\ y\ (errordEntrada = GP) \rightarrow (salida = FGP)$
36. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = GN) \rightarrow (salida = FNP)$
37. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = MN) \rightarrow (salida = FZE)$
38. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = PN) \rightarrow (salida = FPP)$
39. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = ZE) \rightarrow (salida = FMP)$
40. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = PP) \rightarrow (salida = FGP)$
41. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = MP) \rightarrow (salida = FGP)$
42. $Si\ (errorEntrada = MP)\ y\ (errordEntrada = GP) \rightarrow (salida = FGP)$
43. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = GN) \rightarrow (salida = FZE)$
44. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = MN) \rightarrow (salida = FPP)$
45. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = PN) \rightarrow (salida = FMP)$
46. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = ZE) \rightarrow (salida = FGP)$
47. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = PP) \rightarrow (salida = FGP)$
48. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = MP) \rightarrow (salida = FGP)$
49. $Si\ (errorEntrada = GP)\ y\ (errordEntrada = GP) \rightarrow (salida = FGP)$

Defuzzyficación:
Emplearemos el método más empleado y eficiente: el método del centro de masas o centroide. La salida

$$z_o = \frac{\sum z_k \mu_c(z_k)}{\sum \mu_c(z_k)}$$

Lazo de control

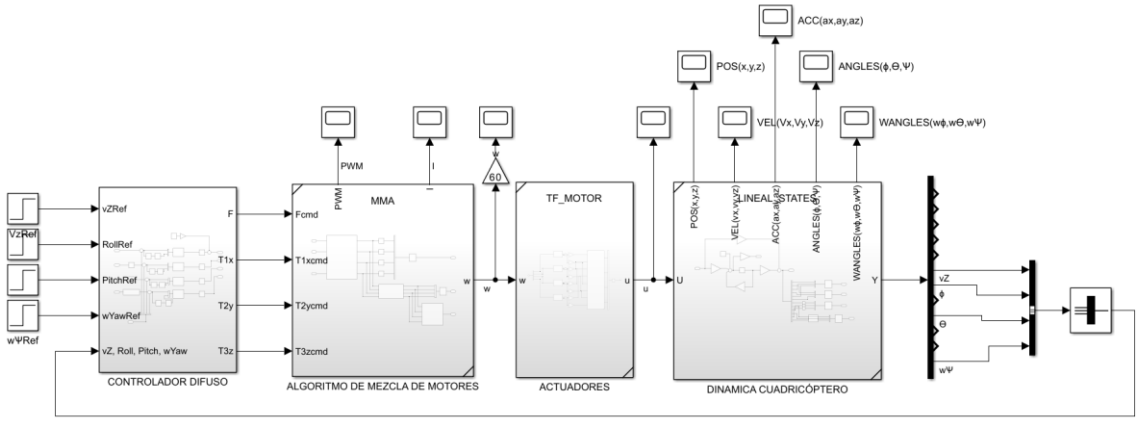


Ilustración 51: Lazo de control difuso desarrollado

Estructura del controlador:

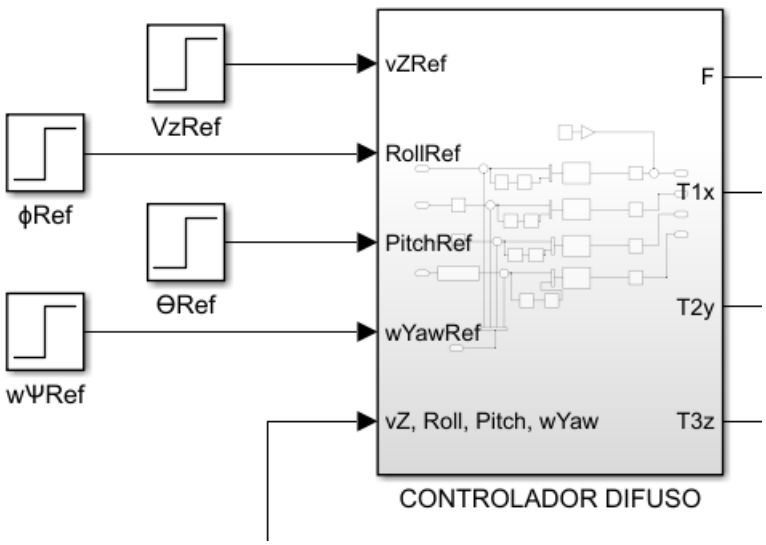


Ilustración 52: Vista del bloque del controlador difuso

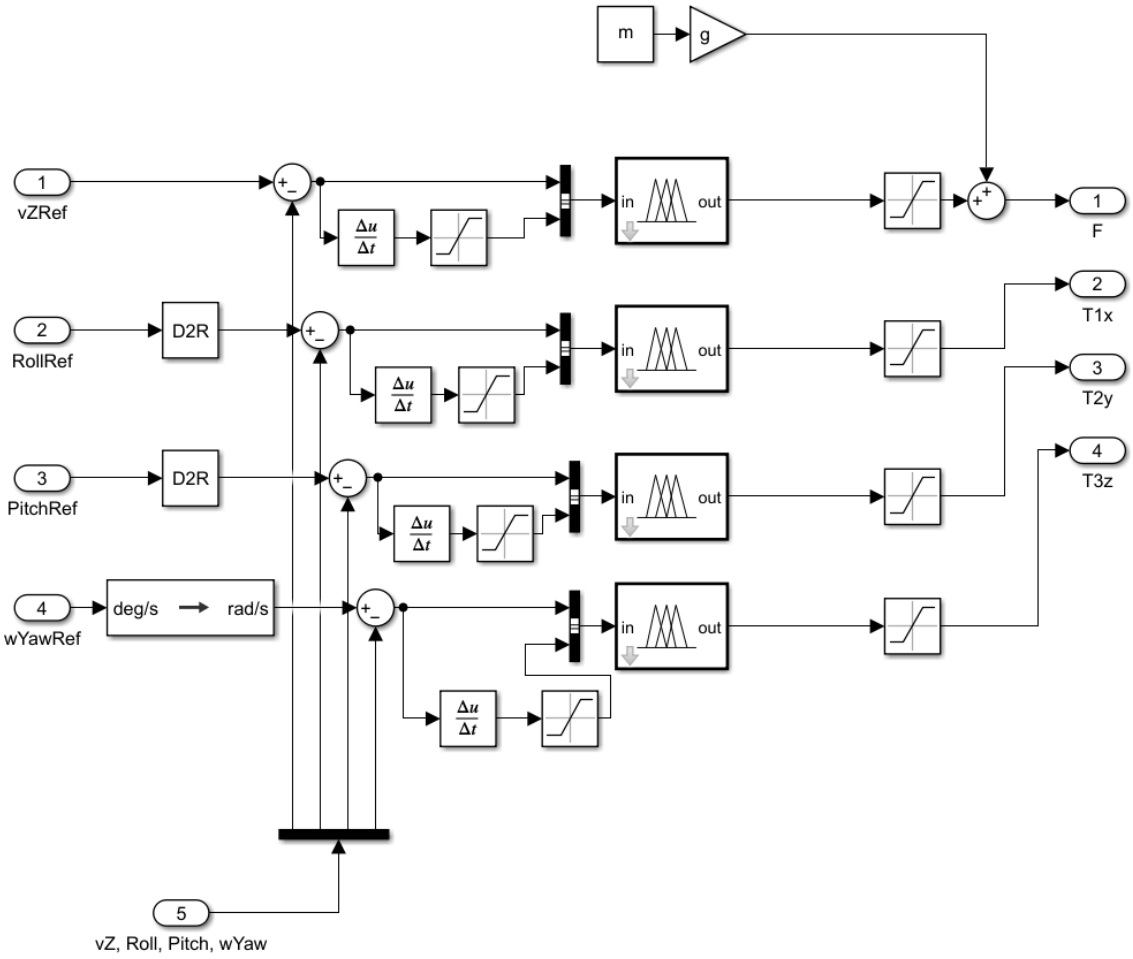


Ilustración 53: Vista de los bloques internos del controlador difuso

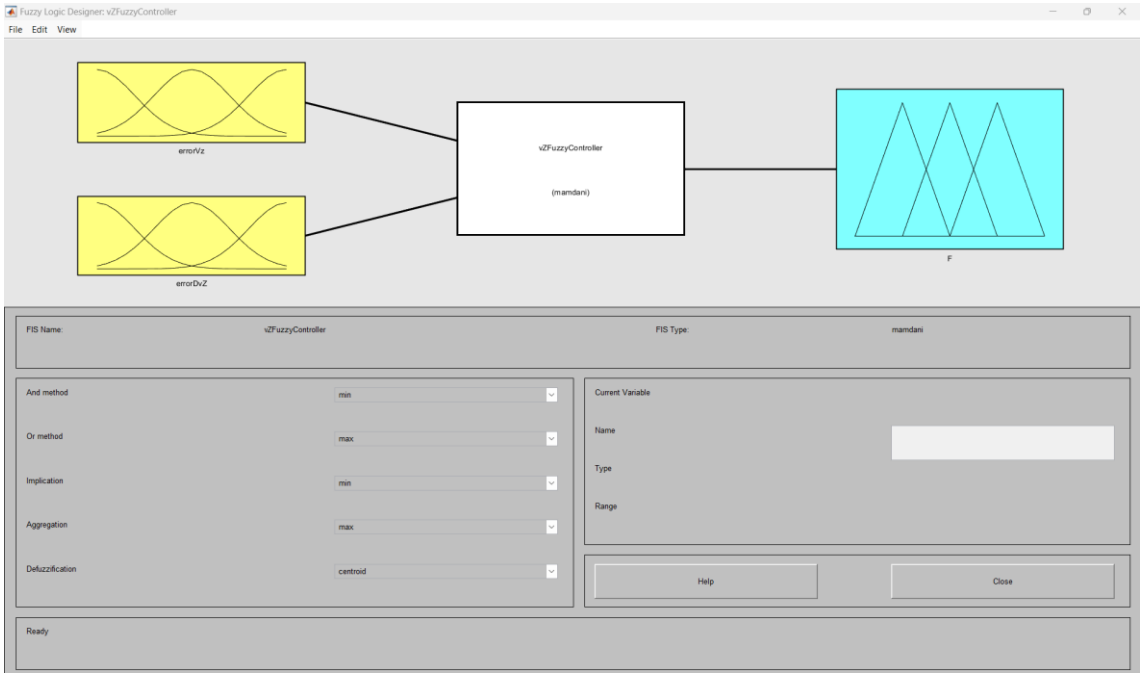


Ilustración 54: Esquema del controlador difuso para la velocidad en Z (MATLAB Fuzzy Logic Toolbox)

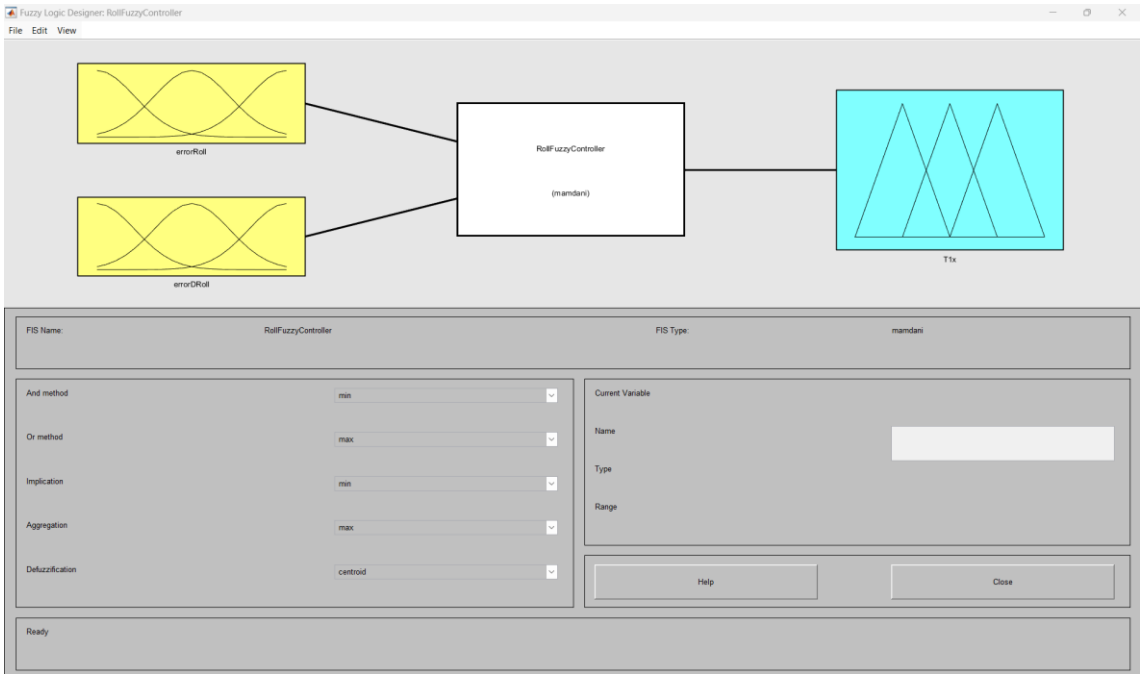


Ilustración 55: Esquema del controlador difuso para el ángulo de alabeo (MATLAB Fuzzy Logic Toolbox)

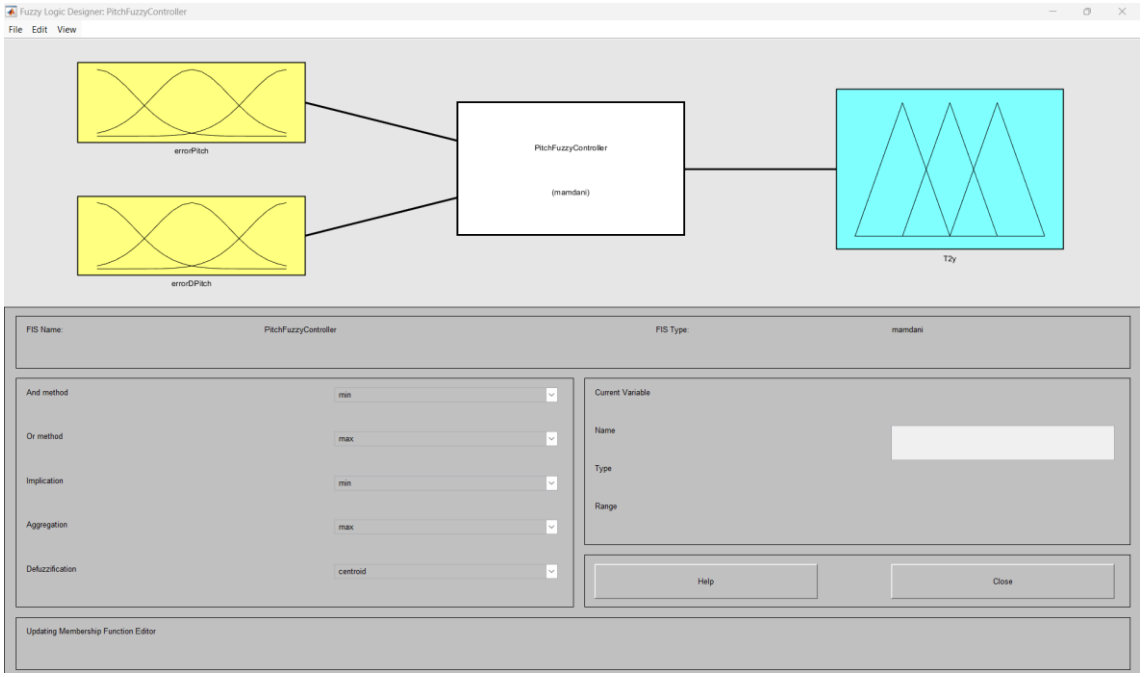


Ilustración 56: Esquema del controlador difuso para el ángulo de cabeceo (MATLAB Fuzzy Logic Toolbox)

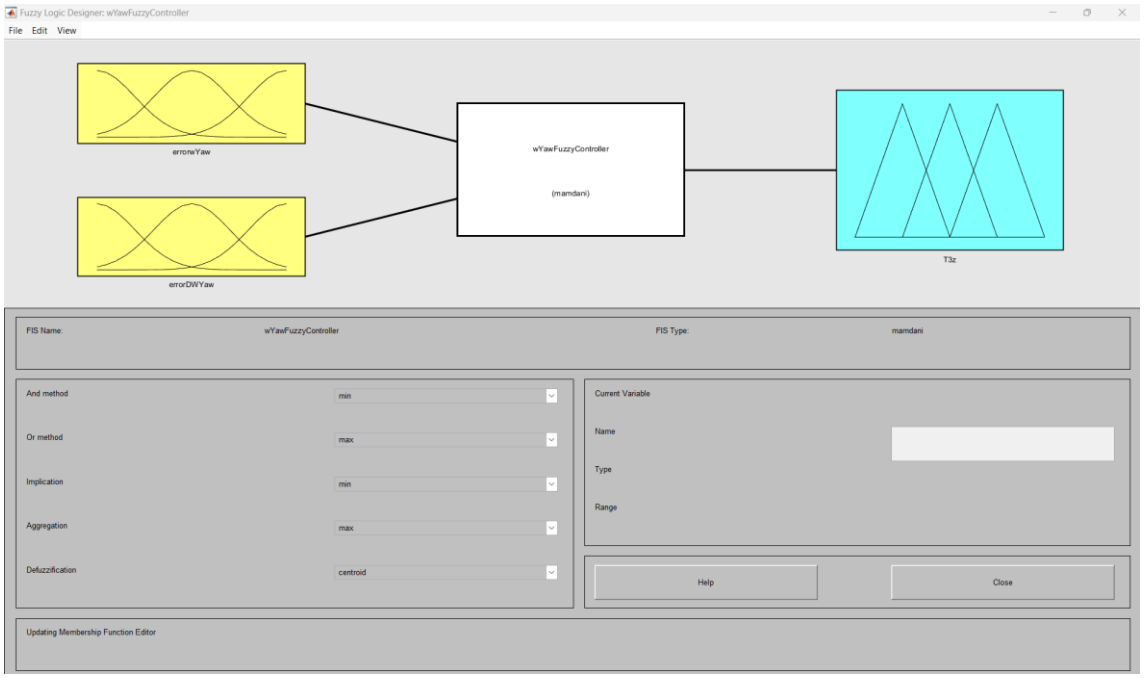


Ilustración 57: Esquema del controlador difuso para la velocidad angular de guiñada (MATLAB Fuzzy Logic Toolbox)

7.8. Respuesta ante entrada escalón

En este apartado se evaluará la respuesta del sistema ante entradas de tipo escalón.

El objetivo es observar y analizar varios parámetros clave que definen el desempeño del sistema de control.

Estos parámetros son:

- **Tiempo de establecimiento:** tiempo que tarda el sistema en alcanzar y mantenerse dentro de una banda de tolerancia del sistema, establecida entre el 2% y el 5% del valor de consigna.

- **Sobrepaso:** medida en la que la respuesta del sistema excede el valor estacionario.

- **Tiempo de subida:** tiempo que tarda la respuesta en pasar del 10% del valor estacionario a un 90% del valor estacionario.

- **Error en estado estacionario:** diferencia entre el valor de consigna y el valor alcanzado por la salida del sistema una vez estabilizada.

Para obtener estos valores, las respuestas se enviarán al *"workspace"* de MATLAB, donde se ejecutará el script *"DATA_STEP.m"*

- **Consumos y/o comandos enviados por el controlador:** A mayores comandos, mayor será el consumo de recursos del sistema para alcanzar la consigna

Las entradas escalón se aplicarán de forma independiente:

Velocidad en Z	Ángulo de Roll	Ángulo de Pitch	Velocidad Angular de Yaw
0.5 m/s	5 º	5 º	90 º/s

Tabla 18: Tabla de entradas escalón aplicadas

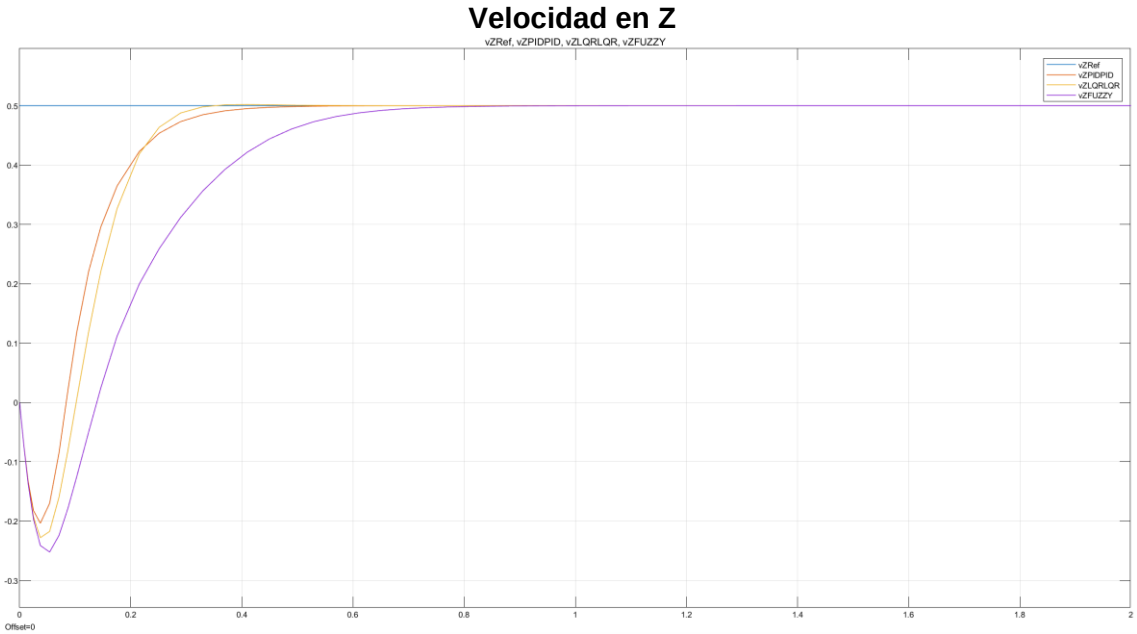


Ilustración 58: Gráfica que muestra la respuesta de la velocidad en Z en los 3 lazos de control al aplicar una entrada de referencia escalón de 0.5 m/s

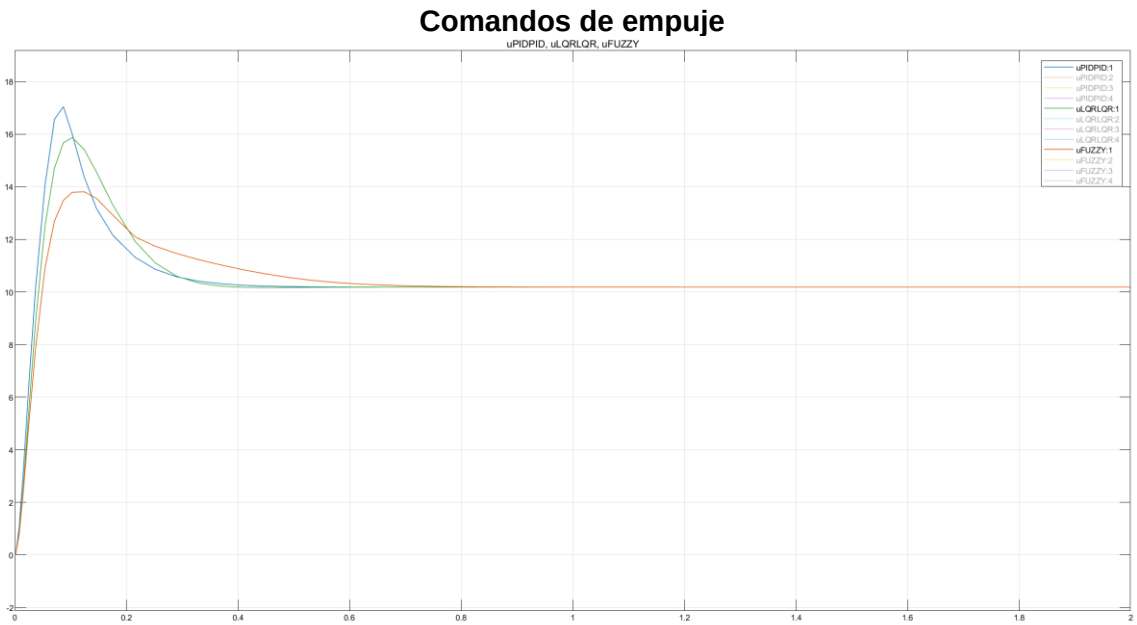


Ilustración 59: Gráfica que muestra las salidas de los controladores para alcanzar la consigna

Lazos	Tiempo de Subida	Tiempo de Establecimiento	Sobrepaso	Error en Estacionario
Lazo con controlador PID en cascada	0.155	0.361	0%	0
Lazo con controlador LQR en cascada	0.129	0.299	0.364 %	0
Lazo con controlador difuso	0.309	0.631	0%	0

Tabla 19: Tabla con datos de respuesta de la velocidad en Z para los controladores PID, LQR y difuso

A la vista de los datos y de las gráficas, podemos sacar algunas conclusiones para la dinámica de la velocidad en Z:

- El tiempo de establecimiento y de subida de la respuesta de la velocidad en Z en los lazos con controles PID en cascada y LQR en cascada es más rápida que para el lazo de control difuso.

- Se observa un mínimo sobrepaso con el lazo de control LQR en cascada y nulo con los otros dos controladores.

- Además, se observa que la respuesta del control PID en cascada es más agresiva, lo cual indican un mayor consumo de recursos que los lazos LQR en cascada y difusos.

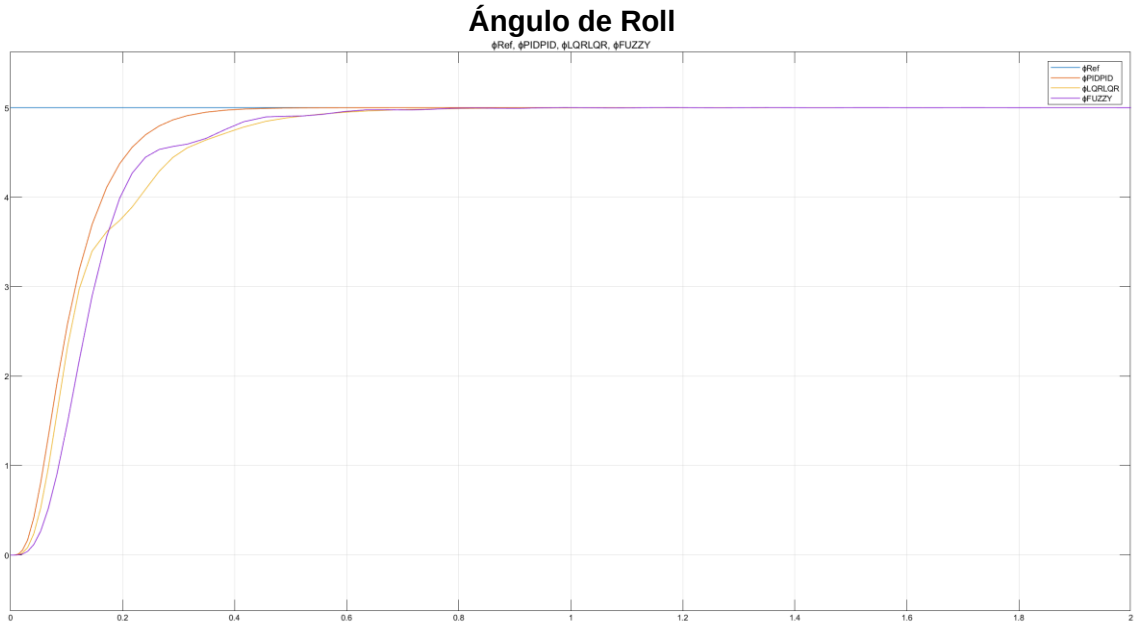


Ilustración 60: Gráfica que muestra la respuesta del ángulo de alabeo en los 3 lazos de control al aplicar una entrada de referencia escalón de 5 º

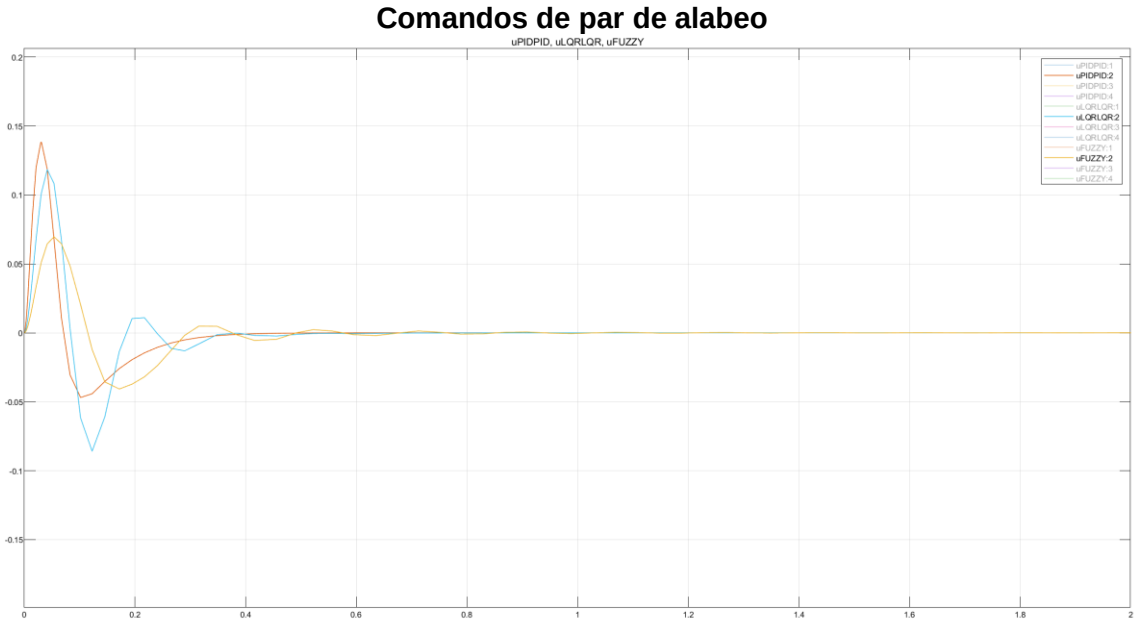


Ilustración 61: Gráfica que muestra las salidas de los controladores para alcanzar la consigna

Lazos	Tiempo de Subida	Tiempo de Establecimiento	Sobrepaso	Error en Estacionario
Lazo con controlador PID en cascada	0.166	0.309	0%	0
Lazo con controlador LQR en cascada	0.25	0.511	0%	0
Lazo con controlador difuso	0.19	0.468	0.025%	0

Tabla 20: Tabla con datos de respuesta del ángulo de alabeo para los controladores PID, LQR y difuso

A la vista de los datos y de las gráficas, podemos sacar algunas conclusiones para la dinámica del ángulo de alabeo:

- El tiempo de establecimiento y de subida de la respuesta del ángulo de alabeo en los lazos con controles PID en cascada y difuso es más rápida que para el lazo LQR en cascada.

- Se observa un mínimo sobrepaso en la respuesta con el lazo de control difuso y nulo con los otros dos controladores.

- Esta vez se observa que las respuestas de los controladores son similares, siendo más suave la del controlador difuso.

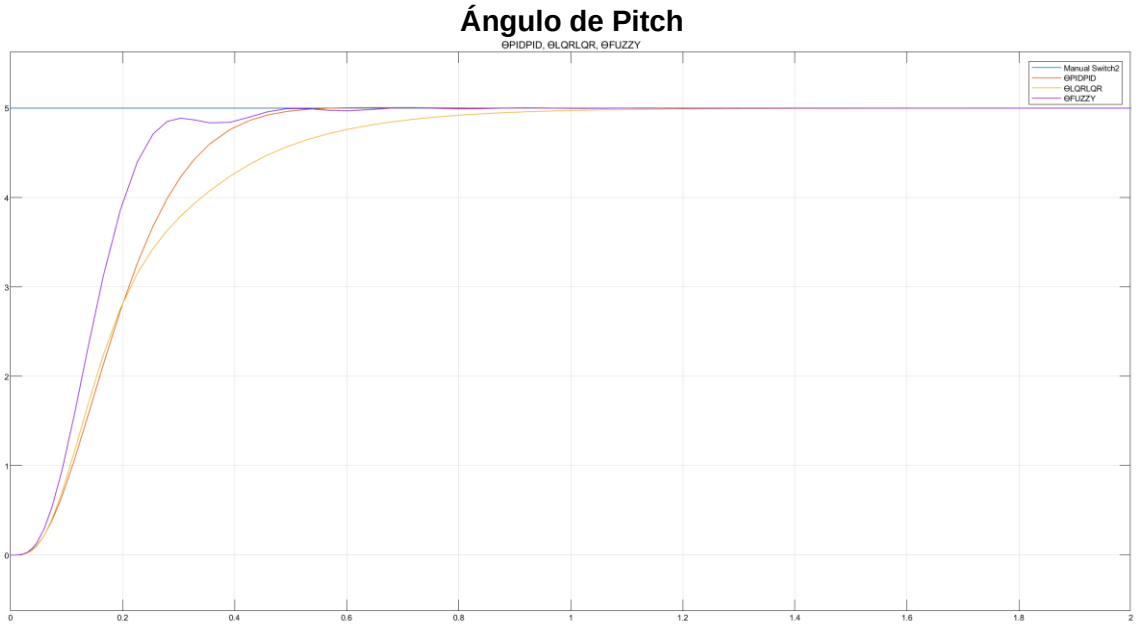


Ilustración 62: Gráfica que muestra la respuesta del ángulo de cabeceo en los 3 lazos de control al aplicar una entrada de referencia escalón de 5 º

Ilustración 63: Gráfica que muestra las salidas de los controladores para alcanzar la consigna

Lazos	Tiempo de Subida	Tiempo de Establecimiento	Sobrepaso	Error en Estacionario
Lazo con controlador PID en cascada	0.258	0.446	0.124%	0
Lazo con controlador LQR en cascada	0.388	0.758	0%	0
Lazo con controlador difuso	0.164	0.426	0.108%	0

Tabla 21: Tabla con datos de respuesta del ángulo de cabeceo para los controladores PID, LQR y difuso

A la vista de los datos y de las gráficas, podemos sacar algunas conclusiones para la dinámica del ángulo de cabeceo:

- El tiempo de establecimiento y de subida de la respuesta del ángulo de alabeo en los lazos con controles PID en cascada y difuso es más rápida que para el lazo LQR en cascada.

- Se observa un mínimo sobrepaso en la respuesta con los lazos de control PID en cascada y control difuso.

- Las respuestas de los controladores son similares, siendo más suave en esta ocasión la de los controladores PID en cascada y LQR en cascada.

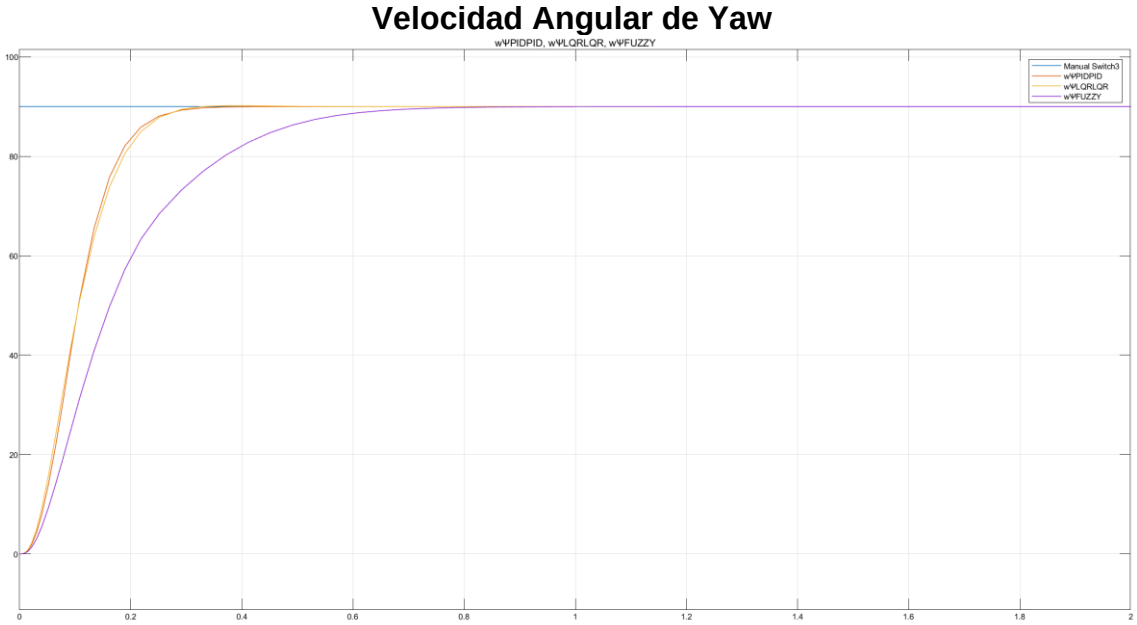


Ilustración 64: Gráfica que muestra la respuesta de la velocidad angular de guiñada en los 3 lazos de control al aplicar una entrada de referencia escalón de 90 º/s

Ilustración 65: Gráfica que muestra las salidas de los controladores para alcanzar la consigna

Lazos	Tiempo de Subida	Tiempo de Establecimiento	Sobrepaso	Error en Estacionario
Lazo con controlador PID en cascada	0.143	0.254	0%	0
Lazo con controlador LQR en cascada	0.153	0.260	0.199%	0
Lazo con controlador difuso	0.332	0.570	0%	0

Tabla 22: Tabla con datos de respuesta de la velocidad angular de guiñada para los controladores PID, LQR y difuso

A la vista de los datos y de las gráficas, podemos sacar algunas conclusiones para la dinámica de la velocidad angular de guiñada:

- El tiempo de establecimiento y de subida de la respuesta de la velocidad angular de guiñada en los lazos con controles PID en cascada y LQR en cascada es más rápida que para el lazo de control difuso.

- Se observa un mínimo sobrepaso con el lazo de control LQR en cascada y nulo con los otros dos controladores.

- Además, se observa que la respuesta del control PID en cascada y LQR en cascada son similares, siendo la respuesta del controlador difuso más suave.

7.9. Respuesta ante entrada rampa

En este apartado se evaluará la respuesta del sistema ante entradas de tipo rampa. El objetivo es observar y analizar cómo responde el sistema al seguimiento de consignas.

Es interesante analizar:

- **Error de seguimiento:** diferencia entre el valor de rampa de referencia y la respuesta del sistema.

Las entradas rampa se aplicarán de forma independiente:

Velocidad en Z	Ángulo de Roll	Ángulo de Pitch	Velocidad Angular de Yaw
Pendiente = 0.1 m/s/s	Pendiente = 0.3 º/s	Pendiente = 0.3 º/s	Pendiente = 10 º/s/s

Tabla 23: Tabla de entradas rampa aplicadas

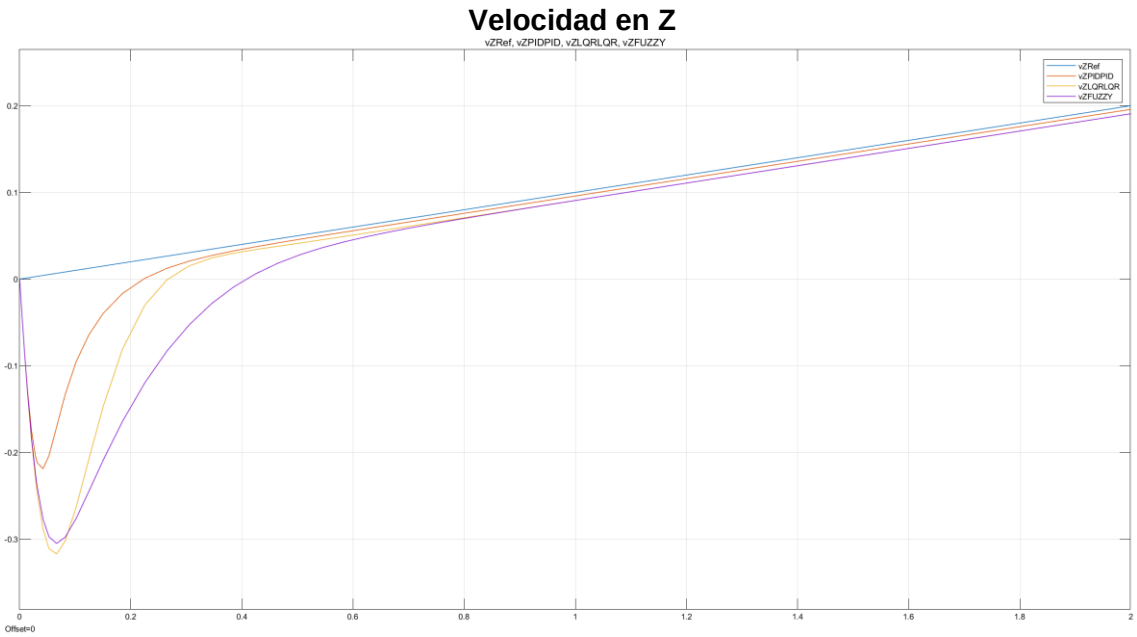


Ilustración 66: Gráfica que muestra la respuesta de la velocidad en Z en los 3 lazos de control al aplicar una entrada de referencia rampa de 0.1 m/s/s

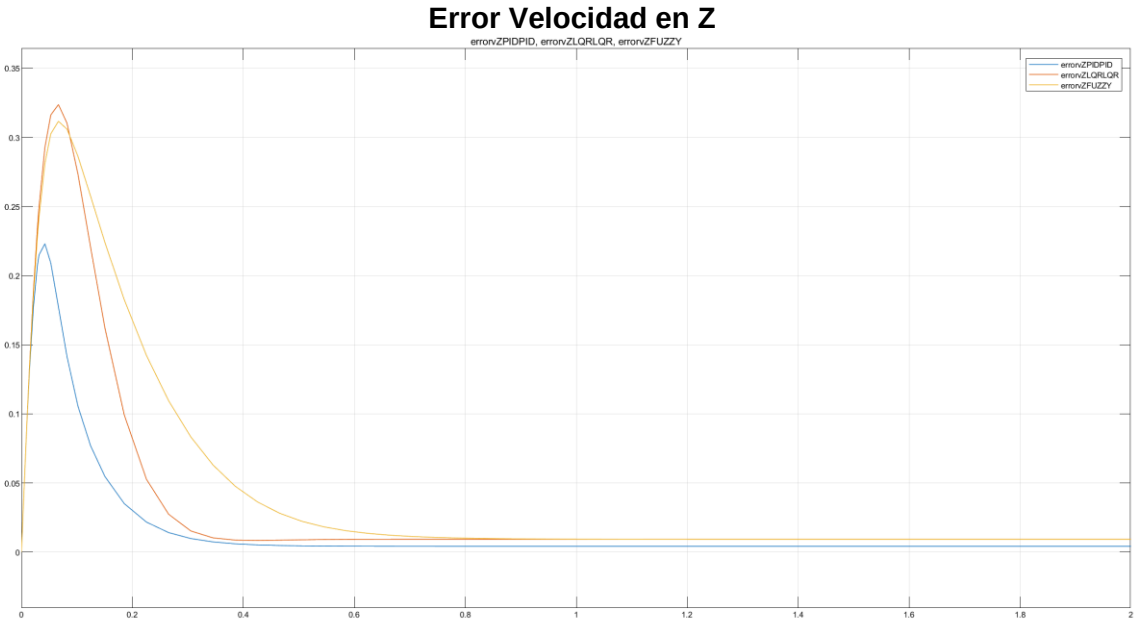


Ilustración 67: Gráfica que muestra el error entre la respuesta en velocidad Z en los 3 lazos de control y la referencia rampa

Tiempo de Subida	Error constante en el seguimiento de la rampa
Lazo con controlador PID en cascada	0.00409
Lazo con controlador LQR en cascada	0.00284
Lazo con controlador difuso	0.00923

Tabla 24: Tabla de errores en el seguimiento de la entrada en rampa con los 3 lazos de control

A la vista de las gráficas y los resultados, la configuración que permite seguir una entrada en rampa con el menor error es el controlador LQR en cascada

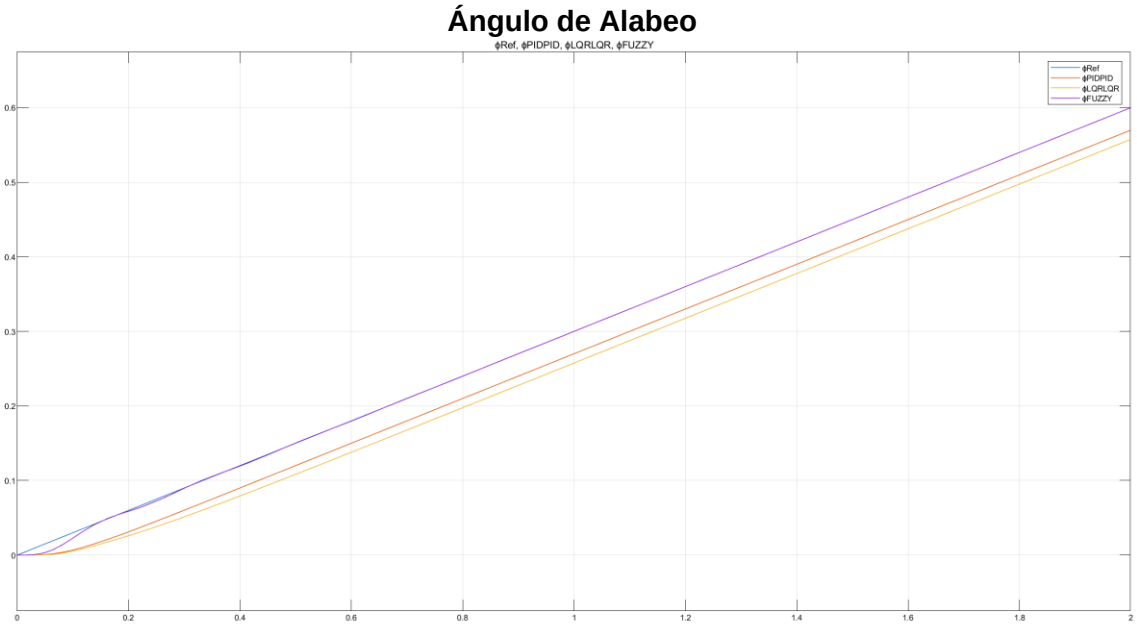


Ilustración 68: Gráfica que muestra la respuesta del ángulo de alabeo en los 3 lazos de control al aplicar una entrada de referencia rampa de 0.3 º/s

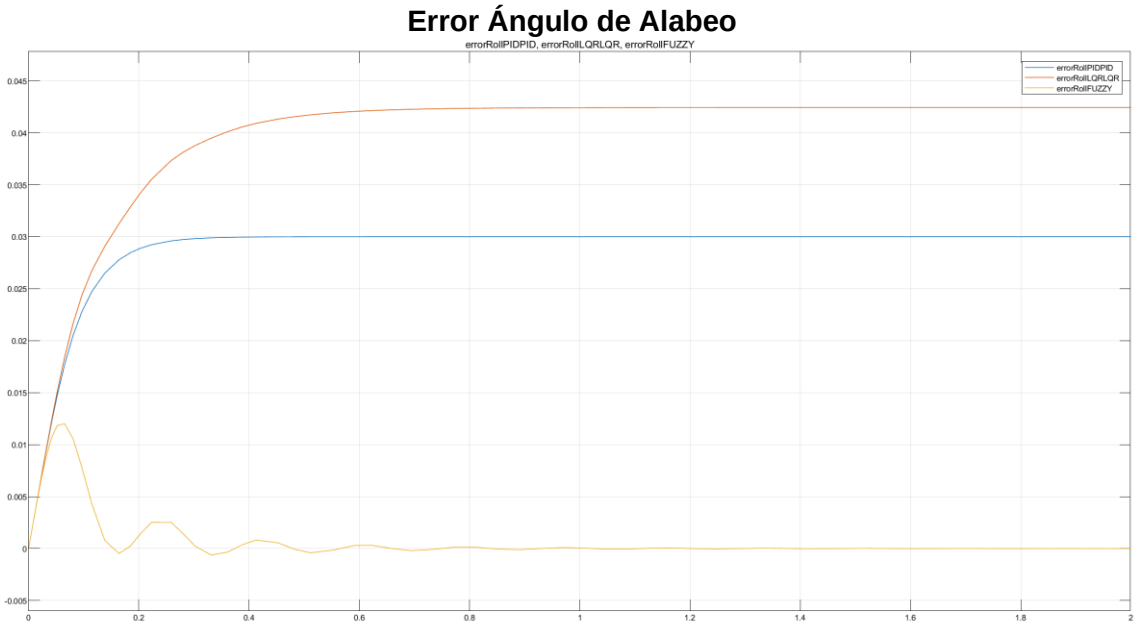


Ilustración 69: Gráfica que muestra el error entre la respuesta en alabeo en los 3 lazos de control y la referencia rampa

Tiempo de Subida	Error constante en el seguimiento de la rampa
Lazo con controlador PID en cascada	0.030
Lazo con controlador LQR en cascada	0.424
Lazo con controlador difuso	0

Tabla 25: Tabla de errores en el seguimiento de la entrada en rampa con los 3 lazos de control

A la vista de las gráficas y los resultados, la configuración que permite seguir una entrada en rampa con el menor error es el controlador difuso

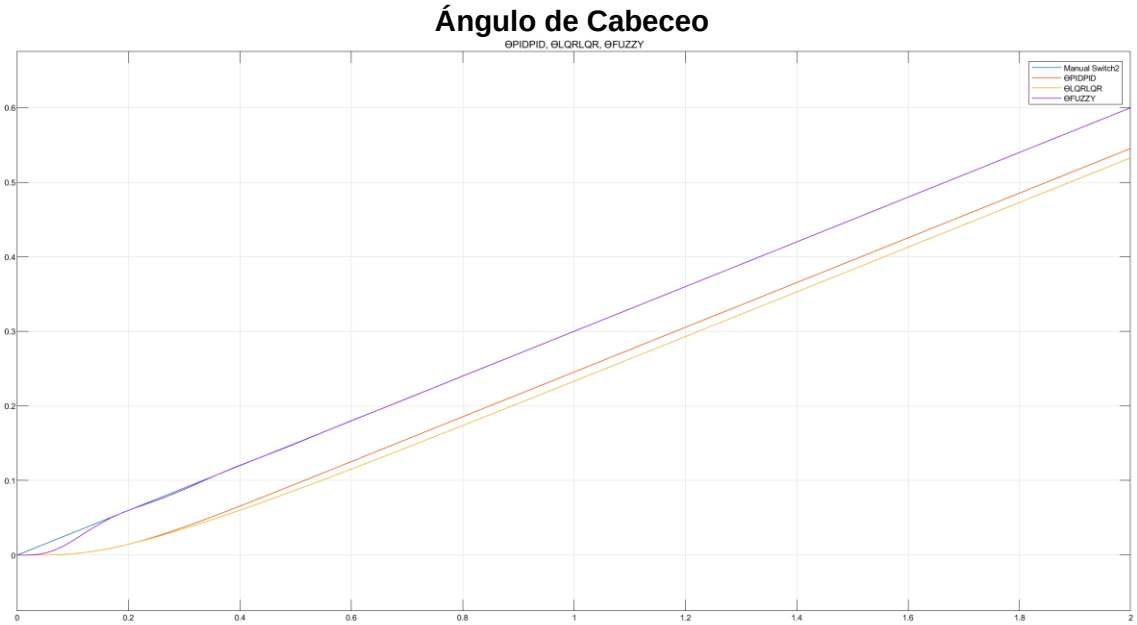


Ilustración 70: Gráfica que muestra la respuesta del ángulo de cabeceo en los 3 lazos de control al aplicar una entrada de referencia rampa de 0.3 º/s

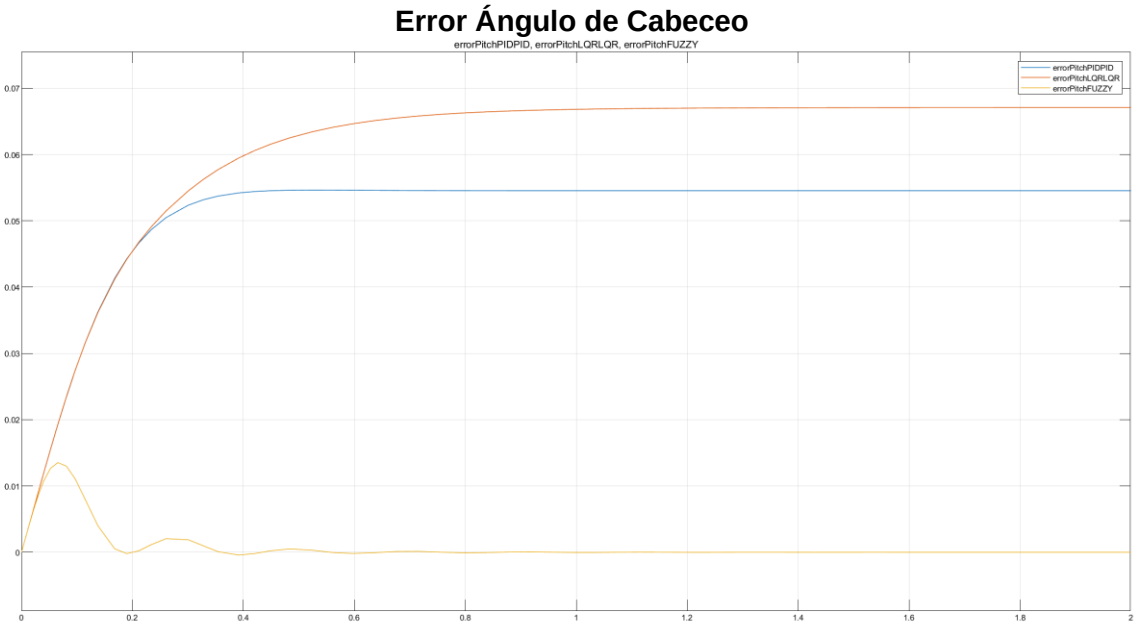


Ilustración 71: Gráfica que muestra el error entre la respuesta en cabeceo en los 3 lazos de control y la referencia rampa

Tiempo de Subida	Error constante en el seguimiento de la rampa
Lazo con controlador PID en cascada	0.0545
Lazo con controlador LQR en cascada	0.0671
Lazo con controlador difuso	0

Tabla 26: Tabla de errores en el seguimiento de la entrada en rampa con los 3 lazos de control

A la vista de las gráficas y los resultados, la configuración que permite seguir una entrada en rampa con el menor error es el controlador difuso

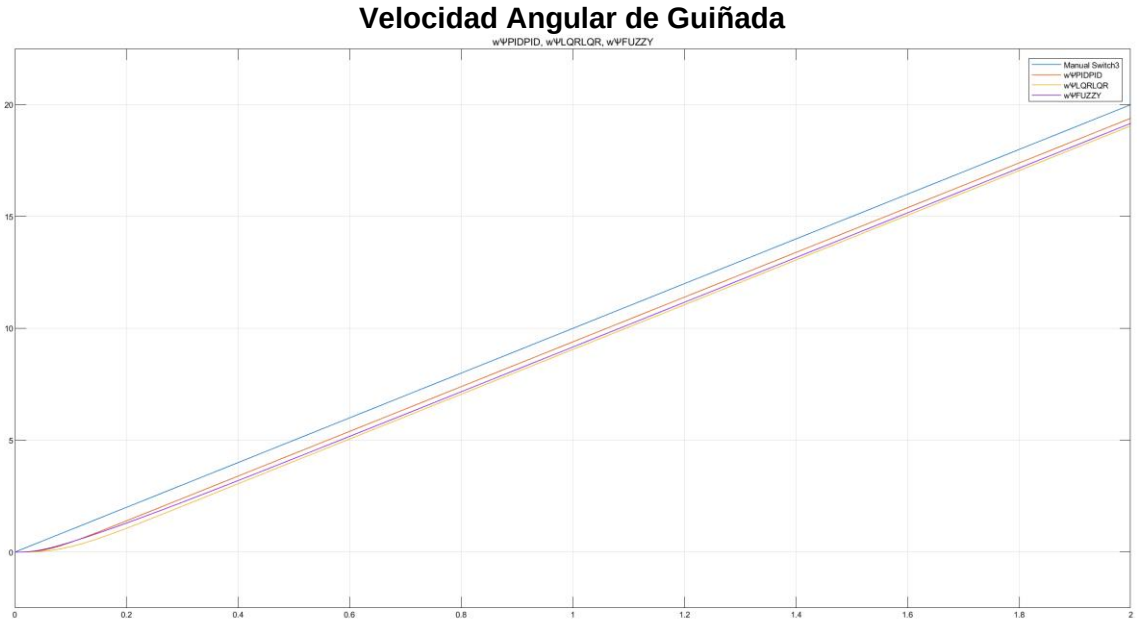


Ilustración 72: Gráfica que muestra la respuesta de la velocidad angular en los 3 lazos de control al aplicar una entrada de referencia rampa de 10 º/s/s

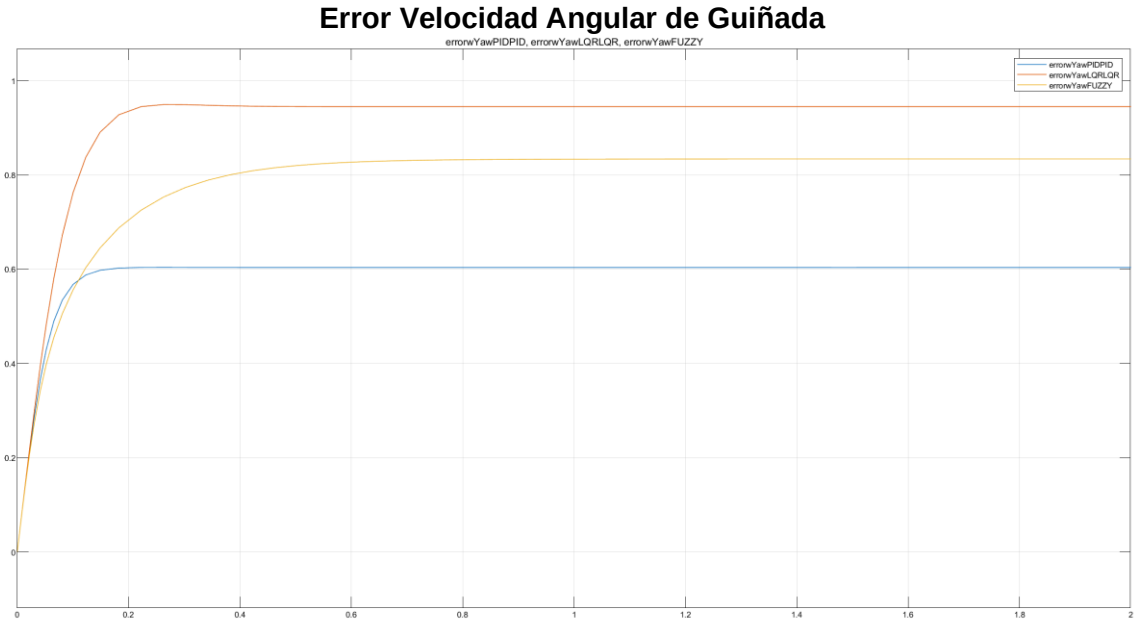


Ilustración 73: Gráfica que muestra el error entre la respuesta en velocidad angular de guiñada en los 3 lazos de control y la referencia rampa

Tiempo de Subida	Error constante en el seguimiento de la rampa
Lazo con controlador PID en cascada	0.6037
Lazo con controlador LQR en cascada	0.9450
Lazo con controlador difuso	0.8338

Tabla 27: Tabla de errores en el seguimiento de la entrada en rampa con los 3 lazos de control

A la vista de las gráficas y los resultados, la configuración que permite seguir una entrada en rampa con el menor error es el controlador PID en cascada

7.10. Respuesta ante perturbaciones

En este apartado se evaluará la respuesta del sistema ante perturbaciones. El objetivo es observar y analizar cómo responde el sistema ante diferentes perturbaciones que intentarán desestabilizar al dron respecto de un vuelo estacionario.
Estas perturbaciones serán aceleraciones lineales y angulares:

- Aceleración angular aplicada en el alabeo
- Aceleración angular aplicada en el cabeceo
- Aceleración angular aplicada en la guiñada
- Desestabilización del dron debido a aceleraciones externas verticales
- Ruido de baja frecuencia (ruido de proceso)
- Ruido blanco de alta frecuencia (ruido de medida)

Estas perturbaciones se aplicarán al modelo linealizado del cuadricóptero de la siguiente manera:

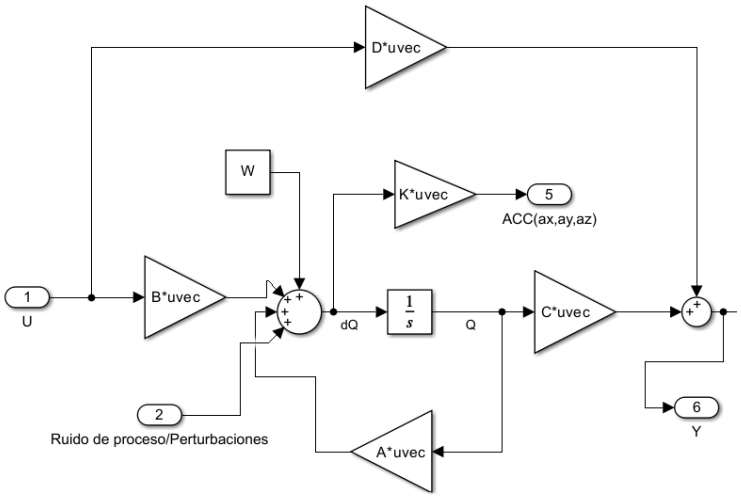


Ilustración 74: Detalle del sistema de estados del cuadricóptero. Nótese la enetrada de perturbaciones en el punto de suma

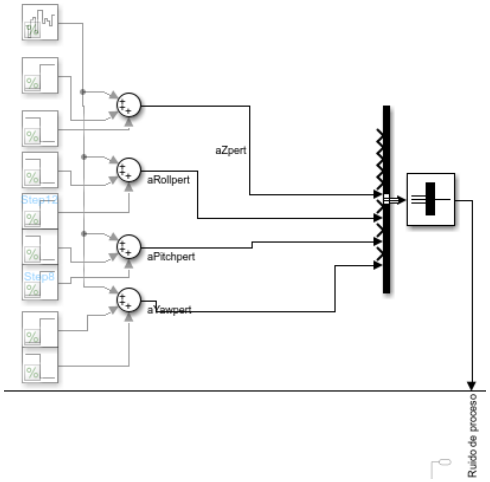


Ilustración 75: entradas de perturbación

- Primero se evaluará la respuesta del sistema ante entradas nulas. Esto significa mantener el UAV en estado estacionario en equilibrio, contrarrestando los efectos gravitatorios. Esta dinámica solo afecta al lazo de la velocidad en Z

- En la simulación no se considera un suelo físico. Podría suponerse que se suelta desde el aire con los siguientes comandos:

Velocidad en Z	Ángulo de Roll	Ángulo de Pitch	Velocidad Angular de Yaw
0 m/s	0 º	0 º	0 º/s

Tabla 28: Tabla de entradas aplicadas

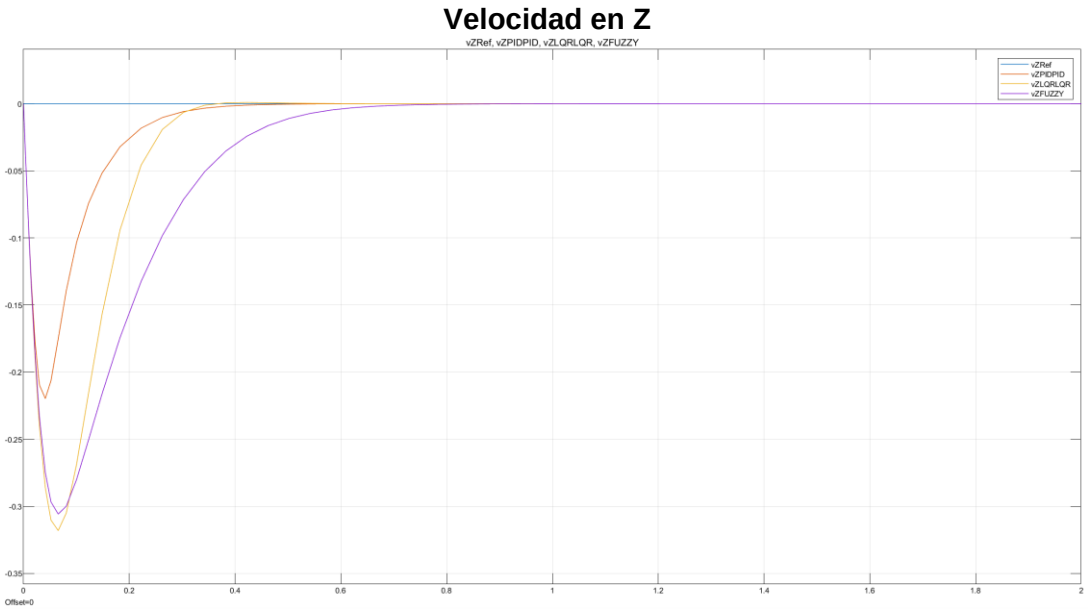


Ilustración 76: Respuesta de la velocidad en Z ante una entrada de 0 m/s

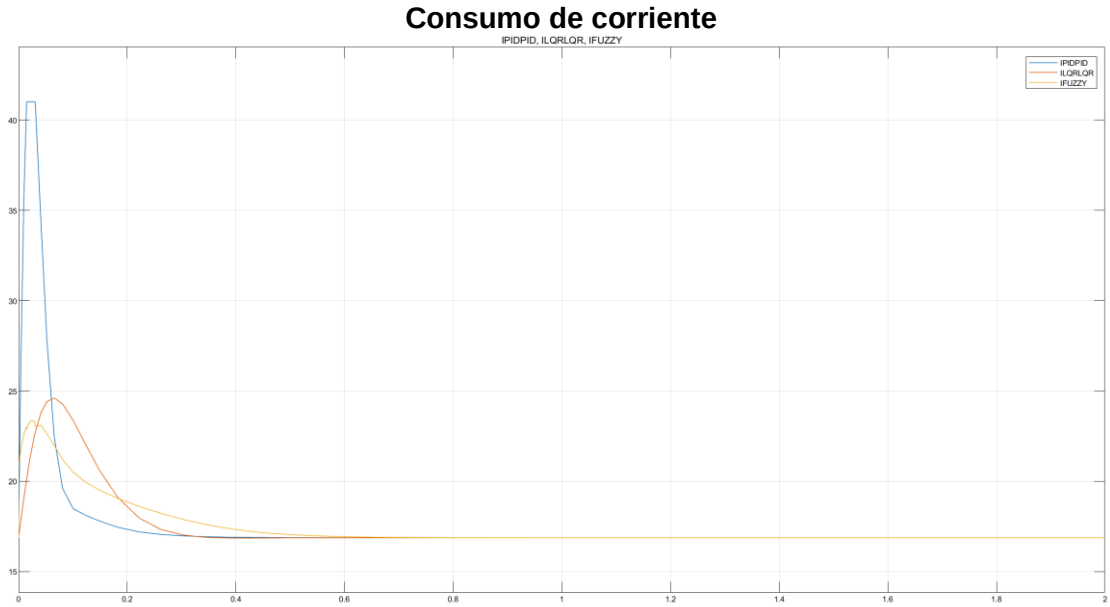


Ilustración 77: Consumo de corriente para estabilizar el dron en vuelo estacionario

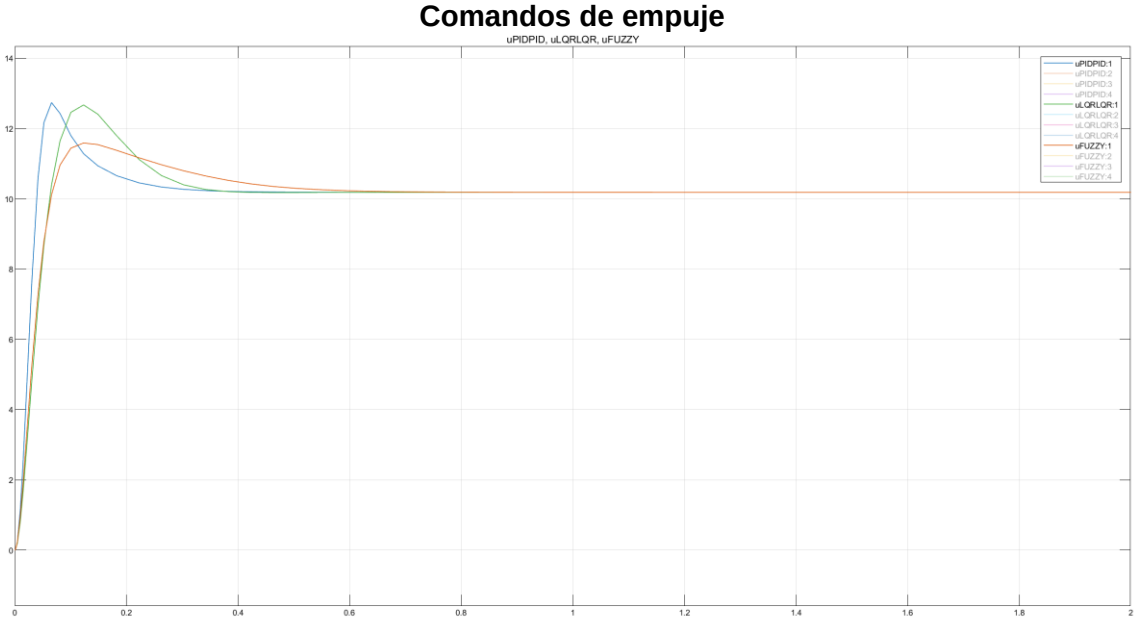


Ilustración 78: Comandos de salida de los controladores para el vuelo estacionario

Podemos observar que para mantener el cuadricóptero en equilibrio en el aire e impedir que caiga, se consumen de forma constante 16 amperios, con un pico inicial muy superior en el controlador PID en cascada. Sin embargo, este controlador estabiliza antes la respuesta del cuadricóptero. En cuanto al resto de controladores, el LQR en cascada y el difuso consumen menos recursos y tardan un poco más de tiempo en estabilizar la respuesta.

- En segundo lugar, se evaluará la respuesta del sistema ante perturbaciones. Estas perturbaciones se pueden traducir en aceleraciones perturbadoras sobre el modelo, tanto lineales como angulares. El objetivo de los controladores será el de contrarrestar estas perturbaciones y mantener la referencia de control.

- Aceleraciones desestabilizadoras / perturbadoras aplicadas:

aZ	**aRoll**	**aPitch**	**aYaw**
Pulso de 5 m/s^2 durante 0.5 s	Pulso de -5 rad/s^2 Durante 0.5 s	Pulso de -5 rad/s^2 Durante 0.5 s	Pulso de 10 rad/s^2 Durante 0.5 s

Tabla 29: Tabla de perturbaciones aplicadas

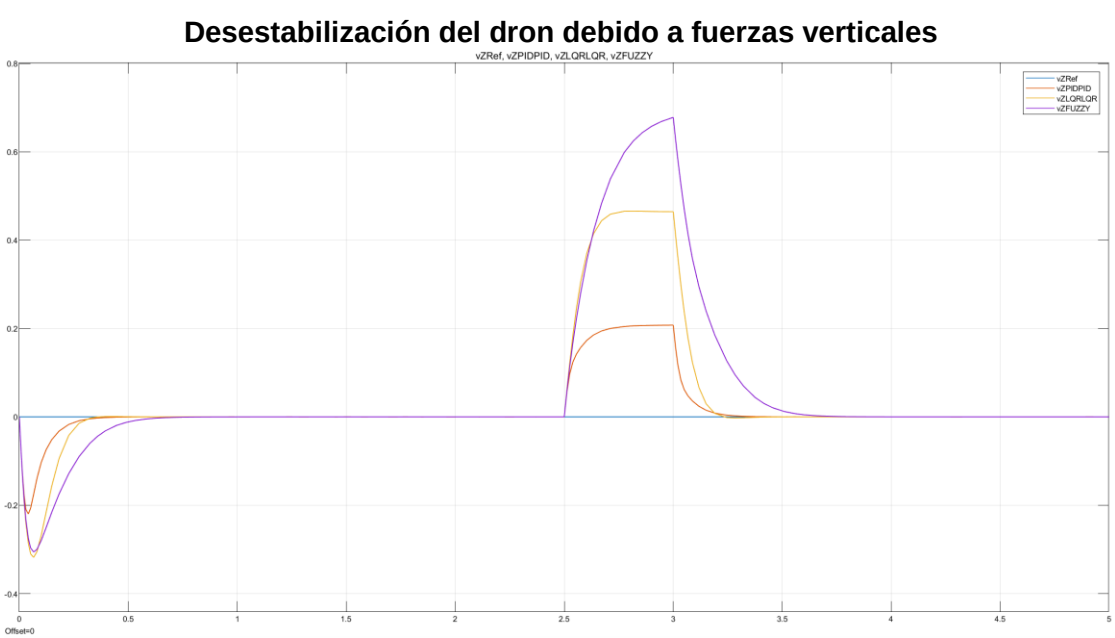


Ilustración 79: Respuesta de la velocidad en Z ante una perturbación de 0.5 segundos y 5 m/s^2

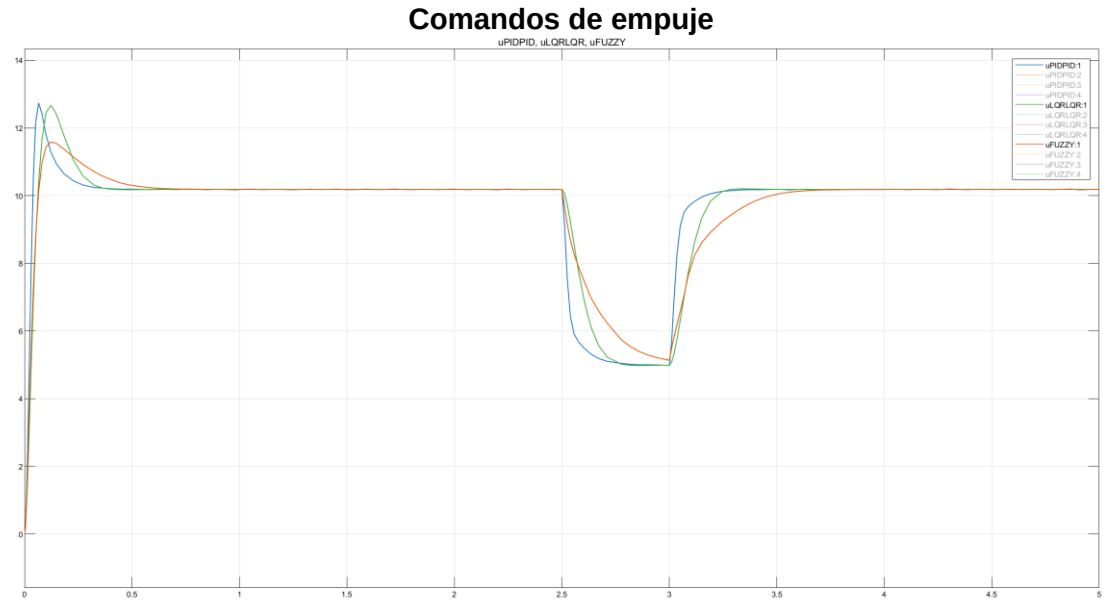


Ilustración 80: Salida de los controladores de la velocidad en Z para contrarrestar la perturbación aplicada

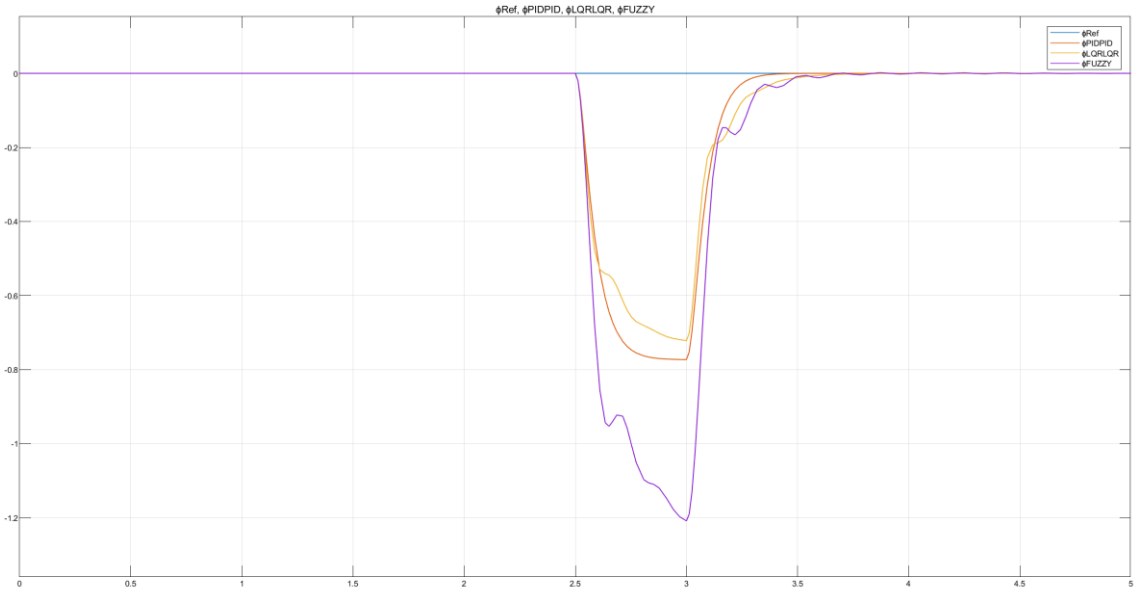


Ilustración 81: Respuesta del ángulo de alabeo ante una perturbación de 0.5 segundos y -5 rad/s^2

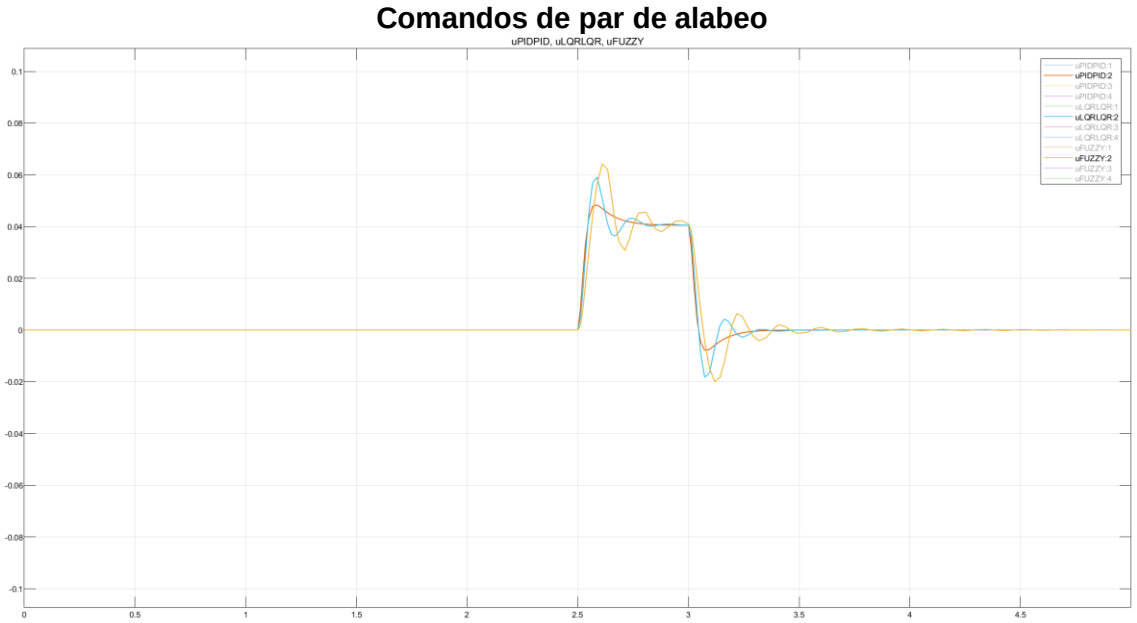


Ilustración 82: Salida de los controladores del ángulo de alabeo para contrarrestar la perturbación aplicada

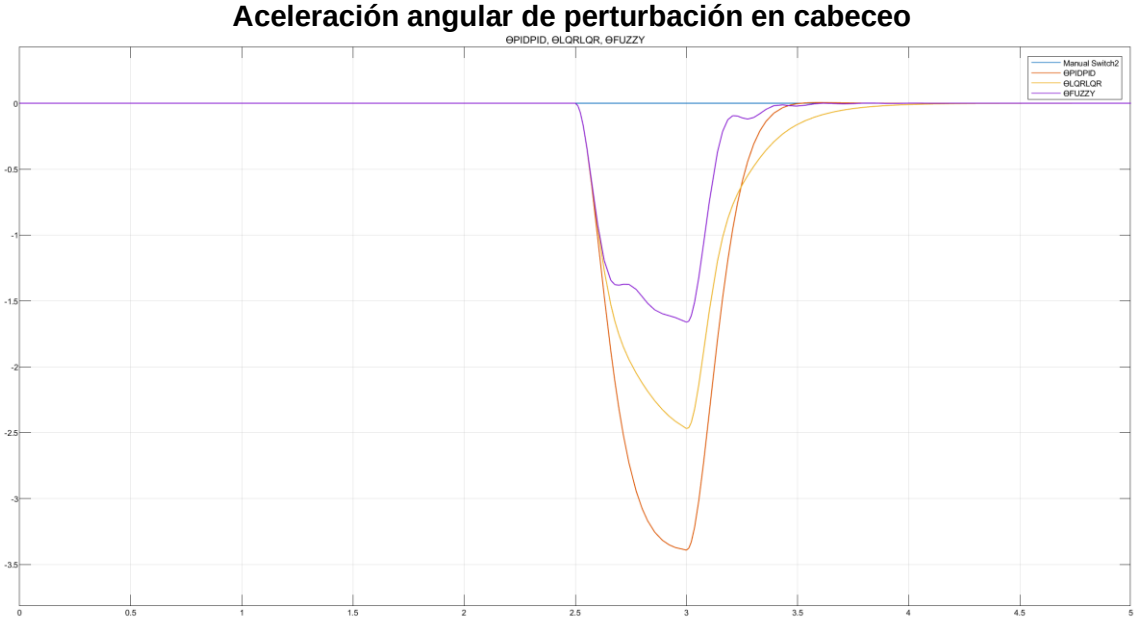


Ilustración 83: Respuesta del ángulo de cabeceo ante una perturbación de 0.5 segundos y -5 rad/s^2

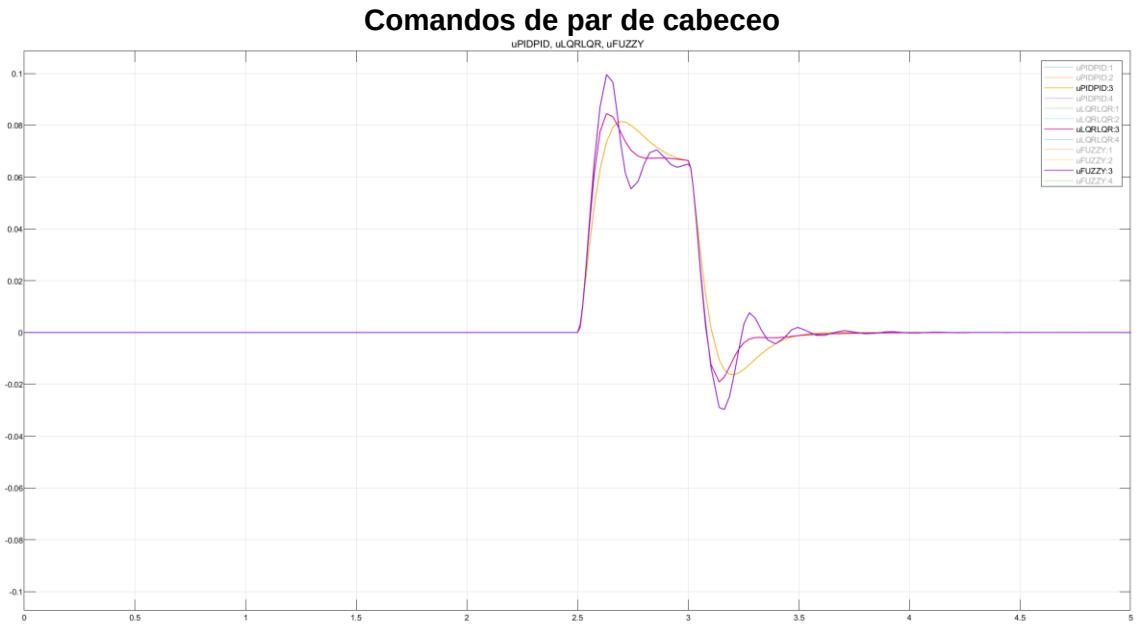


Ilustración 84: Salida de los controladores del ángulo de cabeceo para contrarrestar la perturbación aplicada

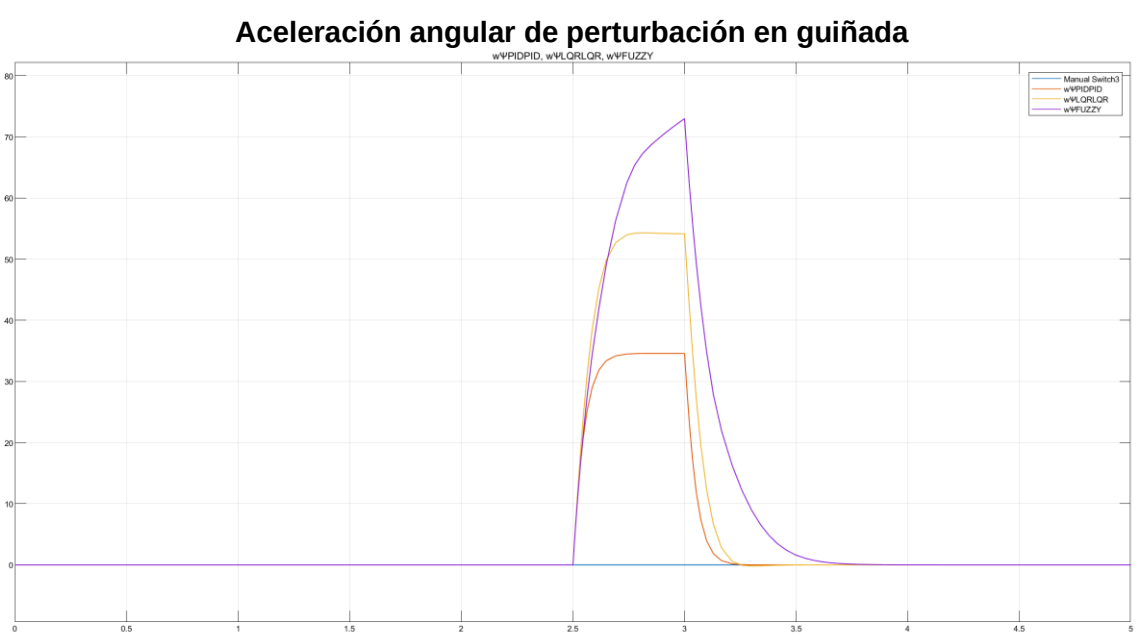


Ilustración 85: Respuesta de la velocidad angular de guiñada ante una perturbación de 0.5 segundos y 10 rad/s^2

Ilustración 86: Salida de los controladores de la velocidad angular de guiñada para contrarrestar la perturbación aplicada

De las anteriores gráficas, podemos deducir que, salvo para el lazo de control del ángulo de cabeceo, los controladores PID en cascada y LQR en cascada responden y estabilizan antes las perturbaciones. Aunque apliquen comandos ligeramente más agresivos, estos no se diferencian en exceso. Se observa que la respuesta del controlador difuso es más suave.

En cuanto al ángulo de cabeceo parece responder mejor con el controlador difuso.

- ***Ruido de proceso de baja frecuencia: turbulencias aleatorias de 1 Hz:***

Los gráficos siguientes muestran las respuestas de la velocidad en Z, ángulo de alabeo, ángulo de cabeceo y la velocidad angular de guiñada en presencia de ruido de proceso de frecuencia 1 Hz y amplitud de 0.3 para todos los lazos.

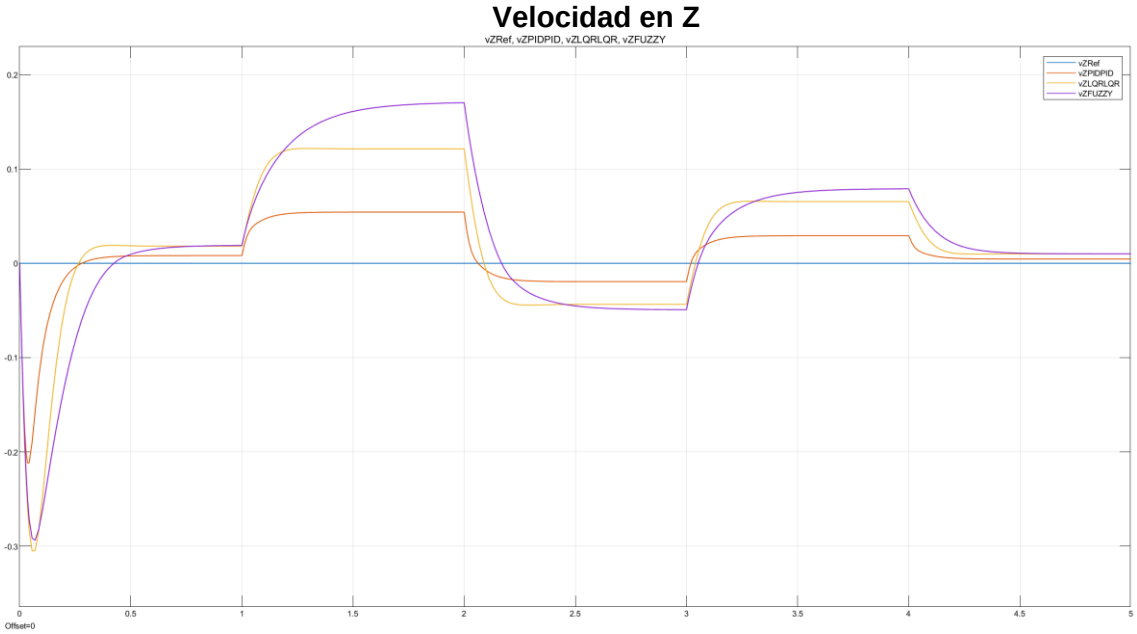


Ilustración 87: Respuesta de la velocidad en Z ante un ruido de proceso de 1 Hz

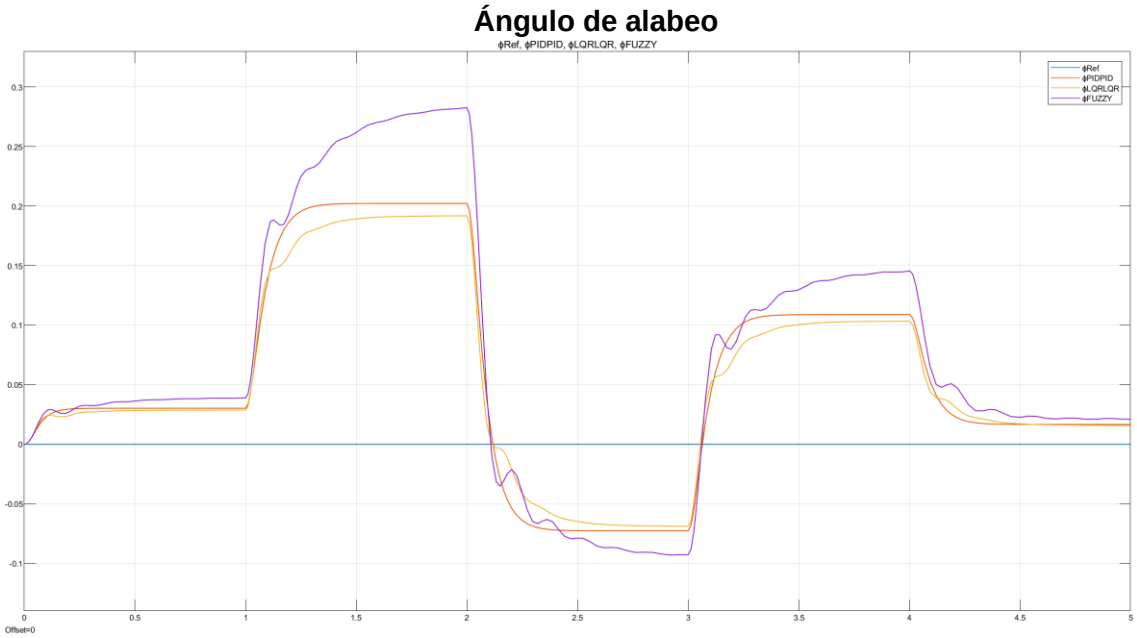


Ilustración 88: Respuesta del ángulo de alabeo ante un ruido de proceso de 1 Hz

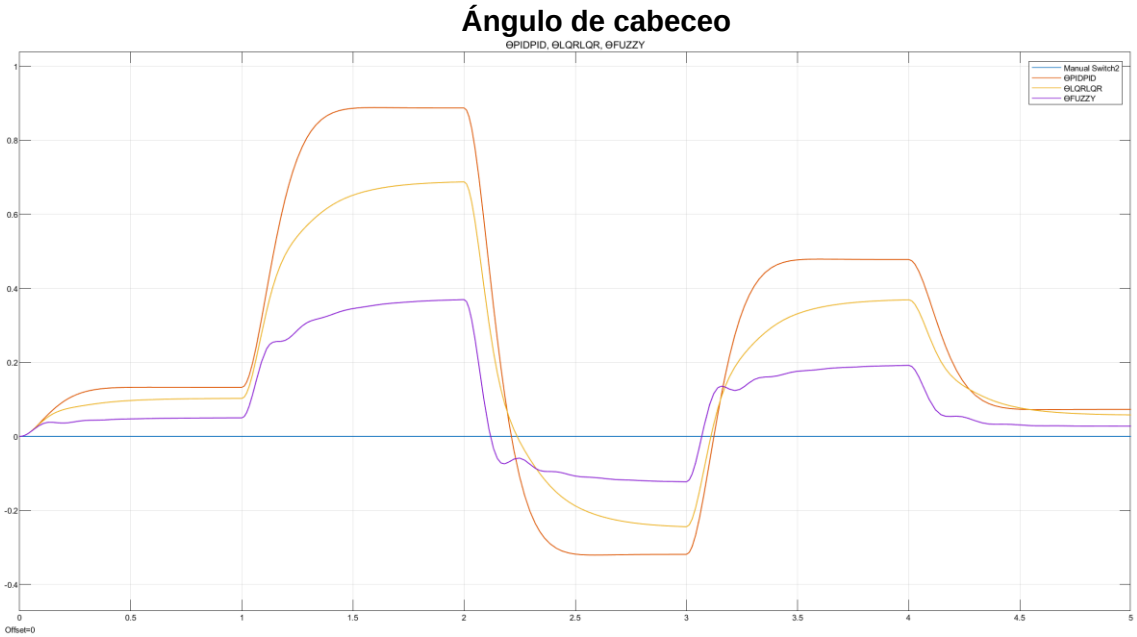


Ilustración 89: Respuesta del ángulo de cabeceo ante un ruido de proceso de 1 Hz

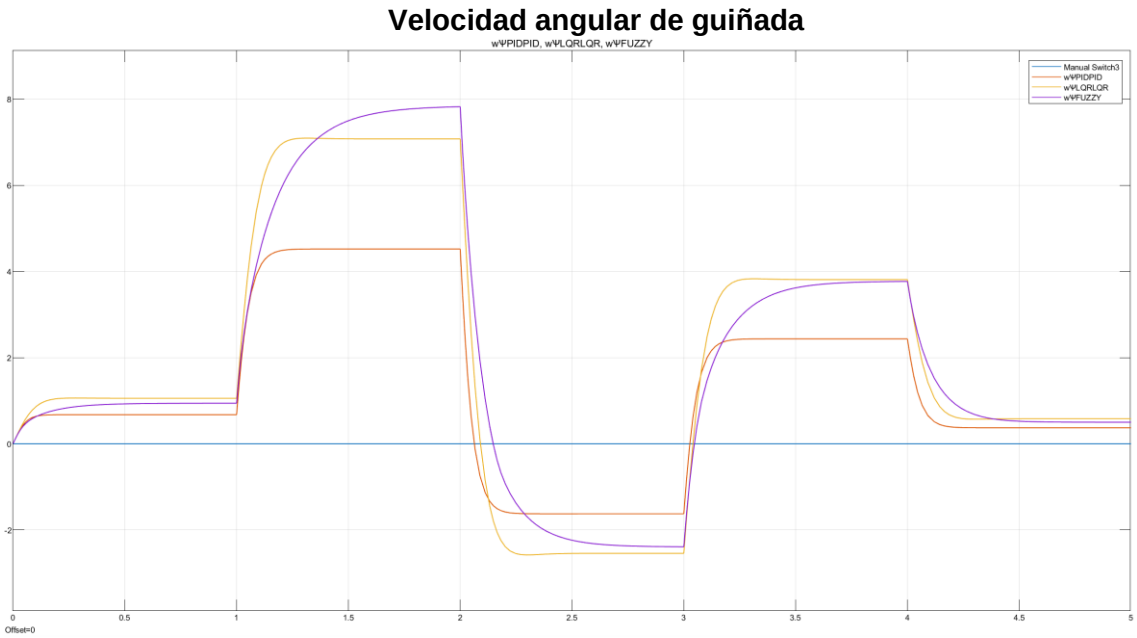


Ilustración 90: Respuesta de la velocidad angular de guiñada ante un ruido de proceso de 1 Hz

Al igual que con la respuesta ante perturbaciones, podemos observar que, de forma general, salvo en la respuesta del ángulo de cabeceo, los lazos de control PID en casaca y LQR en casca responden y tienden a estabilizar el sistema más rápido.

- ***Ruido de proceso de media / alta frecuencia: turbulencias aleatorias de 10 Hz:***

Los gráficos siguientes muestran las respuestas de la velocidad en Z, ángulo de alabeo, ángulo de cabeceo y la velocidad angular de guiñada en presencia de ruido de proceso de frecuencia 10 Hz y amplitud de 0.3 para todos los lazos.

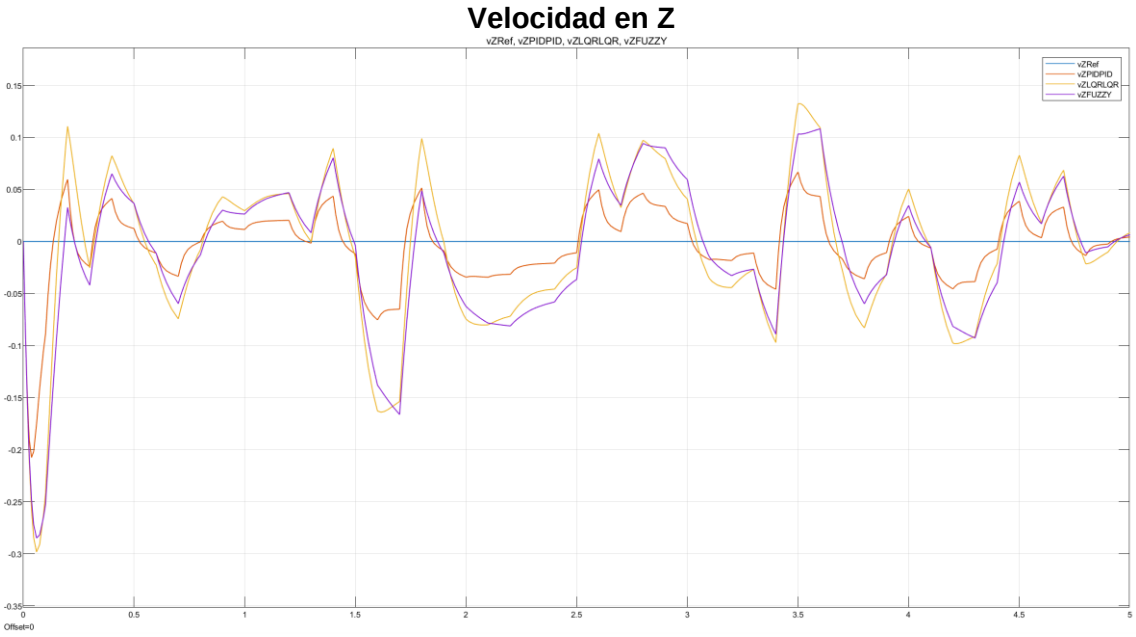


Ilustración 91: Respuesta de la velocidad en Z ante un ruido de proceso de 10 Hz

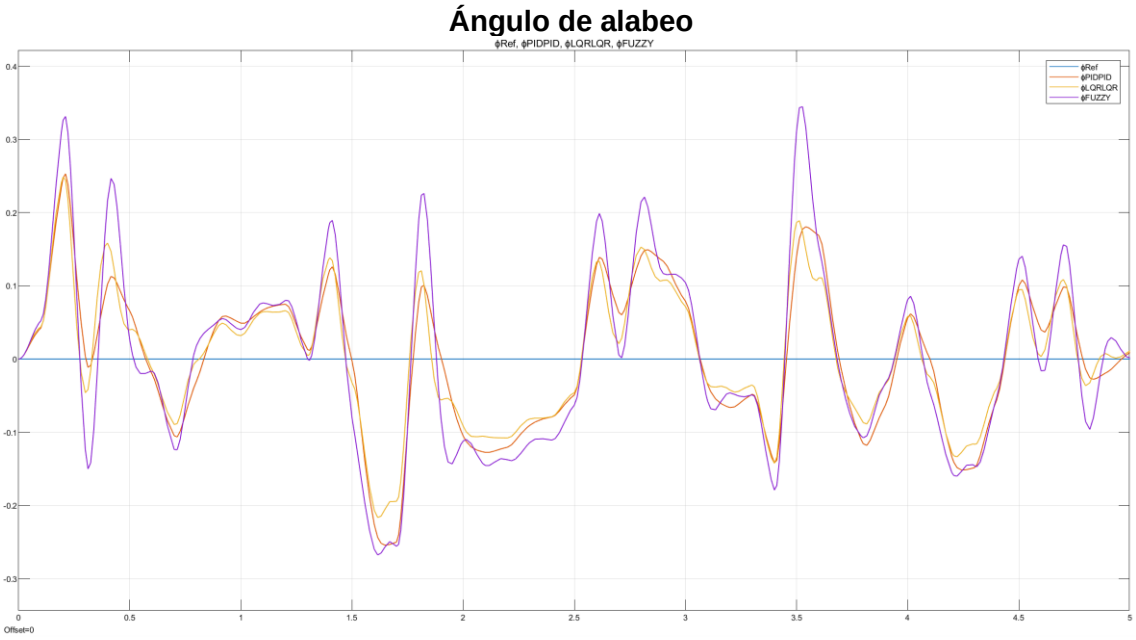


Ilustración 92: Respuesta del ángulo de alabeo ante un ruido de proceso de 10 Hz

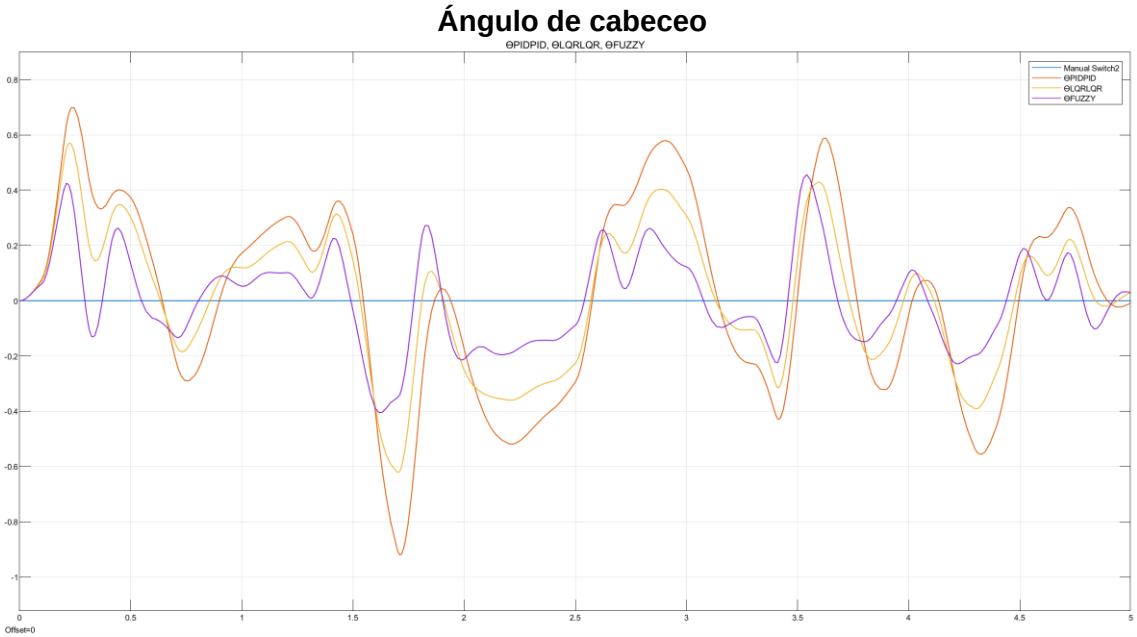


Ilustración 93: Respuesta del ángulo de cabeceo ante un ruido de proceso de 10 Hz

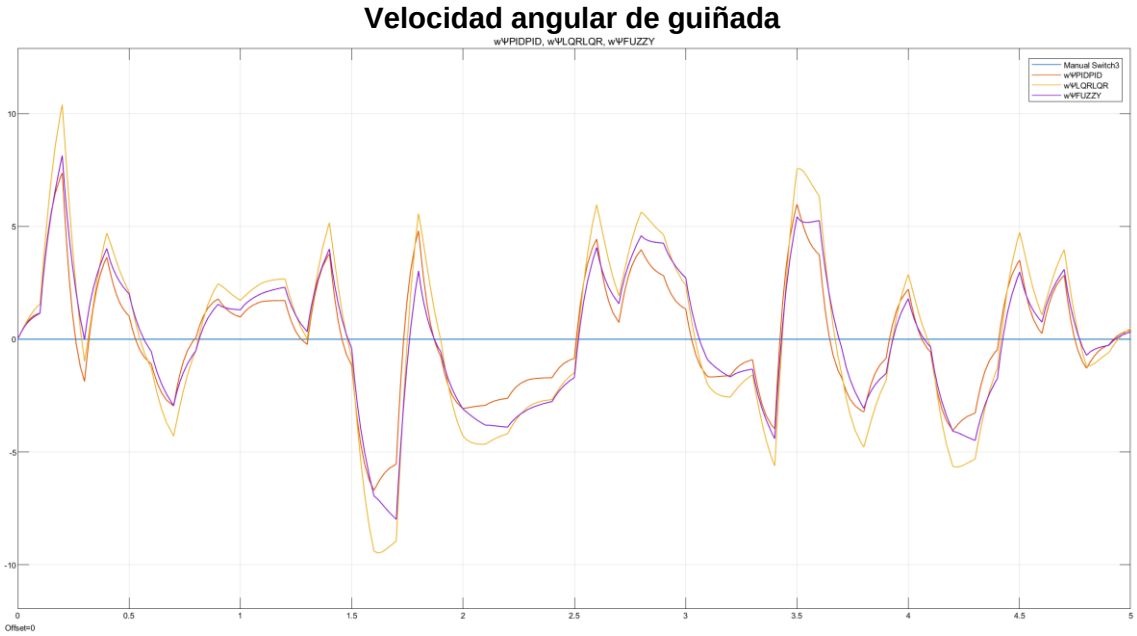


Ilustración 94: Respuesta de la velocidad angular de guiñada ante un ruido de proceso de 10 Hz

Podemos observar que, de forma general, salvo en la respuesta del ángulo de cabeceo, el lazo de control de PID en cascada puede manejar mejor el ruido de proceso de más frecuencia, con una menor amplitud de oscilación entorno a la consigna que el controlador difuso y el LQR en cascada.

- ***Ruido de medida:*** Se aplicará un ruido de medida a los ángulos de alabeo y cabeceo de 0.017 radianes (0. 97º) con una frecuencia de 100 Hz y para la velocidad angular de guiñada de 0.017 radianes/s (0. 97º/s) con una frecuencia de 100 Hz. El objetivo es analizar cómo responde el sistema ante ruidos de medida. Estos valores, aproximadamente, se corresponden con el ruido que recoge el sensor por cada muestra.

    - Sin la aplicación de filtros paso bajo en las lecturas, el ruido de medida torna al sistema inestable e incluso Simulink devuelve errores debido a valores no finitos en la simulación.

    - Se añade un filtro paso bajo de 15 Hz a los ángulos de alabeo y cabeceo, así como a la velocidad angular de guiñada para evitar que el sistema se vuelva inestable.

Ángulo de alabeo con filtro paso bajo de 15 Hz aplicado

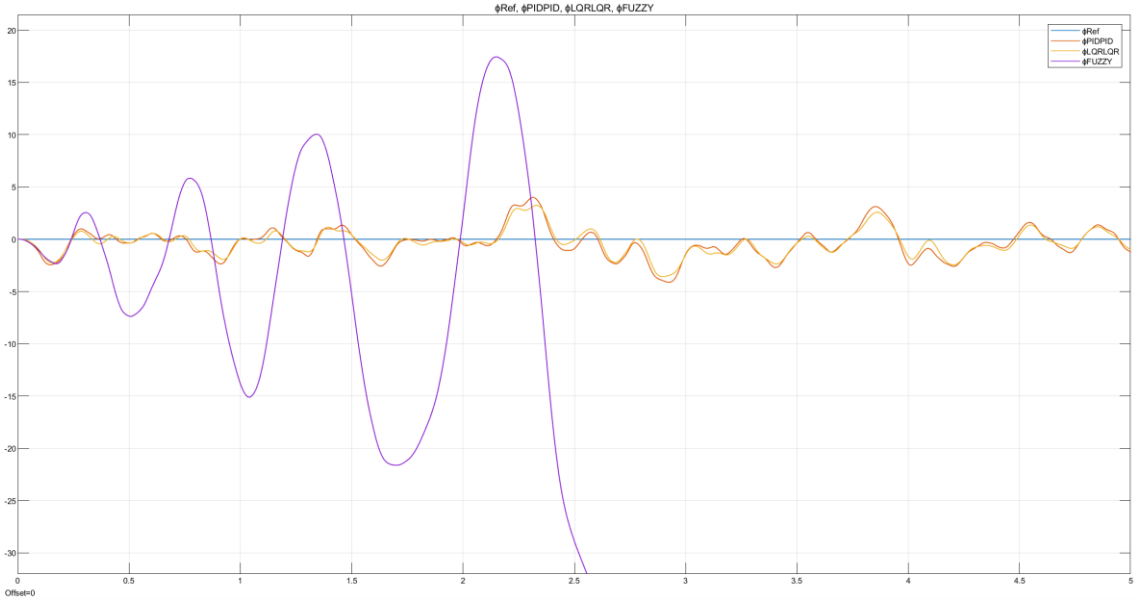


Ilustración 95: Respuesta del ángulo de alabeo ante un ruido de medida de 100 Hz

Ángulo de cabeceo con filtro paso bajo de 15 Hz aplicado

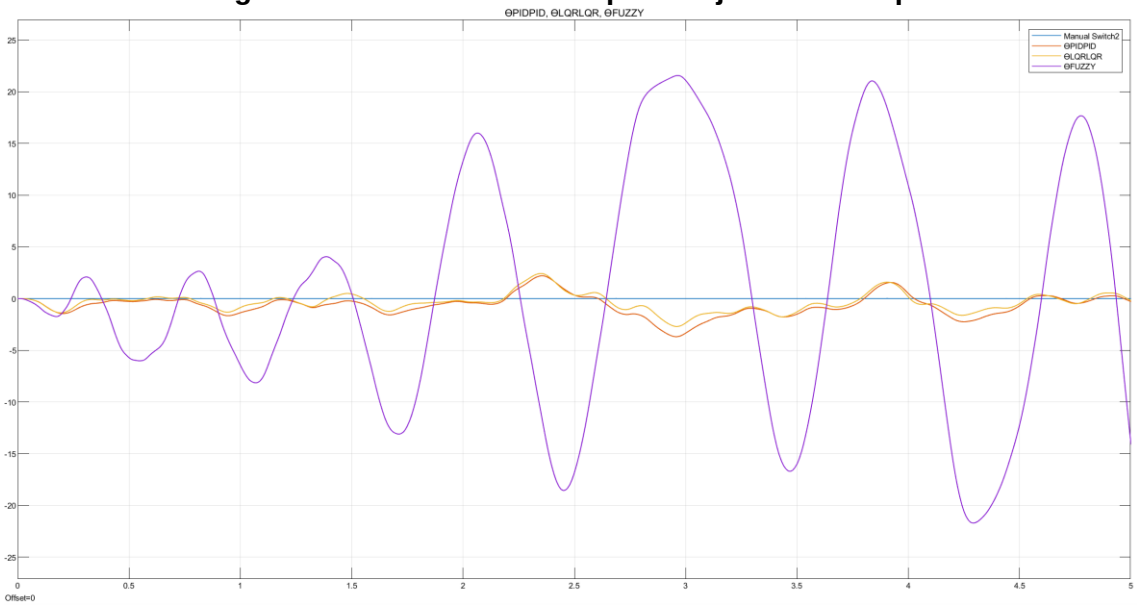


Ilustración 96: Respuesta del ángulo de cabeceo ante un ruido de medida de 100 Hz

Velocidad angular de guiñada con filtro de 15 Hz aplicado

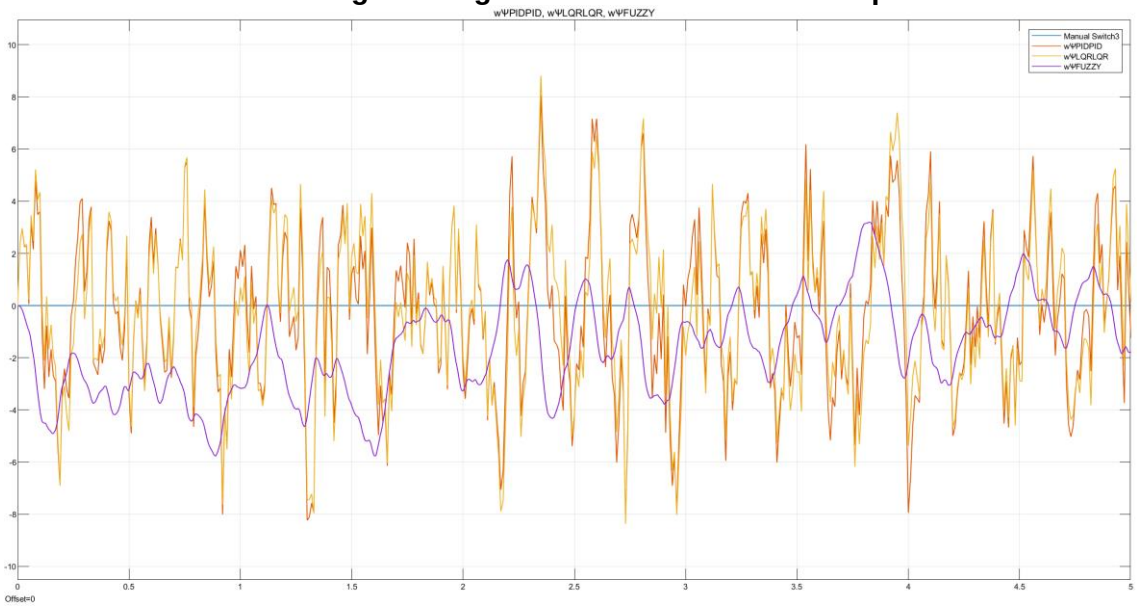


Ilustración 97: Respuesta de la velocidad angular de guiñada ante un ruido de medida de 100 Hz

Observamos que el ruido de medida introduce oscilaciones en el sistema. Sin la aplicación de filtros paso bajo en las lecturas, el sistema se torna fuertemente inestable. Con la sintonización actual, el controlador difuso no es capaz de atenuar y responder adecuadamente al ruido de medida, al contrario que los lazos de control de PID en cascada y LQR en cascada que, junto a un buen filtrado, son capaces de atenuar este ruido de medida.

8. Implementación en microcontrolador

En este apartado se va a explicar los procedimientos y técnicas más importantes llevadas a cabo para la implementación efectiva de los algoritmos de control PID en cascada y LQR en cascada en el prototipo de dron desarrollado, pues son los que obtuvieron un mejor desempeño en las simulaciones. Los códigos completos se podrán encontrar en el anexo al final de este trabajo.

8.1. Calibración inicial de sensores y configuración de los ESC:

Calibración del IMU LSM6DS0X

Para calibrar el IMU debemos establecer el cuadricóptero sobre una superficie plana y estable. Se tomarán 10000 medidas y se calculará la media de las lecturas tomadas para obtener los “offset” del acelerómetro y del giroscopio. Estos valores de calibración se sumarán a cada una de las lecturas obtenidas con la IMU.

```
void calibrarIMU(void) {
  for (int i = 0; i < 10000; i++) {
    if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
      IMU.readGyroscope(gX, gY, gZ);
      IMU.readAcceleration(aX, aY, aZ);
    }
    gXCal += gX;
    gYCal += gY;
    gZCal += gZ;
    aXOffset += aX;
    aYOffset += aY;
    aZOffset += aZ;
  }
  gXCal /= 10000;
  gYCal /= 10000;
  gZCal /= 10000;
  aXOffset/= 10000;
  aYOffset/= 10000;
  aZOffset = (aZOffset / 10000) - 1;
}
```

Calibración de los controladores ESC

Los controladores electrónicos de velocidad de los motores (ESC) son programables y configurables por el usuario. Es necesario su configuración previa antes de su uso para que todos los motores funcionen de igual manera y evitar diferencias de comportamiento significativas durante el vuelo que pueden desestabilizar el dron.

Al no contar con una interfaz visual, estos controladores constan de un código de sonidos para indicar el menú seleccionado y las opciones que se pueden configurar. Dichos sonidos son generados por el motor cuando el ESC alimenta dos fases de forma simultánea, generando una secuencia de sonidos agudos.

Para seleccionar las opciones o menús, se debe mover el joystick que controla el empuje de los motores. Podemos encontrar todas las opciones y menús en el datasheet. Por lo tanto, para calibrar los ESC es necesario una emisora que envié los comandos al microcontrolador.

Para realizar todas estas calibraciones, cargaremos el programa "*calibrar_esc.ino*" al microcontrolador y seguiremos las instrucciones del datasheet del ESC, accediendo al menú de calibración al mover el joystick hacia arriba cuando así nos lo indique el programa del ESC. Este programa puede encontrarse en los anexos.

Se realizaron las siguientes configuraciones:

- El valor mínimo de velocidad de giro de los motores, motores detenidos, se establece con el joystick que controla la velocidad en Z al mínimo, abajo del todo.

- El valor máximo de velocidad de giro de los motores se establece con el joystick que controla la velocidad en z al máximo, arriba del todo.

De esta forma se puede aprovechar el rango completo del joystick.

- Freno de los motores desactivado.

- Resto de valores de fábrica por defecto.

Una vez realizadas todas las configuraciones, estas quedarán guardadas en la memoria del ESC. Cada vez que iniciemos las pruebas con nuestro dron, será necesario enviar el comando mínimo a nuestros motores para que estos se configuren y no accedan al menú de calibración. Esto lo realizaremos de forma automática al alimentar el cuadricóptero con la batería.

```
// Configurar Motores y ESC
motor1.attach(6);
motor2.attach(17);
motor3.attach(16);
motor4.attach(7);
motor1.writeMicroseconds(1000);
motor2.writeMicroseconds(1000);
motor3.writeMicroseconds(1000);
motor4.writeMicroseconds(1000);
delay(10000);
```

8.2. Tiempo de muestro. Bucle principal.

Para garantizar un funcionamiento en tiempo real del sistema, es necesario que la tasa de muestreo sea lo suficientemente pequeña y estable, al igual que la latencia en las comunicaciones, a fin de reducir el tiempo entre bucles y se pueda decir que el control del cuadricóptero es en tiempo real.

Para conseguir esto es necesario optimizar los algoritmos de control y la programación del UAV, así como establecer un tiempo de muestreo estable que varíe lo menos posible entre interacciones. Este tiempo puede variar entre interacciones debido a las incertidumbres de cambio de estado de las puertas lógicas que conforman el microprocesador, las operaciones a realizar en cada bucle, latencia de las comunicaciones, etc.

Estableceremos un tiempo mínimo de bucle de 0.01 segundos, ejecutando el siguiente bucle:

```
//************LOOP************//
void loop() {
  tiempoInicio = micros();
  if ((power == 1)) {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
    APPgetData();
    IMUgetData();
    Controller();
    MMA();
    setMotors();
    PCsendData();
    #ifdef PRINT
      printValues();
    #endif
  } else {
    APPgetData();
    motor1.writeMicroseconds(1000);
    motor2.writeMicroseconds(1000);
    motor3.writeMicroseconds(1000);
    motor4.writeMicroseconds(1000);
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
  }
  tiempoFin = micros();
  //Serial.println(duracion);
  while ((tiempoFin - tiempoInicio) < tiempoDeseado) {
    tiempoFin = micros();
  }
}
```

8.3. Comunicaciones. Envío y recepción de datos en tiempo real

Recepción de comandos de la aplicación

Para la recepción del paquete de datos enviado desde la aplicación móvil se empleará el siguiente código:

```
//Recibir paquetes desde la app
void APPgetData(void) {
  int packetSize = Udp.parsePacket();
  if (packetSize > 0) {
    char incomingPacket[255];
    int len = Udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = 0;
      sscanf(incomingPacket, "%f,%f,%f,%f,%d", &vZRef, &wYawRef, &rollRef,
&pitchRef, &power);
      rollRef *= PI / 180;
      pitchRef *= PI / 180;
      wYawRef *= PI / 180;
    }
  }
}
```

Este código recibe un paquete de datos vía wifi UDP, separando los datos en función del tipo que se espera recibir, flotante en el caso de los comandos de control de movimiento del dron y entero en caso de la parada de emergencia o “power”. Las magnitudes angulares se pasan a radianes para trabajar en unidades del SI en los bucles de control.

Envío de datos al ordenador

Para el envío de la información sensorial proveniente del IMU y del altímetro se empleará el siguiente código:

```
//Enviar paquetes al PC
void PCsendData(void) {
  uint8_t packetBuffer[16];
  memcpy(packetBuffer, &estadoKalmanRoll, sizeof(estadoKalmanRoll));
  memcpy(packetBuffer + 4, &estadoKalmanPitch, sizeof(estadoKalmanPitch));
  memcpy(packetBuffer + 8, &wYaw, sizeof(wYaw));
  memcpy(packetBuffer + 8, &vZKalman, sizeof(vZKalman));
  Udp.beginPacket(receiverIp, clientUdpPort);
  Udp.write(packetBuffer, sizeof(packetBuffer));
  Udp.endPacket();
}
```

Este código crea un paquete de datos del tamaño apropiado del número total de datos que se van a enviar. Finalmente se envía el paquete vía wifi a la dirección IP del ordenador y el puerto empleando el protocolo UDP.

8.4. Filtrado y combinación de lecturas de sensores. IMU y altímetro

Como se comprobó en simulaciones, es vital obtener unas lecturas de la IMU limpias, filtradas con los menores niveles de ruido que sean posible, para asegurar una respuesta estable del sistema. En este caso se aplicarán varias técnicas:

- Selección de una tasa de muestreo adecuada para el acelerómetro para medir el menor ruido posible.
- Aplicación de filtros paso bajo para las lecturas del acelerómetro.
- Selección de una tasa de muestreo adecuada para el giroscopio y medir el menor ruido posible.
- Aplicación de filtros paso alto para las lecturas del giroscopio y reducir la deriva.

Todas estas configuraciones se pueden establecer escribiendo en los registros habilitados para cada configuración, indicados por el datasheet del LSM6DS0X.

Para usar este sensor de forma más simple, emplearemos la librería que provee Arduino para este sensor. Los filtros y muestreos fueron habilitados modificando el método. begin() de esta librería, pues no permite la configuración manual de la tasa de muestro ni de los filtros.

Este es el método modificado en la librería de Arduino:

```
int LSM6DSOXClass::begin()
{
  if (_spi != NULL) {
    pinMode(_csPin, OUTPUT);
    digitalWrite(_csPin, HIGH);
    _spi->begin();
  } else {
    _wire->begin();
  }

  if (!(readRegister(LSM6DSOX_WHO_AM_I_REG) == 0x6C ||
readRegister(LSM6DSOX_WHO_AM_I_REG) == 0x69)) {
    end();
    return 0;
  }

  writeRegister(LSM6DSOX_CTRL2_G, 0b00110100);      //muestro de 56Hz en el
rango de 500 dps

  writeRegister(LSM6DSOX_CTRL1_XL, 0b00111010);     //muestro de 52Hz en el
rango de 4 g y LPF
```

```
  writeRegister(LSM6DSOX_CTRL7_G, 0b01110010);      //HPF para giroscopio conf. corte 1 Hz y acelerometro bypass

  writeRegister(LSM6DSOX_CTRL8_XL, 0b10001001);     //LPF2 acelerometro con f.corte de 52/10 Hz

  return 1;
}
```

Una vez establecidos los filtros, los ángulos de alabeo y de cabeceo se calculan a partir de un análisis trigonométrico a partir de las componentes de la aceleración medidas por el acelerómetro [11]:

$$\phi = \text{atan}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

$$\theta = -\text{atan}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right)$$

Para las lecturas del altímetro láser emplearemos la librería VL53L0X desarrollada por “Pololu”, que implementa todos los cálculos necesarios para estimar la altura.

La función implementada en el microcontrolador para obtener las lecturas de las sensores se denomina “GetData()”:

```
void getData(void) {
  if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
    IMU.readGyroscope(gX, gY, gZ);
    IMU.readAcceleration(aX, aY, aZ);
    gX -= gXCal;
    gY -= gYCal;
    gZ -= gZCal;
    aX -= aXOffset;
    aY -= aYOffset;
    aZ -= aZOffset;
    wRoll = gX * PI / 180;
    wPitch = gY * PI / 180;
    wYaw = -gZ * PI / 180;
  }
  roll = atan(aY / sqrt(aX * aX + aZ * aZ));
  pitch = -atan(aX / sqrt(aY * aY + aZ * aZ));
  kalman1D (estadoKalmanRoll, incertidumbreKalmanRoll, wRoll, roll);
  estadoKalmanRoll = estadoKalman;
  incertidumbreKalmanRoll = incertidumbreKalmanRoll;
  kalman1D (estadoKalmanPitch, incertidumbreKalmanPitch, wPitch, pitch);
  estadoKalmanPitch = estadoKalman;
  incertidumbreKalmanPitch = incertidumbreKalmanRoll;
  AZ = aX * (-cos(roll) * sin(pitch)) + aY * sin(roll) + aZ * (cos(roll) * cos(pitch));
  AZ = (AZ - 1) * 9.81;
  z = ((float)sensor.readRangeContinuousMillimeters() / 1000 ) * cos (roll) * cos(pitch);
  kalman2D();
}
```

Sin embargo, para conseguir mejores lecturas y mejor desempeño del sistema, es necesario obtener lecturas todavía más limpias y estables. Para ello se puede implementar alguno de los siguientes filtros:

- Filtro de Medias
- Filtro Complementario
- Filtro Butterworth
- Filtro de Kalman

De igual manera es necesario combinar las lecturas del acelerómetro y del sensor láser para obtener lecturas precisas de la altura y velocidad en el eje Z, empleando alguno de los filtros anteriores.

8.4.1. Filtro de Medias

Este filtro suaviza la señal haciendo un promedio de un determinado número de valores.
Se puede implementar de la siguiente manera:

$$z(n) = \frac{1}{2M+1}\sum z(n+k)$$

Este filtro es sencillo de implementar, pero presenta algunas limitaciones importantes:

- Sensible a valores extremos de la señal que estén bastante alejados de la media.
- Introduce un retardo considerable proporcional al número de muestras que se van a promediar. A mayor número de muestras, mejor filtrado a coste de un mayor retardo en el proceso.

8.4.2. Filtro Complementario

Se trata de un filtro simple y fácil de implementar, que puede fusionar dos lecturas de dos sensores con diferentes características de ruido y precisión. Típicamente se combinan lecturas de alta frecuencia con lecturas de baja frecuencia.

Esto se consigue ponderando ambas señales y sumándolas:

$$z_{filtrada} = \propto z_{HF} + (1-\propto) z_{LF}$$

Este filtro si bien es muy simple de implementar (y efectivo en ciertas circunstancias), presenta algunas limitaciones:

- No puede adaptarse de forma automática a variaciones en la dinámica del sistema o variaciones en características del ruido, pues el parámetro de ponderación una vez fijado no varía, por lo que es menos preciso.

8.4.3. Filtro Butterworth

Se trata de un filtro digital que proporciona una respuesta en frecuencia lo más plana posible en el rango de frecuencias de la banda pasante y reduciendo las frecuencias de la banda de frecuencias atenuada

Su función de transferencia es la siguiente:

$$H(jw) = \frac{1}{\sqrt{1 + (\frac{\omega}{\omega_{corte}})^{2 \cdot n_{filtro}}}}$$

Los mejores resultados de filtrado se obtienen para filtros de orden mayor a 2.

Limitaciones:

- A mayor orden del filtro, mayor retardo y coste computacional.

8.4.4. Filtro de Kalman

Se trata de un algoritmo recursivo para estimar los estados de un sistema dinámico, incluso cuando las mediciones de dichos estados estén sujetas a imprecisiones, ruido, derivas, etc. debido a las características de los sensores. Un filtro de Kalman consta siempre de 2 etapas [17], [18]:

Etapa de Predicción:

1. Predicción del estado: Esta ecuación predice el estado del sistema en el instante k a partir del estado en el instante k – 1.

$$S_{k|k-1} = A \cdot S_{k-1|k-1} + B \cdot U_k$$

A: $matriz\ de\ transición\ de\ estado$
$S_{k-1|k-1}$: $vector\ de\ estados\ anterior$
B: $matriz\ de\ control$
U_k: $vector\ de\ entradas\ de\ control$

2. Cálculo de la covarianza del error: Esta ecuación predice la incertidumbre del estado en el instante k a partir de la incertidumbre en el instante k – 1.

$$P_{k|k-1} = A \cdot P_{k-1|k-1} \cdot F^T + Q$$

A: $matriz\ de\ transición\ de\ estado$
$P_{k-1|k-1}$: $vector\ de\ covarianzas\ de\ error\ anterior$
Q: $matriz\ de\ ruido\ de\ proceso\ (desviación\ típica)$

Etapa de Actualización

1. Calcular la ganancia de Kalman: Esta ganancia determina cuanto se debe ajustar la predicción del estado en el instante k a partir de una nueva medición.

$$K_k = P_{k|k-1} \cdot H^T \cdot (H \cdot P_{k|k-1} \cdot H^T + R)^{-1}$$

H: $matriz\ de\ observación$
R: $matriz\ de\ covarianza\ del\ ruido\ de\ medición$

2. Actualización del estado: Esta ecuación ajusta la predicción del estado predicho con la nueva medición.

$$S_{k|k} = S_{k|k-1} + K_k \cdot (M_k - H \cdot S_{k|k-1})$$

$S(k)$: $vector\ de\ estados\ actual$
K: $matriz\ de\ ganancias\ de\ Kalman$
M_k: $vector\ de\ mediciones\ actual$
$M_k - H \cdot S_{k|k-1}$: $residuo\ o\ diferencia\ entre\ medición\ real\ y\ la\ predicción$

3. Actualización de la covarianza del error: Esta ecuación actualiza la covarianza de error predicha con la nueva medición.

$$P_{k|k} = (I - K_k \cdot F) \cdot P_{k|k-1}$$

$I - K_k \cdot F$: *ajuste para la covarianza del error*
$P_{k|k}$: *vector de covarianzas de error actual*

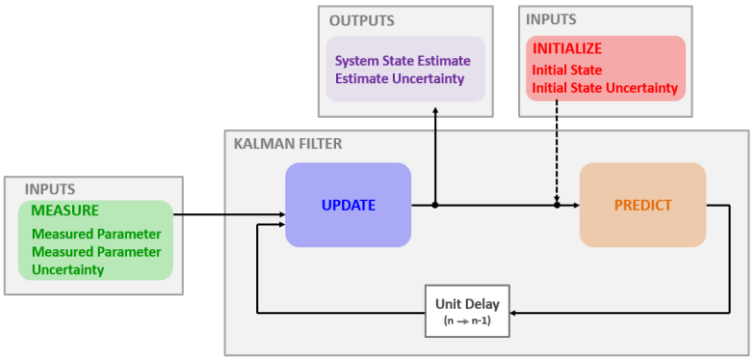


Ilustración 98: Esquema que muestra el proceso recursivo que sigue el filtro de Kalman [https://www.kalmanfilter.net/ES/default_es.aspx]

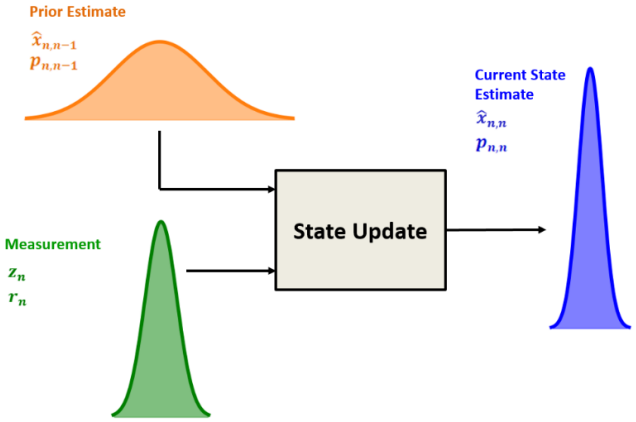


Ilustración 99: Estimación del estado reduciendo la incertidumbre de la medida mediante filtro de Kalman [https://www.kalmanfilter.net/ES/default_es.aspx]

En esta ocasión optaremos por implementar filtro de Kalman a nuestras lecturas. A pesar de tener un coste computacional mayor que el filtro complementario, pero menor que filtros de Butterworth de órdenes elevados, permite obtener mejores lecturas de los sensores en tiempo real, con una excelente precisión y filtrado. Optaremos por implementar este filtro para las lecturas del IMU y del sensor láser.

8.3.5. Implementación del filtro de Kalman

Filtro de Kalman implementado para los ángulos de alabeo y de cabeceo:

El objetivo es estimar los ángulos de alabeo y de cabeceo de nuestro dron. Sin embargo, los sensores tienen algunas limitaciones:

- El acelerómetro introducide ruido en las lecturas, aunque puede moderarse con filtros que incorpora la IMU. El giroscopio introduce un error que se acumula con el tiempo debido a la deriva.
- Datos:
  - A partir del datasheet del IMU, sabemos que el ruido RMS en el giroscopio es de $\pm$0.075 dps ($\pm$0.001309 rad/s) sin filtrar, resultando una varianza de 0.0000017 $\frac{rad^2}{s^2}$
  - A partir del datasheet del IMU, sabemos que el ruido RMS en el acelerómetro es de $\pm$2 mg ($\pm$0.002 g) sin filtrar, resultando una varianza de 0.000004 g^2
  - Se va a fijar una incertidumbre inicial de $\pm$2º ($\pm$0.035 rad)

Estos valores son bastante pequeños, y ante la presencia de vibraciones producidas por los motores y hélices serán mayores, por lo que tomaremos valores más elevados.
Para seleccionar los valores, realizaremos un ensayo con los motores encendidos, cargando el programa "*enviodatosruido.ino*" en el microcontrolador enviando los datos vía Wifi al PC, ejecutando el programa "DATOS_IMU.py". Primero se enviará un comando a baja de velocidad de 1095 µs (velocidad mínima) y un tiempo después un comando de 1500 µs (velocidad media).

A continuación, se muestran las lecturas obtenidas:

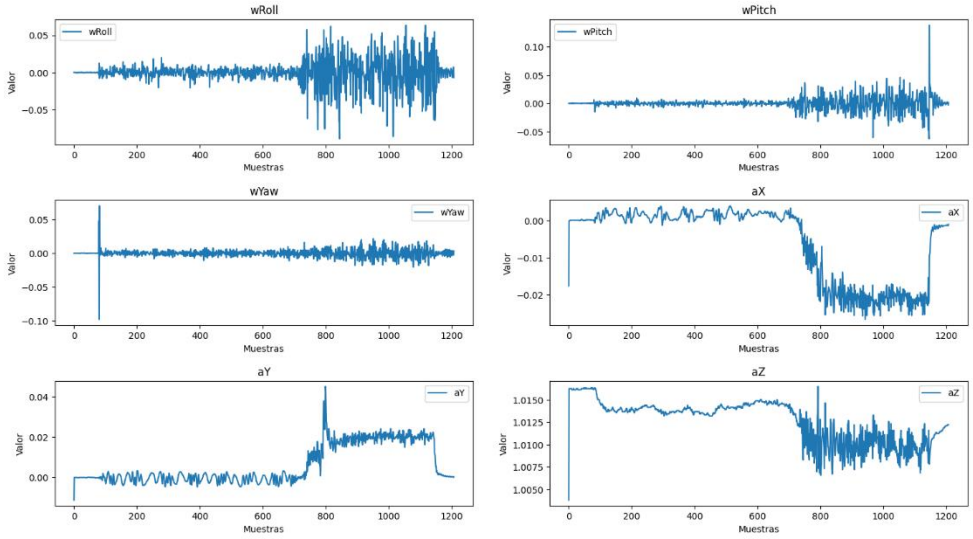


Ilustración 100: Lecturas de ruido de velocidades angulares y aceleraciones lineales medidas con los rotores en funcionamiento

Tras calcular el promedio y la varianza de los datos obtenidos, se han obtenido los siguientes valores:

	Velocidad Angular Alabeo [rad/s]	**Velocidad Angular Cabeceo** [rad/s]	**Velocidad Angular Guiñada** [rad/s]	**Acel. X** [g]	**Acel. Y** [g]	**Acel. Z** [g]
Promedio	-0.000305	0.0000611	-0.000073	-0.00567	0.00599	1.01
Desviación Típica	0.017	0.011	0.0065	0.01	0.0096	0.002

Tabla 30: Tabla que recoge el promedio y las desviaciones típicas de las lecturas del IMU sujetas a ruido y vibraciones al hacer girar los rotores

Desviación típica promedio velocidad angular:

$$\omega_{desv.tip} = 0.0116\ [\frac{rad}{s}]$$

Desviación típica promedio aceleración angular:

$$a_{desv.tip} = 0.007\ [g]$$

Esta se traduce en una variación en los ángulos de entre, aproximadamente $\pm$1º ($\pm 0.0175\ rad$)

<u>Etapa de Predicción:</u>

1. Predicción del estado:

$$S_{k|k-1} = A \cdot S_{k-1|k-1} + B \cdot U_k$$

Sabemos que, el ángulo se puede expresar de forma discreta explícitamente como:

$$\theta_k = \theta_{k-1} + \omega_k \cdot \Delta t$$

Resultando en:

$$A = 1$$

$$S_{k-1|k-1} = \theta_{k-1}$$

$$B = \Delta t$$

$$U_k = \omega_k$$

2. Cálculo de la covarianza del error:

$$P_{k|k-1} = A \cdot P_{k-1|k-1} \cdot A^T + Q$$

$$Q = 0.0116 \cdot 0.0116 = 0.000135\ [(rad/s)^2]$$

Etapa de Actualización

1. Calcular la ganancia de Kalman:

$$K_k = P_{k|k-1} \cdot H^T \cdot (H \cdot P_{k|k-1} \cdot H^T + R)^{-1}$$

$$H = 1$$
$$R = 0.0175 \cdot 0.0175 = 0.000305 \ [rad^2]$$

2. Actualización del estado:

$$S_{k|k} = S_{k|k-1} + K_k \cdot (M_k - H \cdot S_{k|k-1})$$

3. Actualización de la covarianza del error:

$$P_{k|k} = (I - K_k \cdot A) \cdot P_{k|k-1}$$

$$I = 1$$

Filtro de Kalman implementado para la velocidad vertical:

El objetivo es estimar la velocidad en Z de nuestro dron. Sin embargo, los sensores tienen algunas limitaciones:

- Con el sensor láser podemos obtener lecturas precisas de la altura en Z, sujetas a una ligera incertidumbre y ruido.
- La aceleración inercial calculada para el eje Z a partir de las componentes de las aceleraciones en x, y, z y los ángulos de cabeceo y de alabeo está sujeta a un ligero ruido pese a los filtros.
- Los drones que operan en exteriores emplean además un GPS. Este sensor puede indicar la velocidad del dron, su posición, rumbo, etc. Este no es nuestro caso, pues en drones de vuelo en interiores no se recibiría una señal de GPS válida debido a los obstáculos e interferencias.

Datos:

- El datasheet no proporciona información acerca del ruido en unidades fáciles de trabajar (para el sensor láser da el ruido en kilocuentas por cada diodo avalancha sensible a fotón (kcps/SPAD)), por lo que los datos los estimaremos de manera similar al ruido delo acelerómetro y del giroscopio.

Siguiendo la misma metodología que en el método anterior (variando tamaño de los paquetes de envío y recepción de datos), elevando el dron a cierta altura de forma manual, se han obtenido las siguientes medias y varianzas:

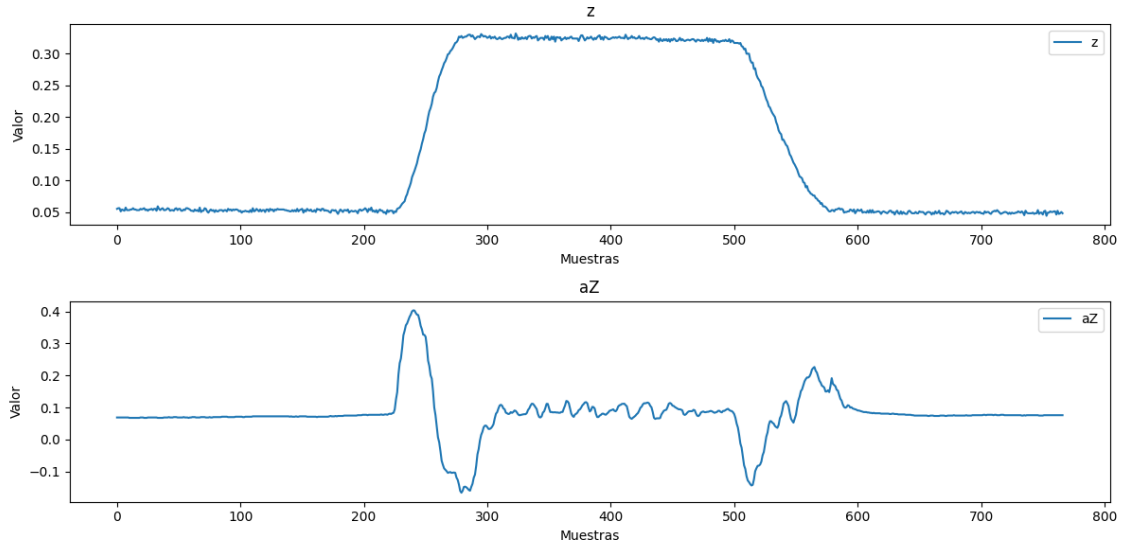


Ilustración 101: Lecturas de ruido en la medición de la altura y aceleración en Z

	Velocidad en Z	**Altura en Z**
Promedio	0.326	0.119
Desviación Típica	0.116	0.050

Tabla 31: Tabla que recoge el promedio y la desviación típicas de las lecturas de velocidad en Z y altura en Z

Desviación típica promedio velocidad angular:

$$a_{zdesv.tip} = 0.116\ [\frac{m}{s^2}]$$

Desviación típica promedio aceleración angular:

$$z_{desv.tip} = 0.050\ [m]$$

En este caso es necesario un filtro de Kalman en 2 dimensiones, para obtener lecturas más precisas tanto en velocidad como en altura.

Etapa de Predicción:

1. Predicción del estado:

$$S_{k|k-1} = A \cdot S_{k-1|k-1} + B \cdot U_k$$

Sabemos que la posición en Z se puede expresar de forma discreta explícitamente como:

$$z_k = z_{k-1} + v_{k-1} \cdot \Delta t + \frac{1}{2} a_{k-1} \Delta t^2$$

Y la velocidad en Z como:

$$v_{zk} = v_{zk-1} + a_{k-1} \Delta t$$

Resultando en:

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \qquad S_k = \begin{bmatrix} z_k \\ v_{zk} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \qquad U_k = a_{k-1}$$

2. Cálculo de la covarianza del error:

$$P_{k|k-1} = A \cdot P_{k-1|k-1} \cdot A^T + Q$$

$$Q = \begin{bmatrix} q_{11} & 0 \\ 0 & q_{22} \end{bmatrix} = \begin{bmatrix} 0.116 \cdot 0.116 & 0 \\ 0 & 0.050 \cdot 0.050 \end{bmatrix} = \begin{bmatrix} 0.0135 & 0 \\ 0 & 0.0025 \end{bmatrix}$$

Etapa de Actualización

1. Calcular la ganancia de Kalman:

$$K_k = P_{k|k-1} \cdot H^T \cdot (H \cdot P_{k|k-1} \cdot H^T + R)^{-1}$$

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

$$R = \begin{bmatrix} r_{11} & 0 \\ 0 & r_{22} \end{bmatrix} = \begin{bmatrix} 0.116 \cdot 0.116 & 0 \\ 0 & 0.050 \cdot 0.050 \end{bmatrix} = \begin{bmatrix} 0.0135 & 0 \\ 0 & 0.0025 \end{bmatrix}$$

2. Actualización del estado:

$$S_{k|k} = S_{k|k-1} + K_k \cdot (M_k - H \cdot S_{k|k-1})$$

3. Actualización de la covarianza del error:

$$P_{k|k} = (I - K_k \cdot A) \cdot P_{k|k-1}$$

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

8.3.6. Algoritmos de control

La elección del algoritmo de control que se desea usar se debe seleccionar definiendo dicho algoritmo de control: PID ó LQR con una directiva de preprocesador.
Mediante estas directivas, evitamos emplear más memoria de la necesaria para almacenar código que no se va a usar, pues los algoritmos de control PID y LQR están implementados en funciones separadas, pero se llaman desde la misma sección del código.

Función común a ambos controladores:

```
// Acciones de Control
void Controller(void) {
 #ifdef PID
   vZError = vZRef - vZKalman;
   PD(vZError, KpVz, KdVz, vZErrorPrev, 11.46, -9.81);
   Thrust = outPD[0] + 9.81;
   vZErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   rollError = rollRef - estadoKalmanRoll;
   P(rollError, KpRoll);
   wRollRef = outP;
   outP = 0;
   wRollError = wRollRef - wRoll;
   PD(wRollError, KpWRoll, KdWRoll, wRollErrorPrev, 1.128, -1.128);
   T1x = outPD[0];
   wRollErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   pitchError = pitchRef - estadoKalmanPitch;
   P(pitchError, KpPitch);
   wPitchRef = outP;
   outP = 0;
   wPitchError = wPitchRef - wPitch;
   PD(wPitchError, KpWPitch, KdWPitch, wPitchErrorPrev, 1.128, -1.128);
   T2y = outPD[0];
   wPitchErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   wYawError = wYawRef - wYaw;
   PD(wYawError, KpWYaw, KdWYaw, wYawErrorPrev, 0.1, -0.1);
   T3z = outPD[0];
   wYawErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
 #endif

 #ifdef LQR
   rollError = - rollRef + estadoKalmanRoll;
```

```
    pitchError = - pitchRef + estadoKalmanPitch;
    LQRAngles(rollError, pitchError);
    vZError = - vZRef + vZ;
    wRollError = - wRollRef + wRoll;
    wPitchError = - wPitchRef + wPitch;
    wYawError = - wYawRef + wYaw;
    LQRVelocities(vZError, wRollError, wPitchError, wYawError);
  #endif
}
```

Algoritmo de control PID cascada:

```
#ifdef PID
  //Controlador PD
  void PD(float error, float Kp, float Kd, float& errorPrev, float max, float min) {
    float P = Kp * error;
    float D = Kd * N * (error - errorPrev) / (N + 0.01);
    float output = P + D;
    if (output > max) {
      output = max;
    }
    if (output < min) {
      output = min;
    }
    outPD[0] = output;
    outPD[1] = error;
  }
  //Controlador P
  void P(float error, float Kp) {
    outP = Kp * error;
  }
#endif
```

Algoritmo de control LQR cascada:

```
//Controlador LQR
#ifdef LQR
  void LQRVelocities(float vZError, float wRollError, float wPitchError, float wYawError)
{
    error1 = {vZError, wRollError, wPitchError, wYawError};
    u1 = - (K2 * error1);
    Thrust = u1(0,0) + 9.81;
    T1x = u1(1,0);
    T2y = u1(2,0);
    T3z = u1(3,0);
  }
  void LQRAngles(float rollError, float pitchError) {
    error2 = {rollError, pitchError};
    u2 = - (K1 * error2);
    wRollRef = u2(0,0);
    wPitchRef = u2(1,0);
  }
#endif
```

8.3.7. Algoritmo de Mezcla de Motores. Actuadores

Para el cálculo del comando PWM se aplicará la siguiente relación obtenida tras la realización de ensayos en el banco de pruebas:

$$PWM_i = 7 \cdot 10^{-15}(\omega_i^2)^3 - 7 \cdot 10^{-10}(\omega_i^2)^2 + 4 \cdot 10^{-5}(\omega_i^2)$$

Se mapeará el valor de PWM calculado entre 1000 y 2000 µs, el rango de valores que admite la función writeMicroseconds ().

```
// Función MMA (Motor Mixing Algorithm)
void MMA(void) {
  w1_2 = Thrust / (4 * kf) - T1x / (4 * kf * L) + T2y / (4 * kf * L) + T3z / (4 * km);
  w2_2 = Thrust / (4 * kf) + T1x / (4 * kf * L) + T2y / (4 * kf * L) - T3z / (4 * km);
  w3_2 = Thrust / (4 * kf) + T1x / (4 * kf * L) - T2y / (4 * kf * L) + T3z / (4 * km);
  w4_2 = Thrust / (4 * kf) - T1x / (4 * kf * L) - T2y / (4 * kf * L) - T3z / (4 * km);

  PWM1 =  (7e-15) * w1_2 * w1_2 * w1_2- (7e-10) * w1_2 * w1_2 + (4e-5) * w1_2;
  PWM2 =  (7e-15) * w2_2 * w2_2 * w2_2- (7e-10) * w2_2 * w2_2 + (4e-5) * w2_2;
  PWM3 =  (7e-15) * w3_2 * w3_2 * w3_2- (7e-10) * w3_2 * w3_2 + (4e-5) * w3_2;
  PWM4 =  (7e-15) * w4_2 * w4_2 * w4_2 -(7e-10) * w4_2 * w4_2 + (4e-5) * w4_2;

  if (PWM1 > 1) {
    PWM1 = 1;
  }
  if (PWM2 > 1) {
    PWM2 = 1;
  }
  if (PWM3 > 1) {
    PWM3 = 1;
  }
  if (PWM4 > 1) {
    PWM4 = 1;
  }
  if (PWM1 < 0) {
    PWM1 = 0;
  }
  if (PWM2 < 0) {
    PWM2 = 0;
  }
  if (PWM3 < 0) {
    PWM3 = 0;
  }
  if (PWM4 < 0) {
    PWM4 = 0;
  }
```

```
  PWM1 = (1000 + (PWM1 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM2 = (1000 + (PWM2 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM3 = (1000 + (PWM3 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM4 = (1000 + (PWM4 - 0) * ((2000 - 1000)/(1 - 0)));

}

// Función para establecer los valores de los motores
void setMotors(void) {
  motor1.writeMicroseconds(PWM1);
  motor2.writeMicroseconds(PWM2);
  motor3.writeMicroseconds(PWM3);
  motor4.writeMicroseconds(PWM4);
}
```

8.4. Primeros ensayos

En los primeros ensayos realizados no se han obtenido las respuestas esperadas con la sintonización de los controladores establecida en simulación, teniendo problemas de estabilidad. Algunas posibles causas podrían ser:

- Diferencias entre el modelo teórico y real.
- Reajuste de las ganancias del controlador PID en cascada y de las matrices de ponderación R y Q del controlador LQR en cascada.
- Desbalance de masas que podría corregirse con contrapesos colocados en ciertas zonas para anular estos desbalances.
- Diferencias de empuje entre motores por posibles diferencias en su fabricación

Sin embargo, se ha conseguido establecer una comunicación exitosa entre la emisora el microcontrolador y el ordenador vía wifi UDP.

Por otro lado, los controladores responden ante las consignas enviadas desde la emisora y las mediciones del IMU, cuyas lecturas son bastante limpias y estables gracias a los filtros paso bajo y algoritmo de Kalman. Además, el tiempo de ejecución es bastante rápido. Podrían establecerse muestreos más cortos.

9. Conclusiones. Trabajo futuro

En base al trabajo realizado, podemos extraer una serie de conclusiones relativas al diseño e implementación de algoritmos de control en cuadricópteros:

- Debido a la naturaleza no lineal y acoplada de las dinámicas de los cuadricópteros, es recomendable simplificar dicha dinámica en torno al punto de equilibrio definido por un vuelo estacionario, asumiendo la condición de ángulos pequeños, para el desarrollo de métodos de control y algoritmos de control lineal de forma satisfactoria, tales como el PID o el LQR en cascada.

- Los métodos de control PID en cascada y LQR en cascada son métodos fiables y robustos ante perturbaciones. Con una correcta sintonización, es posible controlar de forma satisfactoria el cuadricóptero.

- La implementación de un control difuso requiere un conocimiento y experiencias mucho más profundas del sistema para obtener respuestas más satisfactorias y aceptables.

- Para implementar cualquier método de control en un UAV, es imprescindible hacer uso de filtros paso bajo y de filtros adicionales como el filtro de Kalman, para evitar obtener lecturas con elevados niveles de ruido, que puedan volver al sistema incontrolable.

- La experiencia del operador / técnico en el vuelo de UAV es un factor clave para la implementación satisfactoria de algoritmos de control y poder corregir problemas de distribución de masas u otros aspectos que puedan afectar al vuelo de la aeronave.

A partir de este trabajo, se sugieren algunas líneas de trabajo futuro. Entre otras:

- Obtención de sintonizaciones óptimas para los controladores estudiados.

- Implementación con mayor éxito y estabilidad de los controladores estudiados.

- Desarrollo e implementación de otras estrategias de control lineal.

- Desarrollo e implementación de estrategias de control no lineal.

- Desarrollo e implementación de estrategias de control visual.

- Desarrollo e implementación de estrategias de control y modelos basadas en *“Maching Learning”* y redes neuronales.

- Incluir un sistema de posicionamiento como un GPS y un magnetómetro para abordar el problema de seguimiento de trayectorias.

10. Presupuesto

10.1. Cuadro de precios elementales

CÓDIGO	U.M	DESCRIPCIÓN	PRECIO UNITARIO
MATEL01	Ud	Microcontrolador Arduino Nano RP2040	37,46
MATEL03	Ud	Barómetro BMP-280	2,98
MATEL04	Ud	Motor Brushless A2212/10T	19,12
MATEL05	Ud	Controlador ESC XXDD-30 A	15,99
MATEL06	mts	Cable AWG – 16	1,70
MATEL08	mts	Cable AWG – 10	6,64
MATEL09	Ud	Par de Conectores T-Plug Macho y Hembra	1,27
MATEL10	Ud	Conector XT-60 Macho	1,59
MATEL11	Ud	Soporte Arduino Nano	4,22
MATEL12	Ud	Cables Dupont Macho – Macho 10 cm (x20)	2,09
MATEL13	Ud	Cables Dupont Macho – Hembra 10 cm (x20)	2,09
MATEL14	Ud	Batería de Li-Po 3s 3600 mAh 35 C	27,79
MATEL15	Ud	Cargador IMAX-B6AC DUAL POWER	42,21
MATEL16	Ud	Celda de carga de 1 kg	4,60
MATEL17	Ud	Módulo amplificador HX711	3,02
MATES01	Ud	Estructura Dron Impresa en 3D en Garhem3D	49,65
MATES02	Ud	Rótula de bola	8,34
MATES03	Ud	Pata metálica	38,72
MATES04	Ud	Tablón de madera de 40 x 40 cm	9,93
MATES05	Ud	Clema de conexión de 2 terminales	0,99
MO01	hrs	Ingeniero Técnico Industrial Junior	25
MAQ01	Ud	Taladro	49,65
MAQ02	Ud	Destornillador	14,90
MAQ03	Ud	Soldador de estaño	29,79
MAQ04	Ud	Multímetro	39,72
MAQ05	ud	Ordenador Personal	695,10
MATAX01	gr	Estaño (60% Sn y 40% Pb)	0,1
MATAX02	mts	Cinta Aislante	0,14
MATAX03	Ud	Bridas de 100 mm (x100)	3,92
MATAX04	Ud	Tornillería (Tornillos, arandelas y tuercas)	14,90
MATAX05	Ud	Fundas Termoretráctiles	0,69
SFTW01	Lic	MATLAB / Simulink Estudiante + Complementos	415,71
SFTW02	Lic	Microsoft Office 365 Personal	68,52/año
SFTW03	Lic	Autodesk Inventor (Autodesk Product Design Licence)	3500,52/año
SFTW04	Lic	Arduino IDE	0
SFTW05	Lic	Processing IDE (modo Android)	0
SFTW06	Lic	Draw.IO	0
SFTW07	Lic	wxMaxima	0
SFTW08	Lic	KiCad	0

Tabla 32: Cuadro de precios elementales

10.2. Cuadro de precios descompuestos

CÓDIGO	U.M	DESCRIPCIÓN	CANTIDAD	PRECIO UNITARIO
UO01	***Diseño Dron y Soporte de Pruebas***			
SFTW03	Lic	Autodesk Inventor	1	3500,52/año
MO01	hrs	Ingeniero Técnico Industrial Junior	20	25
Costes Indirectos			3%	120,02
Total			**4120,54 €**	
UO02	**Impresión y montaje de la estructura del dron**			
MATES01	Ud	Estructura Dron Impresa en Garhem 3D	1	49,65
MATES02	Ud	Rótula de bola	1	8,34
MATES03	Ud	Pata metálica	1	38,72
MATES04	Ud	Tablón de madera de 40 x 40 cm	1	9,93
MO01	hrs	Ingeniero Técnico Industrial Junior	3	25
Costes indirectos			3%	5,45
Total			**187,09 €**	
UO03	**Montaje y soldadura de cableado de potencia del dron**			
MATEL06	Mts	Cable AWG – 16	2	1,70
MATEL08	Mts	Cable AWG – 10	1	6,64
MATEL09	Ud	Par de Conectores T-Plug Macho y Hembra	4	1,27
MATEL10	Ud	Conector XT-60 Macho	1	1,59
MATAX01	Gr	Estaño (60% Sn y 40% Pb)	100	0,10
MATAX02	Mts	Cinta Aislante	25	0,14
MATAX05	Ud	Fundas Termoretráctiles	2	0,69
MO01	Hrs	Ingeniero Técnico Industrial Junior	6	25
Costes indirectos			3%	5,45
Total			**187,04 €**	

Tabla 33: Cuadro de precios descompuestos

CÓDIGO	U. M	DESCRIPCIÓN	CANTIDAD	PRECIO UNITARIO
UO04	**Montaje de la electrónica del dron**			
MATEL01	Ud	Microcontrolador Arduino Nano RP2040	1	37,46
MATEL03	Ud	Barómetro BMP-280	1	2,98
MATEL04	Ud	Motor Brushless A2212/10T	4	19,12
MATEL05	Ud	Controlador ESC XXDD-30 A	4	15,99
MATEL11	Ud	Soporte Arduino Nano	1	4,22
MATEL12	Ud	Cables Dupont Macho – Macho 10 cm (x20)	1	2,09
MATEL13	Ud	Cables Dupont Macho – Hembra 10 cm (x20)	1	2,09
MO01	Hrs	Ingeniero Técnico Industrial Junior	5	25
Costes indirectos			3 %	9,43
Total			**323,71 €**	
UO05	**Estudio Dron. Sistemas de control**			
SFTW01	Lic	MATLAB / Simulink Estudiante + Complementos	1	415,71
STW02	Lic	Microsoft Office 360 Personal	1	68,52/año
STW06	Lic	Draw.IO	1	0
STW07	Lic	wxMaxima	1	0
STW08	Lic	KiCad	1	0
MO01	Hrs	Ingeniero Técnico Industrial Junior	206	25
Costes indirectos			3 %	169,03
Total			**5803,26 €**	
UO06	**Programación del dron**			
SFTW04	Lic	Arduino IDE	1	0
SFTW05	Lic	Processing IDE (modo Android)	1	0
MO01	Hrs	Ingeniero Técnico Industrial Junior	60	25
Costes indirectos			3 %	45
Total			**1545 €**	
UO07	**Batería y cargadores**			
MATEL14	Ud	Batería de Li-Po 3s 3600 mAh 35 C	2	27,79
MATEL15	Ud	Cargador IMAX-B6AC DUAL POWER	1	42,21
Costes indirectos			3 %	2,93
Total			**100,72 €**	
UO08	**Herramientas y banco de pruebas**			
MAQ01	Ud	Taladro	1	49,65
MAQ02	Ud	Destornillador	1	14,90
MAQ03	Ud	Soldador de estaño	1	29,79
MAQ04	Ud	Multímetro	1	39,72
MAQ05	Ud	Ordenador Personal con Windows 11	1	695,10
MATEL16	Ud	Celda de carga de 1 kg	1	4,60
MATEL17	Ud	Módulo amplificador HX711	1	3,02
Costes indirectos			3 %	25,10
Total			**861,88 €**	

TOTAL DE COSTES	**13129,24 €**

10.3. Presupuesto de ejecución por contrata

Costes	**13129,24 €**
Costes + Gastos Generales (13 %)	**14836,04 €**
Costes + Gastos Generales (13 %) + Beneficio Industrial (6 %)	**15726,20 €**
Costes + Gastos Generales (13 %) + Beneficio Industrial (6 %) + IGIC (7 %)	**16827,03 €**

Tabla 34: Cuadro de presupuesto de ejecución por contrata

TOTAL DEL PRESUPUESTO	**16827,03 €**

El presupuesto total del proyecto se estima en *"**dieciséis mil ochocientos veintisiete euros con tres céntimos**"*.

11. Bibliografía y referencias

[1] H. Shakhatreh et al., "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," in IEEE Access, vol. 7, pp. 48572-48634, 2019, doi: 10.1109/ACCESS.2019.2909530.

[2] Maaruf, M., Mahmoud, M. S., & Ma'arif, A. (2022). A survey of control methods for quadrotor uav. International Journal of Robotics and Control Systems, 2(4), 652-665.

[3] Taha, B., & Shoufan, A. (2019). Machine learning-based drone detection and classification: State-of-the-art in research. IEEE access, 7, 138669-138682.

[4] ArduPilot, «Flight Modes,» [En línea]. Available: https://ardupilot.org/copter/docs/flight-modes.html. [Último acceso: 28 06 2024].

[5] G. M. A. B. H. B. Miranda Colorado.R, Drones. Modelado y control de cuadricópteros, México: Marcombo, 2020.

[6] Barata, Filipe & Quadrado, Jose & Silva, Joao & Conselheiro, R. & Navarro, Emídio. (2005). Brushless DC Motor: Position Linear Control Simulation. 4.

[7] A. B. Sajid, A. Marryam, and M. Ali, "Modelling and Control of Brushless DC Motor," EasyChair Preprint no. 5710, June 4, 2021. [Online]. Disponible: https://easychair.org/publications/preprint/5710

[8] MIT, "Unit Conversion," MIT. [Online]. Disponible: https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html. [Último acceso: 09 07 2024].

[9] X. D. S. W. Quan Quan, Multicopter Design and Control Practice: A Series Experiments based on MATLAB and Pixhawk, Beijing: Springer, 2020.

[10] P. Patonis, P. Patias, I. N. Tziavos, D. Rossikopoulos, and K. G. Margaritis, "A fusion method for combining low-cost IMU/magnetometer outputs for use in applications on mobile devices," *Sensors (Basel)*, vol. 18, no. 8, p. 2616, Aug. 2018, doi: 10.3390/s18082616.

[11] Starlino, "A guide to using IMU (Accelerometer and Gyroscope Devices) in embedded applications." [Online]. Disponible: http://www.starlino.com/imu_guide.html. [Último acceso: 09 07 2024].

[12] P. Millett, "Brushless vs. brushed DC motors: When and why to choose one over the other," MPS. [Online]. Disponible: https://media.monolithicpower.com/mps_cms_document/2/0/2021-brushless-vs-brushed-dc-motors-when-and-why-to-choose-one-over-the-other_r1.0.pdf [Último acceso: 09 07 2024].

[13] «CookieRobotics,» [Online]. Disponible: https://cookierobotics.com/066/. [Último acceso: 09 07 2024].

[14] Max.P, «ArduPilot,» 10 10 2014. [En línea]. Disponible: http://autoquad.org/wiki/wiki/configuring-autoquad-flightcontroller/frame-motor-mixing-table/. [Último acceso: 28 06 2024]

[15] MATLAB, «Diseño de un Regulador Lineal Cuadrático,» [En línea]. Disponible: https://es.mathworks.com/help/control/ref/lti.lqr.html. [Último acceso: 28 06 2024].

[16] Documento cedido por el profesor sobre control difuso. [En línea]. Disponible: https://drive.google.com/file/d/1Oep-cJF7Ez8EohBoLl9WHzcNthGC58-D/view?usp=sharing [Último acceso: 09 07 2024].

[17] G. Welch and G. Bishop, "An introduction to the Kalman filter," TR 95-041, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, Chapel Hill, NC, Mar. 2002

[18] G. Welch and G. Bishop, "An introduction to the Kalman filter," KalmanFilter.net. [Online]. Disponible: https://www.kalmanfilter.net/ES/kalman1d_es.html. [Último acceso: 09 07 2024].

12. Anexos

12.1. Bloques empleados para simulación

12.1.1. MMA (*Motor Mixing Algorithm)*

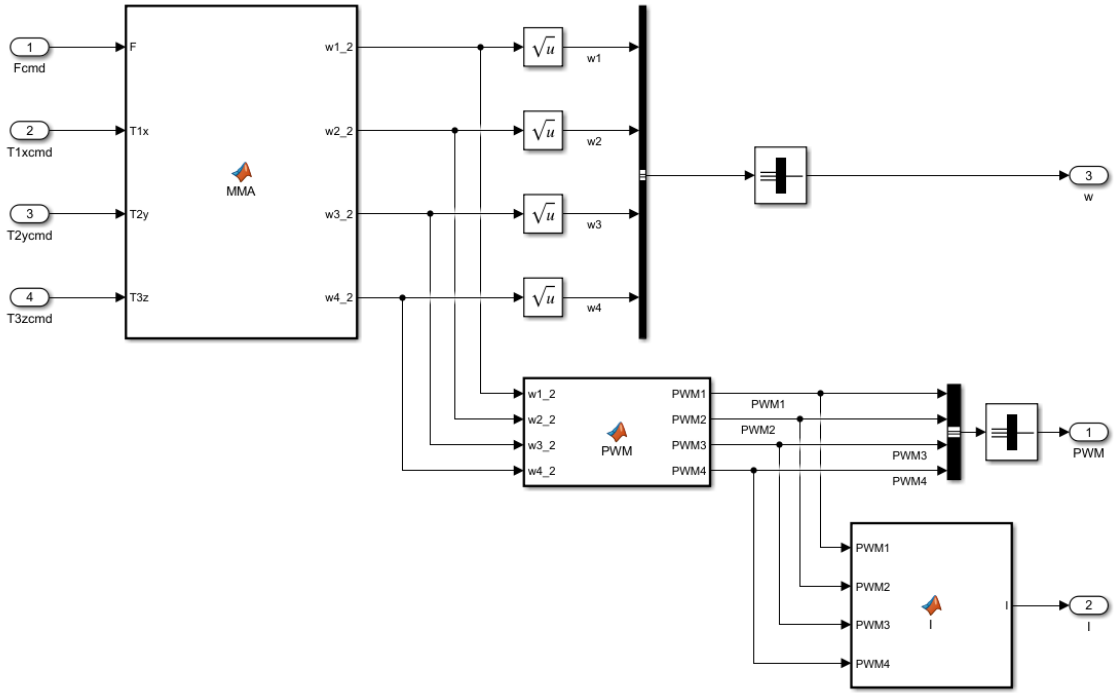


Ilustración 102: Bloque de Algoritmo de Mezcla de Motores (MMA). Simulink

12.1.1.1. MMA

```
function [w1_2, w2_2, w3_2, w4_2] = MMA(F, T1x, T2y, T3z)
kf = 0.00013294;
km = 4.67946*10^-6;
L = 0.15 / sqrt(2);

w1_2 = (1/(4*kf)) * F + (1/(4*kf*L)) * T1x + (1/(4*kf*L)) * T2y + (1/(4*km)) * T3z;
w2_2 = (1/(4*kf)) * F - (1/(4*kf*L)) * T1x + (1/(4*kf*L)) * T2y - (1/(4*km)) * T3z;
w3_2 = (1/(4*kf)) * F - (1/(4*kf*L)) * T1x - (1/(4*kf*L)) * T2y + (1/(4*km)) * T3z;
w4_2 = (1/(4*kf)) * F + (1/(4*kf*L)) * T1x - (1/(4*kf*L)) * T2y - (1/(4*km)) * T3z;
end
```

Ilustración 103: Función que calcula las velocidades angulares a partir de los pares y fuerzas

12.1.1.2. PWM

```
function [PWM1,PWM2,PWM3,PWM4] = PWM(w1_2, w2_2, w3_2, w4_2)

PWM1 = 7*10^(-15) * w1_2 * w1_2 * w1_2 - 7*10^(-10) * w1_2 * w1_2 + 4*10^(-5) * w1_2;
PWM2 = 7*10^(-15) * w2_2 * w2_2 * w2_2 - 7*10^(-10) * w2_2 * w2_2 + 4*10^(-5) * w2_2;
PWM3 = 7*10^(-15) * w3_2 * w3_2 * w3_2 - 7*10^(-10) * w3_2 * w3_2 + 4*10^(-5) * w3_2;
PWM4 = 7*10^(-15) * w4_2 * w4_2 * w4_2 - 7*10^(-10) * w4_2 * w4_2 + 4*10^(-5) * w4_2;
end
```

Ilustración 104: Función para calcular los comandos PWM a enviar a los ESC para alcanzar las velocidades de giro calculadas

12.1.1.3. I

```
function I  = I(PWM1, PWM2, PWM3, PWM4)

I1 = 3.5839 * (PWM1^3) + 6.9761 * (PWM1^2) + 2.1798 * PWM1 + 0.152;
I2 = 3.5839 * (PWM2^3) + 6.9761 * (PWM2^2) + 2.1798 * PWM2 + 0.152;
I3 = 3.5839 * (PWM3^3) + 6.9761 * (PWM3^2) + 2.1798 * PWM3 + 0.152;
I4 = 3.5839 * (PWM4^3) + 6.9761 * (PWM4^2) + 2.1798 * PWM4 + 0.152;

I = I1 + I2 + I3 + I4 + 0.2;

end
```

Ilustración 105: Función para calcular el consumo total de los rotores del cuadricóptero

12.1.2. Actuadores

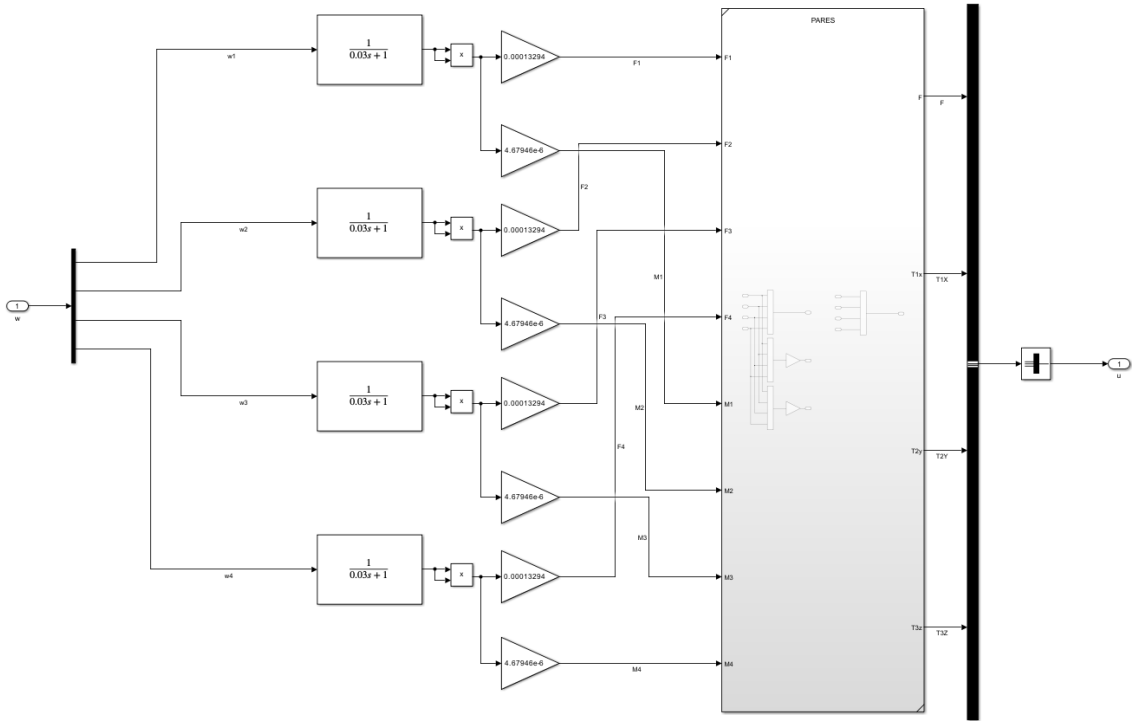


Ilustración 106: Bloque de Actuadores. Simulink

12.1.2.1. Dinámica del motor BLDC

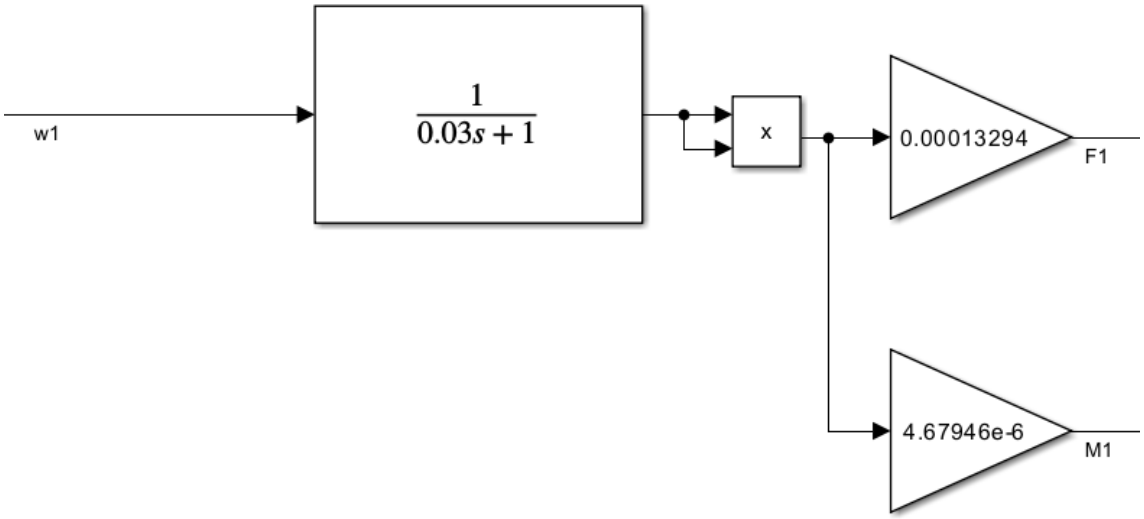


Ilustración 107: Bloques que definen la dinámica de los motores del cuadricóptero

12.1.2.2. Cálculo de los pares

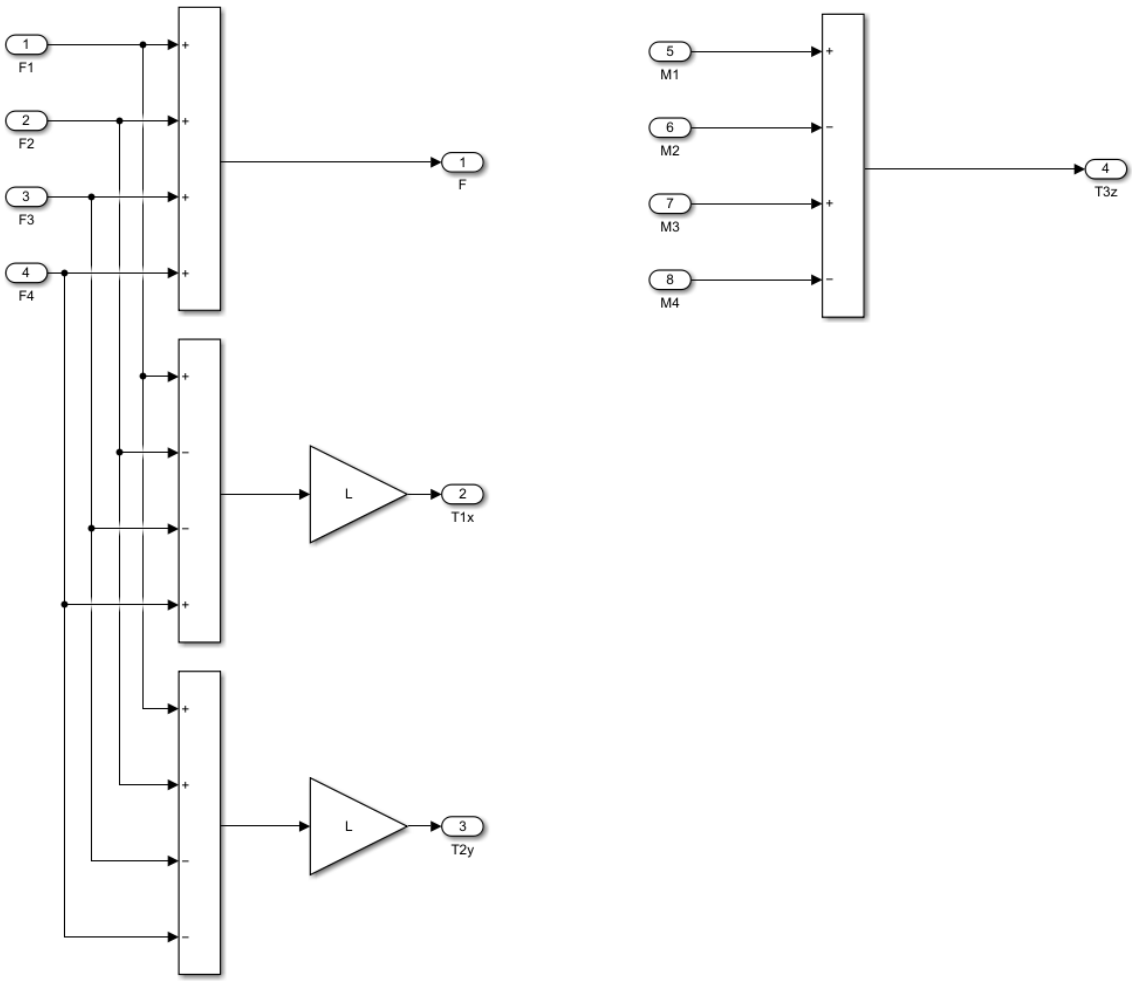


Ilustración 108: Bloque para calcular las fuerzas y pares a partir de las fuerzas y pares generados por los rotores

12.1.3. Modelo dinámico del dron

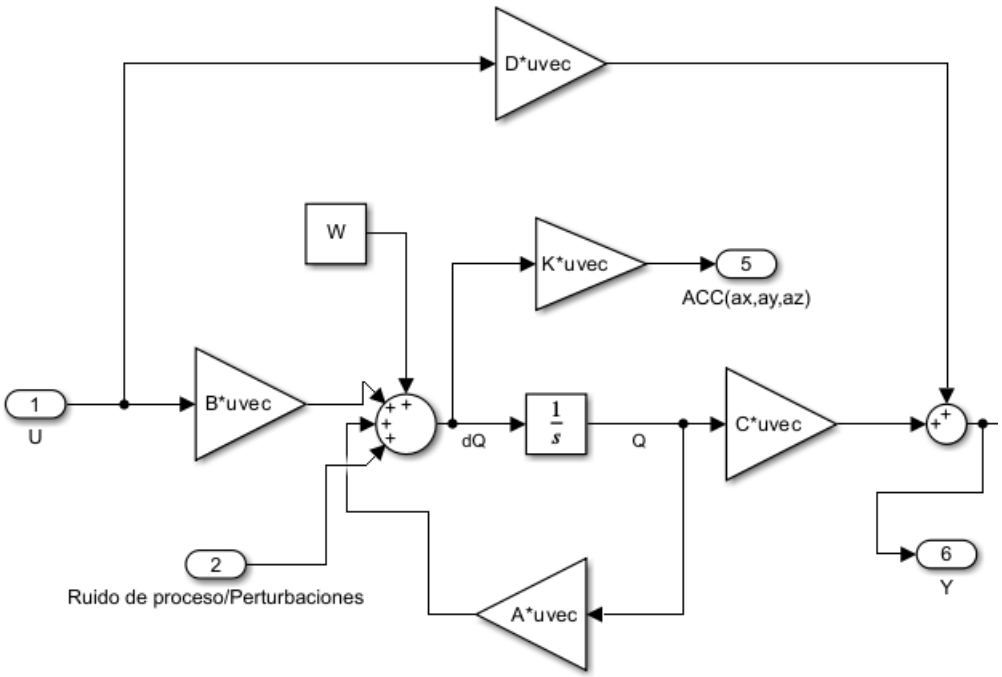


Ilustración 109: Modelo dinámica del cuadricóptero representado en ecuaciones de estado

Para graficar los datos de forma más visual, se añade:

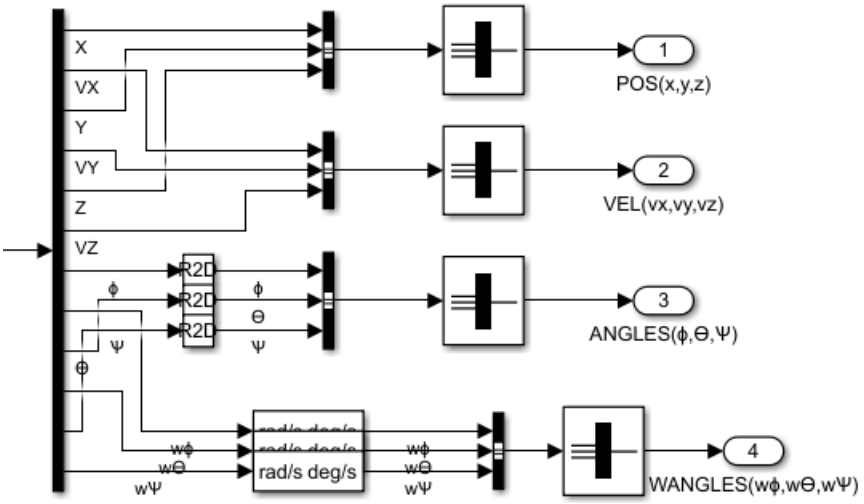


Ilustración 110: Demux para graficar cómodamente los estados del cuadricóptero

12.1.4. Bloque de simulación

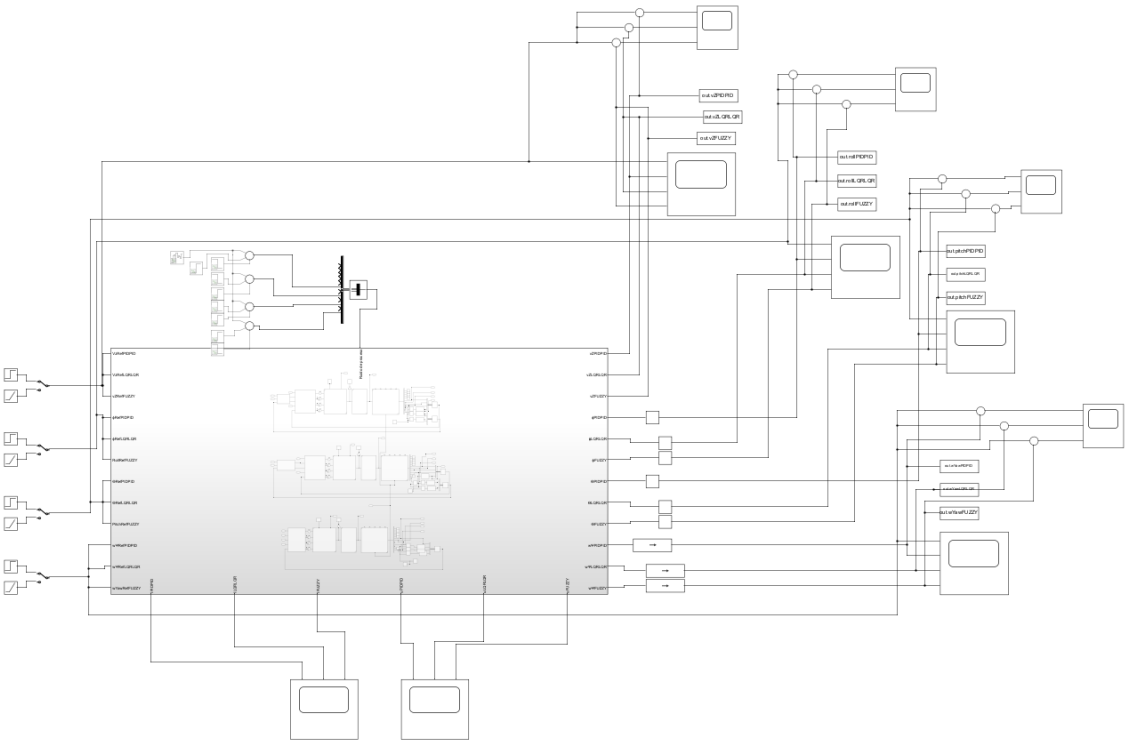


Ilustración 111: Simulación de controladores de forma simultánea

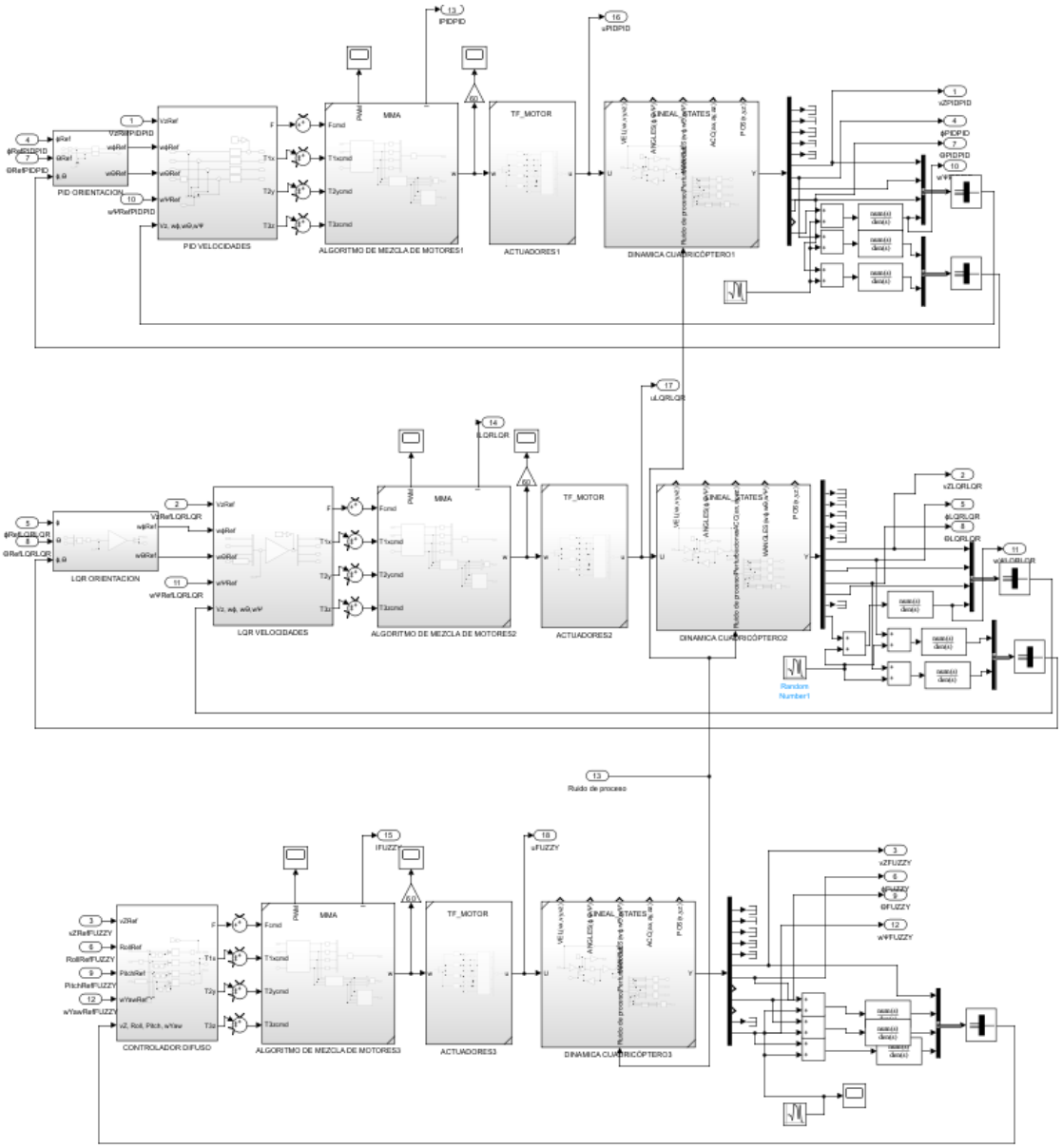


Ilustración 112: Lazos de control simulados

12.2. Códigos

12.2.1. Código banco de pruebas

```
//LIBRERIAS
#include "HX711.h"

//DEFINICIONES
#define DT 18     // DT de HX711
#define SCK 19      // SCK de HX711

//VARIABLES
const int pwmFreq = 50;         // Frecuencia de 50 Hz
const int pwmResolution = 16;   // Resolución de 16 bits
int dutyCycle = 0;
float microSeconds = 1;
float force = 0;

//OBJETOS
HX711 celda;

//SETUP
void setup() {
  //Comunicacion Serial
  Serial.begin(115200);
  delay(2000);
  //Iniciar módulo HX711
  Serial.println("********************INICIANDO BANCO DE PRUEBAS********************");
  celda.begin(DT, SCK);
  //Escala báscula
  celda.set_scale(1545.f);
  //Tarar báscula
  celda.tare();
  //Configurar motor
  ledcAttach(17, pwmFreq, pwmResolution);
  delay(2000);
  Serial.println("********************COMENZANDO EN 10 SEGUNDOS********************");
  delay(10000);
  //Calibración ESC
  Serial.println("Calibrando ESC");
  dutyCycle = (1 * 65536) / 20; // 1 ms (MIN) y 2ms(MAX)
  ledcWrite(17, dutyCycle);
  delay(10000);
}
```

```
//LOOP
void loop() {
  //Bucle para ir recorriendo diferentes velocidades angulares del motor
  for (microSeconds = 1; microSeconds < 2.1; microSeconds+= 0.1) {
    dutyCycle = (microSeconds * 65536) / 20;
    ledcWrite(17, dutyCycle);
    //Bucle para tomar varias lecturas de fuerza ejercida por el motor
    for (int i = 0; i < 10; i++) {
      //Promedio de 5 lecturas
      force = -celda.get_units(5);
      Serial.print("uS: ");
      Serial.print(microSeconds*1000);
      Serial.print("\tValor (gramos): ");
      Serial.println(force, 1);
      //Se resetea (apaga) HX711
      celda.power_down();
      delay(1000);
      //Se enciende HX711
      celda.power_up();
    }
  }
}
```

12.2.2. Código calibración banco de pruebas

Referencia del código: https://github.com/bitwiseAr/Curso-Arduino-desde-cero/blob/aacf3786192a2ef663343723ba904969fba0675a/Capitulo78/Capitulo78-Programa1.txt

```
/*
  Capitulo 78 de Arduino desde cero en Español.
  Programa para obtener el factor de escala en base a un peso conocido. Con el valor
  obtenido se divide por el peso colocado (en gramos) y aplicar en el segundo programa.
  Requiere libreria: HX711 Arduino Library de Bogdan

  Autor: bitwiseAr
  https://www.youtube.com/c/BitwiseAr

*/

#include "HX711.h"    // incluye libreria HX711

#define DT 18     // DT de HX711 a pin digital 2
#define SCK 19      // SCK de HX711 a pin digital 3

HX711 celda;      // crea objeto con nombre celda

void setup() {
  Serial.begin(9600);   // inicializa monitor serie a 9600 baudios

  celda.begin(DT, SCK);   // inicializa objeto con los pines a utilizar

  Serial.println("Para obtener factor de escala:"); // texto estatico descriptivo

  celda.set_scale();    // establece en factor de escala por defecto
  celda.tare();               // realiza la tara o puesta a cero
  Serial.println("Colocar un peso conocido (10 seg.)"); // texto estático descriptivo
  delay(10000);   // demora de 10 segundos para colocar el peso conocido
  Serial.println(celda.get_units(10));  // obtiene valor (promedio de 10 lecturas) para calcular
factor de escala
  Serial.println("Hecho. Dividir el valor mostrado por el peso colocado");  // texto descriptivo
}

void loop() {
    // nada por aqui
}
```

12.2.3. Código para la recepción de datos en tiempo real en Python

Datos ruido IMU: aceleraciones lineales y velocidades angulares

Sketch DATOS_IMU.py

```
#Librerias
import socket
import struct
import time
import matplotlib.pyplot as plt

#Puerto de recepcion
local_port = 54321
#buffer para almacenar datos recibidos
data_buffer = {
    'wRoll': [],
    'wPitch': [],
    'wYaw': [],
    'aX': [],
    'aY': [],
    'aZ': []
}

# Crear punto UDP que escuche cualquier IP que envíe al puerto 54321
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('0.0.0.0', local_port))

try:
    while True:
        data, addr = sock.recvfrom(24)  # Recibir datos del socket (24 bytes para 6
floats de 4 bytes cada uno)
        if len(data) >= 24:  # Asegurar que hay al menos 24 bytes disponibles
            # Desempaquetar los primeros 24 bytes como seis floats
            values = struct.unpack('ffffff', data[:24])
            # Asignar cada valor a variables wRoll, wPitch, wYaw, aX, aY, aZ
            wRoll = values[0]
            wPitch = values[1]
            wYaw = values[2]
            aX = values[3]
            aY = values[4]
            aZ = values[5]
            # Mostrar por pantalla los valores recibidos
            print(f'wRoll: {wRoll}   wPitch: {wPitch}   wYaw: {wYaw}   aX: {aX}   aY:
{aY}   aZ: {aZ}')
            # Actualizar de forma dinámica el buffer de datos con .append
            data_buffer['wRoll'].append(wRoll)
            data_buffer['wPitch'].append(wPitch)
            data_buffer['wYaw'].append(wYaw)
            data_buffer['aX'].append(aX)
```

```
            data_buffer['aY'].append(aY)
            data_buffer['aZ'].append(aZ)

            time.sleep(0.001)
# Permitir la interrupción del bucle con Ctrl+C
except KeyboardInterrupt:
    print('Interrupción del usuario, terminando el programa.')
finally:
    sock.close()  # Cerrar punto

# Graficar los datos recibidos al terminar
plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)
plt.plot(data_buffer['wRoll'], label='wRoll')
plt.title('wRoll')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 2)
plt.plot(data_buffer['wPitch'], label='wPitch')
plt.title('wPitch')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 3)
plt.plot(data_buffer['wYaw'], label='wYaw')
plt.title('wYaw')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 4)
plt.plot(data_buffer['aX'], label='aX')
plt.title('aX')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 5)
plt.plot(data_buffer['aY'], label='aY')
plt.title('aY')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()
```

```
plt.subplot(3, 2, 6)
plt.plot(data_buffer['aZ'], label='aZ')
plt.title('aZ')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.tight_layout()
plt.show()
```

Datos ruido altímetro: altura en Z y velocidad lineal en Z
Sketch DATOS_LASER.py

```
#Librerias
import socket
import struct
import time
import matplotlib.pyplot as plt

#Puerto de recepcion
local_port = 54321
#buffer para almacenar datos recibidos
data_buffer = {
    'z': [],
    'aZ': []
}

# Crear punto UDP que escuche cualquier IP que envíe al puerto 54321
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('0.0.0.0', local_port))
try:
    while True:
        data, addr = sock.recvfrom(8)  # Recibir datos (8 bytes para 2 floats de 4 bytes
cada uno)
        if len(data) >= 8:  # Asegurar que hay al menos 8 bytes disponibles
            # Desempaquetar los primeros 8 bytes como dos floats
            values = struct.unpack('ff', data[:8])
            # Asignar cada valor a las variables z y aZ
            z = values[0]
            aZ = values[1]
            # Mostrar por pantalla los valores recibidos
            print(f'z: {z}   aZ: {aZ}')
            # Actualizar de forma dinámica el buffer de datos con .append
            data_buffer['z'].append(z)
            data_buffer['aZ'].append(aZ)

            time.sleep(0.001)
```

```
# Permitir la interrupción del bucle con Ctrl+C
except KeyboardInterrupt:
    print('Interrupción del usuario, terminando el programa.')
finally:
    sock.close()  # Cerrar punto

# Graficar los datos
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.plot(data_buffer['z'], label='z')
plt.title('z')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(data_buffer['aZ'], label='aZ')
plt.title('aZ')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.tight_layout()
plt.show()
```

Datos IMU: velocidad lineal en Z, velocidad angular de guiñada, ángulo de cabeceo, y ángulo de alabeo

Sketch DATOS_IMU_IMPLEMENTACION.py

```
#Librerias
import socket
import struct
import time
import matplotlib.pyplot as plt

#Puerto de recepcion
local_port = 54321
#buffer para almacenar datos recibidos
data_buffer = {
    'Roll': [],
    'Pitch': [],
    'wYaw': [],
    'vZ': []
}

# Crear punto UDP que escuche cualquier IP que envíe al puerto 54321
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('0.0.0.0', local_port))

try:
    while True:
        data, addr = sock.recvfrom(16)  # Recibir datos del socket (16 bytes para 4 floats
de 4 bytes cada uno)
        if len(data) >= 24:  # Asegurar que hay al menos 24 bytes disponibles
            # Desempaquetar los primeros 24 bytes como seis floats
            values = struct.unpack('ffff', data[:16])
            # Asignar cada valor a variables Roll, Pitch, wYaw, vZ
            Roll = values[0]
            Pitch = values[1]
            wYaw = values[2]
            vZ = values[3]
            # Mostrar por pantalla los valores recibidos
            print(f'wRoll: {wRoll}   wPitch: {wPitch}   wYaw: {wYaw}   aX: {aX}   aY:
{aY}   aZ: {aZ}')
            # Actualizar de forma dinámica el buffer de datos con .append
            data_buffer['Roll'].append(Roll)
            data_buffer['Pitch'].append(Pitch)
            data_buffer['wYaw'].append(wYaw)
            data_buffer['vZ'].append(vZ)



            time.sleep(0.001)
# Permitir la interrupción del bucle con Ctrl+C
```

```
except KeyboardInterrupt:
    print('Interrupción del usuario, terminando el programa.')
finally:
    sock.close()  # Cerrar punto

# Graficar los datos recibidos al terminar
plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)
plt.plot(data_buffer['Roll'], label='Roll')
plt.title('Roll')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 2)
plt.plot(data_buffer['Pitch'], label='Pitch')
plt.title('Pitch')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 3)
plt.plot(data_buffer['wYaw'], label='wYaw')
plt.title('wYaw')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.subplot(3, 2, 4)
plt.plot(data_buffer['vZ'], label='vZ')
plt.title('vZ')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.legend()

plt.tight_layout()
plt.show()
```

12.2.4. Código Aplicación Emisora

```
//********LIBRERIAS********//
import android.view.MotionEvent;    //Empleada para eventos de movimiento en la pantalla del dispositivo
import android.widget.Toast;        //Empleada para mostrar mensajes de la aplicación (errores, informacion de conexión, etc)
import java.net.*;                  //Empleada para manejar operaciones de red (enviar y recibir datos, gestión de conexiones, etc)

//********INSTANCIAS********//
DatagramSocket socket;

//********VARIABLES********//
//WIFI
String serverIP = "192.168.122.77"; // IP receptor
int serverPort = 12345; // Puerto del receptor

//Joysticks
float j1X, j1Y; // Joystick izquierdo
float j2X, j2Y; // Joystick derecho
float jSize = 200; //Tamaño circulo externo
float jRadius = jSize / 2; //Tamaño circulo interno
boolean j1Pressed = false;  //Joystick izquierdo presionado (true o false)
boolean j2Pressed = false;  //Joystick derecho presionado (true o false)
float dY1 = 0;  // Valor eje Y joystick izquierdo
float dX1 = 0;  // Valor eje X joystick izquierdo
float dY2 = 0;  // Valor eje Y joystick derecho
float dX2 = 0;  // Valor eje X joystick derecho

//Botones
float buttonWidth = 100; // Ancho botones
float buttonHeight = 40; // Alto botones
boolean yawDisabled = false;  //Boton yawdisabled presionado (true o false)
boolean rollDisabled = false;  //Boton rolldisabled presionado (true o false)
boolean pitchDisabled = false;  //Boton pitchdisabled presionado (true o false)
float yawButtonX, yawButtonY;    //Boton yawdisabled coordenadas
float rollButtonX, rollButtonY;   //Boton rolldisabled coordenadas
float pitchButtonX, pitchButtonY; //Boton pitchdisabled coordenadas
boolean power0n = false; // Estado del botón de encender/apagar (true o false)
float powerButtonX, powerButtonY; // Coordenadas del botón de encender/apagar
//********SETUP********//
void setup() {
  fullScreen();      //pantalla completa
  //Ubicacion Joysticks izquierdo y derecho en X e Y
  j1X = width / 4;
  j1Y = height / 2;
  j2X = width * 3 / 4;
  j2Y = j1Y;
  //Ubicacion botones en Y
  yawButtonY = height - buttonHeight - 20;
  rollButtonY = height - buttonHeight - 20;
  pitchButtonY = height - buttonHeight - 20;
  powerButtonY = 20;
  //Ubicacion botones en X
  yawButtonX = width / 2 - 1.5 * buttonWidth - 20;
  rollButtonX = yawButtonX + buttonWidth + 20;
  pitchButtonX = rollButtonX + buttonWidth + 20;
  powerButtonX = width - buttonWidth - 20;
  // Iniciar conexión al servidor
  connectToServer();
}
```

```
60  //********DRAW*********//
61  void draw() {
62    //Color fondo
63    background(255);
64    // Dibujamos los botones de desactivar yaw, roll y pitch
65    drawButton(yawButtonX, yawButtonY, "Disable Yaw", yawDisabled);
66    drawButton(rollButtonX, rollButtonY, "Disable Roll", rollDisabled);
67    drawButton(pitchButtonX, pitchButtonY, "Disable Pitch", pitchDisabled);
68    // Dibujamos el botón de encender/apagar mostrando ON u OFF si se presiona
69    drawButton(powerButtonX, powerButtonY, "Power " + (power0n ? "On" : "Off"), power0n);
70    // Dibuja los joysticks
71    drawJoystick(width / 4, height / 2, j1X, j1Y);
72    drawJoystick(width * 3 / 4, height / 2, j2X, j2Y);
73
74    // Si no esta desactivado el yaw, mapear sus valores entre -100 y 100 °/s
75    if (!yawDisabled) {
76      dX1 = map(j1X, width / 4 - jRadius * 3, width / 4 + jRadius * 3, -100, 100);  // Mapear j1X a Yaw (-100 a 100)
77    } else {
78      dX1 = 0;
79    }
80    // Mapear valores throttle enre 1 y -1 m/s
81    dY1 = map(j1Y, height / 2 + jRadius * 3, height / 2 - jRadius * 3, -1, 1);    // Mapear j1Y a Throttle (-1 a 1)
82    // Si no esta desactivado el roll, mapear sus valores entre -10 y 10 °
83    if (!rollDisabled) {
84      dX2 = map(j2X, width * 3 / 4 - jRadius * 3, width * 3 / 4 + jRadius * 3, -10, 10);  // Mapear j2X a Roll (-10 a 10)
85    } else {
86      dX2 = 0;
87    }
88    // Si no esta desactivado el pitch, mapear sus valores entre -10 y 10 °
89    if (!pitchDisabled) {
90      dY2 = map(j2Y, height / 2 + jRadius * 3, height / 2 - jRadius * 3, -10, 10);  // Mapear j2Y a Pitch (-10 a 10)
91    } else {
92      dY2 = 0;
93    }
94    //Reestringir valores de cada joystick
95    dX1 = constrain(dX1, -100, 100);
96    dY1 = constrain(dY1, -1, 1);
97    dX2 = constrain(dX2, -10, 10);
98    dY2 = constrain(dY2, -10, 10);
99    // Mostramos en pantalla el valor de cada joystick mapeado en color negro
100   fill(0);
101   textSize(20);
102   textAlign(LEFT);
103   text("Yaw: " + nf(dX1, 0, 2), 20, 50);
104   text("Throttle: " + nf(dY1, 0, 2), 20, 70);
105   text("Roll: " + nf(dX2, 0, 2), 20, 90);
106   text("Pitch: " + nf(dY2, 0, 2), 20, 110);
107   // Creamos un string para almacenar los valores de cada joystick mapeados y pasados también a string
108   String data = String.valueOf(dY1) + "," + String.valueOf(dX1) + "," + String.valueOf(dX2) + "," + String.valueOf(dY2) + "," + (power0n ? "1" : "0");
109   //Llamamos a la funcion para enviar los datos
110   sendData(data);
111 }
```

```
//********CONECTAR AL RECEPTOR/SERVIDOR********//
//Intentamos conectar al servidor. Si no lanzar excepcion
void connectToServer() {
  new Thread(new Runnable() {
    @Override
    public void run() {
      try {
        socket = new DatagramSocket();
        showToast("Conectado");
      } catch (SocketException e) {
        e.printStackTrace();
        showToast("Error al conectar");
      }
    }
  }).start();
}

//********ENVIAR DATOS********//
//Intentamos enviar los datos. Si no es posible, lanzar excepcion y mostrar en mensaje de la aplicacion llamando a showToast
void sendData(final String data) {
  new Thread(new Runnable() {
    @Override
    public void run() {
      try {
        // Convertimos el String a bytes (mas rópido y eficiente el envío de esta forma)
        byte[] sendData = data.getBytes();
        // Creamos el paquete UDP
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, InetAddress.getByName(serverIP), serverPort);
        // Enviamos el paquete UDP
        socket.send(sendPacket);
        Thread.sleep(10);
      } catch (Exception e) {
        e.printStackTrace();
        showToast("Error al enviar datos por UDP");
      }
    }
  }).start();
}
```

```
//********MOSTRAR MENSAJE DE ERROR*********//
//Si no se pudo enviar el paquete mostrar el estado de error
void showToast(final String message) {
  getActivity().runOnUiThread(new Runnable() {
    @Override
    public void run() {
      Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();
    }
  });
}

//********DIBUJAR JOYSTICK********//
void drawJoystick(float x, float y, float jX, float jY) {
  // Dibujamos el círculo gris del joystick
  fill(200);
  ellipse(x, y, jSize, jSize);
  // Dibujamos el círculo oscuro del joystick
  fill(150);
  ellipse(jX, jY, jSize * 0.6, jSize * 0.6);
}

//********DIBUJAR BOTONES********//
void drawButton(float x, float y, String label, boolean pressed) {
  // Dibujamos el botón. Si se presiona lo coloreamos verde. Si no rojo.
  fill(pressed ? color(0, 255, 0) : color(255, 0, 0));
  //Dimensiones del rectangulo
  rect(x, y, buttonWidth, buttonHeight, 10);
  // Dibujamos el texto del botón de color blanco
  fill(255);
  textSize(16);
  textAlign(CENTER, CENTER);
  text(label, x + buttonWidth / 2, y + buttonHeight / 2);
}
```

```
//********FUNCION SI SE TOCA ALGUN ELEMENTO********//
void touchStarted() {
  //Bucle para recorrer todos los posibles botones que se pueden haber tocado
  for (int i = 0; i < touches.length; i++) {
    //Si se ha tocado el joystick izquierdo (dentro de las dimensiones del joystick)
    if (dist(touches[i].x, touches[i].y, j1X, j1Y) < jRadius) {
      j1Pressed = true;  //Presionado = true
    }
    //Si se ha tocado el joystick derecho (dentro del las dimensiones del joystick)
    if (dist(touches[i].x, touches[i].y, j2X, j2Y) < jRadius) {
      j2Pressed = true;  //Presionado = true
    }
    //Si se ha tocado el botón de Yaw (dentro de las dimensiones del botón)
    if (touches[i].x > yawButtonX && touches[i].x < yawButtonX + buttonWidth && touches[i].y > yawButtonY && touches[i].y < yawButtonY + buttonHeight) {
      yawDisabled = !yawDisabled;  //Si estaba desactivado se activa. Si esta ativado se desactiva
    }
    //Si se ha tocado el botón de Roll (dentro de las dimensiones del botón)
    if (touches[i].x > rollButtonX && touches[i].x < rollButtonX + buttonWidth && touches[i].y > rollButtonY && touches[i].y < rollButtonY + buttonHeight) {
      rollDisabled = !rollDisabled;  //Si estaba desactivado se activa. Si esta ativado se desactiva
    }
    // Si se ha tocado el botón de Pitch (dentro de las dimensiones del botón)
    if (touches[i].x > pitchButtonX && touches[i].x < pitchButtonX + buttonWidth && touches[i].y > pitchButtonY && touches[i].y < pitchButtonY + buttonHeight) {
      pitchDisabled = !pitchDisabled;  //Si estaba desactivado se activa. Si esta ativado se desactiva
    }
    // Si se ha tocado el botón de encender/apagar (dentro de las dimensiones del botón)
    if (touches[i].x > powerButtonX && touches[i].x < powerButtonX + buttonWidth && touches[i].y > powerButtonY && touches[i].y < powerButtonY + buttonHeight) {
      power0n = !power0n;  //Si estaba apagado se enciende. Si estaba encendido se apaga
    }
  }
}
//********FUNCION SI SE ARRASTRA ALGUN ELEMENTO********//
void touchMoved() {
  //Bucle para recorrer todos los posibles joystick que se pueden haber arrastrado
  for (int i = 0; i < touches.length; i++) {
    //Si esta presioando el joystick izquierdo y se arrastra una cierta distancia, cambiamos las coordenadas de los circulos a donde se arrastre
    if (j1Pressed && dist(touches[i].x, touches[i].y, j1X, j1Y) < jRadius) {
      j1X = touches[i].x;
      j1Y = touches[i].y;
    }
    //Si esta presioando el joystick derecho y se arrastra una cierta distancia, cambiamos las coordenadas de los circulos a donde se arrastre
    if (j2Pressed && dist(touches[i].x, touches[i].y, j2X, j2Y) < jRadius) {
      j2X = touches[i].x;
      j2Y = touches[i].y;
    }
  }
}

//********FUNCION SI SE DEJA DE TOCAR ALGUN ELEMENTO********//
void touchEnded() {
  // Si se ha dejado de tocar el joystick izquierdo lo retornamos a su posición de reposo
  if (j1Pressed) {
    j1Pressed = false;  //Presionado = false
    j1X = width / 4;
  }
  // Si se ha dejado de tocar el joystick derecho lo retornamos a su posición de reposo
  if (j2Pressed) {
    j2Pressed = false;  //Presionado = false
    j2X = width * 3 / 4;
    j2Y = height / 2;
  }
}
```

12.2.5. Código Configuración ESC

```
//LIBRERIAS
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <Servo.h>
//**********VARIABLES**********//
float PWM1 = 0;

//Comun
float vZRef = 0;
float wYawRef = 0;
float rollRef = 0;
float pitchRef = 0;
int power = 0;

// Wi-Fi
char ssid[] = "OPPO A53";
char pass[] = "611b10a883c5";
int status = WL_IDLE_STATUS;
unsigned int localUdpPort = 12345;
unsigned int clientUdpPort = 54321;

//**********INSTANCIAS*********//
WiFiUDP Udp;
Servo motor1;
Servo motor2;
Servo motor3;
Servo motor4;

//****PROTOTIPOS FUNCIONES*****//
void APPgetData(void);
void setMotors(void);
//************SETUP************//
void setup() {
  //Iniciar COM Serial
  Serial.begin(115200);
  //RGB
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);
  //Iniciar COM WiFi
  while (status != WL_CONNECTED) {
    Serial.print("Conectando a SSID: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
```

```
    digitalWrite(LEDB, LOW);
    delay(5000);

  }
  Serial.print("Conectado. IP Arduino: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
  // Iniciar puerto
  Udp.begin(localUdpPort);
  Serial.print("Escuchando puerto UDP: ");
  Serial.println(localUdpPort);
  while (power != 1) {
    APPgetData();
  }
  motor1.attach(6);
  motor2.attach(17);
  motor3.attach(16);
  motor4.attach(7);
}

//*************LOOP************//
void loop() {
  if (power == 1) {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
    APPgetData();
    setMotors();
  } else {
    APPgetData();
    motor1.writeMicroseconds(1000);
    motor2.writeMicroseconds(1000);
    motor3.writeMicroseconds(1000);
    motor4.writeMicroseconds(1000);
  }
}

//**********FUNCIONES**********//
void setMotors(void) {
  PWM1 = (1000 + (vZRef - (-1)) * ((2000 - 1000)/(1 - (-1))));
  Serial.println(PWM1);
  motor1.writeMicroseconds(PWM1);
  motor2.writeMicroseconds(PWM1);
  motor3.writeMicroseconds(PWM1);
  motor4.writeMicroseconds(PWM1);
```

```
  delay(50);
}

void APPgetData(void) {
  int packetSize = Udp.parsePacket();
  if (packetSize > 0) {
    char incomingPacket[255];
    int len = Udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = 0;
      sscanf(incomingPacket, "%f,%f,%f,%f,%d", &vZRef, &wYawRef, &rollRef,
&pitchRef, &power);
      rollRef *= PI / 180;
      pitchRef *= PI / 180;
      wYawRef *= PI / 180;
    }
  }
}
```

12.2.6. Código para enviar datos del ruido de sensores

```
//LIBRERIAS
#include <Arduino_LSM6DSOX.h>
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <Servo.h>
//*********VARIABLES*********//

// IMU altimetro
float aXOffset = 0;  float aYOffset = 0; float aZOffset = 0;
float gXCal = 0;       float gYCal = 0;       float gZCal = 0;
float aX = 0;          float aY = 0;          float aZ = 0;          float AZ = 0;
float gX = 0;          float gY = 0;          float gZ = 0;
float roll = 0;        float pitch = 0;       float wRoll = 0;    float wPitch = 0;  float
wYaw = 0;
float vZ = 0;          float z = 0;

// Motor PWM
float PWM1 = 0;

//Comun
float vZRef = 0;       float rollRef = 0;         float wRollRef = 0;        float pitchRef
= 0;         float wPitchRef = 0;         float wYawRef = 0;

//Parada emergencia
int power = 0;

// Wi-Fi
char ssid[] = "OPPO A53";
char pass[] = "611b10a883c5";
int status = WL_IDLE_STATUS;
unsigned int localUdpPort = 12345;
unsigned int clientUdpPort = 54321;

//*********INSTANCIAS*********//
WiFiUDP Udp;
Servo motor1;
Servo motor2;
Servo motor3;
Servo motor4;
IPAddress receiverIp(192,168,45,63);

//****PROTOTIPOS FUNCIONES*****//
void PCsendData(void);
void calibrarIMU(void);
void IMUgetData(void);
void APPgetData(void);
```

```
void MMA(void);
void setMotors(void);

//***********SETUP***********//
void setup() {
  //Iniciar COM Serial
  Serial.begin(115200);
  //RGB
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);
  //Iniciar COM WiFi
  while (status != WL_CONNECTED) {
    Serial.print("Conectando a SSID: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
    digitalWrite(LEDB, LOW);
    delay(5000);

  }
  Serial.print("Conectado. IP Arduino: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
  // Iniciar puerto
  Udp.begin(localUdpPort);
  Serial.print("Escuchando puerto UDP: ");
  Serial.println(localUdpPort);
  while (power != 1) {
    APPgetData();
  }
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, HIGH);
  // Configurar Motores y ESC
  motor1.attach(6);
  motor2.attach(17);
  motor3.attach(16);
  motor4.attach(7);
  motor1.writeMicroseconds(1000);
  motor2.writeMicroseconds(1000);
  motor3.writeMicroseconds(1000);
  motor4.writeMicroseconds(1000);
  delay(10000);
```

```
  //Iniciar IMU
  if (!IMU.begin()) {
    Serial.println("Error al inicializar IMU");
    while (1);
  }
  //Calibrar IMU
  calibrarIMU();
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
}

//************LOOP***********//
void loop() {
  tiempoInicio = micros();
  if ((power == 1)) {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
    APPgetData();
    IMUgetData();
    setMotors();
    PCsendData();
  } else {
    APPgetData();
    motor1.writeMicroseconds(1000);
    motor2.writeMicroseconds(1000);
    motor3.writeMicroseconds(1000);
    motor4.writeMicroseconds(1000);
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
  }
  tiempoFin = micros();
  //Serial.println(duracion);
  while ((tiempoFin - tiempoInicio) < tiempoDeseado) {
    tiempoFin = micros();
  }
}

//*********FUNCIONES**********//
void calibrarIMU(void) {
  for (int i = 0; i < 10000; i++) {
    if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
      IMU.readGyroscope(gX, gY, gZ);
      IMU.readAcceleration(aX, aY, aZ);
```

```
    }
    gXCal += gX;
    gYCal += gY;
    gZCal += gZ;
    aXOffset += aX;
    aYOffset += aY;
    aZOffset += aZ;
  }
  gXCal /= 10000;
  gYCal /= 10000;
  gZCal /= 10000;
  aXOffset/= 10000;
  aYOffset/= 10000;
  aZOffset = (aZOffset / 10000) - 1;
  zIni = 0;
}

void IMUgetData(void) {
  if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
    IMU.readGyroscope(gX, gY, gZ);
    IMU.readAcceleration(aX, aY, aZ);
    gX -= gXCal;
    gY -= gYCal;
    gZ -= gZCal;
    aX -= aXOffset;
    aY -= aYOffset;
    aZ -= aZOffset;
    wRoll = gX * PI / 180;
    wPitch = gY * PI / 180;
    wYaw = -gZ * PI / 180;
  }
}

// Función para establecer los valores de los motores
void setMotors(void) {
  PWM1 = (1000 + (vZRef - (-1)) * ((2000 - 1000)/(1 - (-1))));
  motor1.writeMicroseconds(PWM1);
  motor2.writeMicroseconds(PWM1);
  motor3.writeMicroseconds(PWM1);
  motor4.writeMicroseconds(PWM1);
  delay(10);
}
```

```
//Recibir paquetes desde la app
void APPgetData(void) {
  int packetSize = Udp.parsePacket();
  if (packetSize > 0) {
    char incomingPacket[255];
    int len = Udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = 0;
      sscanf(incomingPacket, "%f,%f,%f,%f,%d", &vZRef, &wYawRef, &rollRef,
&pitchRef, &power);
      rollRef *= PI / 180;
      pitchRef *= PI / 180;
      wYawRef *= PI / 180;
    }
  }
}


void PCsendData(void) {
  uint8_t packetBuffer[24];
  memcpy(packetBuffer, &wRoll, sizeof(wRoll));
  memcpy(packetBuffer + 4, &wPitch, sizeof(wPitch));
  memcpy(packetBuffer + 8, &wYaw, sizeof(wYaw));
  memcpy(packetBuffer + 12, &aX, sizeof(aX));
  memcpy(packetBuffer + 16, &aY, sizeof(aY));
  memcpy(packetBuffer + 20, &aZ, sizeof(aZ));
  Udp.beginPacket(receiverIp, clientUdpPort);
  Udp.write(packetBuffer, sizeof(packetBuffer));
  Udp.endPacket();
}
```

12.2.7. Código del cuadricóptero

```
//DEFINICIONES
#define PID
//#define LQR
#define PRINT

//LIBRERIAS
#include <Arduino_LSM6DSOX.h>
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <Servo.h>
#include <Wire.h>
#include <BasicLinearAlgebra.h>
#include <ElementStorage.h>
#include <VL53L0X.h>

//ESPACIOS DE NOMBRE
using namespace BLA;

//**********VARIABLES**********//
// Matrices del filtro de Kalman
BLA::Matrix<2, 2> A_ = {1, 0.01, 0, 1};
BLA::Matrix<2, 1> B_ = {0.01*0.01/2, 0.01};
BLA::Matrix<2, 2> H_ = {1, 0, 0, 1};
BLA::Matrix<2, 2> Q_ = {0.12*0.12, 0, 0, 0.050*0.050};
BLA::Matrix<2, 2> R_ = {0.12*0.12, 0, 0, 0.050*0.050};
BLA::Matrix<2> s_ = {0, 0};
BLA::Matrix<2, 2> P_ = {1, 0, 0, 1};
float estadoKalmanRoll = 0;
float incertidumbreKalmanRoll = 0.035*0.035;
float estadoKalmanPitch = 0;
float incertidumbreKalmanPitch = 0.035*0.035;
float estadoKalman = 0;
float incertidumbreKalman = 0;

// IMU y LASER
float aXOffset = 0;  float aYOffset = 0; float aZOffset = 0;
float gXCal = 0;       float gYCal = 0;       float gZCal = 0;
float aX = 0;          float aY = 0;          float aZ = 0;          float AZ = 0;
float gX = 0;          float gY = 0;          float gZ = 0;
float roll = 0;        float pitch = 0;       float wRoll = 0;       float wPitch = 0;       float
wYaw = 0;
float vZ = 0;          float z = 0;           float zIni = 0;     float zKalman = 0;  float vZKalman
= 0;

// Motor PWM
float PWM1 = 0;        float PWM2 = 0;          float PWM3 = 0;        float PWM4 = 0;
```

```
float kf = 0.00013294;  float km = 0.00000467946;  float L = 0.1061;
float w1_2 = 0;
float w2_2 = 0;
float w3_2 = 0;
float w4_2 = 0;

// PID
#ifdef PID
  float vZErrorPrev = 0;  float rollErrorPrev = 0;    float wRollErrorPrev = 0;   float
pitchErrorPrev = 0;    float wPitchErrorPrev = 0;    float wYawErrorPrev = 0;
  float DvZprev = 0;      float DrollPrev = 0;        float DwRollPrev = 0;       float
DpitchPrev = 0;        float DwPitchPrev = 0;        float DwYawPrev = 0;
  float KpVz = 15;        float KdVz = 0;
  float KpWRoll = 0.5;    float KdWRoll = 0.0;      float KpRoll = 2.5;
  float KpWPitch = 0.5;   float KdWPitch = 0.0;     float KpPitch = 2.5;
  float KpWYaw = 0.2;    float KdWYaw = 0.0;
  float outPD[] = {0, 0, 0};  float outP = 0;         float N = 100;
#endif

//LQR
#ifdef LQR
  BLA::Matrix<2, 2> K1 = {3.4157, 0 , 0, 3.1623};
  BLA::Matrix<4, 4> K2 = {9.3541, 0, 0, 0, 0, 0.0816, 0, 0, 0, 0, 0.1211, 0, 0, 0, 0,
0.2236};
  BLA::Matrix<4, 1> error1;
  BLA::Matrix<4, 1> u1;
  BLA::Matrix<2, 1> error2;
  BLA::Matrix<2, 1> u2;
#endif

//Comun
float vZRef = 0;        float rollRef = 0;          float wRollRef = 0;         float pitchRef =
0;          float wPitchRef = 0;          float wYawRef = 0;
float vZError = 0;      float rollError = 0;        float wRollError = 0;       float pitchError =
0;        float wPitchError = 0;        float wYawError = 0;
uint32_t tiempoLoop = 0;  uint32_t tiempo = 0;
float Thrust = 0;       float uWRoll = 0;           float T1x = 0;              float uWPitch =
0;          float T2y = 0;                float T3z = 0;

//Tiempo
const unsigned long tiempoDeseado = 10000;
unsigned long tiempoInicio;
unsigned long tiempoFin;

//Parada emergencia
int power = 0;
```

```
// Wi-Fi
char ssid[] = "OPPO A53";
char pass[] = "611b10a883c5";
int status = WL_IDLE_STATUS;
unsigned int localUdpPort = 12345;
unsigned int clientUdpPort = 54321;

//*********INSTANCIAS*********//
WiFiUDP Udp;
Servo motor1;
Servo motor2;
Servo motor3;
Servo motor4;
IPAddress receiverIp(192,168,45,63);
VL53L0X sensor;

//****PROTOTIPOS FUNCIONES*****//
void PCsendData(void);
void calibrarIMU(void);
void getData(void);
void APPgetData(void);
void Controller(void);
#ifdef PID
  void PD(float error, float Kp, float Kd, float& errorPrev, float max, float min);
  void P(float error, float Kp);
#endif
#ifdef LQR
  void LQRVelocities(float vZError, float wRollError, float wPitchError, float wYawError);
  void LQRAngles(float rollError, float pitchError);
#endif
void MMA(void);
void setMotors(void);
#ifdef PRINT
void printValues(void);
#endif
void kalman2d (void);
void kalman1d (float estado, float incertidumbre, float entrada, float medida);

//***********SETUP***********//
void setup() {
  //Iniciar COM Serial
  Serial.begin(115200);
  //RGB
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);
  //Iniciar COM WiFi
```

```
while (status != WL_CONNECTED) {
  Serial.print("Conectando a SSID: ");
  Serial.println(ssid);
  status = WiFi.begin(ssid, pass);
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, LOW);
  delay(5000);

}
Serial.print("Conectado. IP Arduino: ");
Serial.println(WiFi.localIP());
digitalWrite(LEDR, HIGH);
digitalWrite(LEDG, HIGH);
digitalWrite(LEDB, LOW);
// Iniciar puerto
Udp.begin(localUdpPort);
Serial.print("Escuchando puerto UDP: ");
Serial.println(localUdpPort);
while (power != 1) {
  APPgetData();
}
digitalWrite(LEDR, LOW);
digitalWrite(LEDG, LOW);
digitalWrite(LEDB, HIGH);
// Configurar Motores y ESC
motor1.attach(6);
motor2.attach(17);
motor3.attach(16);
motor4.attach(7);
motor1.writeMicroseconds(1000);
motor2.writeMicroseconds(1000);
motor3.writeMicroseconds(1000);
motor4.writeMicroseconds(1000);
delay(10000);
//Iniciar IMU
if (!IMU.begin()) {
  Serial.println("Error al inicializar IMU");
  while (1);
}
//Configurar LIDAR
sensor.setTimeout(500);
if (!sensor.init())
{
  Serial.println("Failed to detect and initialize sensor!");
  while (1) {}
}
```

```
  sensor.startContinuous();
  //Calibrar IMU
  calibrarIMU();
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);
}

//************LOOP***********//
void loop() {
  tiempoInicio = micros();
  if ((power == 1)) {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
    APPgetData();
    IMUgetData();
    Controller();
    MMA();
    setMotors();
    PCsendData();
    #ifdef PRINT
      printValues();
    #endif
  } else {
    APPgetData();
    motor1.writeMicroseconds(1000);
    motor2.writeMicroseconds(1000);
    motor3.writeMicroseconds(1000);
    motor4.writeMicroseconds(1000);
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, LOW);
  }
  tiempoFin = micros();
  //Serial.println(duracion);
  while ((tiempoFin - tiempoInicio) < tiempoDeseado) {
    tiempoFin = micros();
  }
}

//**********FUNCIONES**********//
void calibrarIMU(void) {
  for (int i = 0; i < 10000; i++) {
    if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
      IMU.readGyroscope(gX, gY, gZ);
      IMU.readAcceleration(aX, aY, aZ);
```

```
    }
    gXCal += gX;
    gYCal += gY;
    gZCal += gZ;
    aXOffset += aX;
    aYOffset += aY;
    aZOffset += aZ;
    //zIni += (float)sensor.readRangeContinuousMillimeters() / 1000;
  }
  gXCal /= 10000;
  gYCal /= 10000;
  gZCal /= 10000;
  aXOffset/= 10000;
  aYOffset/= 10000;
  aZOffset = (aZOffset / 10000) - 1;
  zIni = 0;
}

void getData(void) {
  if ((IMU.gyroscopeAvailable()) && (IMU.accelerationAvailable())) {
    IMU.readGyroscope(gX, gY, gZ);
    IMU.readAcceleration(aX, aY, aZ);
    gX -= gXCal;
    gY -= gYCal;
    gZ -= gZCal;
    aX -= aXOffset;
    aY -= aYOffset;
    aZ -= aZOffset;
    wRoll = gX * PI / 180;
    wPitch = gY * PI / 180;
    wYaw = -gZ * PI / 180;
  }
  roll = atan(aY / sqrt(aX * aX + aZ * aZ));
  pitch = -atan(aX / sqrt(aY * aY + aZ * aZ));
  kalman1D (estadoKalmanRoll, incertidumbreKalmanRoll, wRoll, roll);
  estadoKalmanRoll = estadoKalman;
  incertidumbreKalmanRoll = incertidumbreKalmanRoll;
  kalman1D (estadoKalmanPitch, incertidumbreKalmanPitch, wPitch, pitch);
  estadoKalmanPitch = estadoKalman;
  incertidumbreKalmanPitch = incertidumbreKalmanRoll;
  AZ = aX * (-cos(roll) * sin(pitch)) + aY * sin(roll) + aZ * (cos(roll) * cos(pitch));
  AZ = (AZ - 1) * 9.81;
  z = ((float)sensor.readRangeContinuousMillimeters() / 1000 ) * cos (roll) * cos(pitch);
  kalman2D();
}

// Acciones de Control
```

```
void Controller(void) {
 #ifdef PID
   vZError = vZRef - vZKalman;
   PD(vZError, KpVz, KdVz, vZErrorPrev, 9.817, -10.183);
   Thrust = outPD[0] + 10.183;
   vZErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   rollError = rollRef - estadoKalmanRoll;
   P(rollError, KpRoll);
   wRollRef = outP;
   outP = 0;
   wRollError = wRollRef - wRoll;
   PD(wRollError, KpWRoll, KdWRoll, wRollErrorPrev, 1, -1);
   T1x = outPD[0];
   wRollErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   pitchError = pitchRef - estadoKalmanPitch;
   P(pitchError, KpPitch);
   wPitchRef = outP;
   outP = 0;
   wPitchError = wPitchRef - wPitch;
   PD(wPitchError, KpWPitch, KdWPitch, wPitchErrorPrev, 1, -1);
   T2y = outPD[0];
   wPitchErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
   wYawError = wYawRef - wYaw;
   PD(wYawError, KpWYaw, KdWYaw, wYawErrorPrev, 0.3, -0.3);
   T3z = outPD[0];
   wYawErrorPrev = outPD[1];
   outPD[0] = 0;
   outPD[1] = 0;
 #endif

 #ifdef LQR
   rollError = - rollRef + estadoKalmanRoll;
   pitchError = - pitchRef + estadoKalmanPitch;
   LQRAngles(rollError, pitchError);
   vZError = - vZRef + vZ;
   wRollError = - wRollRef + wRoll;
   wPitchError = - wPitchRef + wPitch;
   wYawError = - wYawRef + wYaw;
   LQRVelocities(vZError, wRollError, wPitchError, wYawError);
 #endif
}
```

```
#ifdef PID
  //Controlador PD
  void PD(float error, float Kp, float Kd, float& errorPrev, float max, float min) {
    float P = Kp * error;
    float D = Kd * N * (error - errorPrev) / (N + 0.01);
    float output = P + D;
    if (output > max) {
      output = max;
    }
    if (output < min) {
      output = min;
    }
    outPD[0] = output;
    outPD[1] = error;
  }
  //Controlador P
  void P(float error, float Kp) {
    outP = Kp * error;
  }
#endif

//Controlador LQR
#ifdef LQR
  void LQRVelocities(float vZError, float wRollError, float wPitchError, float wYawError)
{
    error1 = {vZError, wRollError, wPitchError, wYawError};
    u1 = - (K2 * error1);
    Thrust = u1(0,0) + 9.81;
    T1x = u1(1,0);
    T2y = u1(2,0);
    T3z = u1(3,0);
  }
  void LQRAngles(float rollError, float pitchError) {
    error2 = {rollError, pitchError};
    u2 = - (K1 * error2);
    wRollRef = u2(0,0);
    wPitchRef = u2(1,0);
  }
#endif

// Función MMA (Motor Mixing Algorithm)
void MMA(void) {
  w1_2 = Thrust / (4 * kf) - T1x / (4 * kf * L) + T2y / (4 * kf * L) + T3z / (4 * km);
  w2_2 = Thrust / (4 * kf) + T1x / (4 * kf * L) + T2y / (4 * kf * L) - T3z / (4 * km);
  w3_2 = Thrust / (4 * kf) + T1x / (4 * kf * L) - T2y / (4 * kf * L) + T3z / (4 * km);
  w4_2 = Thrust / (4 * kf) - T1x / (4 * kf * L) - T2y / (4 * kf * L) - T3z / (4 * km);
```

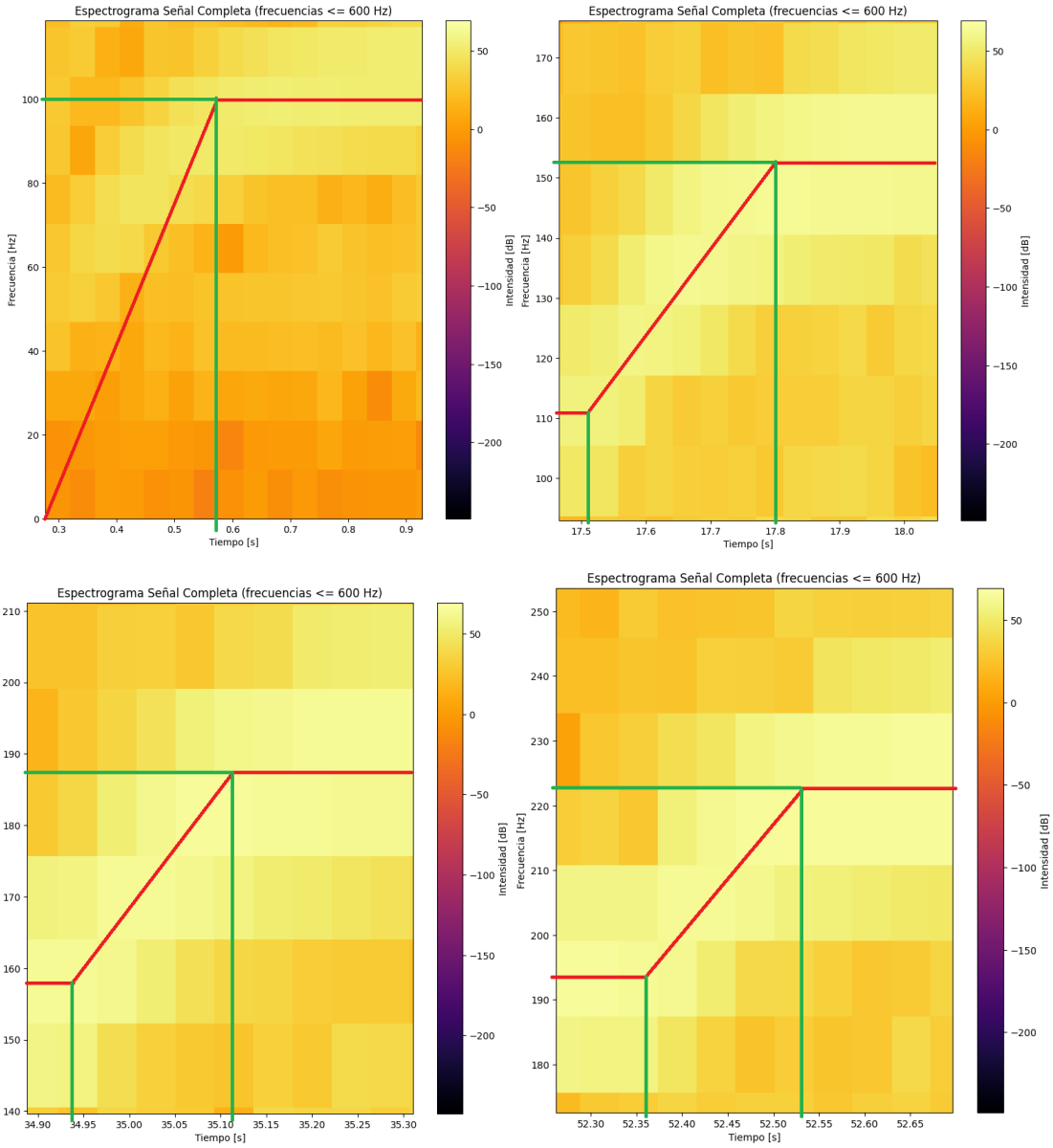
```
  PWM1 =  (7e-15) * w1_2 * w1_2 * w1_2- (7e-10) * w1_2 * w1_2 + (4e-5) * w1_2;
  PWM2 =  (7e-15) * w2_2 * w2_2 * w2_2- (7e-10) * w2_2 * w2_2 + (4e-5) * w2_2;
  PWM3 =  (7e-15) * w3_2 * w3_2 * w3_2- (7e-10) * w3_2 * w3_2 + (4e-5) * w3_2;
  PWM4 =  (7e-15) * w4_2 * w4_2 * w4_2 -(7e-10) * w4_2 * w4_2 + (4e-5) * w4_2;

  if (PWM1 > 1) {
    PWM1 = 1;
  }
  if (PWM2 > 1) {
    PWM2 = 1;
  }
  if (PWM3 > 1) {
    PWM3 = 1;
  }
  if (PWM4 > 1) {
    PWM4 = 1;
  }
  if (PWM1 < 0) {
    PWM1 = 0;
  }
  if (PWM2 < 0) {
    PWM2 = 0;
  }
  if (PWM3 < 0) {
    PWM3 = 0;
  }
  if (PWM4 < 0) {
    PWM4 = 0;
  }
  PWM1 = (1000 + (PWM1 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM2 = (1000 + (PWM2 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM3 = (1000 + (PWM3 - 0) * ((2000 - 1000)/(1 - 0)));
  PWM4 = (1000 + (PWM4 - 0) * ((2000 - 1000)/(1 - 0)));

}

// Función para establecer los valores de los motores
void setMotors(void) {
  motor1.writeMicroseconds(PWM1);
  motor2.writeMicroseconds(PWM2);
  motor3.writeMicroseconds(PWM3);
  motor4.writeMicroseconds(PWM4);
}

// Función para imprimir valores
#ifdef PRINT
```

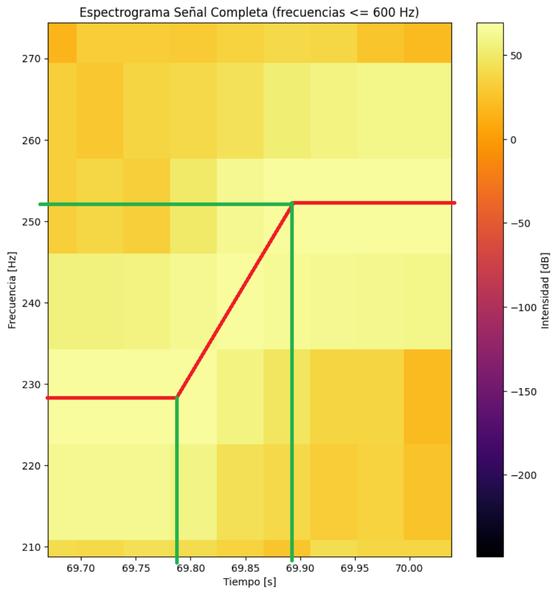
```
void printValues(void) {
  Serial.print("vZRef: "); Serial.print(vZRef); Serial.print("\t");
  Serial.print("wYawRef: "); Serial.print(wYawRef); Serial.print("\t");
  Serial.print("rollRef: "); Serial.print(rollRef); Serial.print("\t");
  Serial.print("pitchRef: "); Serial.print(pitchRef); Serial.print("\t");
  Serial.print("Z: "); Serial.print(zKalman); Serial.print("\t");
  Serial.print("vZ: "); Serial.print(vZKalman); Serial.print("\t");
  Serial.print("wYaw: "); Serial.print(wYaw); Serial.print("\t");
  Serial.print("Roll: "); Serial.print(estadoKalmanRoll*180/PI); Serial.print("\t");
  Serial.print("Pitch: "); Serial.print(estadoKalmanPitch*180/PI); Serial.print("\t");
  Serial.print("gX, gY, gZ: "); Serial.print(gX); Serial.print("\t");Serial.print(gY);
Serial.print("\t");Serial.print(gZ); Serial.print("\t");
  Serial.print("aX, aY, aZ: "); Serial.print(aX); Serial.print("\t");Serial.print(aY);
Serial.print("\t");Serial.print(aZ); Serial.print("\t");
  Serial.print("Thrust: "); Serial.print(Thrust); Serial.print("\t");
  Serial.print("Troll: "); Serial.print(T1x); Serial.print("\t");
  Serial.print("Tpitch: "); Serial.print(T2y); Serial.print("\t");
  Serial.print("Tyaw: "); Serial.print(T3z); Serial.print("\t");
  Serial.print("PWM1: "); Serial.print(PWM1); Serial.print("\t");
  Serial.print("PWM2: "); Serial.print(PWM2); Serial.print("\t");
  Serial.print("PWM3: "); Serial.print(PWM3); Serial.print("\t");
  Serial.print("PWM4: "); Serial.println(PWM4);
}
#endif

//Recibir paquetes desde la app
void APPgetData(void) {
  int packetSize = Udp.parsePacket();
  if (packetSize > 0) {
    char incomingPacket[255];
    int len = Udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = 0;
      sscanf(incomingPacket, "%f,%f,%f,%f,%d", &vZRef, &wYawRef, &rollRef,
&pitchRef, &power);
      rollRef *= PI / 180;
      pitchRef *= PI / 180;
      wYawRef *= PI / 180;
    }
  }
}

//Enviar paquetes al PC
void PCsendData(void) {
  uint8_t packetBuffer[16];
  memcpy(packetBuffer, &estadoKalmanRoll, sizeof(estadoKalmanRoll));
  memcpy(packetBuffer + 4, &estadoKalmanPitch, sizeof(estadoKalmanPitch));
```

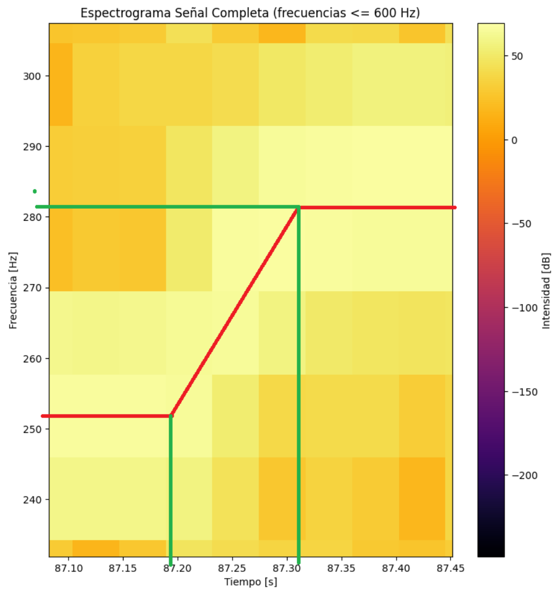
```
  memcpy(packetBuffer + 8, &wYaw, sizeof(wYaw));
  memcpy(packetBuffer + 8, &vZKalman, sizeof(vZKalman));
  Udp.beginPacket(receiverIp, clientUdpPort);
  Udp.write(packetBuffer, sizeof(packetBuffer));
  Udp.endPacket();
}

void kalman1D (float estado, float incertidumbre, float entrada, float medida) {
  estado = estado + 0.01 * entrada;
  incertidumbre = incertidumbre + (0.0116*0.0116);
  float gananciaKalman = incertidumbre * 1 / ( 1 * incertidumbre + 0.0175 * 0.0175);
  estado = estado + gananciaKalman * (medida - estado);
  incertidumbre = (1 - gananciaKalman) * incertidumbre;
  estadoKalman = estado;
  incertidumbreKalman = incertidumbre;
}

void kalman2D (void) {
  BLA::Matrix<2> Z_ = {z, AZ};
  s_ = A_ * s_ + B_ * AZ;
  P_ = A_ * P_ * (~A_) + Q_;
  BLA::Matrix<2, 2> S_ = H_ * P_ * (~H_) + R_;
  BLA::Matrix<2, 2> K_ = P_ * (~H_) * Inverse(S_);
  BLA::Matrix<2, 2> I_ = {1,0,0,1};
  s_ = s_ + K_ * (Z_ - H_ * s_);
  P_ = (I_ - K_ * H_) * P_;
  zKalman = s_(0);
  vZKalman = s_(1);
}
```

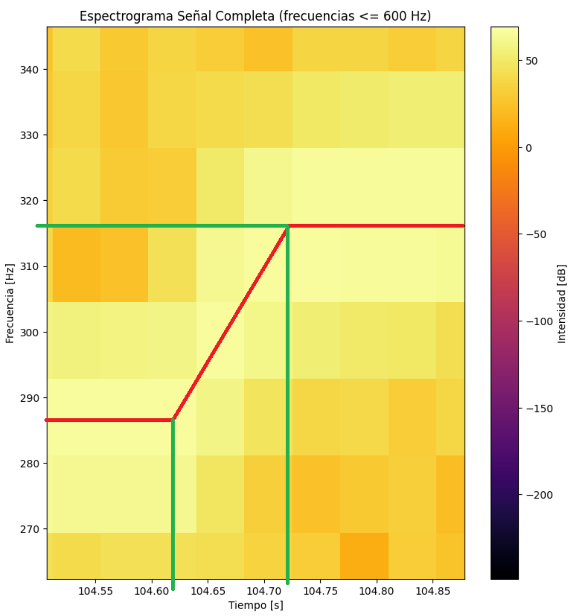
12.3. Análisis espectral para calcular la constante de tiempo

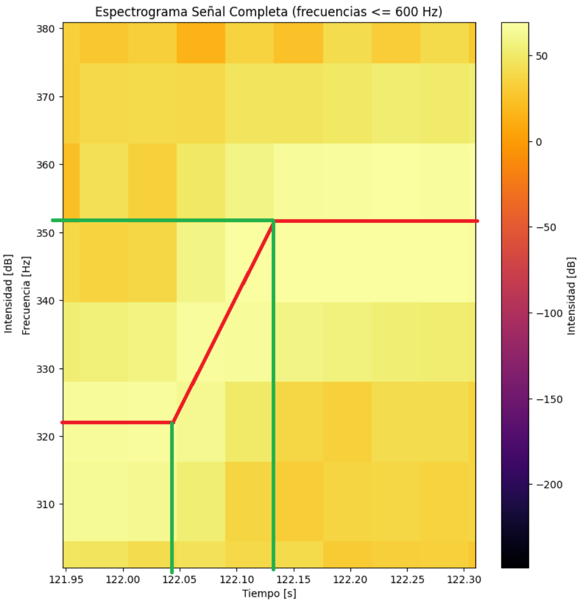
Se adjuntan el resto de las figuras las transiciones entre velocidades angulares, empleadas para medir el tiempo transitorio y conocer la constante de tiempo del sistema de propulsión y los datos calculados:

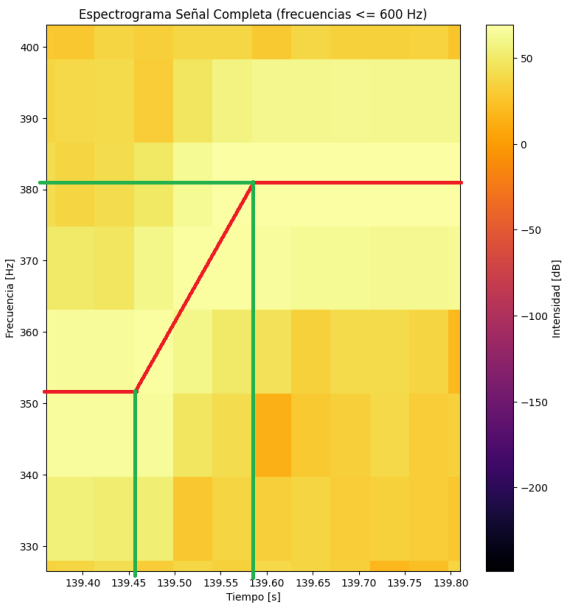


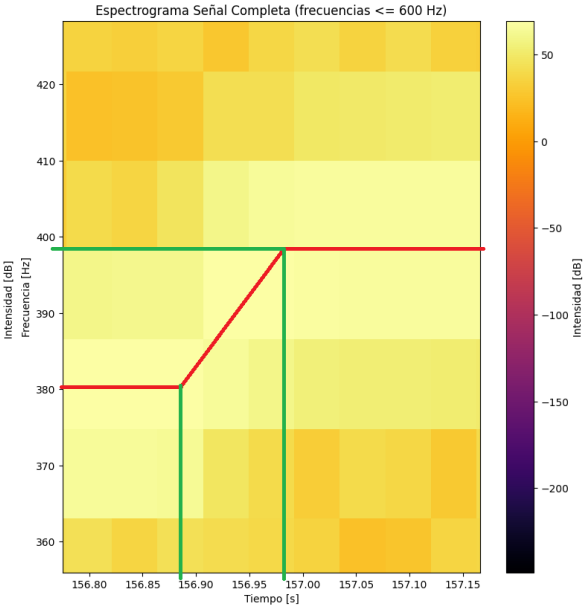

Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]
270
260
250
240
230
220
210
69.70
69.75
69.80
69.85
69.90
69.95
70.00
50
0
−50
−100
−150
−200


Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]
300
290
280
270
260
250
240
87.10
87.15
87.20
87.25
87.30
87.35
87.40
87.45
50
0
−50
−100
−150
−200

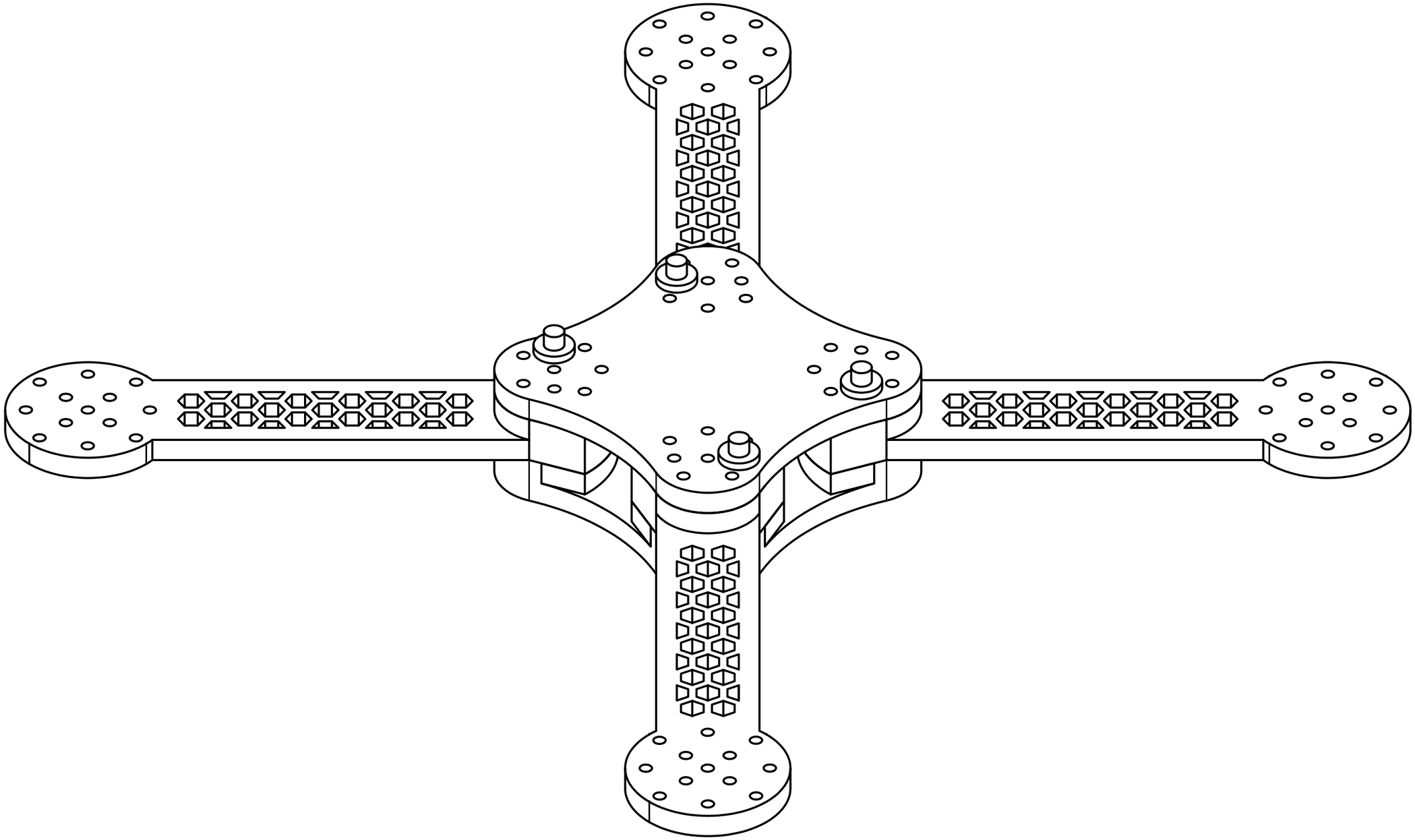

Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]
340
330
320
310
300
290
280
270
104.55
104.60
104.65
104.70
104.75
104.80
104.85
50
0
−50
−100
−150
−200


Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]
380
370
360
350
340
330
320
310
121.95
122.00
122.05
122.10
122.15
122.20
122.25
122.30
50
0
−50
−100
−150
−200


Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]


Espectrograma Señal Completa (frecuencias <= 600 Hz)
Frecuencia [Hz]
Tiempo [s]
Intensidad [dB]

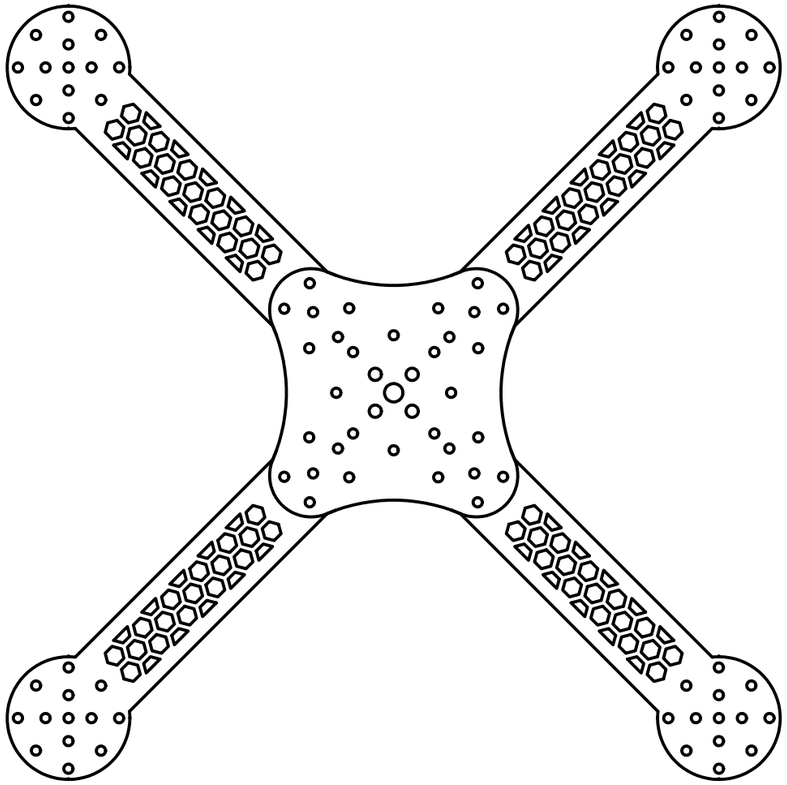
12.4. Planos

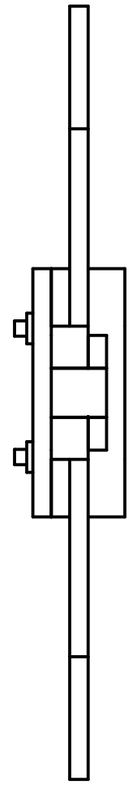


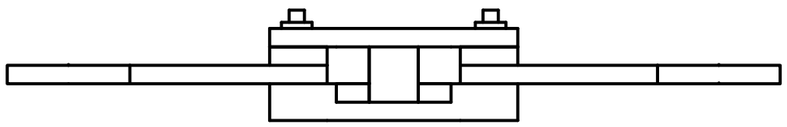
LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	BASE DOWN	
2	1	BRAZO 1	
3	1	BRAZO 2	
4	1	BRAZO 3	
5	1	BRAZO 4	
6	1	BASE UP	

Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Rubén Sahuquillo					25/06/2024	

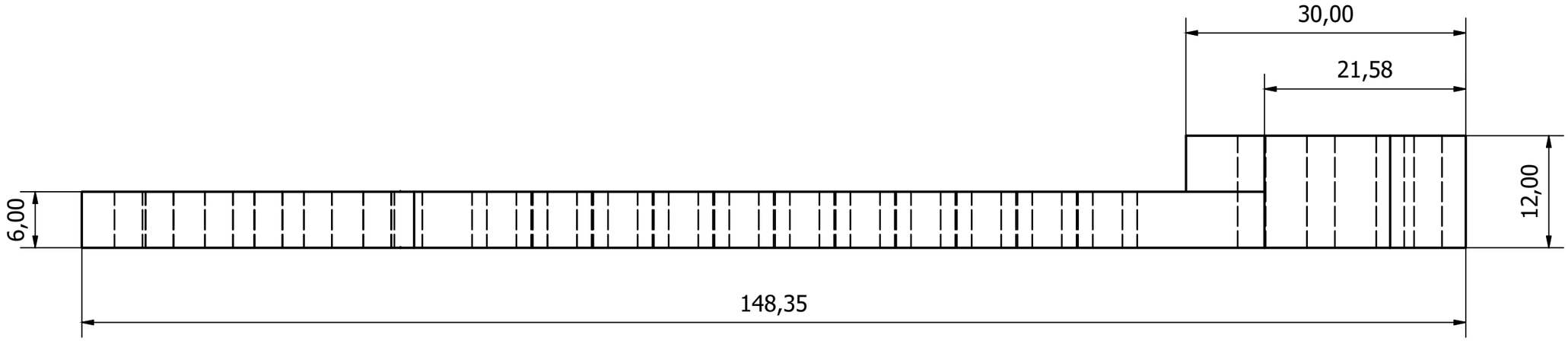
ULL	Prototipo de Dron. Vistas		
	General	Edición	Hoja 1 / 2

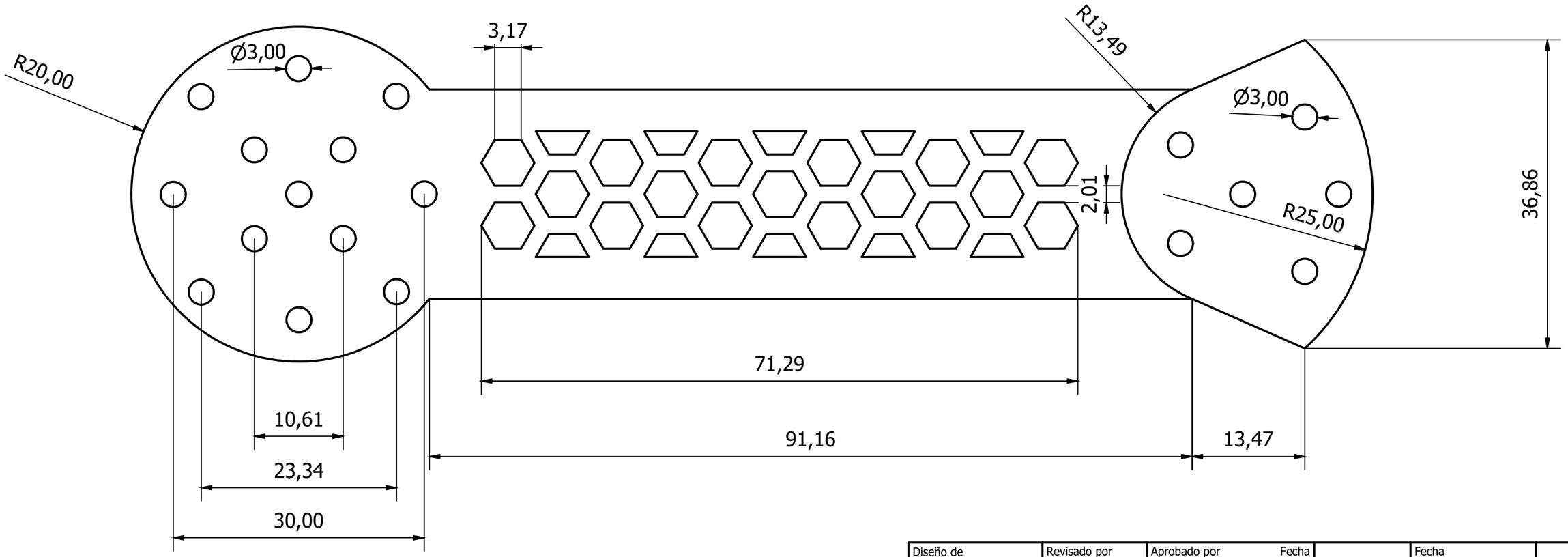






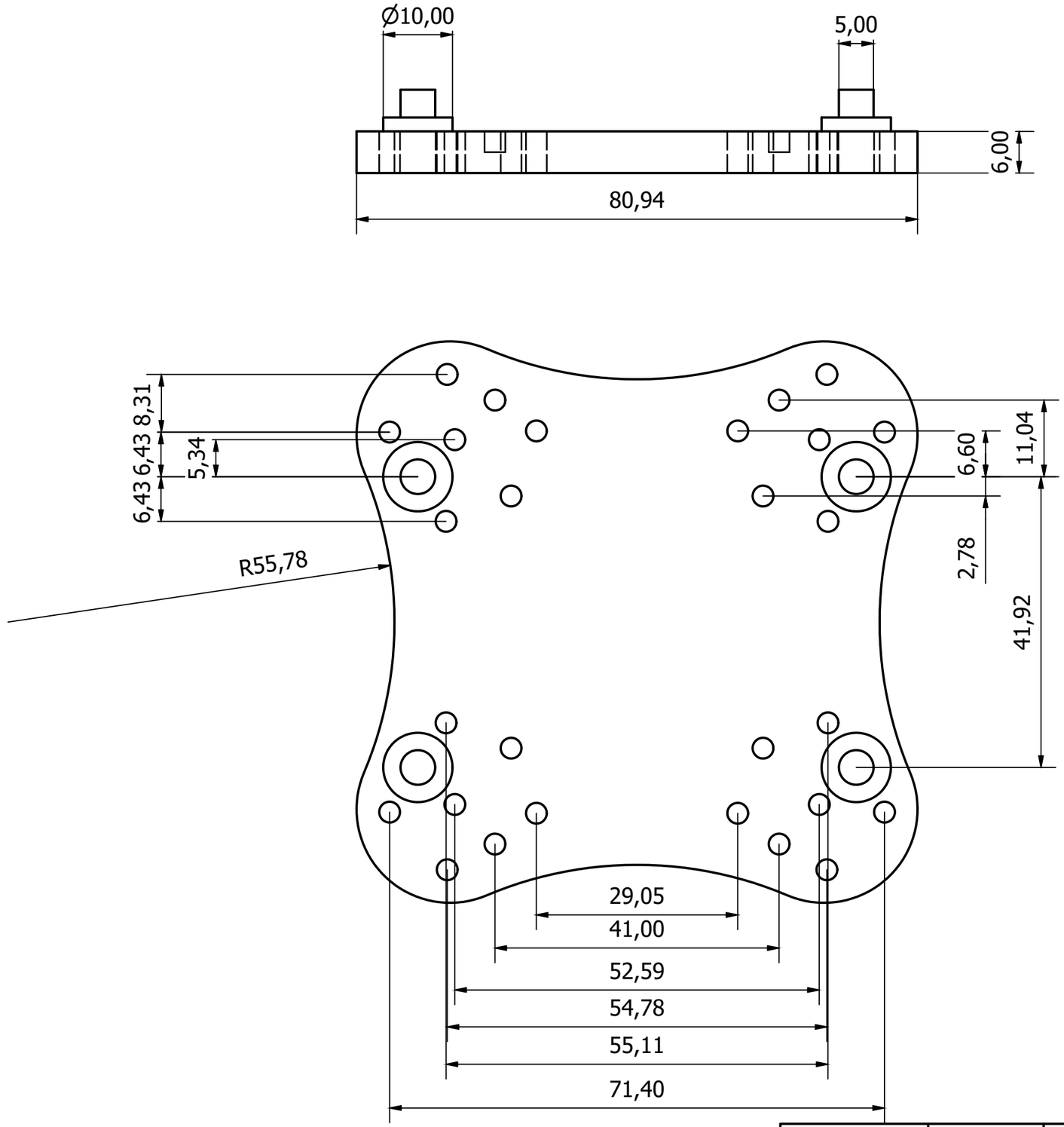
Diseño de Rubén Sahuquillo	Revisado por	Aprobado por	Fecha	Fecha 25/06/2024	
ULL		Prototipo de Dron. Vistas			
		General		Edición	Hoja 2 / 2





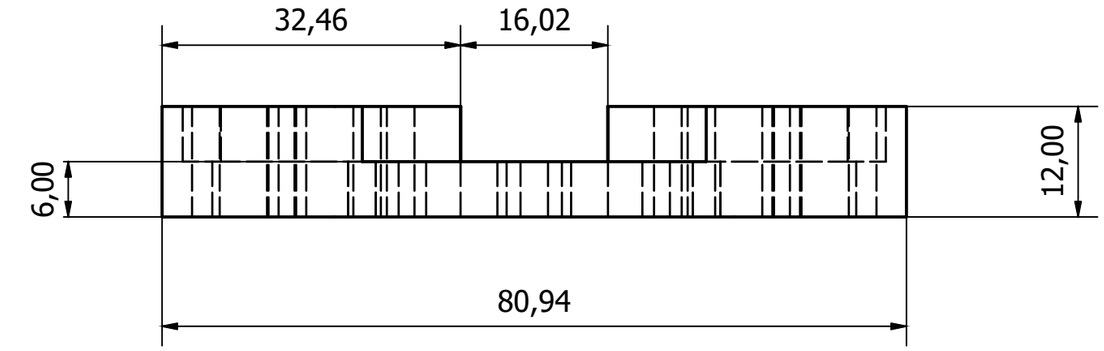
Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Ruben Sahuquillo					26/06/2024	

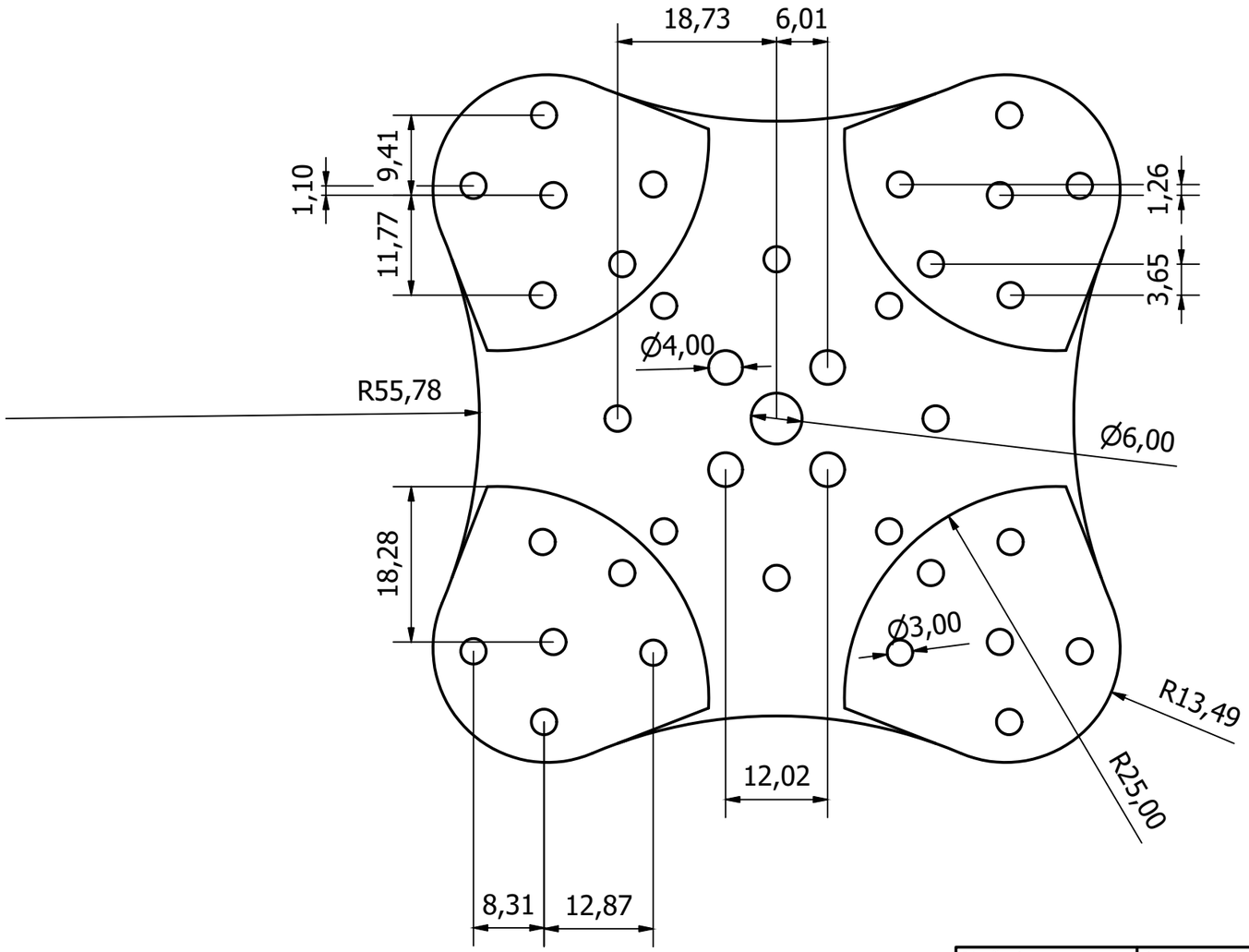
ULL	Brazo		
	BRAZO	Edición	Hoja 1 / 1



Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Ruben Sahuquillo					26/06/2024	

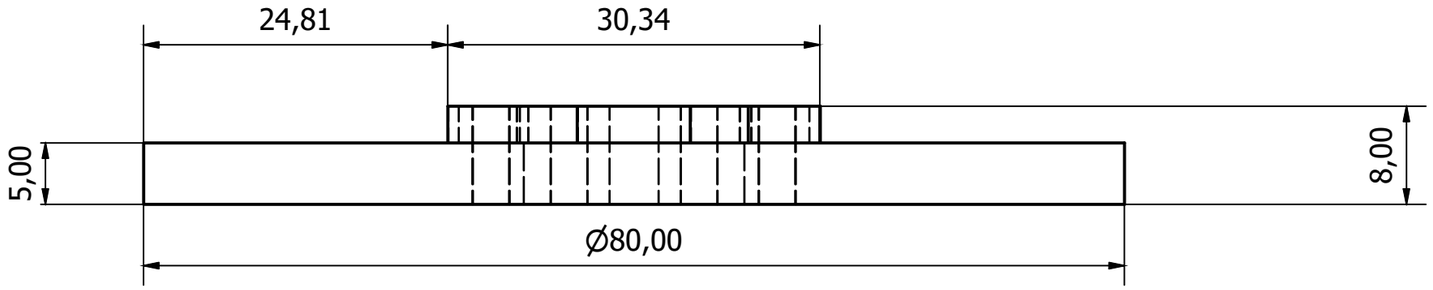
ULL	Base Superior		
	BASE_UP	Edición	Hoja 1 / 1

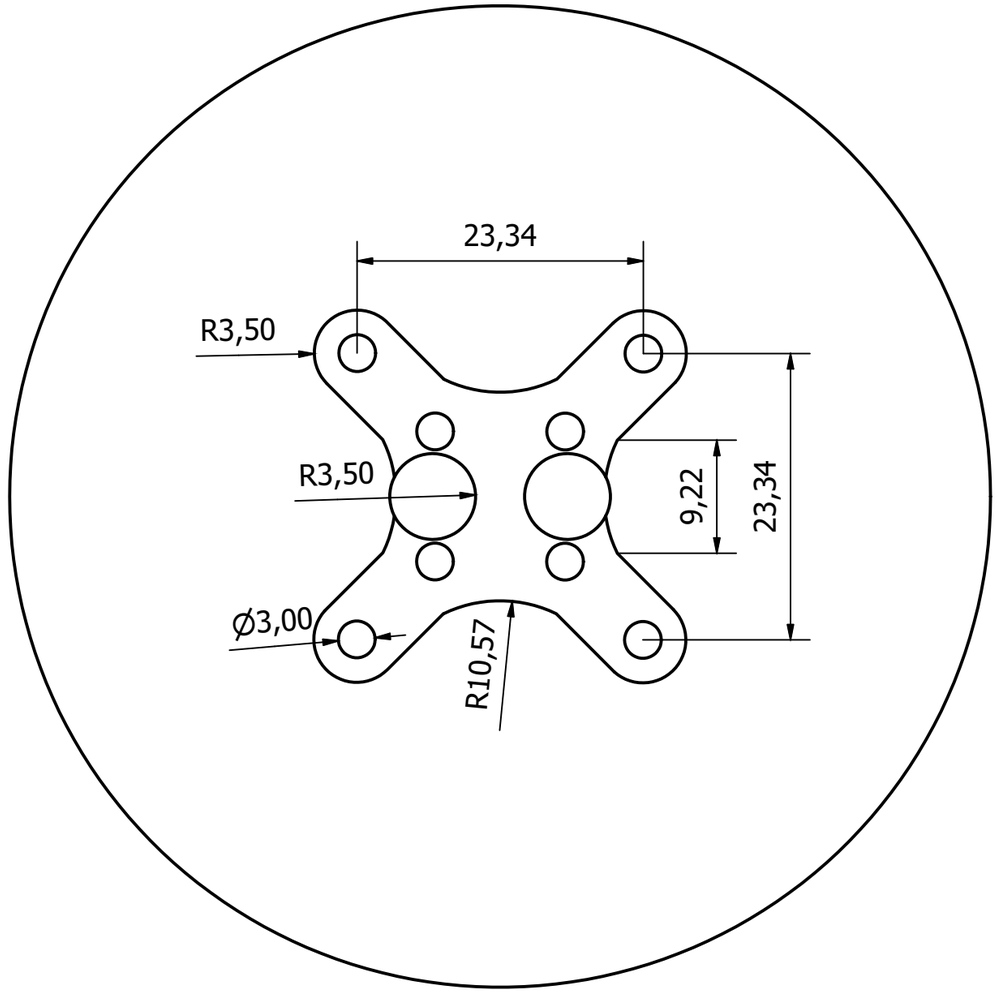




Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Ruben Sahuquillo					26/06/2024	

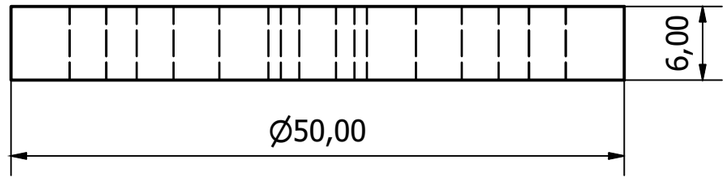
ULL	Base Inferior		
	BASE_DOWN	Edición	Hoja 1 / 1

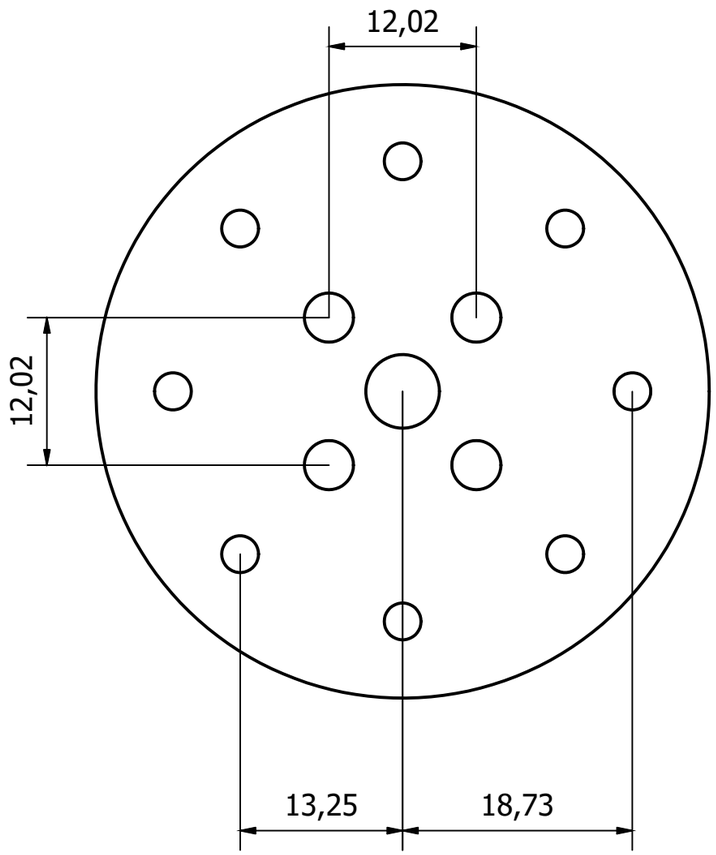




Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Ruben Sahuquillo					26/06/2024	

ULL	Báscula		
	Bascula	Edición	Hoja 1 / 1





Diseño de	Revisado por	Aprobado por	Fecha	Fecha	
Ruben Sahuquillo				26/06/2024	

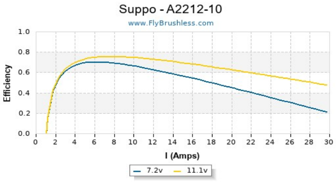
Base Rotula

Base Rotula	Edición	Hoja
		1 / 1

12.5. Datos del estudio de motores

	Datos Internet				interpolacion	
	VOLTS [V]	AMPS [A]	RPM	THRUST [g]	PWM (%)	
APC 7x5 E	0	0	0	0	0,00	0%
	6,9	7,1	8280	308	62,16	62,16%
	7,9	8,55	9210	390	71,17	71,17%
	8,8	10,15	10080	472	79,28	79,28%
	9,8	11,75	10920	561	88,29	88,29%
	10,8	13,35	11700	653	97,30	97,30%
	11,1				100,00	100%

https://www.flybrushless.com/motor/view/228



Velocidades analisis de frecuencias				
De	A	tinicio	tfinal	Transitorio
1000	1100	0,28	0,56	0,28
1100	1200	17,51	17,8	0,29
1200	1300	34,94	35,12	0,18
1300	1400	52,36	52,53	0,17
1400	1500	69,76	69,89	0,13
1500	1600	87,19	87,32	0,13
1600	1700	104,62	104,72	0,1
1700	1800	122,04	122,13	0,09
1800	1900	139,46	139,58	0,12
1900	2000	156,88	156,97	0,09

Promedio de todos los valores	0,158
Promedio de los valores no extremos	0,126
Mediana	0,130

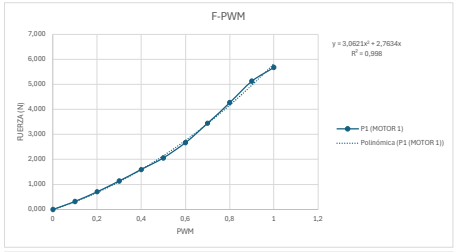
Tiempo Transitorio (4τ)	0,13
Constante de Tiempo (τ)	0,03

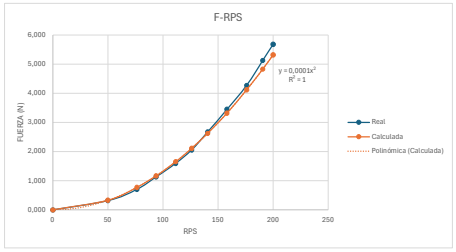
PWM	PWM	Frecuencia	RPM
1000	0%	0	0
1100	10%	100	3000
1200	20%	152,5	4575
1300	30%	187,5	5625
1400	40%	223	6690
1500	50%	252	7560
1600	60%	281	8430
1700	70%	316	9480
1800	80%	352	10560
1900	90%	381	11430
2000	100%	400	12000

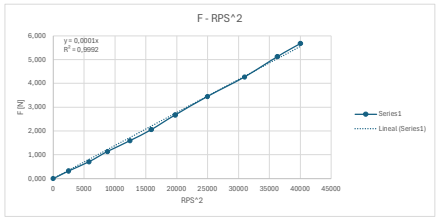
PWM	PRUEBA 1 (12,5 V)			PRUEBA 2 (12,31 V)			PRUEBA 3 (12,17 V)			PRUEBA 4 (11,96 V)			PRUEBA 5 (12,47 V)		
	V (V)	I (A)	F (g)	V (V)	I (A)	F (g)	V (V)	I (A)	F (g)	V (V)	I (A)	F (g)	V (V)	I (A)	F (g)
1000	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
0%	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
	12,50	0,05	0,00	12,31	0,05	0,00	12,17	0,05	0,00	11,96	0,05	0,00	12,47	0,05	0,00
		0,00	0,00		0,00	0,00		0,00	0,00		0,00	0,00		0,00	0,00
1100	12,47	0,49	16,40	12,27	0,54	16,70	12,12	0,47	15,00	11,93	0,47	14,80	12,43	0,50	18,80
10%	12,47	0,48	32,60	12,28	0,52	34,40	12,12	0,47	32,30	11,93	0,46	31,30	12,43	0,50	35,70
	12,47	0,48	32,90	12,28	0,52	34,20	12,12	0,47	32,10	11,93	0,46	31,20	12,43	0,50	35,70
	12,47	0,48	32,40	12,28	0,52	33,50	12,12	0,47	32,00	11,93	0,47	31,00	12,43	0,50	35,70
	12,47	0,49	32,20	12,28	0,52	33,00	12,12	0,47	32,40	11,93	0,47	30,70	12,43	0,50	35,80
	12,47	0,49	32,10	12,28	0,52	32,60	12,12	0,47	32,40	11,93	0,47	30,20	12,43	0,50	35,80
	12,46	0,49	32,40	12,28	0,52	32,20	12,12	0,48	32,90	11,93	0,47	30,50	12,43	0,50	36,00
	12,46	0,49	32,00	12,28	0,53	31,70	12,12	0,47	32,70	11,93	0,48	30,20	12,43	0,50	36,10
	12,46	0,49	32,40	12,27	0,53	31,40	12,12	0,48	32,80	11,93	0,48	29,90	12,43	0,50	36,10
	12,46	0,49	32,50	12,27	0,53	30,60	12,12	0,48	33,10	11,93	0,48	29,90	12,43	0,50	36,20
	12,47	0,49	32,39	12,28	0,53	32,62	12,12	0,47	32,52	11,93	0,47	30,54	12,43	0,50	35,90
		66,51	0,32		62,14	0,32		68,76	0,32		64,85	0,30		71,80	0,35
1200	12,42	1,00	64,80	12,24	1,03	61,50	12,06	0,96	63,60	11,89	0,96	59,50	12,38	1,02	67,90
20%	12,42	1,00	70,60	12,24	1,02	66,70	12,06	0,96	69,10	11,89	0,95	65,40	12,38	1,02	74,10
	12,42	1,00	71,30	12,24	1,03	67,10	12,06	0,97	69,50	11,89	0,95	65,60	12,38	1,02	74,60
	12,42	1,00	71,50	12,24	1,03	67,30	12,06	0,97	70,20	11,89	0,96	65,80	12,38	1,02	74,90
	12,42	1,00	71,50	12,23	1,03	66,60	12,06	0,97	70,50	11,89	0,96	65,60	12,37	1,02	74,10
	12,42	1,01	71,70	12,23	1,03	66,30	12,06	0,97	70,70	11,89	0,96	66,10	12,37	1,03	75,30
	12,42	1,01	72,20	12,23	1,03	66,30	12,06	0,97	71,20	11,89	0,96	65,90	12,37	1,03	75,80
	12,42	1,01	72,60	12,23	1,03	66,20	12,05	0,98	71,10	11,88	0,97	65,80	12,37	1,03	76,20
	12,42	1,01	72,40	12,23	1,03	66,20	12,05	0,98	71,50	11,88	0,97	66,10	12,37	1,03	75,50
	12,41	1,01	72,80	12,23	1,03	66,30	12,05	0,98	71,90	11,88	0,97	65,50	12,37	1,03	75,50
	12,42	1,01	71,84	12,23	1,03	66,56	12,06	0,97	70,63	11,89	0,96	65,76	12,37	1,03	75,11
		71,49	0,70		64,68	0,65		72,74	0,69		68,42	0,65		73,28	0,74
1300	12,37	1,62	109,20	12,19	1,63	101,90	11,99	1,56	105,90	11,84	1,54	100,40	12,31	1,65	112,20
30%	12,36	1,62	114,20	12,18	1.64	106,20	11,99	1,56	110,30	11,84	1,54	104,20	12,31	1,65	116,80
	12,36	1,62	114,90	12,13	1,63	104,90	11,99	1,57	111,40	11,84	1,55	104,30	12,31	1,65	117,60
	12,36	1,62	114,40	12,12	1,63	104,80	11,99	1,57	111,90	11,84	1,55	104,80	12,31	1,67	117,70
	12,36	1,63	115,20	12,12	1,63	105,30	11,98	1,57	112,10	11,83	1,55	104,00	12,31	1,67	117,40
	12,36	1,63	115,40	12,12	1,64	104,50	11,98	1,57	111,80	11,83	1,56	105,10	12,30	1,68	118,10
	12,36	1,64	116,40	12,11	1,63	104,50	11,98	1,58	111,80	11,83	1,56	105,20	12,31	1,68	119,20
	12,36	1,64	115,40	12,11	1,63	106,10	11,98	1,58	113,00	11,83	1,56	105,80	12,30	1,68	119,10
	12,35	1,64	116,50	12,11	1,64	105,20	11,98	1,58	111,90	11,83	1,56	105,70	12,30	1,68	119,50
	12,35	1,65	117,40	12,11	1,64	104,90	11,98	1,58	112,60	11,83	1,56	105,40	12,30	1,68	119,80
	12,36	1,63	115,53	12,13	1,63	105,16	11,98	1,57	111,87	11,83	1,55	104,94	12,31	1,67	118,36
		70,84	1,13		64,38	1,03		71,16	1,10		67,58	1,03		70,91	1,16
1400	12,29	2,41	159,10	12,03	2,39	146,70	11,90	2,32	152,70	11,78	2,29	146,00	12,24	2,45	161,20
40%	12,29	2,41	160,70	12,02	2,39	150,20	11,90	2,32	156,30	11,77	2,29	148,50	12,24	2,45	162,80
	12,29	2,41	161,90	12,02	2,39	148,90	11,90	2,32	155,60	11,77	2,29	149,10	12,23	2,46	162,50
	12,29	2,42	162,30	12,03	2,39	149,80	11,90	2,32	157,30	11,77	2,29	149,30	12,22	2,46	163,20
	12,29	2,42	161,40	12,03	2,39	148,80	11,90	2,32	156,60	11,77	2,30	148,10	12,24	2,47	164,70
	12,29	2,42	162,00	12,01	2,39	148,60	11,89	2,32	157,90	11,77	2,29	148,70	12,24	2,46	164,40
	12,29	2,42	164,40	12,02	2,39	149,70	11,89	2,32	157,30	11,77	2,30	149,70	12,24	2,46	164,60
	12,28	2,43	163,70	12,02	2,39	150,30	11,89	2,32	158,20	11,77	2,30	150,40	12,23	2,47	165,70
	12,28	2,42	163,40	12,04	2,40	151,10	11,89	2,33	159,10	11,76	2,30	149,70	12,23	2,47	167,10
	12,28	2,43	164,60	12,05	2,40	151,40	11,89	2,33	158,70	11,76	2,31	150,90	12,23	2,48	166,40
	12,29	2,42	162,35	12,03	2,39	149,87	11,90	2,32	157,44	11,77	2,30	149,38	12,23	2,46	164,26
		67,11	1,59		62,65	1,47		67,81	1,54		65,06	1,47		66,69	1,61
1500	12,21	3,28	207,20	11,97	3,22	192,80	11,80	3,11	199,30	11,71	3,09	190,40	12,16	3,37	211,30
50%	12,21	3,29	208,80	11,96	3,22	193,30	11,80	3,12	198,70	11,70	3,10	190,40	12,16	3,36	209,90
	12,21	3,27	209,80	11,97	3,21	191,50	11,80	3,12	199,70	11,70	3,10	191,70	12,16	3,37	211,30
	12,21	3,27	209,50	11,98	3,22	193,20	11,79	3,12	200,20	11,70	3,09	191,60	12,15	3,35	212,10
	12,20	3,27	209,70	11,99	3,22	192,90	11,79	3,12	199,60	11,69	3,10	191,60	12,15	3,34	215,40
	12,20	3,28	209,30	11,99	3,21	193,30	11,79	3,13	200,50	11,69	3,10	191,90	12,15	3,35	214,00
	12,20	3,28	211,00	11,99	3,22	195,20	11,79	3,11	201,60	11,69	3,10	192,30	12,15	3,36	215,40
	12,19	3,28	211,40	11,98	3,22	193,90	11,78	3,12	200,90	11,68	3,11	191,80	12,15	3,38	214,40
	12,19	3,29	210,50	11,98	3,22	194,60	11,78	3,12	200,60	11,69	3,10	192,60	12,14	3,39	214,80
	12,19	3,29	210,60	11,98	3,23	194,30	11,77	3,14	203,20	11,68	3,10	192,10	12,15	3,36	214,70
	12,20	3,28	209,78	11,98	3,22	193,50	11,79	3,12	200,43	11,69	3,10	191,64	12,15	3,36	213,33
		63,96	2,06		60,11	1,90		64,22	1,97		61,84	1,88		63,43	2,09
1600	12,09	4,51	270,30	11,87	4,45	251,70	11,65	4,29	256,90	11,60	4,27	246,80	12,05	4,62	273,70
60%	12,08	4,51	270,80	11,87	4,45	253,60	11,65	4,29	255,80	11,59	4,27	248,70	12,05	4,64	273,30
	12,08	4,52	270,60	11,87	4,46	254,20	11,64	4,30	258,00	11,59	4,27	248,50	12,04	4,65	276,80
	12,08	4,51	270,90	11,87	4,47	253,40	11,64	4,28	258,00	11,58	4,28	248,40	12,04	4,65	276,10
	12,08	4,50	272,60	11,86	4,47	250,40	11,64	4,28	256,30	11,58	4,28	249,40	12,04	4,65	276,20
	12,07	4,52	272,80	11,86	4,47	253,00	11,63	4,27	256,00	11,57	4,28	248,40	12,03	4,66	276,70
	12,07	4,51	272,50	11,86	4,47	252,40	11,63	4,28	257,10	11,56	4,28	249,70	12,03	4,64	277,70
	12,07	4,51	271,70	11,86	4,48	253,20	11,62	4,29	258,40	11,56	4,28	248,90	12,03	4,64	277,10
	12,07	4,52	275,40	11,86	4,48	253,60	11,62	4,29	259,30	11,55	4,28	249,60	12,03	4,63	275,60
	12,06	4,51	273,80	11,86	4,46	253,10	11,61	4,29	258,70	11,55	4,28	251,70	12,03	4,63	277,80
	12,08	4,51	272,14	11,86	4,47	252,86	11,63	4,29	257,45	11,57	4,28	249,01	12,04	4,64	276,10
		60,31	2,67		56,62	2,48		60,07	2,53		58,22	2,44		59,49	2,71
1700	11,93	6,21	349,30	11,72	6,12	325,70	11,46	5,83	329,00	11,43	5,82	320,00	11,91	6,36	351,70
70%	11,92	6,22	350,70	11,72	6,07	327,10	11,46	5,82	328,10	11,43	5,82	319,10	11,90	6,34	354,40
	11,91	6,22	352,50	11,72	6,10	325,10	11,45	5,83	326,30	11,42	5,81	320,60	11,89	6,34	349,90
	11,91	6,23	349,30	11,73	6,10	325,40	11,46	5,83	329,10	11,42	5,83	321,20	11,88	6,35	350,10
	11,90	6,21	351,30	11,72	6,13	326,30	11,45	5,86	328,30	11,42	5,83	318,80	11,88	6,35	351,20
	11,90	6,23	352.6	11,72	6,12	331,90	11,45	5,86	329,30	11,42	5,84	321,40	11,87	6,36	351,10
	11,90	6,23	351,90	11,72	6,12	328,70	11,44	5,83	328,90	11,42	5,84	319,20	11,87	6,34	351,10
	11,90	6,20	351,60	11,72	6,10	326,70	11,44	5,87	331,10	11,41	5,83	320,80	11,87	6,34	348,90
	11,90	6,22	353,20	11,71	6,10	325,80	11,44	5,86	327,90	11,41	5,82	319,90	11,86	6,35	354,50
	11,90	6,22	354,10	11,71	6,10	326,80	11,43	5,86	330,70	11,51	5,82	319,50	11,85	6,53	354,70
	11,91	6,22	351,54	11,72	6,11	326,95	11,45	5,85	328,87	11,43	5,83	320,05	11,88	6,37	351,76
		56,53	3,45		53,55	3,21		56,27	3,23		54,93	3,14		55,26	3,45
1800	11,74	8,28	435,90	11,53	8,14	410,30	11,29	7,84	412,60	11,27	7,80	402,20	11,71	8,49	435,00
80%	11,74	8,30	436,30	11,52	8,14	411,20	11,29	7,83	414,60	11,27	7,81	404,60	11,71	8,46	434,60
	11,73	8,30	435,10	11,53	8,16	409,90	11,29	7,83	411,30	11,26	7,81	405,00	11,71	8,38	427,70
	11,73	8,32	435,10	11,54	8,18	412,10	11,29	7,81	412,00	11,26	7,82	404,80	11,69	8,45	433,10
	11,72	8,34	435,90	11,54	8,16	417,00	11,28	7,82	410,50	11,25	7,81	405,00	11,68	8,47	331,50
	11,72	8,33	434,40	11,54	8,16	412,60	11,27	7,83	411,10	11,24	7,79	402,00	11,68	8,45	428,10
	11,71	8,32	436,70	11,54	8,13	411,40	11,27	7,84	413,30	11,23	7,79	402,00	11,68	8,47	435,70
	11,71	8,30	434,60	11,53	8,16	411,90	11,26	7,85	412,80	11,23	7,79	401,70	11,67	8,45	432,00
	11,70	8,35	434,90	11,52	8,16	413,90	11,26	7,83	414,30	11,22	7,81	404,70	11,67	8,47	432,80
	11,70	8,35	433,30	11,52	8,18	414,20	11,25	7,85	413,50	11,22	7,80	407,70	11,66	8,48	433,00
	11,72	8,32	435,22	11,53	8,16	412,45	11,28	7,83	412,60	11,25	7,80	403,97	11,69	8,46	422,35
		52,32	4,27		50,56	4,05		52,67	4,05		51,77	3,96		49,94	4,14
1900	11,51	10,85	520,50	11,33	10,54	491,40	11,05	10,28	495,50	11,05	10,26	489,40	11,48	11,02	519,60
90%	11,51	10,84	522,60	11,32	10,60	494,20	11,06	10,24	495,60	11,05	10,23	485,80	11,48	11,05	524,30
	11,50	10,86	520,70	11,32	10,57	493,90	11,05	10,18	495,50	11,04	10,18	484,40	11,47	10,95	518,70
	11,50	10,83	521,10	11,31	10,59	495,00	11,04	10,22	495,80	11,03	10,16	485,20	11,47	10,94	520,10
	11,49	10,83	521,80	11,32	10,57	498,50	11,03	10,23	494,70	11,03	10,19	486,00	11,46	11,00	521,00
	11,49	10,81	526,80	11,30	10,61	499,20	11,03	10,22	495,20	11,02	10,17	486,10	11,45	10,99	519,80
	11,49	10,82	523,00	11,30	10,64	495,90	11,03	10,19	492,30	11,01	10,13	483,60	11,45	10,98	519,90
	11,49	10,80	520,50	11,30	10,62	497,80	11,02	10,19	492,80	11,01	10,12	484,40	11,45	10,95	522,20
	11,48	10,89	525,20	11,29	10,62	495,10	11,02	10,19	493,50	11,00	10,14	483,20	11,45	10,92	517,20
	11,48	10,88	522,50	11,28	10,63	498,00	11,01	10,18	492,10	11,00	10,15	484,50	11,43	11,00	522,00
	11,49	10,84	522,47	11,31	10,60	495,90	11,03	10,21	494,30	11,02	10,17	485,26	11,46	10,98	520,48
		48,19	5,13		46,79	4,86		48,40	4,85		47,70	4,76		47,40	5,11
2000	11,32	12,66	581,80	11,13	12,39	554,20	10,88	11,94	548,70	10,87	11,93	540,30	11,30	12,76	573,70
100%	11,31	12,63	577,90	11,13	12,37	550,60	10,88	11,93	545,80	10,87	11,89	539,90	11,29	12,75	576,50
	11,31	12,59	580,30	11,14	12,33	551,10	10,87	11,92	549,60	10,86	11,89	541,30	11,29	12,71	573,30
	11,31	12,58	579,70	11,13	12,35	550,10	10,87	11,92	548,30	10,86	11,86	541,00	11,28	12,76	573,70
	11,31	12,57	573,40	11,13	12,37	551,10	10,86	11,92	547,80	10,85	11,88	539,30	11,27	12,72	573,10
	11,31	12,58	579,70	11,13	12,34	549,60	10,85	11,93	548,40	10,85	11,88	537,90	11,26	12,76	576,00
	11,31	12,60	579,30	11,13	12,36	551,80	10,86	11,89	548,80	10,85	11,88	541,10	11,26	12,73	573,80
	11,31	12,59	578,70	11,12	12,37	551,50	10,86	11,85	548,30	10,85	11,85	540,00	11,26	12,70	570,00
	11,31	12,59	577,70	11,12	12,27	546,60	10,85	11,86	543,80	10,84	11,86	536,40	11,25	12,70	572,30
	11,31	12,55	580,30	11,12	12,28	551,50	10,84	11,84	548,20	10,84	11,85	538,50	11,25	12,65	570,90
	11,31	12,59	578,88	11,13	12,34	550,81	10,86	11,90	547,77	10,85	11,88	539,57	11,27	12,72	573,33
		45,96	5,68		44,63	5,40		46,03	5,37		45,43	5,29		45,06	5,62

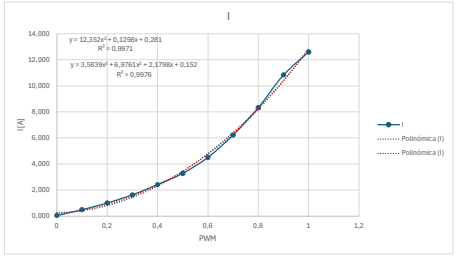
P1 (MOTOR 1)															
PWM	**V**	**I**	**P**	**RPM**	**RPS**	**rad/s**	**F (g)**	**F (N)**	**M (Nm)**	**Kt**	**Km**	**Fexp**	**RPS^2**	**Mexp**
0	12,500	0,050	0,625	0,000	0	0,000	0,000	0,000	0,000	0	0	0,000	0	0
0,1	12,470	0,490	6,110	3000,000	50	314,159	32,390	0,318	0,024	0,0001271	9,72484E-06	0,33235438	2500	0,02431211
0,2	12,420	1,010	12,544	4575,000	76,25	479,093	71,840	0,705	0,033	0,00012121	5,62929E-06	0,77293165	5814,0625	0,03272904
0,3	12,360	1,630	20,147	5625,000	93,75	589,049	115,530	1,133	0,043	0,00012895	4,86432E-06	1,16843336	8789,0625	0,04275284
0,4	12,290	2,420	29,742	6690,000	111,5	700,575	162,350	1,593	0,053	0,00012811	4,26848E-06	1,65276509	12432,25	0,05306675
0,5	12,200	3,280	40,016	7560,000	126	791,681	209,780	2,058	0,063	0,00012963	3,97972E-06	2,11058324	15876	0,06318199
0,6	12,080	4,510	54,481	8430,000	140,5	882,788	272,140	2,670	0,077	0,00013524	3,90791E-06	2,6243034	19740,25	0,07714314
0,7	11,910	6,220	74,080	9480,000	158	992,743	351,540	3,449	0,093	0,00013814	3,73647E-06	3,31875788	24964	0,09327714
0,8	11,720	8,320	97,510	10560,000	176	1105,841	435,220	4,270	0,110	0,00013783	3,5583E-06	4,11800368	30976	0,11022203
0,9	11,490	10,840	124,552	11430,000	190,5	1196,947	522,470	5,125	0,130	0,00014123	3,58422E-06	4,82448939	36290,25	0,1300722
1	11,310	12,590	142,393	12000,000	200	1256,637	578,880	5,679	0,142	0,00014197	3,54102E-06	5,31767005	40000	0,14164084
										0,00013294	**4,67946E-06**			

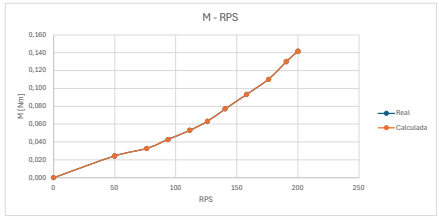
P2 (MOTOR 3)											
PWM	**V**	**I**	**P**	**RPM**	**RPS**	**rad/s**	**F (g)**	**F (N)**	**M (Nm)**	**Kt**	**Km**
0	12,470	0,050	0,624	0,000	0	0,000	0,000	0,000	0,000	0	0
0,1	12,430	0,500	6,215	3000,000	50	314,159	35,900	0,352	0,025	0,00014087	9,89148E-06
0,2	12,370	1,030	12,741	4575,000	76,25	479,093	75,110	0,737	0,033	0,00012673	5,71765E-06
0,3	12,310	1,670	20,558	5625,000	93,75	589,049	118,360	1,161	0,044	0,00013211	4,96353E-06
0,4	12,230	2,460	30,086	6690,000	111,5	700,575	164,260	1,611	0,054	0,00012961	4,31785E-06
0,5	12,150	3,360	40,824	7560,000	126	791,681	213,330	2,093	0,064	0,00013182	4,06008E-06
0,6	12,040	4,640	55,866	8430,000	140,5	882,788	276,100	2,709	0,079	0,00013721	4,00724E-06
0,7	11,880	6,370	75,676	9480,000	158	992,743	351,760	3,451	0,095	0,00013823	3,81693E-06
0,8	11,690	8,460	98,897	10560,000	176	1105,841	422,350	4,143	0,112	0,00013376	3,60892E-06
0,9	11,460	10,980	125,831	11430,000	190,5	1196,947	520,480	5,106	0,131	0,0001407	3,62103E-06
1	11,270	12,720	143,354	12000,000	200	1256,637	573,330	5,624	0,143	0,00014061	3,56493E-06

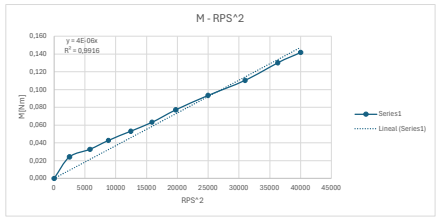


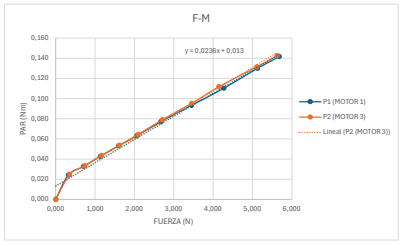


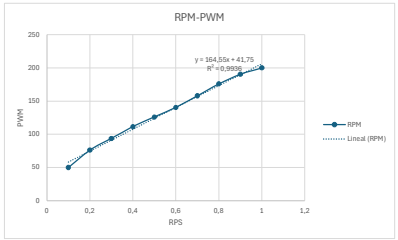


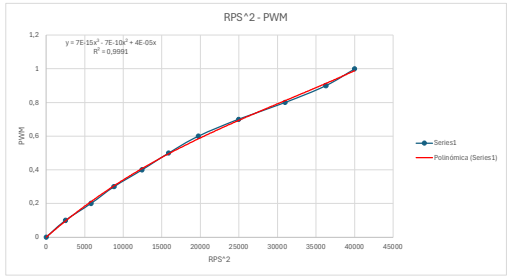












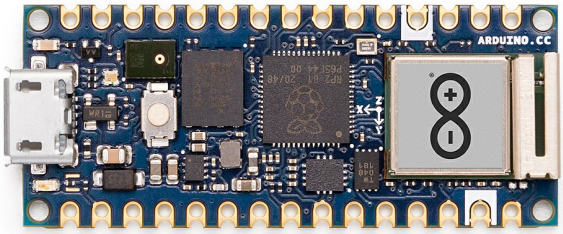
12.5. Portadas datasheets

Arduino® Nano RP2040 Connect

Product Reference Manual
SKU: ABX00053

Description

The feature packed **Arduino® Nano RP2040 Connect** brings the new **Raspberry Pi RP2040** microcontroller to the Nano form factor. Make the most of the dual core **32-bit Arm® Cortex®-M0+** to make Internet of Things projects with Bluetooth and WiFi connectivity thanks to the **U-blox® Nina W102** module. Dive into real-world projects with the onboard accelerometer, gyroscope, RGB LED and microphone. Develop robust embedded AI solutions with minimal effort using the **Arduino® Nano RP2040 Connect**!

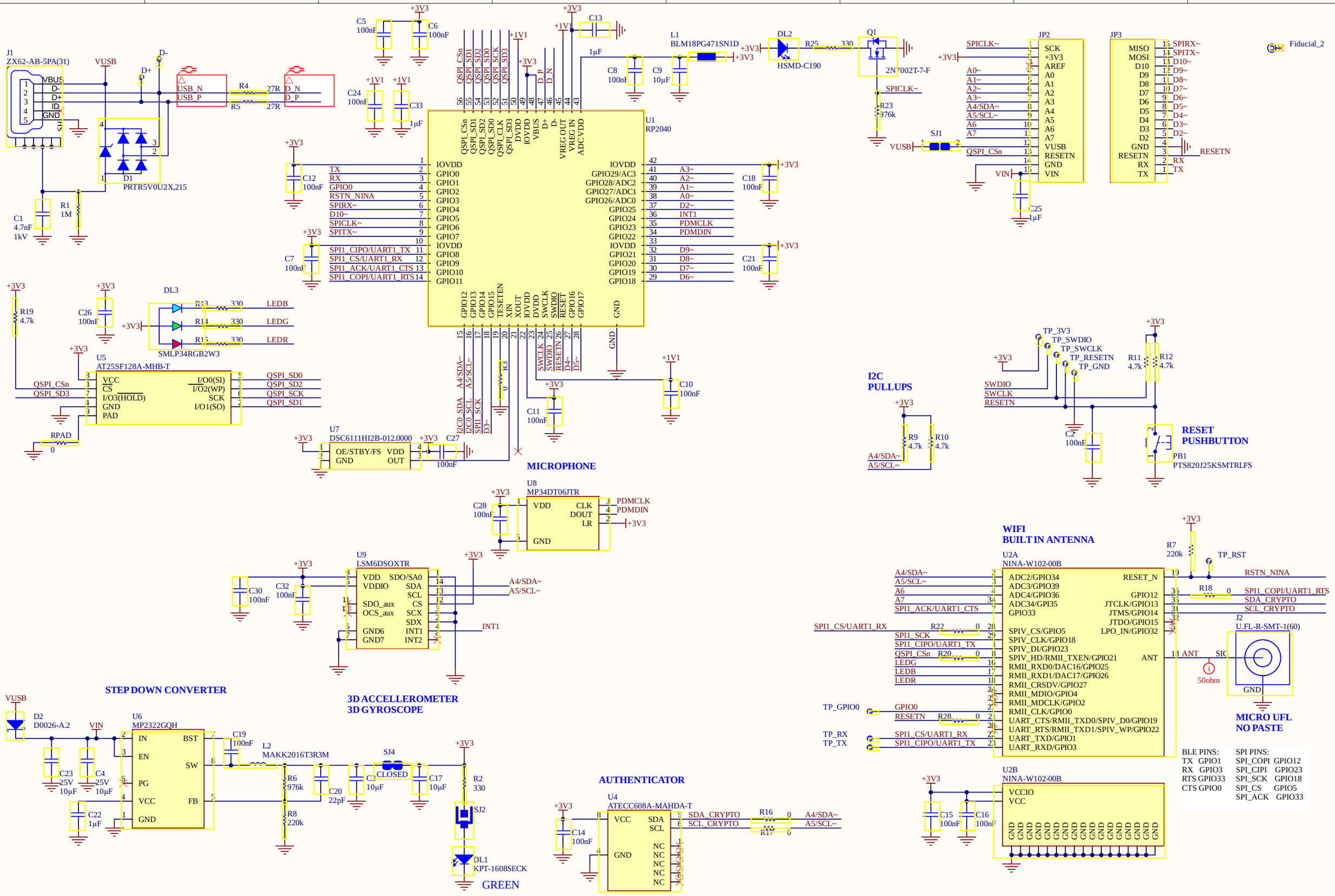


Target Areas

Internet of Things (IoT), machine learning, prototyping,

Features

- **Raspberry Pi RP2040** Micrcontroller
  - 133MHz 32bit Dual Core Arm® Cortex®-M0+
  - 264kB on-chip SRAM
  - Direct Memory Access (DMA) controller
  - Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
  - USB 1.1 controller and PHY, with host and device support
  - 8 PIO state machines
  - Programmable IO (PIO) for extended peripheral support
  - 4 channel ADC with internal temperature sensor, 0.5 MSa/s, 12-bit conversion
  - SWD Debugging
  - 2 on-chip PLLs to generate USB and core clock
  - 40nm process node
  - Multiple low power mode support
  - USB 1.1 Host/Device
  - Internal Voltage Regulator to supply the core voltage
  - Advanced High-performance Bus (AHB)/Advanced Peripheral Bus (APB)



Title: Nano RP2040 Connect		
ID: ABX00053	Revision: V3.4	
Date: 5/10/2021 Time: 2:08:50 PM	Sheet 1 of 1	
File: TOP.SchDoc	Author: Arturo Guadalupi	RevAuthor: Silvio Navaretti

ARDUINO

life.augmented

LSM6DSOX

Datasheet

6-axis IMU (inertial measurement unit) with embedded AI: always-on 3-axis accelerometer and 3-axis gyroscope

LGA-14L
(2.5 x 3.0 x 0.83 mm) typ.

Product status link
LSM6DSOX

Product summary		
Order code	LSM6DSOX	LSM6DSOXTR
Temperature range [°C]	-40 to +85	
Package	LGA-14L (2.5 x 3.0 x 0.83 mm)	
Packing	Tray	Tape and reel

Product resources
AN5272 (device application note)
AN5273 (finite state machine)
AN5259 (machine learning core)
TN0018 (design and soldering)

Product label
SUSTAINABLE TECHNOLOGY

Features

- Supply current: 0.55 mA in combo high-performance mode
- “Always-on" experience with low power consumption for both accelerometer and gyroscope
- Smart FIFO up to 9 KB
- Android compliant
- ±2/±4/±8/±16 *g* full scale
- ±125/±250/±500/±1000/±2000 dps full scale
- Analog supply voltage: 1.71 V to 3.6 V
- Independent IO supply (1.62 V)
- Compact footprint: 2.5 mm x 3 mm x 0.83 mm
- SPI / I²C & MIPI I3C^{SM} serial interface with main processor data synchronization
- Auxiliary SPI for OIS data output for gyroscope and accelerometer
- OIS configurable from aux SPI, primary interface (SPI / I²C & MIPI I3C^{SM})
- Advanced pedometer, step detector, and step counter
- Significant motion detection, tilt detection
- Standard interrupts: free-fall, wake-up, 6D/4D orientation, click and double-click
- Programmable finite state machine: accelerometer, gyroscope, and external sensors
- Machine learning core
- S4S data synchronization
- Embedded temperature sensor
- ECOPACK and RoHS compliant

Applications

- Motion tracking and gesture detection
- Sensor hub
- Indoor navigation
- IoT and connected devices
- Smart power saving for handheld devices
- EIS and OIS for camera applications
- Vibration monitoring and compensation

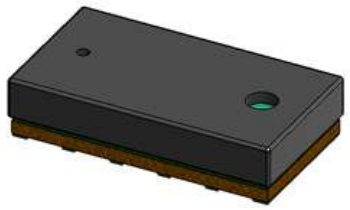
Description

The LSM6DSOX is a 6-axis IMU (inertial measurement unit) system-in-package featuring a 3-axis digital accelerometer and a 3-axis digital gyroscope, boosting performance at 0.55 mA in high-performance mode and enabling always-on low-power features for an optimal motion experience for the consumer.

VL53L0X

Datasheet

Time-of-Flight ranging sensor



Product status link
VL53L0X

Features

Fully integrated miniature module

- 940 nm laser VCSEL (vertical-cavity surface-emitting laser)
- VCSEL driver
- Ranging sensor with advanced embedded microcontroller
- 4.4 x 2.4 x 1.0 mm

Fast, accurate distance ranging

- Measures absolute range up to 2 m
- The reported range is independent of the target reflectance
- Advanced embedded optical crosstalk compensation to simplify cover glass selection

Eye safety

- Class 1 laser device compliant with latest standard IEC 60825-1:2014 - 3rd edition

Easy integration

- Single reflowable component
- No additional optics
- Single power supply
- I²C interface for device control and data transfer
- Xshutdown (reset) and interrupt GPIO
- Programmable I²C address

Application

- Access control (system activation and presence detection)
- Robotics (collision avoidance, wall tracking, and cliff detection)
- Home appliance and home automation
- Inventory management and liquid level monitoring

Description

The VL53L0X is a Time-of-Flight (ToF) laser-ranging module housed in the smallest package on the market today, providing accurate distance measurement whatever the target reflectance, unlike conventional technologies. It can measure absolute distances up to 2 m, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The VL53L0X integrates a leading-edge SPAD array (single photon avalanche diodes) and embeds ST's second generation FlightSense patented technology.

The VL53L0X's 940 nm VCSEL emitter (vertical cavity surface-emitting laser), is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distances, higher immunity to ambient light, and better robustness to cover glass optical crosstalk.

24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales

DESCRIPTION

Based on Avia Semiconductor's patented technology, HX711 is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

The input multiplexer selects either Channel A or B differential input to the low-noise programmable gain amplifier (PGA). Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of ±20mV or ±40mV respectively, when a 5V supply is connected to AVDD analog power supply pin. Channel B has a fixed gain of 32. On-chip power supply regulator eliminates the need for an external supply regulator to provide analog power for the ADC and the sensor. Clock input is flexible. It can be from an external clock source, a crystal, or the on-chip oscillator that does not require any external component. On-chip power-on-reset circuitry simplifies digital interface initialization.

There is no programming needed for the internal registers. All controls to the HX711 are through the pins.

FEATURES

- **Two selectable differential input channels**
- **On-chip active low noise PGA with selectable gain of 32, 64 and 128**
- **On-chip power supply regulator for load-cell and ADC analog power supply**
- **On-chip oscillator requiring no external component with optional external crystal**
- **On-chip power-on-reset**
- **Simple digital control and serial interface: pin-driven controls, no programming needed**
- **Selectable 10SPS or 80SPS output data rate**
- **Simultaneous 50 and 60Hz supply rejection**
- **Current consumption including on-chip analog power supply regulator: normal operation < 1.5mA, power down < 1uA**
- **Operation supply voltage range: 2.6 ~ 5.5V**
- **Operation temperature range: -40 ~ +85℃**
- **16 pin SOP-16 package**

APPLICATIONS

- **Weigh Scales**
- **Industrial Process Control**

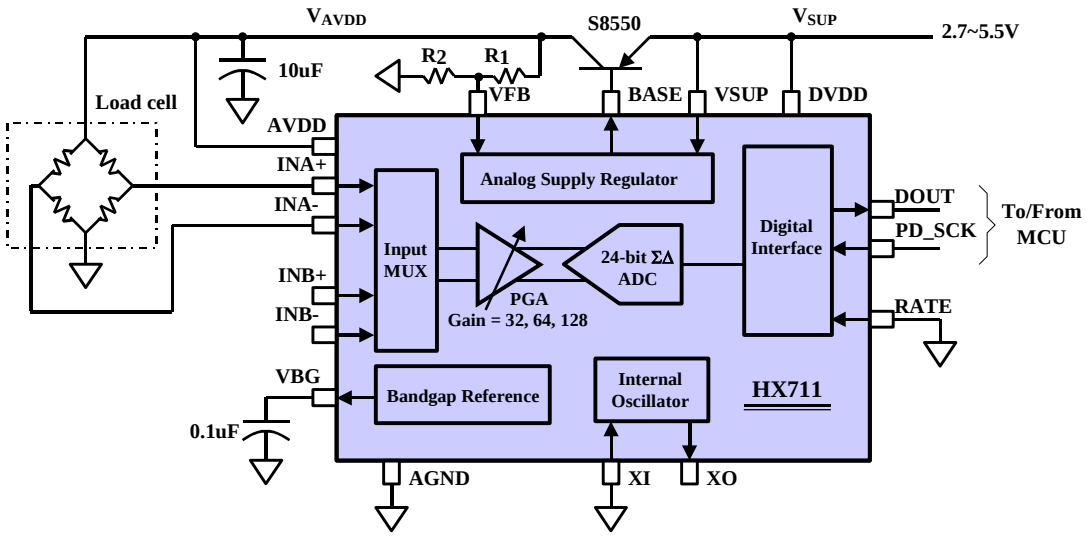


Fig. 1 Typical weigh scale application block diagram

30A BLDC ESC

Figure 1: 30A BLDC ESC

Introduction

This is fully programmable 30A BLDC ESC with 5V, 3A BEC. Can drive motors with continuous 30Amp load current. It has sturdy construction with 2 separate PCBs for Controller and ESC power MOSFETs. It can be powered with 2-4 lithium Polymer batteries or 5-12 NiMH / NiCd batteries. It has separate voltage regulator for the microcontroller for providing good anti-jamming capability. It is most suitable for UAVs, Aircrafts and Helicopters.

Specifications

- Output: 30A continuous; 40Amps for 10 seconds
- Input voltage: 2-4 cells Lithium Polymer / Lithium Ion battery or 5-12 cells NiMH / NiCd
- BEC: 5V, 3Amp for external receiver and servos
- Max Speed: 2 Pole: 210,000rpm; 6 Pole: 70,000rpm; 12 Pole: 35,000rpm
- Weight: 32gms
- Size: 55mm x 26mm x 13mm

Features:

- High quality MOSFETs for BLDC motor drive
- High performance microcontroller for best compatibility with all types of motors at greater efficiency
- Fully programmable with any standard RC remote control
- Heat sink with high performance heat transmission membrane for better thermal management
- 3 start modes: Normal / Soft / Super-Soft, compatible with fixed wing aircrafts and helicopters
- Throttle range can be configured to be compatible with any remote control available in the market

A2212 1400KV Brushless Motors

PRODUCT DESCRIPTION

- 1400KV UAV brushless motor
- Ideal to work with 30A ESC
- Efficiency current (maximum): 6-12A (>75%)
- Current (no load): 0.7A

The UAV Brushless Motor A2212/10 1400Kv provides excellent performance, quality, and dependability at an affordable price. Manufactured with high quality bearings and components, this brushless motor is one of the smoothest and most powerful in its class. Each motor is pre-solderd with 3.5mm male gold bullet connectors and includes an accessory pack with collet-style prop adapter, X-mount plate and matching set of female 3.5mm bullet connectors.

Specifications :

- Number of cells: 2-3s Lipo; 6-10 cell NiMh
- Maxium efficiency current: 6-12A (>75%)
- No load current (10v): 0.7A
- Maximum current: 16A/60s

- Maximum efficiency: 78%
- Internal resistance: 65mΩ
- Poles: 14
- Maximum watts: 180W
- Motor bolt pattern (back of motor): 16mm x 19mm (hole to hole)
- X-mount hole pattern: 32mm x 32mm (hole to hole)

Dimensions

- Motor size: 27.5mm x 30mm
- Shaft size: 3.17mm
- Weight: 47g